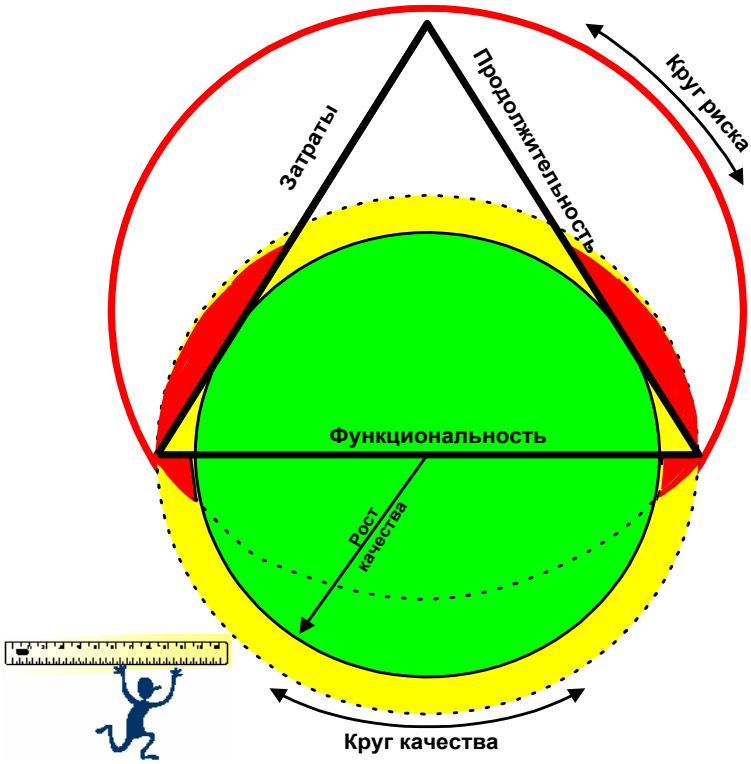
Ф.И. Андон
Г.И. Коваль
Т.М. Коротун
Е.М. Лаврищева
В.Ю. Суслов

ОСНОВЫ ИНЖЕНЕРИИ КАЧЕСТВА ПРОГРАММНЫХ СИСТЕМ

Второе издание



НАЦИОНАЛЬНАЯ АКАДЕМИЯ НАУК УКРАИНЫ
ИНСТИТУТ ПРОГРАММНЫХ СИСТЕМ

Ф.И. Андон, Г.И. Коваль, Т.М. Коротун,

Е.М. Лаврищева, В.Ю. Суслов

Основы инженерии качества программных систем

Второе издание

Издательский дом
«Академпериодика»

Киев • 2007

ББК 32.973.
УДК 681.3.06
0753

Андон Ф.И., Коваль Г.И., Коротун Т.М., Лаврищева Е.М., Суслов В.Ю. Основы инженерии качества программных систем. 2-е изд., перераб. и доп. - К.: Академпериодика, 2007. – 672 с.

Определяется ядро знаний в области инженерии качества программных систем. Рассматриваются процессы жизненного цикла, связанные с обеспечением качества – процессы верификации, валидации, тестирования, измерения, управления риском, обеспечения гарантии качества, управления качеством и др. Предлагаются обзоры методов, используемых при выполнении этих процессов. Подробно рассматриваются методологии GQM, FPA, COCOMO II, CMM и др. Обзоры основаны на материалах (отчетах, публикациях) зарубежных организаций-лидеров в области исследования вопросов инженерии качества – SEI, ISO, IEEE, NIST и др.

Для руководителей организаций-разработчиков программных систем, менеджеров проектов, инженеров по качеству, аспирантов и студентов вузов.

Визначається ядро знань у галузі інженерії якості програмних систем. Розглядаються процеси життєвого циклу, що стосуються забезпечення якості – процеси верифікації, валідації, тестування, вимірювання, управління ризиком, забезпечення гарантії якості, управління якістю та ін. Пропонуються огляди методів, що використовуються при виконанні цих процесів. Докладно розглядаються методології GQM, FPA, COCOMO II, CMM та інші. Огляди базовані на матеріалах (звітах, публікаціях) провідних зарубіжних організацій у галузі дослідження питань інженерії якості – SEI, ISO, IEEE, NIST тощо.

Для керівників організацій-розробників програмних систем, менеджерів проектів, інженерів з якості, аспірантів та студентів вузів.

Ответственный редактор академик НАН Украины И.В. Сергиенко

Утверждено к печати ученым советом
Института программных систем НАН Украины

ISBN 966-8002-41-5

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Неуклонное повышение качества выпускаемых программных продуктов - основная цель программной инженерии. Вопросами качества озабочены не только разработчики ПС для зарубежного заказчика (уровень требований которого всегда был высок), но и организации, работающие для украинского заказчика или непосредственно на украинский рынок потребителей. Это, в первую очередь, объясняется обострением конкуренции разработчиков, а, кроме того, - повышением уровня знаний потребителей о качестве ПС и, как следствие, ростом их требований к качеству.

В последнее время происходит консолидация усилий передовых фирм-разработчиков ПС по созданию национальной системы обеспечения качества. Об этом свидетельствуют сообщения на сайтах Украинской Ассоциации Производителей программного обеспечения (УАППО)[1] и Ассоциации «Информационные технологии Украины»[2], объединяющих такие известные компании, как «Миратех», «Софтлайн», «Мирасофт», «Профикс», «Укрсофт», «Софтсерв» и др. Нужно надеяться, что слияние научного и производственного потенциала Украины в области программной инженерии позволит создавать программные продукты, отвечающие международным стандартам и способные конкурировать на международном рынке программной продукции.

За рубежом существует множество фирм, осуществляющих регулярные обзоры и исследование состояния дел по различным аспектам программной индустрии в разных странах мира, например, NASSCOM (National Association of software and service companies)[3], Standish Group International, Inc.[4], IBM's Consulting Group[5]. По результатам этих исследований можно судить о том, произошли ли существенные изменения в области качества зарубежной программной индустрии.

Для сравнения, обратимся к результатам исследований, проведенных Standish Group в 1994 и в 2003 годах.

В 1994 году обзор 8380 программных проектов, разрабатываемых в государственном и частном секторах США, показал, что[6]:

- 31% всех программных проектов были закрыты до их полного завершения;
- 53% успешно завершенных проектов в среднем на 18% превысили начальную стоимость;
- из этих 53% проектов только 42% обеспечили в программных продуктах все изначально требуемые свойства и функции;
- только 9% проектов были завершены в срок и уложились в предусмотренные для проекта денежные средства.

Обзор 13522 проектов в 2003 году показал, что в 34% проектов состояние разработок улучшилось на 16% по сравнению в 1994 годом. Однако 51% проектов все еще сталкивались с проблемами. Из них:

---

[1] Адрес УАППО в Интернет: http://www.uaswd.org.ua

[2] Адрес «ИТ-Украины» в Интернет: http://www.itukraine.org.ua

[3] Адрес NASSCOM в Интернет: http://www.nasscom.org

[4] Адрес Standish Group в Интернет: http://www.standishgroup.com

[5] Адрес в Интернет: http://www.ibm.com/annualreport/2004/ или http://www-1.ibm.com

[6] *Gibbs W.* Software's Chronic Crisis // Scientific America, September 1994. - P. 86 - 95

- 15% проектов были закрыты до полного завершения;
- 46% успешно завершенных проектов превысили начальную стоимость в среднем на 10-15%;
- из 85% завершенных проектов изначально требуемые свойства и функции были обеспечены в программных продуктах только на 52%.

Таким образом, несмотря на достигнутый прогресс, разработка качественной программной продукции все еще остается проблемой.

Авторы не располагают данными о результатах подобных исследований в Украине, однако нет оснований считать, что состояние дел с отечественными программными проектами лучше. К сожалению, отечественная индустрия программного обеспечения пока не обладает достаточной нормативно-методической поддержкой разработки высококачественных и конкурентоспособных программных систем. Хотя за последние годы Госстандартом Украины[7] принято более 10 новых государственных стандартов в области программной инженерии и качества, их практическое применение затруднено не только отсутствием каких-либо методических рекомендаций по каждому стандарту, но и единого концептуального взгляда на проблему качества ПС и пути ее решения. Не определена сама парадигма качества ПС.

На вопросы о том, как повысить конкурентоспособность украинских программных продуктов, как снизить риск проектов, как достичь баланса сторон треугольника «продолжительность – стоимость – цели проекта», в течение последних двух десятилетий пытаются найти ответ руководители организаций-разработчиков ПС, менеджеры проектов, а также заказчики и потребители, приобретающие программные продукты невысокого качества.

Тайна нестабильности проектов, недолговечности и низкого качества программных продуктов кроется в недрах *процессов* разработки. Основные проблемы коренятся в *неспособности эффективно управлять* разработкой ПС. Даже самые хорошие методы, инструменты и технологии не спасают положения, поскольку *не могут* быть рационально использованы в рамках недисциплинированного, хаотического проекта. Качество программного продукта остается непредсказуемым, так как нет объективного базиса для его достижения и оценивания.

Изменить ситуацию можно только в результате концентрации усилий на создании инфраструктуры для *поддержки процесса* эффективной программной инженерии и инженерии качества ПС.

Нужно заметить, что подходы к усовершенствованию процессов создания ПС разрабатывались в Украине еще в 80-е годы прошлого столетия. Направление исследований тогда определяла международная Комплексная программа научно-технического прогресса стран-членов СЭВ до 2000 года. В ней программная продукция впервые была отнесена к продукции производственно-технического назначения и была сформулирована проблема «Развитие технологии разработки и *промышленного производства* программных средств вычислительной техники».

---

[7] Правильнее было бы дать украинское наименование организации – «Держспоживстандарт України»

В рамках этой программы в СКТБ ПО Института кибернетики АН УССР им. В.М.Глушкова[8] были выработаны основы *технологической подготовки* разработки программных изделий (ТПР ПИ), - этапа, предшествующего непосредственной разработке ПС, и предназначенного для построения базового процесса программной инженерии в организации и его адаптации для каждого конкретного программного проекта. Отделом технологии программирования под руководством Е.М. Лаврищевой были разработаны серии документов ТПР (стандартов предприятия)[9], в которых регламентировались процессы ТПР и собственно разработки ПС в СКТБ ПО и устанавливались требования к структуре и содержанию программно-технологических документов – карт технологических линий и процессов, технологических маршрутов, форм и т.д. С развалом СССР эти работы были приостановлены, их результаты остались не востребованы, а «процессный подход» забыт.

Теперь, спустя годы, виток спирали развития программной инженерии возвращает нас к процессо-ориентированному подходу, полагаемому в основу современной парадигмы качества ПС. Его отражение можно найти, например, во множестве международных стандартов ISO, а также гармонизованных с ними стандартов стран СНГ, включая Украину. Методологические основы применения процессо-ориентированного подхода в разработке ПС в Украине развиваются в Институте программных систем НАН Украины (ИПС НАНУ) в рамках выполняемых академических научно-исследовательских программ.

Авторы книги (сотрудники ИПС НАНУ) предлагают читателю материал, который поможет сформировать представление об *инженерии качества* ПС, как об одном из основных направлений деятельности в программных проектах, а также получить некоторые методические рекомендации по применению приобретенных знаний на практике.

В книге сконцентрирован материал, полученный в результате анализа и обобщения международного, отечественного и собственного опыта авторов в разработке высококачественных программных систем, а также стандартов в области инженерии качества. В нем, в частности, нашли отражение идеи известнейших специалистов в области качества ПС – Н. Фентона, У. Хамфри, М.Полка, В.В. Липаева и многих-многих других, опубликованные в открытой печати и Интернет.

Основная цель написания этой книги – показать место инженерии качества ПС среди множества (более сорока) процессов программной инженерии, представленных в базовом международном стандарте ISO/IEC 12207 «Процессы жизненного цикла программных средств». Поскольку этот стандарт определяет только то, ЧТО (какие действия) должны выполняться в процессах, и не указывает КАК (какими методами) их выполнять, авторы предлагают обзоры самых современных методов выполнения процессов, связанных с качеством, разработанных ведущими организациями, специализирующимися в области программной инженерии – SEI (США), Microsoft, ISO, IEEE, NIST и другими. Наиболее известные методы (например, метод измерений

---

[8] в 1992 году на базе СКТБ ПО создан Институт программных систем (ИПС) НАН Украины.

[9] Е.М.Лаврищева. Основы технологической подготовки разработки прикладных программ систем обработки данных // Препринт / АН УССР, Ин-т кибернетики им. В.М.Глушкова; 87-5. – Киев, 1987. – 30с.

GQM или метод инспекции) излагаются достаточно подробно, «по шагам», и сопровождаются множеством форм (шаблонов) и метрик, применяемых при выполнении процессов. Однако нужно отметить, что все описания процессов, методов и форм должны *адаптироваться* для применения в реальных проектах.

Перед Вами второе *переработанное* и *дополненное* издание книги «Основы инженерии качества программных систем». Можно утверждать, что интерес к проблеме качества со стороны отечественных специалистов в области разработки программных систем существенно возрос со времени ее первого издания (в 2002 году).

Книга состоит из 12 глав.

В *первой главе* «Парадигма качества в программной инженерии» определено множество взаимосвязанных понятий в области инженерии качества ПС: процессы ЖЦ, модели ЖЦ, проект, измерение, риск, качество и другие. Акцент сделан на *процессо-ориентированном* подходе к разработке ПС. Указаны процессы, которые должны институциализироваться в организации и *интегрироваться* в проекты ПС для обеспечения качества. Центральное место отведено процессу измерения, результаты выполнения которого используются всеми другими процессами в проекте. Сформулированы основные положения концепции инженерии качества ПС. В приложении 1 к главе дан краткий обзор моделей ЖЦ ПС.

Во *второй главе* «Инженерия качества. Ядро профессиональных знаний» представлены основные элементы ядра профессиональных знаний в областях программной инженерии (SWEBOK) и управления проектами (PMBOK) и на их основе синтезировано множество элементов, составляющих, по мнению авторов, *ядро знаний в области инженерии качества* ПС.

Третья глава «Модели и метрики качества» посвящена метрикам – основе измерений продуктов, процессов и ресурсов в проекте. В ней приведена классификация мер и метрик качества. Дан краткий обзор существующих моделей качества, включая графические. Представлена парадигма «встраивания» качества в программные продукты QFD (Quality Function Deployment). В помощь менеджерам проектов в приложении к главе (приложение 2) приведены примеры метрик для некоторых характеристик качества ПС.

В *четвертой главе* «Измерения в программной инженерии» рассмотрен основополагающий процесс для инженерии качества - *процесс измерения*. Описаны его цели, задачи и объекты, а также место в архитектуре процессов ЖЦ. В центре внимания главы – *целеориентированный* подход к измерениям, основанный на методологии «цель – вопрос – мера» GQM (Goal - Question - Metric).

Пятая глава «Контроль и гарантия качества» посвящена процессам и методам обеспечения качества. В ней разграничены понятия обеспечения гарантии качества (SQA, Software Quality Assurance) и управления качеством (Quality Management), определено место одноименных процессов в архитектуре процессов ЖЦ. Введено понятие *профиля* процесса, связывающего воедино определение процесса, план его выполнения, метрики, а также требования (стандарты) для процесса и его рабочих продуктов, методы и инструменты, и, наконец, требования к профессиональной зрелости кадров, выполняющих процесс. Дана характеристика каждого элемента профиля. Очерчен примерный круг знаний, которыми (по мнению авторов) должны

обладать инженеры по качеству (или SQA-менеджеры). Рассмотрены методы и инструменты анализа качества (от простейших графических, до методов datamaning (добычи данных) и байесовских сетей). В помощь менеджерам в приложении 3 представлен перечень международных и отечественных *стандартов в области инженерии качества* и дана краткая характеристика проектов ISO/IEC по этому направлению. В приложении 4 кратко описаны средства инструментальной поддержки качества в продуктах ведущих поставщиков – Oracle (Oracle Designer'2000), Microsoft (Microsoft Visual Studio 2005), Rational (Rational Suite), Borland (Borland Suite)

В *шестой главе* «Процессы и методы проверки» рассмотрен другой очень важный аспект обеспечения качества – его *проверка* на стадиях ЖЦ. Процедуры проверки качества сосредоточены в процессах верификации, валидации, совместного просмотра и аудита. Установлены общие черты и отличительные особенности каждого из этих процессов. Подробно описаны задачи верификации и валидации рабочих продуктов ПС на стадиях ЖЦ. Дана классификация и краткий обзор методов, применяемых в процессах проверки. Это методы индивидуального анализа рабочих продуктов, а также методы коллективной проверки (технический и управленческий обзоры, сквозной контроль, аудиторская проверка). Подробно рассмотрен метод *инспекции* – наиболее формализованный и широко применяемый метод проверки рабочих продуктов ПС, а в приложении 5 приведены примеры вопросов для ее проведения.

Седьмая глава «Тестирование программных систем» посвящена тестированию. В ней предложен подход к тестированию с учетом риска отказов программного продукта и риска срыва проекта. Тестирование рассматривается как *интегрированный* процесс, действующий *на всех* стадиях ЖЦ ПС. В нем собраны воедино все действия группы тестирования, «размытые» по другим процессам ЖЦ, начиная с процесса выявления требований и кончая процессом инсталляции ПС. Такой подход не только позволяет четко определить задачи группы тестирования, но и служит залогом правильного определения трудоемкости тестирования. В главе представлен также обзор методов тестирования.

Восьмая, девятая и десятая главы охватывают вопросы оценивания размера, стоимости и риска разработки ПС. Хотя они, казалось бы, не имеют непосредственного отношения к обеспечению качества, однако тесно связаны с управлением проектом ПС. Проект должен обеспечивать разработку ПС *определенного размера* (здесь размер выступает мерилом функциональности ПС), *требуемого качества*, в *отведенные сроки*, за *выделенные деньги*, и с *наименьшим риском* для того же размера, качества, сроков и финансирования.

Восьмая глава «Методы оценки размера программной системы» посвящена методам определения функционального размера ПС, в основе которых лежит методология *анализа показателей функциональности* FPA (Function Points Analysis), впервые использующая для оценки размера взгляд *пользователей* на множество функций, которые должна выполнять система. Читателям предлагается обзор методов, используя которые можно оценить (прогнозировать) размер ПС не подсчитывая число строк исходного кода (который появляется, увы, на более поздних стадиях разработки), а с самого начала – анализируя предметную область, функциональные требования и интервьюируя пользователей будущей ПС. Размер определяется в

условных единицах функциональности, которые, при необходимости, могут конвертироваться в строки кода для применения в моделях надежности, стоимости и тестирования.

В девятой главе «Оценка затрат на разработку программных систем» представлены модели и методы расчета трудоемкости и затрат на разработку ПС. Кратко описаны методы Дельфи, SLIM, ANGEL, а также одна из самых известных моделей - трехуровневая модель COCOMO II и ее расширения. В основе расчетных формул модели – размер ПС, а параметрами настройки служат факторы, учитывающие особенности самого процесса разработки.

В *десятой главе* «Управление риском проектов» рассмотрены основные понятия и концепции парадигмы управления риском проектов, предложенной SEI (Software Engineering Institute) - одним из ведущих институтов США в области программной инженерии. В приложении к главе (приложение 6) содержится вопросник, согласующийся с таксономией риска, который может быть использован для анализа и оценки риска отечественных проектов.

Одиннадцатая глава «Методологии повышения качества в современной парадигме» посвящена современным подходам к управлению качеством ПС (статистический контроль качества, всеобщее управление качеством TQM, лестницы восхождения к качеству, MSF, RUP, а также Agile-подходы XP, SCRUM, DSDM и др.). Дана их краткая характеристика и сравнительный анализ.

В *двенадцатой главе* «Подходы к оценке качества программных систем» рассмотрены вопросы *оценки качества*. Здесь с понятием *качество* связываются не только характеристики качества готовых программных продуктов, но и *зрелость* организации-разработчика ПС, и *мощность* процессов ЖЦ. В главе кратко описана широко известная модель зрелости - CMM (Capability Maturity Model), основные процедуры оценивания мощности процессов ЖЦ, регламентируемые стандартом ISO/IEC 15504, а также методы оценки качества продуктов и сертификации программных продуктов и систем качества.

Книга предназначена для руководителей организаций-разработчиков программных систем, менеджеров проектов и инженеров по качеству. Она может служить основой для построения спецкурсов по инженерии качества ПС, поскольку содержит множество ссылок на литературные источники (более 200 ссылок) и адреса сайтов в Интернете, где можно получить более детальную информацию по рассматриваемым вопросам. Книга может также использоваться студентами старших курсов и аспирантами по специальности 01.05.03 – Математическое и программное обеспечение вычислительных машин и систем - не только для расширения кругозора, но и в качестве отправной точки для выбора направлений собственных исследований в области инженерии качества.

Авторы будут благодарны читателям за *любые* отклики на книгу. Просим направлять их по адресу *Института программных систем*

03187, Украина, г. Киев, просп. Академика В.М. Глушкова, 40; ИПС НАНУ

телефон (044) 526 55 07; факс 526 62 63;

e-mail: iss@isofts.kiev.ua (адрес ИПС НАН Украины)

Глава 1. ПАРАДИГМА КАЧЕСТВА В ПРОГРАММНОЙ ИНЖЕНЕРИИ

1.1. Основные понятия в области качества

1.1.1. Инфраструктура разработки

В последние годы программная индустрия достигла такого уровня развития, при котором требования к обеспечению качества стали обязательным пунктом заключаемых договоров на разработку *программных систем* (ПС).

Программная система - группа *интегрированных* программных средств, поддерживающих определенный деловой процесс потребителя (или его часть) и использующих общие хранилища данных. Пример ПС - группа программных приложений, составляющих автоматизированное рабочее место (АРМ) специалиста в определенной предметной области.

Термин *программное обеспечение* (ПО) используется применительно к совокупности *программных средств*, приобретаемых или разрабатываемых с целью эксплуатации в составе системы (системы автоматизации, информационные системы и др.). Примеры программных средств: шрифт, драйвер, программное приложение. Мы говорим «программное обеспечение системы», если хотим подчеркнуть его принадлежность к системе (включающей также техническое, организационное и другие виды обеспечения системы).

Основа качественной разработки программных систем – рациональная *инфраструктура программной инженерии* как вида бизнеса.

Желание компаний получить наивысшую отдачу от инвестиций в создание ПС побуждает их постоянно возвращаться к вопросу о текущей практике разработки. Разработка ПС как вид бизнеса прошла за последние 30 лет тернистый путь от «закрученных» кодов на Коболе, Фортране и ПЛ/1, создаваемых исключительно в отделах Вычислительно-информационных центров, до приложений, проектируемых, моделируемых и разрабатываемых профессионалами в области информационных систем практически в постоянном взаимодействии с конечными пользователями.

К сожалению, разработчики ПС нечасто завершают программные проекты вовремя, укладываются в рамки бюджета и полностью отражают в программных продуктах требования спецификаций заказчика. Последствия поставки плохо разработанных программных продуктов - это всегда убытки клиентов: не только материальные и финансовые потери, но и падение престижа. Эти проблемы, в свою очередь, негативно отражаются на конкурентоспособности организации-разработчика программных продуктов.

И практикующие специалисты в области программных систем, и пользователи сходятся во взглядах на понятие плохого программного продукта как такого, который:

- не обеспечивает поддержку стратегии бизнеса или потребностей пользователей или
- недостаточно надежен, гибок, эффективен и плохо сопровождается или
- является дорогим и слишком долго разрабатываемым.

Причины плохого состояния практики разработки ПС заключаются в том, что организация-разработчик страдает синдромом «сапожник без сапог» из-за отсутствия надлежащей инфраструктуры разработки. Разработчики программных продуктов испытывают недостаток в современных автоматизированных инструментальных средствах, методологии разработки, обучении специалистов. Кроме того, из-за неадекватной инфраструктуры, сообществу их потенциальных пользователей не отводится ведущая роль в определении методологий эксплуатации ПС, стандартных процедур обработки деловой информации, планов обеспечения ее безопасности и пр. Поэтому многие пользователи прерывают применение программных продуктов, которые не только плохо разработаны и несовместимы, но и подвергают опасности целостность данных компании. Пользователи сталкиваются с недостатками в программных продуктах из-за того, что те не имеют завершенной технологии поддержки *всего жизненного цикла ПС*, начиная от ее замысла и кончая списанием. Многие организации-разработчики продолжают использовать тот же ограниченный и не интегрированный набор инструментальных средств и методологий для построения ПС, что и десятилетие тому назад. Эта практика продолжается, несмотря на существенный прогресс в применяемых пользователями основных информационных технологиях и повышение их требований к разрабатываемым по договорам программным продуктам. Руководство организаций-разработчиков по техническим, экономическим и социальным причинам запаздывает с созданием эффективной инфраструктуры, способной обеспечить специалистов инструментальными средствами и методологиями, поддерживающими разработку высококачественных ПС.

Инфраструктура программной инженерии - интегрированный набор общедоступных технических, технологических и методологических ресурсов организации-разработчика, которые делают возможным выполнение *процесса программной инженерии* коллективами *проектов*, открываемых по договорам с заказчиком программных продуктов.

В данном изложении ***проект*** – это ограниченная временными рамками деятельность, цель которой состоит в создании уникального программного продукта; ***процесс программной инженерии*** – это множество логически связанных видов деятельности по определению, проектированию и построению программных продуктов (прикладных программных систем).

Совместное использование ресурсов инфраструктуры позволяет организации повышать продуктивность исполнения программных проектов, создавая потенциал для совершенствования качества программных систем и снижения их стоимости. Концептуальный взгляд на инфраструктуру программной инженерии представлен на рисунке 1.1.

Нет однозначного ответа на вопрос, какова наиболее рациональная инфраструктура разработки (ИР) программных систем. Варианты ее построения зависят от таких факторов, как корпоративный взгляд на информационную технологию, типы проектов ПС, организационная структура проектов (роли, распределяемые между различными специалистами в проекте, и уровни их ответственности). В таблице 1.1 описано множество компонентов инфраструктуры, представленных на рисунке 1.1, а в таблице 1.2 – объяснены роли специалистов в организационных структурах уровня отдельных проектов и организации в целом, имеющих непо-

средственное отношение к обеспечению качества разработки. В небольших проектах или организациях допускается совмещение нескольких ролей одним лицом.

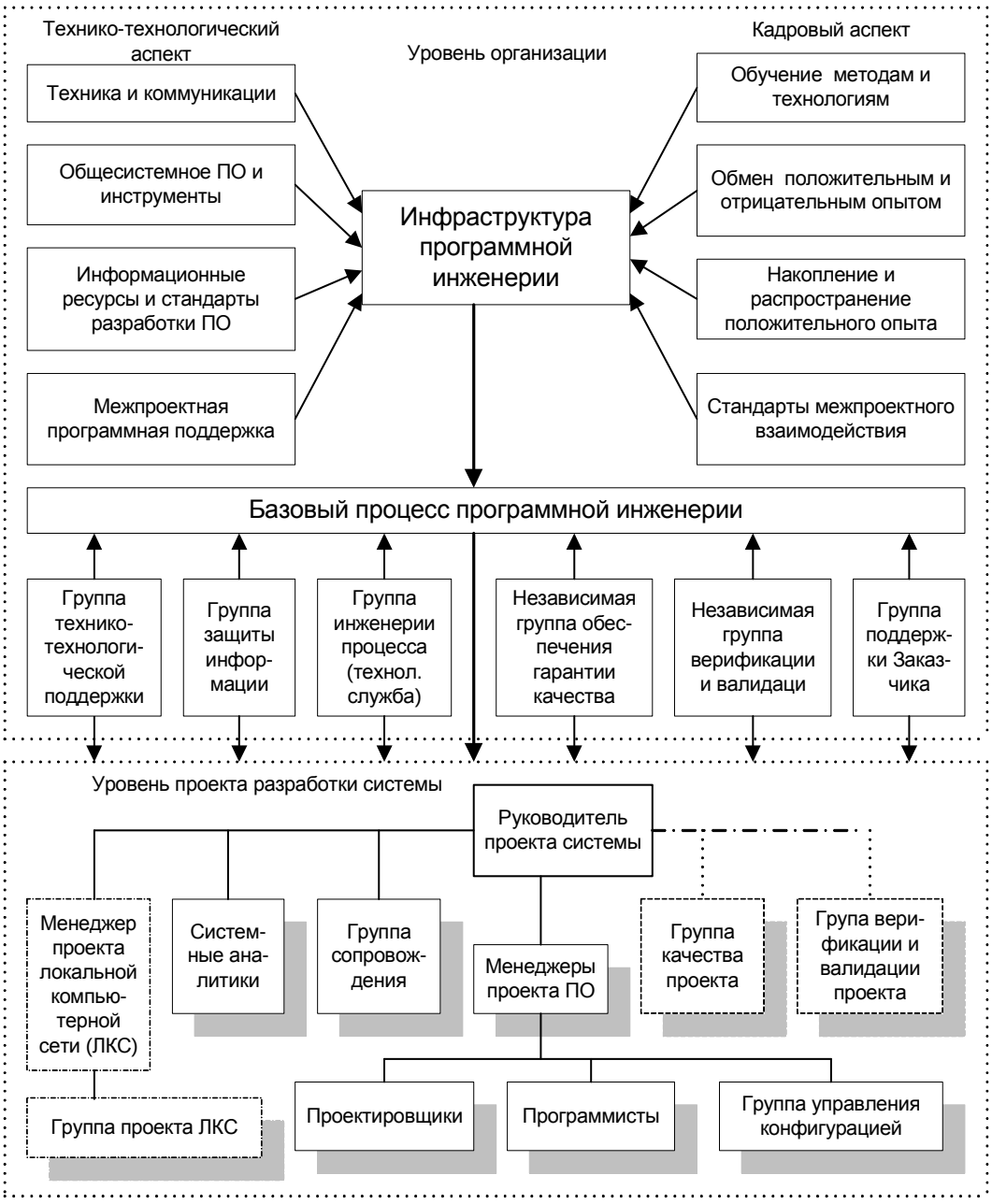


Рис. 1.1. **Инфраструктура разработки программных систем**

Инфраструктура образует фундамент для осуществления процесса программной инженерии в организации.

Таблица 1.1. **Компоненты инфраструктуры разработки ПС**

Технико-технологический аспект ИР	**Описание возможностей**
Техника и коммуникации	• Компьютеры пользователей, файловые серверы • Локальные компьютерные сети (ЛКС) • Глобальная компьютерная сеть (ГКС) • Электронная почта (e-mail) • Техника для тестирования (тестеры) • Офисная техника (сканеры, принтеры, проекторы и т.п.) • Другие составляющие комплекса технических средств (КТС)
Общесистемное ПО (ОПО) и инструменты	•Клиент/Серверные технологии • Операционные системы • Офисные системы (включая ридеры/райтеры) • Системы документооборота • Утилиты (архиваторы, программы записи на оптические носители и т.п.) • Средства защиты информации (антивирусные средства) • CASE-инструменты, системы программирования (4GL) • СУБД • Графические инструменты
Информационные ресурсы и стандарты разработки	• Методологии разработки (в частности, поддержанные CASE-средствами) • Инструменты управления проектами, конфигурациями • Системы поддержки использования ресурсов Интернет (выгрузка/загрузка информации) • Нормативные документы, касающиеся технических, программных, коммуникационных средств, данных и защиты информации • Нормативные документы (стандарты) оформления материалов • Методические материалы, шаблоны и заготовки документов
Межпроектная программная поддержка	• Разработанные программы (модули), признанные пригодными для общего использования, документированные и помещенные под контроль конфигурации
Кадровый аспект	**Описание возможностей**
Обучение методам и технологиям	• Возможности организации по обучению специалистов методам и приемам разработки ПО • Возможности изучения (освоения) специалистами технико-технологических компонентов инфраструктуры (включая CASE-инструменты)
Обмен положительным и отрицательным опытом	• Культура «открытого» восприятия/передачи приобретенного опыта, знаний, характерных ошибок. Содействие распространению положительного опыта. Не укрывательство собственных ошибок и не перекладывание ответственности за них. Желание обучаться/обучать

Кадровый аспект	Описание возможностей
Накопление и закрепление положительного опыта	• Определение форматов и средств накопления и сохранения приобретенного опыта (опросы, семинары, шаблоны (формы), прототипы и т.п.) • Создание библиотек активов организации по принципу «лучший объект». Включение их в сферу управления конфигурацией. Обеспечение доступности
Стандарты межпроектного взаимодействия	• Определение стандартов (границ компетенции, знаний) по процессам ЖЦ создаваемой ПС. Унификация и стандартизация приемов работы с целью построения и поддержки базового процесса программной инженерии • Профилирование знаний для обеспечения заменяемости специалистов в проектах. Соблюдение принципа «глубокие знания в узкой сфере». Определение интерфейсов (учитывая межпроектные отличия)

Таблица 1.2. **Роли специалистов в организационной структуре разработки**

Роли на уровне организации	Обязанности
Группа технико-технологической поддержки	• Изучение рынка услуг и спроса в организации относительно техники и общесистемного ПО (ОПО) • Приобретение/установка/поддержка техники • Приобретение/установка/поддержка ОПО • Обучение/консультационные услуги сотрудникам • Рекомендации по применению техники и технологий в проектах
Группа защиты информации	• Изучение состояния дел в области защиты информации и накопление опыта (наработка активов для общего использования) • Обеспечение защиты информации в организации • Проверка защиты информации в организации • Поддержка проектов в вопросах защиты информации
Группа инженерии процесса (технологическая служба)	• Определение, сопровождение и совершенствование базового процесса программной инженерии. Обеспечение нормативно-методической поддержки выполнения процессов ЖЦ. Организация и пополнение хранилища (библиотеки) активов организации • Помощь менеджерам проектов в адаптации базового процесса к потребностям проектов. Подбор или изготовление форм (шаблонов) документов для инженерии проектов • Поддержка процесса документирования в проектах, в частности выполнение трудоемких графических работ, оформление документов в соответствии со стандартами оформления. Нормоконтроль и печать документов • Межпроектная координация в части накопления опыта и организации обучения • Поддержка управления конфигурацией в проектах (по договоренности с руководителями проектов)

Роли на уровне организации	**Обязанности**
Независимая группа качества (SQA-группа)	• Планирование и выполнение действий по контролю и гарантированию соблюдения дисциплины создания программной продукции в проектах (организация проверок работ в контрольных точках проектов, определенных календарными планами) • Контроль документов и продуктов ПО (определенных в календарных планах как результаты работ) в контрольных точках проектов на предмет соблюдения действующих стандартов и других нормативных документов, установленных в требованиях Заказчика • Беспристрастие, отчетность непосредственно перед руководителем организации
Независимая группа верификации и валидации (V&V-группа)	• Выполнение функций верификации (по договоренности с группой SQA) • Планирование и проведение независимого квалификационного тестирования интегрированных компонентов ПО или программных продуктов с целью определения их соответствия требованиям Заказчика • Координирование планов работ с менеджерами проектов относительно требований к тестовой среде, сроков и порядка передачи ПО на тестирование • Предоставление отчетов (результатов) тестирования менеджерам проектов для принятия мер по исправлению ПО • Независимость от менеджеров проектов в части определения объемов и методов тестирования • Отчетность перед руководителем организации за соблюдение порядка тестирования и состояние разработанных программных продуктов
Группа поддержки Заказчика	• Связь с Заказчиком по вопросам автоматизации деловых процессов • Поддержка процессов управления требованиями, обучения пользователей, сопровождения (или помощь в их выполнении на уровне отдельных проектов)
Роли на уровне проекта	**Обязанности**
Руководитель проекта системы	• Полная финансовая ответственность за выполнение проектных соглашений перед Заказчиком • Управление разработкой составляющих создаваемой продукции – проектов ПО, КТС, средств защиты информации (если проект предусматривает создание системы «под ключ» с поставкой всех составляющих) • Ответственность за действия соисполнителей проекта
Системные аналитики	• Обследование условий и потребностей автоматизации деятельности организации-потребителя • Системный анализ требований потребителя и формирование концепции системы • Контроль обоснованности принимаемых проектных решений

Роли на уровне проекта	Обязанности
Группа качества проекта	• Контроль качества рабочих продуктов, произведенных процессами ЖЦ (на соответствие стандартам, методикам и т.п.) • Подотчетность только руководителю проекта • Может отсутствовать, если на уровне организации действует независимая группа качества
Группа V&V проекта	• Проверка соответствия рабочих продуктов, произведенных на определенном этапе ЖЦ, требованиям к ним, установленным на предыдущем этапе (например, соответствие решений в пояснительной записке к проекту требованиям, установленным в техническом задании) • Может выполнять тестирование отдельных компонентов ПО, а также системное (интеграционное) тестирование ПО, произведенного в проекте • Подотчетна только руководителю проекта
Менеджер проекта ПО	• Полная ответственность за все проектные решения и действия, связанные с разработкой ПО в проекте • Подбор и контроль ресурсов проекта (относительно ОПО), а также графика работ • В больших или распределенных (по соисполнителям) программных проектах может быть несколько менеджеров (по подсистемам или уровням проекта ПО)
Проектировщики	• Принятие и документирование проектных решений по архитектуре и функциям ПО. Согласование решений с менеджером проекта ПО • Соблюдение стандартов качества (обеспечение достижения характеристик качества)
Программисты	• Программирование или моделирование (макетирование, быстрое прототипирование) компонентов ПО по проектным спецификациям (постановкам задач), подготовленным проектировщиками • Соблюдение стандартов качества при программировании (по удобству сопровождения кода, удобству применения программ и др.) • Отладка и автономное тестирование разработанных компонентов
Группа управления конфигурацией	• Выполнение процесса конфигурационного управления версий ПО и рабочих продуктов проекта ПО
Группа сопровождения	• Выполнение процесса сопровождения версий ПО и рабочих продуктов проекта ПО во время опытной эксплуатации и в течение установленного периода сопровождения • Обучение пользователей • Выполнение процесса решения проблем • Могут быть членами группы поддержки Заказчика
Группа проекта ЛКС	• При разработке системы «под ключ» проектирование и монтаж ЛКС для установки в организации потребителя • Закупка и установка КТС и ОПО, пусконаладочные работы и т.п.

1.1.2. Архитектура процессов жизненного цикла

Процесс программной инженерии имеет иерархическую архитектуру, включая множество *процессов жизненного цикла* (ЖЦ) программной системы.

Требования к процессам ЖЦ ПС определяет международный стандарт ISO/IEC 12207. Нужно отметить, что в Украине принят стандарт ДСТУ 3918-99 - аналог ISO/IEC 12207, утвержденного ISO в 1995 году [1]. Однако со времени своего выхода, стандарт ISO/IEC 12207 подвергался пересмотру, и в «текущей» редакции имеет расширенный состав процессов ЖЦ, сформированный с учетом потребностей других международных стандартов (таблицы 1.3 – 1.5) [2, 3].

Процессы ЖЦ распределены по трем группам, что отражает функциональную направленность видов деятельности, которые эти процессы регламентируют:

- основные процессы;
- поддерживающие процессы;
- организационные процессы.

Основные процессы ЖЦ охватывают действия по разработке, поставке, приобретению, эксплуатации и сопровождению программных продуктов (таблица 1.3).

Таблица 1.3. **Основные процессы ЖЦ**

№ п/п	Наименование процесса (подпроцесса)
1.1	Приобретение
1.1.1	Подготовка приобретения
1.1.2	Выбор поставщика
1.1.3	Мониторинг деятельности поставщика
1.1.4	Приемка потребителем
1.2	Поставка
1.2.1	Участие в тендере
1.2.2	Заключение договора
1.2.3	Выпуск продукта (релиза)
	Поддержка приемки продукта
1.3	Разработка
1.3.1	Выявление требований
1.3.2	Анализ требований к системе
1.3.3	Проектирование архитектуры системы
1.3.4	Анализ требований к ПО системы
1.3.5	Проектирование ПО
1.3.6	Программирование (кодирование) ПО
1.3.7	Интеграция ПО
1.3.8	Тестирование ПО
1.3.9	Системная интеграция
1.3.10	Системное тестирование
1.3.11	Инсталляция ПО
1.4	Эксплуатация
1.4.1	Функциональное использование
1.4.2	Поддержка потребителя
1.5	Сопровождение

Поддерживающие процессы интегрируются с любыми другими процессами, решая задачи, вспомогательные по отношению к задачам этих процессов, и обеспечивая качество их решения в конкретных проектах (таблица 1.4).

Таблица 1.4. **Поддерживающие процессы ЖЦ**

№ п/п	Наименование процесса
2.1	Документирование
2.2	Управление конфигурацией
2.3	Обеспечение гарантии качества
2.4	Верификация
2.5	Валидация
2.6	Совместный просмотр
2.7	Аудит
2.8	Управление решением проблем
2.9	Управление запросами на изменение
2.10	Обеспечение применимости продукта (поддержка пользователя)
2.11	Оценивание продукта

Организационные процессы направлены на формирование производственной структуры, включающей множество процессов ЖЦ и персонал, а также на ее поддержку и совершенствование (таблица 1.5).

Таблица 1.5. **Организационные процессы ЖЦ**

№ п/п	Наименование процесса (подпроцесса)
3.1	Управление
3.1.1	Организационное строительство (формирование корпоративной политики, целей, процессов, стандартов, этики)
3.1.2	Управление организацией
3.1.3	Управление проектом
3.1.4	Управление качеством
3.1.5	Управление риском
3.1.6	Измерение
3.2	Поддержка инфраструктуры
3.3	Совершенствование
3.3.1	Учреждение процессов
3.3.2	Оценивание процессов
3.3.3	Улучшение процессов
3.4	Обеспечение трудовыми ресурсами
3.4.1	Управление кадрами
3.4.2	Обучение
3.4.3	Управление знаниями (распространение знаний)
3.5	Управление достоянием (активами организации)
3.6	Управление программой повторного использования
3.7	Доменная инженерия

Как видно из таблиц, часть процессов ЖЦ (особенно процессов поддержки) имеют непосредственное отношение к обеспечению качества программных продуктов и будут подробнее представлены в последующих главах.

1.1.3. Парадигмы программирования и качество

Парадигма программирования – набор *элементов* понятийного аппарата, способ их *представления* и концепция *взаимодействия* (взаимосвязи), лежащие в основе *базовых правил* построения программ из сформированного множества *базовых конструкций*.

Исторически первая парадигма программирования — *императивное программирование*. Методологии императивного программирования учитывают принципы последовательного пошагового изменения состояния вычислительной среды компьютера и полного управления этими изменениями. Примеры таких методологий – *модульное (процедурное) программирование*, *структурное программирование*, а также *неструктурное программирование*[1]. К императивным относят также *объектно-ориентированное программирование*, определяемое зачастую как самостоятельная парадигма программирования.

Другая очень «старая», хотя и до недавнего времени менее широко используемая парадигма программирования – *декларативное программирование*. В отличие от императивного программирования, где программист должен четко определить *алгоритм* решения задачи, в декларативном программировании четко определяются *структуры данных*, *цели решения* или *свойства решения* задачи, а не способ ее решения. Примеры методологий декларативного программирования - *функциональное программирование* (например, на языке Lisp), *логическое программирование* (например, на языке Prolog) и др. Декларативные языки могут быть *доменно-ориентированы* (специализированы для определенного домена), как, например, SQL (для работы с базами данных), HTML (для описания web-страниц) и др.

Каждая парадигма программирования ориентирована на определенную аппаратную архитектуру и вычислительную модель и поддерживается одним или несколькими языками программирования (ЯП). В то же время, один ЯП может применяться для программирования во многих парадигмах, как, например, C++ используемый и в процедурном, и в объектно-ориентированном программировании.

Существует множество других современных парадигм программирования, в числе которых *компонентное программирование*, *аспектно-ориентированное программирование*, *порождающее программирование* и другие.

Появление новой парадигмы программирования служит толчком к разработке методологий, методов и языков программирования, к созданию новых подходов в программной инженерии в целом, и в инженерии качества, в частности.

Выбор методологий программирования, основанных на той или иной парадигме программирования, определяет выбор методов верификации кода, объектов и методов тестирования, а также измеряемых атрибутов программ и используемых метрик.

Конечно, в этой книге невозможно ни охарактеризовать все существующие парадигмы программирования, ни описать все особенности обеспечения качества

---

[1] Языками, поддерживающими неструктурное программирование, являются Ассемблеры.

программ, разработанных в этих парадигмах. Тем не менее, мы посчитали нужным дать очень краткий обзор и широко, и мало известных совсем новых парадигм и методологий программирования (глава 2), но ограничиться при описании методов обеспечения качества в основном традиционными методологиями императивного и (отчасти) объектно-ориентированного программирования. Надеемся, что читатели, «уловив» общие идеи, смогут на практике нужным образом адаптировать описанные в книге подходы к качеству или обратиться к специализированным литературным источникам.

1.1.4. Инженерия процессов разработки

Современной концепцией процесса программной инженерии является построение ***базового процесса организации*** (БПО), *разрабатываемого*, *сопровождаемого*, *оцениваемого* и *улучшаемого* подобно тому, как разрабатываются, сопровождаются, оцениваются и модернизируются программные продукты [4].

БПО служит основой для определения процессов всех программных проектов. *Процессы на уровне проектов* разрабатываются путем *адаптации БПО* к характеристикам конкретного проекта.

Определение БПО - это формализованное описание состава процессов ЖЦ, из которых должны выстраиваться процессы разработки в программных проектах, а также взаимосвязей (порядка следования и интерфейсов) между элементами этих процессов. Описание обеспечивает согласованность выполнения работ в организации, стабильность процессов и фундамент для их улучшения.

С каждым процессом разработки связываются:

- *требования* к процессу, которые указывают, «что» собой представляет процесс (что он будет *делать*);
- *архитектура* и проект процесса, которые описывают, «как» процесс будет определен (каковы будут элементы процесса и как они будут взаимосвязаны);
- *реализация* описания (проекта) процесса в рамках организации или программного проекта (создание элементов процесса и установление интерфейсов);
- *проверка* и утверждения определения процесса;
- *внедрени*е процесса в среду конкретного проекта.

На рисунке 1.2 представлены основные элементы описанной концептуальной модели инженерии процессов разработки программных систем:

- базовый процесс (включая архитектуру процесса и его элементы);
- описания моделей ЖЦ, рекомендованных для использования в организации;
- руководства и критерии для адаптации БПО;
- база данных БПО;
- библиотека документации, связанной с процессом разработки.

Элементы БПО открыты для использования проектами при разработке, сопровождении и реализации собственных процессов для проектов.

Архитектура БПО представляет собой высокоуровневое описание базового процесса организации, касающееся приоритетов, интерфейсов, взаимозависимостей и других взаимоотношений между элементами БПО и с другими внешними по отношению к нему процессами (например, системной инженерии, инженерии аппаратного обеспечения и др.).

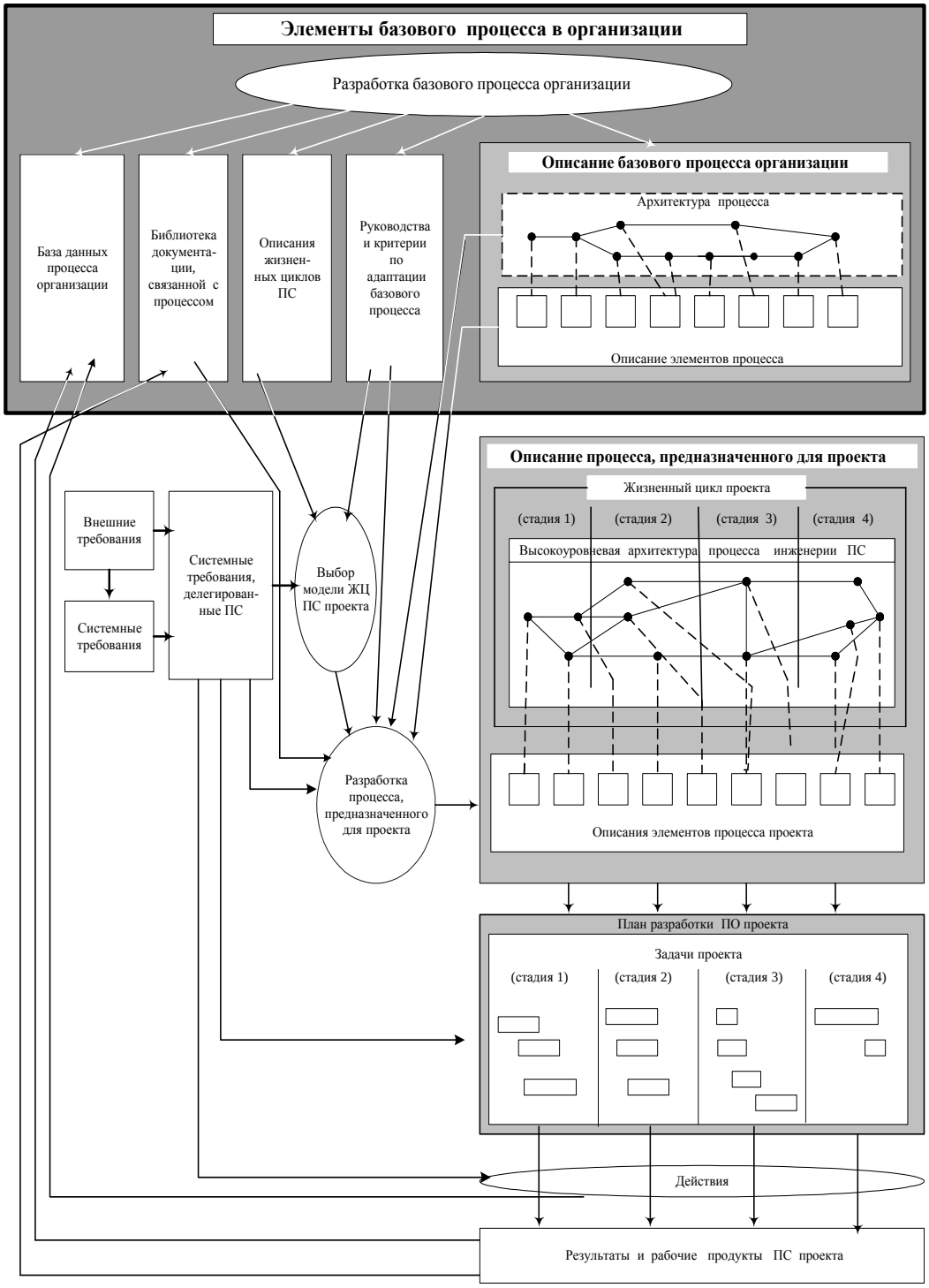


Рис. 1.2. **Концептуальная модель инженерии процессов разработки**

Элемент БПО - это составной элемент описания процесса. Каждый элемент процесса охватывает четко определенное, ограниченное и связное множество задач. Описания элементов процесса могут представлять собой:

- шаблоны форм, подлежащих заполнению;
- фрагменты, требующие укомплектования;
- описания, выполненные на высоком уровне абстракции и нуждающиеся в уточнении;
- полностью сформированные описания, которые могут быть модифицированы или использованы без изменений.

Описание моделей ЖЦ, рекомендованных к использованию - это набор описаний известных из литературных источников и адаптированных к нуждам организации моделей. По согласованию с заказчиком или пользователем конкретного проекта одна из этих моделей должна быть использована в сочетании с базовым процессом при построении процесса для проекта.

Руководства и критерии адаптации БПО призваны помочь руководителям проектов выбрать модель ЖЦ для использования и адаптировать БПО и выбранную модель к характеристикам конкретного проекта. Эти руководства и критерии обеспечивают общий базис для планирования, реализации, измерения, анализа и совершенствования процессов разработки ПС проектов.

База данных БПО - это БД, предназначенная для сбора и предоставления информации о процессах и полученных в ходе их выполнения рабочих продуктах, в частности, тех, которые имеют непосредственное отношение к БПО.

Эта БД содержит или ссылается как на данные реальных измерений, так и на информацию, необходимую для понимания этих данных и оценивания их обоснованности и применимости. БД может содержать, например, оценки размера ПС, трудоемкости и стоимости ее разработки, реальные данные о размере, трудоемкости и стоимости ПС, данные о производительности, полноте охвата и эффективности проверок ПС, о числе и серьезности дефектов, обнаруженных в коде, и др.

Библиотека документации процесса создается для хранения документов процесса, которые могут быть полезными для других действующих или будущих проектов, особенно имеющих отношение к базовому процессу организации (это могут быть процессы проектов, стандарты, процедуры, планы разработки, планы измерений и материалы по процессу обучения). Использование фонда библиотеки должно помочь снизить затраты труда на развертывание нового проекта, демонстрируя примеры успешно выполненных проектов, которые могут быть взяты за образец.

Описание процесса предназначенного для проекта - это используемое проектом операционное определение процесса, изложенное в терминах стандартов по программной инженерии, процедур, инструментов и методов. Оно разрабатывается путем адаптации определения БПО. Описание процесса проекта создает базу для планирования, выполнения и улучшения деятельности исполнителей проекта.

Стадии выполнения проекта отражают распределение усилий на разработку. Они имеют регулируемый размер и представляют собой измеримое множество связанных между собой задач, выполняемых проектом. Обычно стадия - это фрагмент ЖЦ, как правило, заканчивающийся формальным обзором, предшествующим началу следующей стадии.

Задача (или технологическая операция) - это четко определенная единица работы в процессе, завершение которой представляет собой для руководства проектом видимую контрольную точку для оценки состояния проекта. С задачей связывается критерий готовности (предусловие выполнения) и критерий завершения (постусловие выполнения).

Рабочие продукты (*РП*) - это результаты деятельности или решения задач в ходе определения, сопровождения или использования процесса, включая описания, планы, процедуры, компьютерные программы и связанную с ними документацию, которые предназначаются (или не предназначаются) к поставке заказчику или пользователю. Рабочие продукты образуют вход для следующего шага процесса или архивную информацию по проекту ПС для использования будущими проектами (например, планы, оценки, данные о реальной трудоемкости, документы требований и др.).

Продукты ПС (или поставляемые *программные продукты* (*ПП*)) представляют собой полное множество или любой из отдельных элементов множества компьютерных программ, процедур, данных и соответствующей документации, *предназначенных для передачи* заказчику или конечному пользователю. Все поставляемые продукты являются в то же время рабочими продуктами, но не все рабочие продукты представляют собой поставляемые продукты.

Собственно выполнению процесса в проекте предшествует построение *плана разработки ПС*. Этот план в виде одного или группы документов образует мост между процессом в проекте и конкретной реализацией этого процесса (с указанием исполнителей и графика выполнения задач). Только сочетание точного определения процесса и плана разработки дает возможность реально выполнить процесс программной инженерии.

1.1.5. Интеграция процессов жизненного цикла

Каждое программное приложение ПС, программный комплекс (ПК) в составе ПО системы или отдельное программное средство проходит следующие ***стадии ЖЦ*** независимо от выбранной модели ЖЦ.

- определение требований,
- проектирование,
- кодирование, автономное тестирование и отладка,
- интеграция (если ПО - часть системы),
- тестирование на соответствие функциональным и не функциональным (техническим) требованиям,
- внедрение,
- сопровождение,
- списание (замена новым).

Работы на этих стадиях ЖЦ соотносятся с действиями в основных процессах (см. табл. 1.3). Нормативной базой для их выполнения является стандарт ДСТУ 3918-99.

При построении БПО руководители организации-разработчика ПС изначально определяют состав *основных процессов* из числа рекомендованных стандартом ДСТУ 3918-99, а затем поддерживающих и организационных процессов. В своих решениях они руководствуются соображениями целесообразности включения того или иного процесса в БПО с позиций необходимости выполнения регламентируе-

мых процессом действий и их достаточности для достижения целей разработки качественного программного продукта.

В зависимости от стратегических целей усовершенствования БПО, а также наличия соответствующей организационной структуры (например, групп качества, управления конфигурацией, тестирования и др.) процесс может быть рекомендован для внедрения в БПО в полном объеме (все действия) или частично (не все действия). Действия процессов могут объединяться, образуя новый процесс в любой категории процессов (основных, поддерживающих, организационных). Кроме того, отдельные действия (или группы действий) поддерживающих процессов могут *делегироваться* основным или организационным процессам. Это возможно, например, в организациях, разрабатывающих однотипные программные продукты в рамках коротких проектов, где многие поддерживающие процессы (например, верификация, валидация, управление конфигурацией и др.) могут быть *инкорпорированы* в процесс *управления проектом*. В ***минимальной конфигурации процессов ЖЦ***, кроме процесса управления проектом, в БПО обязательно должно найтись место процессу *проверки* (интеграции процессов поддержки, направленных на проверку правильности каждого рабочего продукта процесса), а также процессу *управления конфигурацией* (рисунок 1.3).

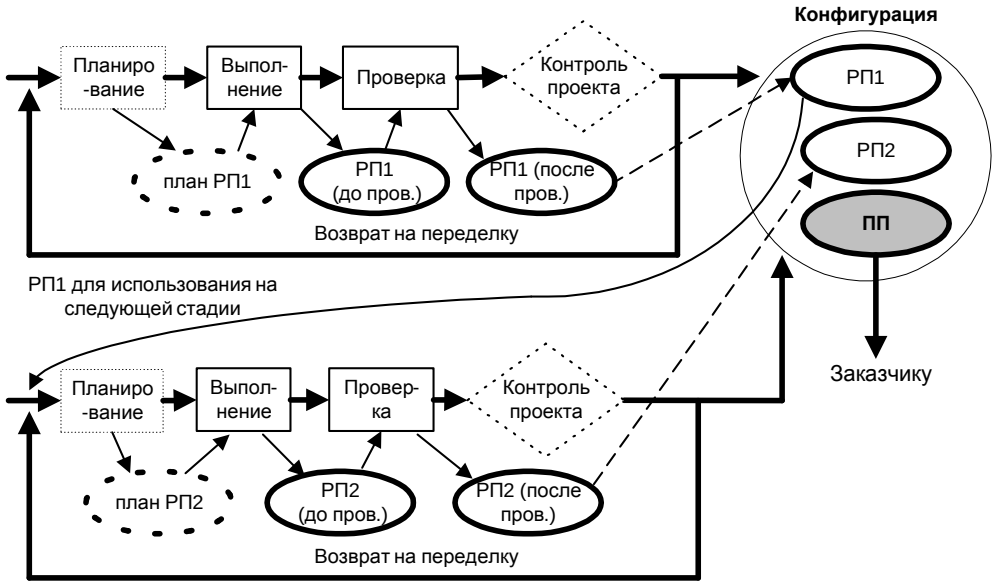


Рис. 1.3. **Жизненный цикл рабочих продуктов БПО**

Каждый создаваемый при выполнении основного процесса рабочий продукт, который вносит вклад в построение конечного программного продукта, должен пройти стадии проверки (лучше независимой), утверждения и помещения под управление конфигурацией. И, только рабочий продукт, находящийся под управление м конфигурацией, может быть использован для построения конечного ПП.

Принцип интеграции процессов в БПО состоит в инкорпорации нужных направлений деятельности в совершенствуемый процесс (в случае отсутствия и нецелесообразности формирования смежных процессов), или, наоборот, в делегировании ряда действий процесса другим, уже существующим, процессам ЖЦ.

В описании БПО определяются также требования к квалификации специалистов и их ***ролям в процессах ЖЦ***. С понятием роли мы связываем перечень обязанностей в процессе ЖЦ, используемых методов и средств, ответственности за рабочие продукты процесса, взаимодействие с исполнителями ролей в других процессах, а также возможное совмещение ролей.

Построенный БПО должен поддерживать использование ***моделей ЖЦ***, применение которых допускается в проектах организации. Для этого в описании БПО должны специально оговариваться возможные точки соприкосновения процессов (например, рабочие продукты и моменты их передачи от одного процесса другому процессу), условия повторного выполнения процессов и методы выполнения работ, обусловленные выбором той или иной модели ЖЦ.

Широко распространены следующие основные классы моделей ЖЦ:

- ✓ каскадные
  - ✓ стандартная
  - ✓ с обратной связью
  - ✓ пилообразная
- ✓ итерационные
  - ✓ с приращениями
  - ✓ эволюционные
    - ✓ спиральная
    - ✓ быстрой разработки приложений (RAD)

Краткий обзор этих моделей можно найти в приложении 1.

Выбор модели ЖЦ существенно зависит от двух факторов (если не принимать в расчет финансовые соображения):

А) можно ли *изначально* определить *практически полный набор функций*, подлежащих реализации в программном продукте:

$PP = \{F_1, F_2,\ldots F_n\}$

Б) должны ли все затребованные функции $F_1, F_2,\ldots F_n$ *поставляться заказчику одновременно*.

Если выполняются условия ***А и Б*** (требования известны (*А*) и нужна одновременная поставка всех функций (*Б*)), - для применения в проекте рекомендуется выбирать каскадные модели.

Если выполняются условия ***А и !Б*** (не *Б*) (требования известны (*А*) и согласована поставка итерациями (!*Б*)), - может быть выбрана итерационная модель с приращениями.

Если выполняются условия ***!А и Б*** (не все требования известны *(!А)*, но все известные - должны быть реализованы и поставлены одновременно (*Б*)), а также *желательна разработка прототипов для моделирования требований* - возможно применение спиральной модели, при условии, что требования будут меняться нечасто и будут выполняться оценки риска реализуемости и планирование разработки.

Если выполняются условия ***!А и !Б*** (не все требования известны *(!А)*, а все известные реализуются и поставляются итерациями (*!Б*), *и желательна разработка действующих прототипов* - возможно применение модели быстрой разработки приложений (RAD), при условии, что прототипы будут охватывать каждый набор требований и сроки разработки не будут четко установлены.

Методология экстремального программирования (XP) (которая получила заслуженное признание программистов) безусловно пригодна для использования в проектах с быстрой разработкой приложений, однако требует практически постоянного участия заказчика и не предполагает документирования проекта, а это – существенная преграда для применения XP в проектах, выполняемых по государственным заказам. Подробнее об этой методологии – в главе 11.

Выбранная для проекта модель ЖЦ должна согласовываться с заказчиком и *фиксироваться в договорных документах*. Кроме того, в них должны фиксироваться стандарты разработки, а также механизмы проверки рабочих продуктов.

Если модель – заведомо не каскадная, с заказчиком нужно оговаривать, каким образом будут появляться (исчезать) и расширяться функции ПС, установленные в требованиях (например, путем оформления дополнительных соглашений к договору и переноса сроков)? Каким должен быть уровень поддерживаемой целостности проекта, то есть, какие документы проекта должны переделываться и повторно утверждаться при изменениях? То есть, какова высота уровня возвратов в проекте – возврат на переделку постановок задач для приложений? Возврат на корректировку технического задания? и др.

Очевидно, что целостность проекта ПС должна поддерживаться на уровне ***документа исходных требований к ПС***. Это тот основной документ, с положениями которого должны сверяться разработчики проекта на всех стадиях построения ПС, а также тот единственный документ, который используется тестировщиками при проверке соответствия программного продукта исходным требованиям (действительно, может ли, например, тестировщик определить полноту функций приложения, если нигде не указано, какие именно функции приложение должно выполнять?).

Практика показывает, что в крупных проектах систем исходные требования достаточно четко формулируются только на уровне ***постановок задач*** для каждого приложения (то есть *верхнеуровневого* описания функций и данных приложений). Это тот уровень разработки, на котором уже ясны (хотя и не всегда стабилизированы) функции приложений, информационные объекты и условия среды их функционирования, но еще не приняты окончательные решения относительно структуры приложений, информационно-программной среды реализации и др.

После выбора модели ЖЦ для проекта, должна быть выполнена ***адаптация БПО*** к условиям этого проекта и утвержденной модели ЖЦ. Описание БПО нужно переработать таким образом, чтобы не допускать неоднозначного толкования действий, ролей, а также состава, структуры и содержания рабочих продуктов ПС.

Для выполнения адаптированного процесса в проекте разрабатывается план управления проектом.

1.1.6. Управление проектами

Применительно к программной инженерии виды деятельности по управлению образуют двухуровневую структуру.

Аспекты общего *организационного управления* касаются механизмов воздействия на процессы разработки программных систем в организации, например, стратегического, тактического и оперативного планирования работ, корпоративной культуры и поведения коллектива организации, функционального и производст-

венного управления приобретением, снабжением, торговлей, сбытом и распределением продукции.

Поскольку создание программной продукции обычно облекается в форму отдельных проектов, второй уровень управления образует деятельность по *управлению выполнением программных проектов*: графиком разработки программной системы, материальными и трудовыми ресурсами, процессами в жизненном цикле проекта, риском проекта, качеством программных продуктов и пр.

Управление проектами (в любой отрасли) – это область знаний, навыков, инструментария и приемов, используемых для достижения целей проектов в рамках согласованных параметров качества, бюджета, сроков и прочих ограничений [5].

Современная концепция управления проектом основана на принципе *информативного измерения* процессов ЖЦ проекта, его ресурсов и создаваемых рабочих продуктов. По существу, управление без измерения, качественного и количественного, страдает недостаточной обоснованностью принимаемых решений, а измерение без управления, в свою очередь, не имеет цели и контекста применения. Кроме того, и управление, и измерение, одинаково безрезультатны без использования накопленных *исторических данных* для проведения экспертного сопоставительного оценивания результатов измерений и управления проектом в целом.

Управление проектом включает следующие шаги:

1. Инициация проекта и определение его границ. Предполагает определение требований к проекту посредством применения методов извлечения требований, оценивания их осуществимости с различных точек зрения (технической, технологической, финансовой, социально-политической).

2. Планирование. Предполагает:

- выбор модели ЖЦ проекта и оценивание процессов ЖЦ в контексте пригодности для удовлетворения требований к проекту, адаптацию базовых процессов организации и их институциализацию в проекте;
- иерархическую декомпозицию задач проекта, их спецификацию наряду с установленными требованиями, ассоциируемыми рабочими и поставляемыми продуктами;
- оценки объемов работ, трудоемкости и стоимости реализации проекта;
- распределение ресурсов по задачам с учетом графика проекта и с позиций рационального использования персонала проекта, оборудования и материалов;
- управление риском проекта, по крайней мере, по критериям стоимости разработки, продолжительности проекта и качества программных продуктов;
- управление качеством - применение процедур планирования и контроля качества выполнения процессов и любых рабочих продуктов этих процессов, а также верификация и валидация (V&V) продуктов. Управление качеством и V&V осуществляется по соответствующим планам, синхронизируемым с планом проекта;
- управление планом проекта. Необходимость управления планом проекта обусловлена часто изменяющимися требованиями заказчика и условиями выполнения проекта. Это делает процесс планирования проекта итеративным.

3. Ввод в действие. Предполагает выполнение выбранных процессов ЖЦ в соответствии с планом наряду с измерениями, мониторингом и регулированием процессов и составлением отчетов для заинтересованных сторон (руководства организации, заказчиков, соисполнителей).

Процесс мониторинга заключается в систематическом периодическом оценивании соблюдения процессами утвержденных планов. Анализируются результаты и условия завершения каждой задачи, определяются характеристики создаваемых рабочих продуктов, исследуются финансовые и временные затраты, использование ресурсов, соответствие требованиям и стандартам по качеству. Оцениваются отклонения измеренных значений величин от ожидаемых (в частности, стоимости, сроков, качества). Пересматриваются величины и приоритеты рисков.

Результаты процесса мониторинга создают базис для принятия решения о регулировании процессов в проекте - повторном выполнении определенных действий (например, повторном тестировании компонентов) или инкорпорации новых действий в процесс (например, разработке прототипов компонентов с целью идентификации и утверждения требований к ним). В ходе регулирования возможны также пересмотры плана проекта и сопутствующих планов. Во всех случаях регулирование должно сопровождаться процедурами контроля изменений и управления конфигурацией ПС. Принимаемые решения документируются и доводятся до всех заинтересованных лиц.

4. Обзор и оценка выполнения проекта. Предполагает выполнение всеобъемлющей проверки деятельности по проекту и оценки его продвижения к установленным целям. Проводится в установленных критических точках проекта. Оцениваются не только результаты выполнения процессов, но и эффективность применяемых методов и инструментов, а также продуктивность работы коллектива проекта и возможные трудности (например, конфликты между членами проекта). При необходимости, осуществляется регулирование.

5. Закрытие проекта. Предполагает прекращение проекта по завершении всех процессов и достижении целей, отраженных в плане проекта и сопутствующих планах. Оценивается успешность проекта по отношению к установленным критериям, касающимся состава и качества выпускаемой программной продукции. База данных измерений по выполняемым в организации проектам пополняется новыми данными, которые подвергаются анализу с целью извлечения и обобщения знаний о процессах ЖЦ ПС и необходимого совершенствования базового процесса организации. ПС передается в эксплуатацию.

1.1.7. Процесс измерения при управлении проектами

Современный уровень развития программной инженерии позволяет вывести проблему качества ПС за рамки простого вопроса, «работает система или нет?», формулируя ее следующим образом: «насколько точно созданная система удовлетворяет установленным измеримым требованиям к ее качеству?». Откладывать поиск ответа на этот вопрос до момента завершения разработки – слишком большой риск как для заказчика, так и для разработчика, особенно в условиях все возрастающей сложности и масштабов ПС. Очевидно, что цели по качеству ПС должны определяться в таком виде, чтобы их достижение можно было своевременно прогнозировать и оценивать в ходе проекта. Процессы разработки и рабочие продукты, создаваемые при их выполнении, должны, в свою очередь, допускать контроль качества и регулирование при обнаружении отклонений от планов на всех стадиях ЖЦ. Следовательно, измерения должны стать неотъемлемой частью ЖЦ проекта.

Измерение – получение объективных данных о состоянии продуктов, процессов и ресурсов разработки ПС с целью построения прогнозных и оценочных моде-

лей, применяемых для управления проектом и совершенствования процессов в организации.

Учреждение и выполнение измерений в проекте представляет собой многошаговый процесс, включающий следующие шаги:

1. Определение целей программы измерения. Несистематизированные подходы к измерению не способствуют проникновению в суть процессов, выполняемых в организации, и дают результаты, которые невозможно оценить и обобщить. Все измерения должны мотивироваться *высшими целями* организации и трансформироваться в *частные цели* улучшения тех процессов ЖЦ ПС, выполнение которых будет способствовать достижению высших целей.

2. Построение процесса измерений. Модель измерений является целеориентированной, основанной на процессе декомпозиции целей и построении вопросов, поиск ответов на которые способствует пониманию и достижению целей. Модель разрабатывается таким образом, чтобы обеспечить построение точных и аргументированных ответов на вопросы, подкрепленных количественными оценками, основанными на измеримых величинах в модели.

3. Выбор базовых мер. Базовые меры выбираются таким образом, чтобы результаты измерений рабочих продуктов, ресурсов и процессов ЖЦ ПС позволяли, с одной стороны, оценивать их соответствие целям проектов, а с другой - определять направления и масштабы усовершенствований процессов программной инженерии во всей организации. В число базовых мер входят, как минимум:

- размер и сложность программного продукта, на основании которых может быть оценена трудоемкость и стоимость разработки;
- производительность труда отдельных специалистов и коллектива проекта в целом;
- меры измерения атрибутов качества;

4. Организация сбора данных для выполнения измерений. Собираемые данные должны обеспечивать не только высокие предсказательные возможности измерений, но и быть достаточно «дешевыми» с точки зрения их получения, а также допускать их повторное применение для других измерений. В качестве вспомогательных средств при извлечении и регистрации данных обычно используются формы, вопросники, графики, диаграммы и другие автоматизированные и ручные инструменты, способные обеспечить минимальную трудоемкость сбора, обработки и интерпретации данных. Тем не менее, сбор и обработка данных для измерений остается достаточно трудоемким процессом, требующим поддержки руководителей организации и дополнительных инвестиций в разработку ПС.

5. Применение моделей. Основная цель применения моделей измерений в проектах – обеспечение адекватного управления проектами на всем протяжении их жизненного цикла. По данным об однотипных проектах модели калибруются и оцениваются с позиций эффективности их применения. Основная цель применения моделей на уровне организации в целом – совершенствование процессов программной инженерии.

1.1.8. Аспекты определения качества

Качество, по определению стандарта ДСТУ 2844, это совокупность свойств программной системы, обеспечивающих ее способность удовлетворять установленные или предполагаемые потребности в соответствии с назначением [6].

При определении требований к качеству, обычно перечисляются ***внешние характеристики*** качества, отражающие требования к функционирующему программному продукту. Далее, для того чтобы количественно определить (квантифицировать) критерии качества, по которым будет осуществляться проверка и подтверждение соответствия ПС предъявляемым к ней требованиям, специфицируются: подходящие внешние измеримые свойства (*внешние атрибуты*) ПС и связанные с ними *метрики* (*или показатели*), представляющие собой модели оценивания атрибутов; приемлемые диапазоны изменения *значений* (мер) соответствующих атрибутов.

Метрики, определение и применение которых возможно только для *работающего* на компьютере программного обеспечения, находящегося на стадии тестирования или функционирования в составе системы, называются *внешними метриками*.

После определения требований к внешним метрикам специфицируются ***внутренние характеристики*** качества и *внутренние атрибуты* ПС. Они используются для планирования достижения требуемых внешних характеристик качества конечного программного продукта и их встраивания в промежуточные (рабочие) продукты ПС в ходе разработки. Далее специфицируются *внутренние метрики* для квантификации внутренних характеристик качества, которые могли бы использоваться для проверки соответствия промежуточных продуктов спецификациям внутренних требований к качеству на стадиях разработки.

Понятия внутренних характеристик качества, атрибутов и метрик связывается с *не работающими* на компьютере рабочими продуктами ПС (документами, текстами кода, тестами и др.), получаемыми на стадиях разработки, предшествующих тестированию (определение требований, проектирование, кодирование).

Нужно отметить, что внешние и внутренние характеристики качества касаются свойств самой ПС (работающей или не работающей). Они отражают, в большей мере, взгляд заказчика и разработчика на программную систему. Однако непосредственный конечный пользователь ждет достижения максимального совокупного эффекта от применения ПС - повышения продуктивности работы и общей удовлетворенности программным продуктом. Такой взгляд на качество ПС обозначается термином «качество в использовании» или «эксплуатационное качество» ПС.

Качество ПС в использовании - совокупный эффект характеристик качества для конечного пользователя, измеряемый в терминах *результата использования*, а не свойств самой программной системы. На качество в использовании может влиять любая характеристика качества. Это понятие шире, чем какая-либо одна характеристика (например, удобство применения или надежность).

Качество ПС должно оцениваться исходя из определенной *модели качества*.

В соответствии со стандартом ISO/IEC 9126 ***модель качества*** представляет собой множество взаимосвязанных характеристик, образующих базис для спецификации требований к качеству и оценивания качества ПС [7].

На качество ПС, а также на выбор модели качества, множества внешних и внутренних метрик качества влияют следующие ***факторы, определяющие требования к качеству*** [8]:

1. Прикладная область и категории пользователей. Модель качества может включать множество характеристик качества. Но, каждая из этих характеристик имеет тот или иной вес (значимость) в определенной прикладной сфере и для соот-

ветствующего сообщества пользователей и отражает точку зрения на качество отдельных категорий пользователей.

Например, для критических систем качество обычно связывается с удовлетворением требований к защищенности информации, безопасности функционирования, достоверности, производительности и др. Причем, атрибут производительность (реактивность) наиболее важен для жестких систем реального времени, достоверность - для сверхнадежных и отказоустойчивых систем, защищенность информации - для учрежденческих и банковских систем. Безопасность функционирования важна с позиций анализа угроз и инженерии безопасных систем. Этот фактор необходимо учитывать как один из основных при формировании модели качества конкретной ПС.

2. Цель моделирования качества. В зависимости от цели моделирования качества, модель может в разной степени отражать взгляд на качество различных категорий участников процессов разработки и эксплуатации (сопровождения) ПС - менеджеров, разработчиков, персонала сопровождения, пользователей, - и содержать упорядоченное множество характеристик, обеспечивающих непротиворечивость их интересов. Стратегия измерения и повышения качества должна основываться на учете приоритета (по значимости, по времени удовлетворения) тех или иных интересов.

В большинстве случаев приоритетным, с точки зрения участников проекта ПС, является взгляд менеджера и его интересы в управлении риском проекта.

3. Стадия жизненного цикла. Для того чтобы должным образом управлять качеством на каждой стадии ЖЦ ПС, необходимо рассматривать разнообразные аспекты качества и учитывать изменение представлений о качестве с течением ЖЦ. Стандарт ISO/IEC 9126 предлагает варьировать взгляды на качество продукта по стадиям ЖЦ следующим образом:

- *целевое качество* - необходимое и достаточное качество, которое отражает *реальные* потребности пользователя. Поскольку потребности, заявленные заказчиком, не всегда отражают реальные нужды пользователей относительно качества программного продукта, и эти нужды могут изменяться после того, как заявлены (установлены), - целевое качество рассматривается разработчиками как концептуальная сущность, которая не может быть полностью определена в начале проекта и должна рассматриваться как ориентир. Требования к целевому качеству должны по возможности включаться в спецификацию требований к качеству ПС и оцениваться путем измерения эксплуатационного качества по завершении разработки программного продукта;
- *затребованное* (*установленное*) *качество продукта* - это тот уровень значений характеристик внешнего качества, который фактически заявлен в спецификации требований к качеству и должен использоваться как цель для его проверки. Требования ко всем или некоторым характеристикам качества указываются в спецификации требований к ПС. Наряду с оптимальными значениями характеристик должны также указываться минимальные значения, что поможет заказчикам и разработчикам избежать ненужного превышения стоимости и сроков разработки;
- *качество программного проекта* – внутреннее качество ПС, представленное в описании основных частей или всего проекта, например, архитектуры программного обеспечения, структуры программ, стратегии проектирования интерфейса пользователя и т.п. Качество проекта – основа качества программного

продукта, которое может быть лишь незначительно улучшено в ходе кодирования и тестирования;

- *оцененное* (*или прогнозируемое*) *качество продукта* - качество, которое оценивается или предсказывается как качество конечного программного продукта на каждой стадии разработки на основании характеристик качества программного проекта;
- *качество поставляемого продукта* - это качество готового к поставке продукта, как правило, протестированного в моделируемой среде на моделируемых данных.
- *качество в использовании (эксплуатационное качество)* - качество ПС, измеряемое в терминах результатов ее использования, а не свойств. Качество ПС в пользовательской среде может отличаться от качества в среде разработки из-за недоучета особенностей среды и сценариев применения ПС и, как следствие, неадекватного тестирования. Пользователь оценивает только те атрибуты ПС, которые видны ему в ходе фактического использования.

4. Размер и сложность. За исключением очень простых проектов, различные компоненты ПС обычно имеют совершенно разные требования к характеристикам качества. Например, компоненты, обеспечивающие интерфейс с пользователем, характеризуются высокими требованиями к удобству применения, а базы данных - повышенными требованиями к целостности. Концептуальное разделение единого программного продукта на компоненты, характеризующиеся однотипными требованиями к качеству, помогает снизить конфликты между требованиями.

5. Методология проектирования и наличие CASE-инструментов. В пределах одной организации процессы разработки могут иметь различное наполнение и варьироваться в зависимости от применяемых в проектах методологий и интегрированных инструментов. Эти различия ограничивают возможность использования одного и того же множества метрик по различным проектам. Очевидно, например, что не имеет смысла сравнивать число строк кода, разработанного с применением структурных методов, с числом строк, разработанных объектно-ориентированным методом.

6. Уровень зрелости организации-разработчика. Существует множество моделей зрелости организаций-разработчиков ПС, определяющих критерии оценивания мощности процессов и эффективности проектов разработки. Стремление организаций-разработчиков обеспечить конкурентоспособность выпускаемой программной продукции служит хорошим стимулом не только для повышения технического и технологического уровня проектов, но и для постоянного совершенствования процессов организации и управления разработкой, их оценивания по моделям зрелости и получения соответствующих сертификатов (например, сертификатов системы качества). Очевидно, что чем выше уровень зрелости организации, тем совершеннее процессы, связанные с обеспечением качества ПС.

7. Исторический опыт разработки. В достаточно зрелой организации-разработчике ПС, специализирующейся на выпуске однотипных программных продуктов, складываются определенные подходы к обучению сотрудников, стратегии проектирования и программирования, приемы сбора и регистрации всевозможных данных, методологии выполнения измерений ПС и т.д. Это дает им возможность использовать собственный опыт разработки программных систем со схожими характеристиками, накапливать полезную историческую информацию по проектам,

относящуюся к качеству, и, таким образом, более точно строить модель качества, выбирать подходящие метрики, устанавливать количественные меры для всех характеристик качества ПС и *прогнозировать* достижение основных характеристик качества (например, надежности) уже на ранних стадиях разработки.

8. Конфликты требований. Множество функциональных и нефункциональных требований к программной системе (в том числе требований к качеству) могут вступать в конфликты, которые должны разрешаться на всех стадиях ее разработки. Очевидно, что высокие требования к качеству ПС в условиях ограниченных ресурсов проекта всегда связаны с риском проекта, например по его продолжительности и стоимости. Для выявления и устранения такого рода конфликтов обычно используются матрицы конфликтов или другие инструменты контроля качества и управления риском.

1.1.9. Взаимосвязь понятий в парадигме качества

Для того чтобы завершить краткий обзор множества понятий в области программной инженерии и качества разрабатываемых программных систем, мы начертили схему, логически объединяющую все элементы понятийного аппарата, лежащего в основе современной парадигмы качества (рисунок 1.4).

На этой схеме закругленными прямоугольниками обозначены *объекты* (продукты, процессы), по отношению к которым будут определяться концепции повышения качества. Кружками (овалами) обозначены *атрибуты* (свойства) этих объектов, оценивание которых необходимо для обеспечения качества разрабатываемых ПС. Бесцветными прямоугольниками обозначены *роли субъектов* процессов разработки и эксплуатации ПС.

1.2. Подходы к повышению качества программных систем

1.2.1. Применение процессов контроля качества

Первый шаг к повышению качества ПС состоит во внедрении в повседневную практику организаций-разработчиков специально предназначенных для этого *поддерживающих процессов* ЖЦ ПС, регламентируемых стандартом ISO/IEC 12207 (или ДСТУ 3918) по процессам жизненного цикла ПС, а именно процессов *гарантирования качества*, *верификации*, *валидации*, *совместного просмотра* и *аудита* [9]. Применение этих процессов обеспечивает руководство организации и проекта методами и средствами контроля за соблюдением требований к процессам разработки и качеству рабочих продуктов на каждой стадии ЖЦ проекта.

Применяемый далее термин «гарантирование качества» соответствует обычно используемому за рубежом термину «Software Quality Assurance» (SQA), получившему в издаваемой у нас переводной литературе по программной инженерии и отечественных стандартах разные варианты перевода, например, «обеспечение качества», «контроль качества». Однако существующие переводы не в полной мере отражают суть деятельности по SQA.

Может сложиться впечатление, что именно (и только) процесс SQA должен *обеспечивать* качество программных продуктов или *контролировать* качество, что неверно. Неверно также все задачи обеспечения качества возлагать на одноименную группу – SQA. Без вовлечения в контроль качества других процессов ЖЦ достижение целей качества было бы невозможным.

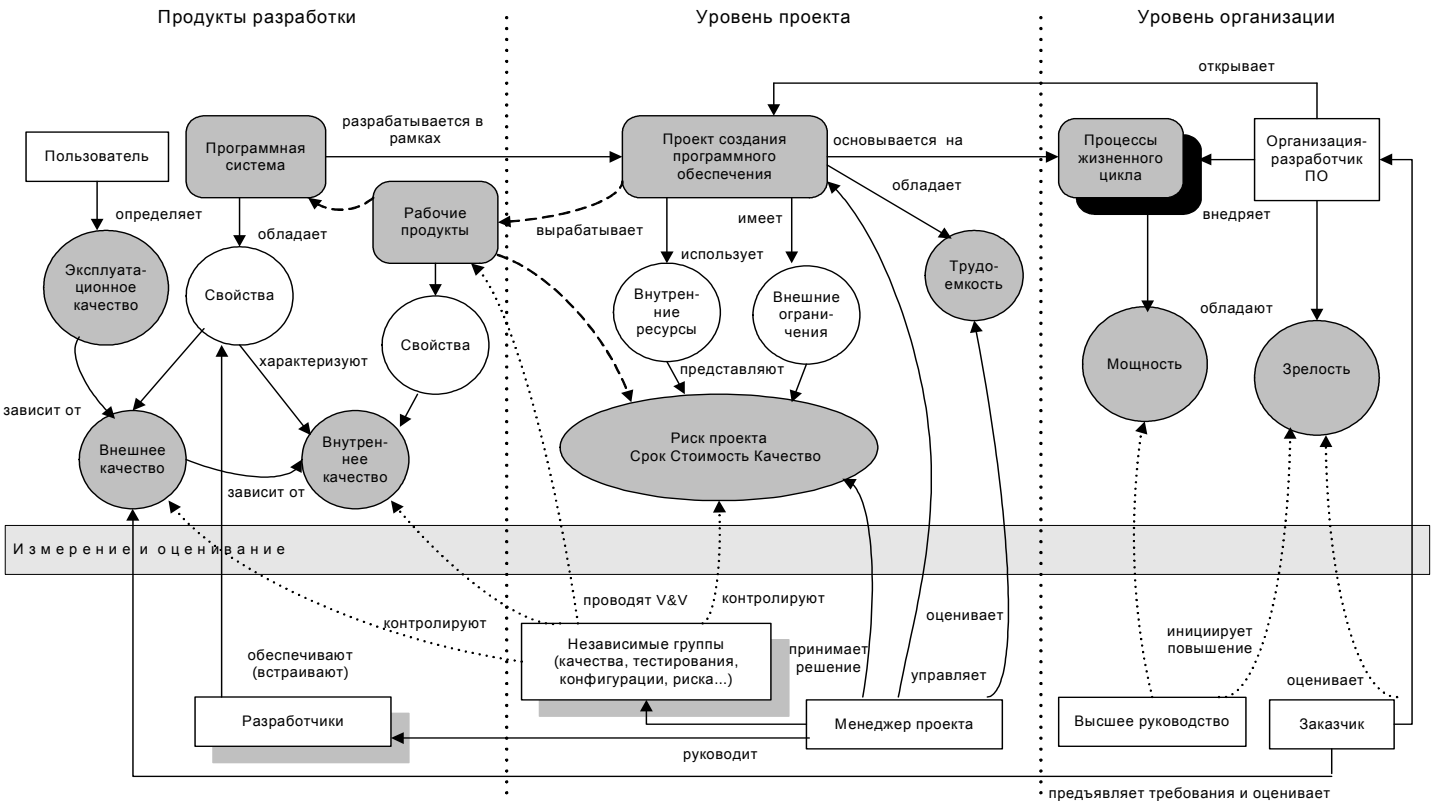


Рис. 1.4. Связь понятий в парадигме качества

Процесс SQA обеспечивает гарантии (от assurance – давать заверения, гарантировать), что программные продукты и процессы в жизненном цикле проекта соответствуют предъявляемым к ним требованиям. Эти гарантии основаны на том, что SQA планирует и осуществляет *контроль и оказание помощи в той деятельности* по проекту, которая непосредственно связана со «встраиванием» в программные продукты свойств, обеспечивающих качество. SQA *устанавливает стандарты*, отвечающие передовой практике программной инженерии (нормы, правила, требования - как к процессам, так и к их продуктам) и контролирует их соблюдение в ходе ЖЦ. SQA гарантирует, что проблемы, неизбежно возникающие в любой комплексной деятельности (каковой является совместное выполнение процессов ЖЦ), своевременно идентифицируются и устраняются, что запланированные процессы пригодны для применения и надлежащим образом выполняются в соответствии с планом и что результаты выполняемых в проекте измерений могут быть использованы для регулирования деятельности в процессах ЖЦ. Цель SQA – установить, *почему* допускаются ошибки и как они могут быть устранены. Таким образом, объектом исследований SQA являются в основном *процессы* ЖЦ ПС, а не программные продукты (в том числе рабочие продукты), качество которых должно быть обеспечено.

Для контроля качества программных *продуктов* (включая все рабочие продукты) предназначены процессы верификации и валидации [10].

Процессы верификации и валидации определяют, действительно ли продукты определенного этапа процесса разработки и сопровождения ПС соответствуют виду и потребностям данного шага и отвечают требованиям, предъявляемым к ним в начале этапа, а также в начале процесса. Задача верификации и валидации – *проверить и подтвердить*, что конечный программный продукт *будет* отвечать своему назначению и удовлетворять пользователей. Эти процессы контролируют качество путем выявления ошибок в продуктах процессов ЖЦ, не исследуя коренных причин появления этих ошибок.

Цель верификации – исследуя трансформации одних (входных) рабочих продуктов в другие (выходные) рабочие продукты проверить, *правильно ли разрабатывается* программный продукт (например, правильный ли код программного модуля по отношению к спецификации проекта этого модуля).

Цель валидации – исследуя совокупность рабочих продуктов, полученных на определенном этапе процесса разработки, убедиться в том, что они *разработаны правильно*, то есть отвечают назначению и специфицированным *исходным требованиям* к программному продукту (например, совокупность программных модулей действительно представляет требуемый программный продукт, совокупность выполненных тестов действительно достаточна для проверки всех функций ПС и т.п.).

И процесс верификации, и процесс валидации должны выполняться, начиная с самых ранних стадий ЖЦ ПС и применительно ко всем рабочим продуктам разработки (включая, например, планы). Эти процессы тесно взаимосвязаны и обычно определяются одним термином «верификация и валидация», которому соответствует термин «Verification and Validation» (V&V), используемый в зарубежной литературе.

Нужно отметить, что контроль процессов SQA и V&V осуществляет руководство проекта или организации (для независимого SQA или независимой верификации и валидации), а сами процессы *координируют* свои действия по проверке

рабочих продуктов и других процессов. Схематически соотношение подходов к контролю качества ПС представлено на рисунке 1.5.

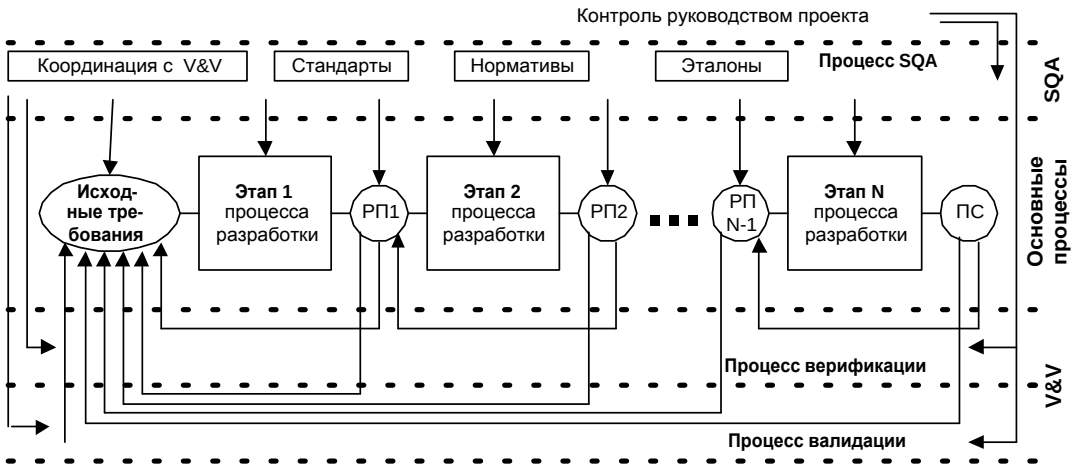


Рис. 1.5. **Взаимосвязь SQA и V&V**

Существует множество методов, пригодных для применения как в целях SQA, так и в целях V&V. Один из видов их классификации – деление на статические и динамические, не связанные и связанные с выполнением программ на компьютере, соответственно.

Статические методы касаются исследования документации по проекту (процессам, рабочим продуктам и пр.) одним лицом или группой лиц (коллективно), возможно с применением инструментальных средств поддержки.

К методам *коллективной работы* относятся инспекции, ревизии, сквозной контроль и др. Они могут также определяться и выполняться как самостоятельные поддерживающие процессы ЖЦ.

Методы *индивидуальной работы* предполагают проведение аналитических исследований (например, анализ потоков данных, причинно-следственный анализ, анализ деревьев событий и деревьев отказов, анализ сложности алгоритмов, формальное доказательство корректности и др.).

К ***динамическим методам*** обычно относят тестирование, имитационное моделирование и символическое выполнение.

Примечательно, что тестированию, с одной стороны, отводится место среди основных процессов ЖЦ ПС, а с другой - его принято считать ключевым методом V&V [11]. На наш взгляд тестирование, как процесс, недостаточно четко определено в действующих стандартах, поскольку эффективное осуществление процесса тестирования требует выполнения определенных действий практически на всех стадиях ЖЦ проекта и во многих процессах, например, в процессах определения требований, SQA и V&V. Парадигма тестирования сформулирована в главе 7.

Рассмотренные процессы и методы контроля кажутся пассивными по природе, поскольку касаются контроля уже завершенного этапа деятельности и уже созданных рабочих продуктов, однако их ценность состоит в обеспечении *обратной связи* при управлении проектом и стимула к повышению качества программных продуктов и усовершенствованию процессов.

1.2.2. Использование прогнозирования при управлении проектом

Прогнозирование (от греческого prognosis – знание наперед, предвидение) заключается в специальном исследовании перспектив какого-либо явления. Научно-техническое прогнозирование – один из ключевых этапов управления. Осуществляется методами экстраполяции, интерполяции, моделирования, обобщения мнений экспертов, исторической аналогии, построения прогнозных сценариев и т.п.

В практической программной инженерии обычно используется так называемое *нормативное* прогнозирование. Оно сводится к определению возможных путей решения проблем управления проектами разработки ПС с целью достижения желательного результата посредством сопоставления реальных данных и нормативов с учетом заранее заданных критериев. Такое прогнозирование основывается на обобщении экспериментальных данных и знании объективных закономерностей развития наблюдаемых явлений в проекте.

Прогнозирование в области программной инженерии и качества не следует путать с оцениванием, хотя в основе и того, и другого лежат *измерения* свойств существующих объектов.

Оценивание обычно предполагает получение численных оценок или экспертных заключений о свойствах, которыми *уже обладают* рассматриваемые объекты, например, объем программного кода, вычисленный в единицах KSLOC (тысяч строк исходного кода), или существующий уровень зрелости организации, оцененный в баллах от 2 до 5.

Прогнозирование предполагает предсказание значений определенных величин *в будущем* по измеренным данным и полученным оценкам реально существующих в настоящем объектов. Для примера с объемом кода прогнозирование заключается в ответе на вопрос, каким будет размер разрабатываемой ПС (выраженный в единицах KSLOC), если известно число функций, которые она должна будет выполнять, а также объемы и структура информации, которая должна будет обрабатываться при выполнении каждой функции?

Таким образом, основа прогнозирования – измерение и оценивание.

С целью повышения качества разрабатываемых ПС, в ходе управления проектом нужно прогнозировать влияние на качество всех ранее упомянутых факторов (см. раздел 1.1), однако отрасль прогнозирования в программной инженерии (на данной стадии ее развития) имеет пока еще слишком слабую экспериментальную базу для формирования каких-либо нормативов, а измерение, как процесс, вообще отсутствует в отечественной практике разработки ПС.

В данной работе рассматриваются подходы к прогнозированию трех ключевых величин – *уровня дефектов в ПС* (как одной из основных характеристик ее качества) [12], *размера* ПС (как базовой величины для определения надежности, объема тестирования ПС, сроков завершения разработки ПС и др.) [13] и *трудоемкости* (стоимости) создания ПС [14]. Очевидно, что уровень дефектов, размер и трудоемкость – ключевые критерии управления проектом. Поэтому умение прогнозировать значения этих величин – первостепенная задача руководителей и менеджеров проектов.

1.2.3. Управление риском в проекте

Существует множество определений *риска*, ни одно из которых не является универсальным. Однако все они сходятся в том, что риск возникает там, где есть

неопределенность, связанная с наступлением какого-либо нежелательного события, и есть возможность понести *ущерб* вследствие наступления этого события.

Риск проекта ПС можно определить как возможность *снижения качества* конечного продукта, *превышения стоимости* его разработки, *задержки окончания* разработки или *срыва* проекта (то есть, отказа от проекта) из-за неэффективности, несовершенства процессов ЖЦ ПС. Несовершенные процессы характеризуются нерациональным выбором и неэффективным использованием аппаратного и общесистемного программного обеспечения и компьютерных технологий, низкой профессиональной культурой и дисциплиной, как разработки, так и управления разработкой, несогласованной деятельностью заказчика и разработчика ПС. Все это вместе взятое образует потенциальный источник риска проекта.

Величина риска представляет собой произведение *серьезности последствий* нежелательного события в проекте и *вероятности* наступления этого события.

Серьезность последствий рассматривается в контексте влияния нежелательного события на характеристики качества ПС – надежность, удобство применения, сопровождаемость, переносимость, а *вероятность* - как степень определенности, с которой можно прогнозировать проявление риска в проекте, то есть перерастание данного риска в проблему для проекта.

Секрет эффективного управления риском состоит в принятии (по каждому риску) компромиссных решений по оценке трудоемкости устранения определенного риска, с одной стороны, и величины потенциального отрицательного воздействия этого риска на качество рабочих продуктов и процессов в проекте, - с другой, а также в правильной оценке взаимозависимости устраняемых рисков и возможного влияния принятых в определенный (текущий) момент времени решений на состояние проекта в будущем.

С ростом размера и сложности программных проектов наметилась тенденция к переходу от эвристических методов управления риском, применяемых отдельными *лицами, принимающими решение (ЛПР)* исходя из собственных знаний и опыта управления разработкой, к использованию систематизированных, гибких и легко адаптируемых методов управления риском, обеспечивающих ЛПР всей необходимой информацией для своевременной идентификации и устранения риска проекта. Современное управление риском основано на измерении, оценивании и прогнозировании [15].

В новой версии стандарта ISO/IEC 12207 управление риском представлено одноименным процессом ЖЦ, что подтверждает признание его значимости для разработки высококачественных программных систем.

1.2.4. Совершенствование процессов жизненного цикла

Усовершенствование действующих в организации процессов - одна из основных задач инженерии процессов разработки ПС.

Цели и приоритеты внесения усовершенствований всегда устанавливаются исходя из конкретных потребностей организации, то есть в ответ на вопрос, *для чего* нужны усовершенствования. Перед проведением усовершенствований формулируются количественные показатели эффективности процесса, например, продолжительность, производительность работы и др. А, кроме того, - критерии повышения эффективности (например, повысить производительность на одну треть). Эффективность процесса (выраженная в установленных показателях) должна измеряться по ходу усовершенствований. Измерения эффективности процесса позволя-

ют идентифицировать и определить приоритеты действий по усовершенствованию.

Усовершенствование процесса - *непрерывный процесс*, представляющий собой многократные циклы действий по планированию, осуществлению и контролю усовершенствований.

Цели усовершенствования, определенные количественно и согласованные всеми заинтересованными сторонами, оформляются в виде *программы усовершенствования процесса*. Действия по усовершенствованию, установленные в программе усовершенствования, осуществляются как *проекты по усовершенствованию* процесса.

Для отслеживания процесса усовершенствования используются *метрики*, что дает возможность оценить, достигнут ли какой-либо прогресс, и сделать необходимые корректировки в проекте усовершенствований.

Один из элементов программы усовершенствования процесса - снижение риска, причем риск оценивается с двух точек зрения:

- риск не достичь целей усовершенствования (усовершенствования проведены, но общая цель организации не достигнута);
- риск не осуществить усовершенствования (риск потерпеть неудачу при попытке усовершенствовать процесс).

Стандарт ДСТУ ISO/IEC TR 15504-7 определяет следующие *шаги усовершенствования процессов* ЖЦ [16]:

- инициация усовершенствования процесса. Ключевая фигура в усовершенствовании процессов – инициатор, лицо из руководства организации, которое утверждает программу усовершенствования процесса, предоставляет для этого ресурсы и контролирует выполнение программы;
- выполнение оценивания тех процессов, усовершенствование которых может дать выгоду;
- построение плана программы усовершенствования, его наполнение конкретными мероприятиями исходя из результатов анализа оценивания. Обеспечение проекта всеми необходимыми ресурсами;
- выполнение усовершенствований согласно планам проектов усовершенствования;
- подтверждение усовершенствования;
- поддержка нового, усовершенствованного уровня выполнения, пока не будет достигнута стабильность в выполнении процесса, а также мониторинг выполнения путем сравнения результатов с измеримыми целями плана программы усовершенствования процесса.

С точки зрения усовершенствования программа усовершенствования процесса может охватывать всю организацию, часть организации, отдельный проект или даже часть проекта.

Во главе проведения усовершенствований лежит предварительное ***оценивание мощности процессов***, поскольку именно результаты оценивания мощности позволяют осуществить выбор направлений усовершенствования процессов. Мощность всегда оценивается в контексте достижения определенных высших целей организации.

Стандарт ДСТУ ISO/IEC TR 15504-2 предлагает двумерную модель оценивания мощности процессов [17].

Сначала отождествляются характерные *признаки наличия процесса*, как такового, в соответствии с эталонной моделью (определением) процесса – процесс должен достигать *цели* и выполнять *множество действий* (соответствующих *передовой практике* в программной инженерии), объявленных в эталонной модели процесса.

После того как получены подтверждения существования процесса, определяется, насколько четко процесс организован, устойчив и управляем – насколько сложившаяся *руководящая практика* способствует его выполнению. Руководящая практика оценивается по *множеству атрибутов*, свидетельствующих о степени, в которой руководящую практику можно считать совершенной. Чем с большим *количеством атрибутов* можно ассоциировать руководящую практику, тем выше *уровень мощности* процесса. Каждый атрибут процесса описывает определенный аспект общей возможности управления и совершенствования эффективности процесса в достижении целей процесса и высших целей организации.

Таким образом, в двумерной модели оценивания процесса одно измерение – это *измерение процессов* (по их действиям и конкретным наработкам), а второе – *измерение мощности процессов* (по количеству атрибутов, свойственных руководящей практике). Чем выше расхождение в целевой и оцененной мощности процесса, тем больше вероятность того, что процесс неустойчив и может не соответствовать тем целям, для которых его хотят применить.

С другой стороны, чем на более низком уровне мощности находится процесс, и чем больше величина расхождения целевой и ожидаемой мощности, тем большими могут быть потери, больше риск того, что процесс не будет эффективен в достижении тех целей, для которых он выбран.

Термин мощность (синонимы – зрелость, совершенство, потенциал) процесса – в данном случае возможно наиболее удачный перевод термина «process capability» (возможности процесса), применяемого в упомянутом стандарте, поскольку уместна аналогия с математическим понятием - мощность множества (количество элементов множества).

Подробно процедура оценивания мощности процессов ЖЦ рассматривается в главе 12.

1.2.5. Повышение зрелости организации

Концепция зрелости организаций-разработчиков ПС стала популярной благодаря работам Уотса Хамфри из института SEI (Software Engineering Institute) (США), хотя ее истоки – в 60 годах (работы Р. Лайкерта).

У. Хамфри разработал пятиуровневую модель зрелости - CMM (от Capability Maturity Model), которая определяет характеристики общего процесса программной инженерии в организации, находящейся на определенном уровне зрелости, и указывает эволюционный путь его улучшения [4].

Зрелость организации можно охарактеризовать как степень четкости (ясности) определения, управления, измерения, контроля и выполнения процесса разработки ПС в организации. Она свидетельствует, с одной стороны, о совершенстве всей совокупности процессов в организации в целом, а, с другой стороны, о степени их применимости (адаптируемости) к конкретным проектам организации. Знание степени зрелости организации помогает предсказать возможности каждого проекта в достижении поставленных перед ним целей.

Модель зрелости представляет собой шаблон для оформления маленьких эволюционных шагов по улучшению процесса в виде *пяти уровней зрелости*, каждый из которых является четко определенной платформой для достижения зрелого процесса программной инженерии. Такая структура CMM определяет *приоритетность* действий в направлении повышения зрелости процесса.

Пять уровней зрелости по модели CMM таковы:

- уровень 1 - *начальный*. Он включен в модель только с целью образования точки отсчета (базы) для оценивания последующих улучшений процесса на более высоких уровнях модели. Характеризуется тем, что процесс разработки ПС неструктурирован и хаотичен, а бюджет, график и качество разработки непредсказуемы;
- уровень 2 - *повторяемый*. На этом уровне управление проектом нацелено на контроль соблюдения планов по стоимости, продолжительности и функциональности разработки. Дисциплина разработки позволяет применять отработанные приемы управления неоднократно для схожих между собой проектов. Разработка новых проектов ведется на основе ранее накопленного опыта и в соответствии с основными стандартами в области технологии программирования;
- уровень 3 - *фиксированный*. Процесс программной инженерии (охватывающий деятельность как в части управления разработкой, так и в части собственно разработки) институциализирован в организации (утвержден, стандартизован и документирован). Внедрена программа обучения штата разработчиков ПС и менеджеров. Коллективы отдельных проектов следуют базовому процессу разработки в организации и настраивают его для достижения целей конкретного проекта;
- уровень 4 - *управляемый*. Достигается цель количественной оценки качества программных продуктов и процесса разработки в рамках единой программы измерения. Осуществляется сбор и анализ данных по проектам, что дает возможность управлять риском проекта и, при необходимости, “возвращать” процесс в установленные рамки;
- уровень 5 - *оптимизируемый*. Обеспечивается непрерывное улучшение процесса благодаря наличию средств количественной оценки его слабых и сильных сторон. Данные об эффективности процесса разработки используются для проведения анализа в целях перехода на новые технологии и совершенствования процесса разработки в организации. Данные о новых приемах инженерии изучаются и распространяются по организации. Коллективами проектов производится причинно-следственный анализ ошибок в проектах.

Таким образом, уровни зрелости в CMM описывают *характеристики организации* на соответствующих уровнях. Каждый уровень зрелости определяет проблемы, которые *преобладают* на уровне, и образует *фундамент* для эффективной реализации процессов на последующих уровнях, поэтому пропуск уровней противоестественен.

Уровни зрелости, за исключением первого, касаются ряда *ключевых направлений процесса* программной инженерии. Каждое направление, в свою очередь, представлено пятью *общими разделами*, а каждый раздел определяет перечень рекомендуемых *практических действий* (приемов, процедур), при совместном выполнении которых достигаются *цели*, поставленные во главу угла по соответствующему ключевому направлению процесса.

CMM предлагает *критерии*, позволяющие оценить зрелость организаций-разработчиков. Эти критерии могут использоваться организациями-разработчиками для улучшения процессов разработки и сопровождения ПС, а также государственными и коммерческими организациями-заказчиками для оценки рисков заключения договоров на разработку программных проектов с определенными организациями-исполнителями.

Предложенный SEI подход к достижению и оцениванию зрелости не предполагает исследования характеристик разработанных продуктов, а опирается на совокупность 18 направлений деятельности, следование которым безусловно обеспечит разработку высококачественных продуктов.

В отличие от стандарта ISO/IEC TR 15504, CMM допускает оценку зрелости не в виде профиля мощности процессов ЖЦ по совокупности атрибутов (векторная оценка), а интегральную численную оценку (скаляр) уровня зрелости организации – от 2 до 5.

Подробно модель CMM, а также ее модификации рассматриваются в главе 12.

1.2.6. Управление качеством и внедрение системы качества

Полноценное обеспечение качества можно связывать только с непрерывным управлением качеством на количественной основе.

Управление качеством ПС - целенаправленная систематическая деятельность по планированию, учету, контролю и регулированию качества ПС в ходе ее разработки.

Способность непрерывно *управлять* происходящими процессами в условиях ограниченных стоимостных, временных, трудовых и других ресурсов и обеспечивать постоянную оптимизацию пользовательских требований и ограничений - свидетельство высокой зрелости организации.

В соответствии с моделью CMM, чем выше уровень управления и шире спектр аспектов управления, тем выше уровень зрелости организации-разработчика ПС. Например, принадлежность ко второму уровню зрелости по модели CMM характеризуется наличием, как минимум, таких ключевых направлений – «управление требованиями», «управление работами соисполнителей» и «управление конфигурацией», а также присутствием отдельных элементов управления – «планирование проекта», «гарантирование (контроль) качества (SQA)» и «мониторинг (учет и контроль) проекта». По мере создания и организационного оформления *процессов* ЖЦ и применения *инженерного* подхода к разработке у организации-разработчика появляется возможность дополнить деятельность по управлению элементом *регулирования* и сделать ее полноценной. Такая возможность объясняется *прозрачностью* (просматриваемостью насквозь) процессов и *измеримостью* как процессов, так и продуктов ПС. Так, на третьем уровне модели CMM, планирование и мониторинг проекта перерастают в «интегрированное управление проектом», а затем в «управление процессами» (на четвертом уровне CMM) и «управление изменениями» (на пятом уровне CMM).

То же можно сказать и о качестве ПС, управляемость которой повышается от уровня к уровню – от SQA (на втором уровне CMM) к улучшению качества за счет осуществления инженерии процессов (на третьем уровне CMM) и, наконец, к управлению качеством ПС (на четвертом уровне CMM).

Только *управление* качеством, как направление деятельности организации-разработчика ПС, может стать залогом создания конкурентоспособных программных продуктов, качество которых может непрерывно улучшаться.

Основополагающими стандартами в области качества продукции являются стандарты ISO серии 9000. Эти стандарты рекомендуют каждой разрабатывающей или поставляющей программные продукты организации иметь свою систему качества.

Система качества ПС по определению стандарта ДСТУ 2844 - это совокупность организационной структуры, ответственности, процедур, процессов и ресурсов, направленных на реализацию управления качеством ПС. Действующая в организации-разработчике ПС система качества *обеспечивает* решение задач контроля, инженерии и управления качеством и действительно может *гарантировать* высокое качество программных продуктов. Вопросы построения и сертификации системы качества подробнее рассматриваются в главе 12.

1.3. Концепция инженерии качества

Под ***инженерией качества*** программных систем понимается управляемый процесс *инкорпорации* в ПС на каждой стадии ЖЦ определенных свойств, характеризующих качество, наличие и уровень достижения которых измеряется, прогнозируется, оценивается и регулируется с помощью совокупности *интегрированных* процессов, методов, средств и систематизированных подходов.

Процесс инженерии качества должен гарантировать что:

- требования по качеству учитывают взгляд на качество заказчиков, пользователей, менеджеров и разработчиков ПС;
- все требования по качеству определены таким образом, что могут быть *измерены* количественно или верифицированы;
- процессы ЖЦ содержат процедуры, ориентированные на выявление требований к качеству и непрерывный анализ их достижимости;
- стандарты в области качества процессов и программных продуктов идентифицированы и соблюдаются;
- разработчики ПС понимают цели качества и выполняют *встраивание* в программные продукты свойств, обеспечивающих внутреннее, внешнее и эксплуатационное качество;
- группа качества (возможно совместно со специально создаваемыми руководителем проекта или организации группами) выполняет измерения и другие функции, связанные с контролем, оцениванием и прогнозированием качества;
- результаты измерений, оценивания и прогнозирования качества используются для управления программными проектами и совершенствования базовых процессов в организации;
- выполняется верификация и валидация рабочих продуктов и тестирование ПС с целью проверки согласованности с требованиями качества.

Нужно отметить, что стандарт ISO/IEC 12207 не выделяет инженерию качества в виде отдельного поддерживающего или организационного процесса ЖЦ.

Однако из представленных в нем процессов, непосредственно инженерии качества касаются следующие:

- процесс управления проектом;

- процесс управления качеством;
- процесс управления риском;
- процесс гарантирования качества;
- процессы V&V;
- процесс анализа требований к ПС;
- процесс измерения;
- процесс усовершенствования процессов ЖЦ.

Для достижения целей инженерии качества эти процессы должны институциализироваться в организации и интегрироваться (совместно применяться) в проектах (рисунок 1.6).

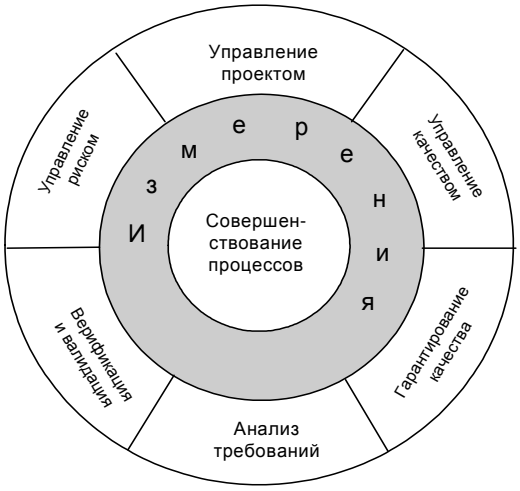


Рис. 1.6. **Интеграция процессов ЖЦ в концепции инженерии качества**

Центральное место в интеграции отводится процессу измерения, результаты выполнения которого используются всеми другими процессами в проекте, а также организацией для совершенствования базовых процессов ЖЦ.

Процесс управления проектом неразрывно связывается с управлением качеством и управлением риском по факторам качества, продолжительности и трудоемкости проекта.

Управление качеством опирается на процессы SQA и V&V.

Процесс SQA взаимодействует с процессом анализа требований к ПС для своевременного контроля за изменением требований.

Связь между процессами отражается в иерархии планов выполнения процессов (например, в плане управления проектом используется ссылка на план качества, в плане качества – на план управления риском и план V&V и т. д.).

Таким образом, ***основные положения концепции инженерии качества*** можно сформулировать следующим образом:

1. *Процессо-ориентированная программная инженерия. Интеграция процессов*. Все участники процессов разделяют цели качества и выполняют специальные функции, связанные с обеспечением качества.

2. *Целе-ориентированная программа измерения*. Цели качества установлены количественно и с учетом потребностей прогнозирования и оценивания качества.

3. *Измерение – ключевое звено цикла управления проектом и совершенствования процессов ЖЦ.*

4. Совершенствование процессов ЖЦ в организации - в контексте достижения целей качества и повышения эффективности управления проектом.

Литература к главе 1

1. *ДСТУ 3918-99.* Інформаційні технології. Процеси життєвого циклу програмного забезпечення.
2. *ISO/IEC 12207:1995* / Amd.1:2002 Information technology – Software life cycle processes.
3. *ISO/IEC 12207:1995* / Amd.2:2004 Information technology – Software life cycle processes.
4. *Модель* оценки технологической зрелости организаций-разработчиков ПО / Андон Ф.И., Суслов В.Ю., Коротун Т.М., Коваль Г.И., Слабоспицкая О.А. // Проблемы программирования. - 1998. - № 4. - С. 46 - 57.
5. *A guide* to the Project Management Body of Knowledge // PMBOK GUIDE. Third Edition (http://www.pmi.org/emeaelink/pmiE-link10-04.pdf).
6. *ДСТУ 2844-94.* Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення.
7. *ISO/IEC 9126-1:2001* Software engineering - Product quality -Part 1: Quality model.
8. *Парадигма* качества программного обеспечения / Ф.И. Андон, В.Ю. Суслов, Т.М. Коротун, Г.И. Коваль // Проблемы программирования. - 1999. - № 2. - С. 51 - 62.
9. *ISO/IEC 12207:1995*. Information technologies. Software life cycle processes. // ISO/IEC. - 1995. – 61 p.
10. *Коротун Т.М.* Верификация и валидация ППО автоматизированных систем организационного управления //Сб. материалов конференции. УкрПРОГ'98, 2-4 сентября 1998 г. -Киев. - 1998. - С. 362 - 367.
11. *Коротун Т.М.*, Лаврищева Е.М. Построение процесса тестирования информационных систем //Проблемы программирования. –2002. - № 1 – 2. –С. 272 – 281.
12. *Коваль Г.І.* Байєсівські мережі як засіб оцінювання та прогнозування якості ПЗ // Проблемы программирования. –2005. - № 2. –С. 15 – 23.
13. *Коваль Г.И.* Методы определения размера ПО // Проблемы программирования. -1999. - № 1. -С. 63 - 71.
14. *Визначення* витрат на створення ПЗ АС / Андон П.І.,Суслов В.Ю., Коротун Т.М., Коваль Г.І. // Проблемы программирования. - 1998. – № 3 -С. 23 - 34.
15. *Управление* риском проектов ПО / Андон Ф.И., Суслов В.Ю., Коротун Т.М., Коваль Г.И. // Проблемы программирования. - 1999. – № 1 -С. 53 - 62.
16. *ДСТУ ISO/IEC TR 15504-7:2002*. Інформаційні технології – Оцінювання процесів життєвого циклу програмних засобів - Частина 7: Настанови з удосконалення процесу.
17. *ДСТУ ISO/IEC TR 15504-2:2002*.Інформаційні технології – Оцінювання процесів життєвого циклу програмних засобів - Частина 2: Еталонна модель процесів та потужності процесу.

Глава 2. ИНЖЕНЕРИЯ КАЧЕСТВА. ЯДРО ПРОФЕССИОНАЛЬНЫХ ЗНАНИЙ

Для того чтобы своевременно учитывать факторы, влияющие на качество, и действительно *управлять* качеством выпускаемой продукции в ходе ЖЦ, необходимо владеть знаниями, составляющими *ядро профессиональных знаний* в области качества программных систем.

Знания разработчиков ПС отличаются большим разнообразием, но, как правило, не полны, не согласованы и разнородны, специализированы в отдельных предметных областях, начиная от операционных систем и кончая современными отраслевыми бизнес-системами. И, что самое важное, эти знания в процессе инженерной деятельности постепенно уточняются, видоизменяются и пополняются.

В последнее десятилетие ведущие зарубежные профессиональные объединения и производители программной продукции работали над определением ядра профессиональных знаний (Body of Knowledge – BOK), составляющих предмет *программной инженерии*, а также *управления проектами создания промышленной продукции*. Ими подготовлены два *Руководства* (*Guide*), соответственно *Guide for SWEBOK* и *Guide for PMBOK*, содержащие разделы и тематические рубрики, определяющие *ядро знаний* в указанных областях[1].

Однако в области *качества ПС* подобное ядро знаний не имеет четких контуров и документально не оформлено как таковое.

В этой главе мы даем краткую характеристику программной инженерии как дисциплины, предлагаем обзор основных положений SWEBOK и PMBOK и представляем наше видение элементов ядра знаний в *области качества*.

2.1. Программная инженерия – дисциплина и специальность

Термин *программная инженерия* прочно вошел в жизнь в начале 70-х годов с выходом в свет первого профессионального журнала в этой области - *Transactions on Software Engineering* (Труды по программной инженерии).

Программная инженерия **–** область компьютерных наук, изучающая вопросы построения компьютерных программ как *инженерной* регламентированной деятельности коллективов разработчиков программного обеспечения. Это инженерная дисциплина, охватывающая все *аспекты создания ПО*, начиная от формирования требований к программному продукту и заканчивая его сопровождением в период использования вплоть до снятия с эксплуатации.

Основной акцент в программной инженерии делается на повышении качества ПО и производительности труда коллективов проектов путем применения передовых методов проектирования ПО; готовых компонентов и методов их генерации; методов эволюции ПО; методов верификации и тестирования ПО; интегрированных инструментальных средств поддержки разработки; методов управления проектами; методов оценки качества продуктов, производительности труда и стоимости работ, а также путем стандартизации процессов разработки ПО, их оценивания и постоянного совершенствования.

---

[1] Мы используем понятие «ядро» (в смысле «основная часть» знаний по предмету), а не «свод» (в смысле набор «упорядоченных текстов» по предмету), как, например, у С.Орлика в работе [1].

Возникновение и высокие темпы развития программной инженерии предопределены несколькими важными факторами:

- накоплением значительного объема знаний в области практического создания ПО, нуждающихся в систематизации;
- появлением разнообразных методов анализа, моделирования и проектирования ПО, а также высоко технологических средств и инструментов разработки ПО, не обеспеченных рекомендациями по эффективному применению;
- высоким уровнем обнаружения дефектов в ПО несмотря на использование прогрессивных методов проектирования и программирования;
- не эффективной организацией труда коллективов разработчиков ПО (менеджеров, проектировщиков, программистов, тестировщиков, технологов и др.);
- использованием готовых программных компонентов, подлежащих идентификации и систематизированному ведению (в библиотеках, хранилищах и др.);
- применением реинженерии существующих компонентов как средства их адаптации к новым, быстро изменяющимся условиям и средам.

В определении понятия *программная инженерия* существенны два ключевых момента:

1) программная инженерия – это *инженерная дисциплина*, суть которой состоит в том, что инженеры (будь то программисты, тестировщики и т.п.) применяют теоретически обоснованные методологии, методы и средства для разработки ПО, руководствуясь конкретными стандартами, правилами и методиками. Программная инженерия «предоставляет» всю необходимую информацию для выбора наиболее подходящего метода, средства и инструмента для реализации разных практических задач программирования ПО на инженерной основе, хотя и не исключает и творческий, неформальный подход к созданию ПО;

2) программная инженерия охватывает множество *аспектов создания ПО*, дополняя процесс собственно разработки (анализа, проектирования и программирования) такими процессами, как управление проектом, конфигурацией, качеством ПО и др., включая оценку результатов труда и затраченных ресурсов. К ключевым аспектам разработки ПО, как инженерной деятельности, относятся, прежде всего, планирование и сопровождение. Первый из них (планирование) предполагает анализ целей и задач разработки, возможностей ее реализации и необходимых для этого ресурсов, а второй (сопровождение) - устранение найденных недостатков в системе и реализацию изменений, обусловленных эволюцией ПО, среды и деятельности пользователей. Один из метров программной инженерии М.Джексон определил золотое правило программирования: *всякая только что законченная программная система сразу требует изменений* [2].

Инженерный аспект деятельности в области программирования делает разработку ПО близкой по своей сущности к *инженерной деятельности*, как она определена в толковом словаре:

1) инженерия есть применение научных результатов, что позволяет получать пользу от свойств материалов и источников энергии;

2) инженерия есть деятельность по созданию машин для предоставления полезных услуг.

В программной инженерии, инженеры – это специалисты, выполняющие практические работы по реализации программ с применением теории, методов и средств компьютерной науки, которая охватывает теорию и методы построения

вычислительных и программных систем. Знание компьютерной науки необходимо специалистам в области ПО так же, как знание физики – инженерам-электронщикам. Если для решения конкретных задач программирования не существует подходящих методов или теории, инженеры применяют собственные знания, накопленные ими в процессе конкретных разработок ПО, а также свой опыт применения соответствующих инструментально-программных средств. Кроме того, инженеры работают в жестких условиях заключенных контрактов и выполняют задачи с учетом их ограничений.

В отличие от других наук, целью которых есть получение знаний, в программной инженерии *знание* есть способ получения некоторой общественной пользы. Ф.Брукс считает, что «ученый строит, чтобы научиться, инженер учится, чтобы строить».

Хотя разработку ПО можно по праву считать инженерной деятельностью, она имеет важные отличия от традиционной «технической» инженерии:

- традиционные «ветви» инженерии имеют высокую степень специализации, а у программной инженерии специализация заметна только в довольно узких применениях (например, операционные системы, трансляторы);
- объекты традиционной инженерии хорошо определены и манипуляции с ними происходят в узком контексте типичных проектных решений, которые отвечают типовым требованиям заказчиков и касаются отдельных деталей, а не общих вопросов, тогда как у программной инженерии подобная типизация отсутствует;
- отдельные готовые решения и изделия в традиционной инженерии классифицированы и каталогизированы, а в программной инженерии каждое новое решение и разработка некоторого элемента ПО - это новая проблема, для которой довольно трудно установить аналогию с ранее выполненными разработками таких продуктов, как программа, компонент, система и т.п.

Приведенные отличия требуют значительных усилий для их нивелирования и превращения программной инженерии в ***инженерную специальность***.

Мировая компьютерная общественность признала целесообразность и своевременность таких усилий. Подтверждением этого является создание ядра SWEBOK, всевозможных программ обучения, институтов и комитетов, международных профессиональных объединений в области информатики. Их главная цель - проведение работ по преобразованию программной инженерии в специальность, которая имела бы зафиксированные признаки для ее распознания и официального признания в мировом сообществе специалистов.

Сложившаяся практика специализации профессиональной деятельности позволяет считать профессию «зрелой», если для нее существуют:

– система начального обучения специальности;

– механизмы развития умений и навыков персонала, которые необходимы для его практической деятельности;

– лицензирование специалистов, организованное под управлением соответствующих государственных органов;

– системы профессионального повышения квалификации персонала и отслеживания современного уровня знаний и технологий по специальности дабы специалисты могли выжить в условиях интенсивного развития специальности;

– этический кодекс специалистов;

– профессиональное объединение.

В последнее десятилетие инженерная деятельность в программировании приблизилась к энциклопедическому определению, ее становление как специальности не вызывает сомнения у большинства преподавательского состава ВУЗов, связанных с информатикой. Сформированы профессиональные организации и объединения, которые работают над определением ядра знаний в программной инженерии и созданием руководства SWEBOK [3], приняли этический Кодекс специалистов по программной инженерии [4], разработали руководства для обучения программной инженерии [5–7], а также создали программу обучения Computing Curricula (CC) 2001 [8]. Кроме того, в США работает комитет по сертификации учебных заведений Computing Accreditation Commission of the Accreditation Board for Engineering and Technology [6]. Программная инженерия как дисциплина тесно связана со *смежными дисциплинами*: компьютерные науки; математика; менеджмент; когнитивные науки; управление проектом; телекоммуникации и сети; электротехническая инженерия и другие инженерные дисциплины.

Таким образом, можно сделать вывод о том, что содержание новой инженерной дисциплины «программная инженерия» сформировано и общепризнано в мировой практике программирования.

2.2. Ядро знаний по программной инженерии (SWEBOK)

2.2.1. Структура руководства по SWEBOK

В начале 90-х годов мировое компьютерное сообщество пришло к необходимости систематизировать накопленные разнородные знания в отраслях компьютерных наук и информатики и зафиксировать их в виде *ядер знаний*.

Для создания ядра знаний по программной инженерии в 1993 году совместными усилиями ACM (Association for Computing Machinery) и компьютерного сообщества института инженеров по электронике и электротехнике IEEE (Computer Society) был образован объединенный координационный комитет по программной инженерии SWECC (Software Engineering Coordinating Committee). В него вошли специалисты мирового уровня в области разработки ПО.

Основная задача SWECC - определение множества критериев и норм профессиональной практики программной инженерии, которые могли бы послужить основой для принятия практических решений при разработке ПО, сертификации продуктов и процессов, аттестации специалистов, а также организации учебного процесса.

В рамках комитета SWECC были организованы специализированные группы по следующим направлениям исследований:

1. Определение необходимого ядра знаний и рекомендуемых практических приемов деятельности в программной инженерии.
2. Определение норм профессиональной этики и стандартов по программной инженерии.
3. Определение Программ обучения студентов ВУЗов по специальности.

Руководство по SWEBOK (2004 года) стало результатом работы по первому направлению [3]. В 1998 был завершен Кодекс этики (как результат работы по второму направлению) [4], а 2004 году – опубликованы новые рекомендации по подготовке программ обучения дисциплине «программная инженерия», разработанные совместными усилиями IEEE и ACM [5].

Ядро знаний SWEBOK составляют знания из *десяти* различных *Областей знаний*, а именно:

1. Программные требования
2. Проектирование (дизайн) ПО
3. Конструирование ПО
4. Тестирование ПО
5. Сопровождение ПО
6. Управление конфигурацией ПО
7. Управление инженерией ПО
8. Процесс инженерии ПО
9. Инструменты и методы инженерии ПО
10. Качество ПО

Каждой Области знаний в руководстве посвящена отдельная глава, структурированная по разделам и рубрикам. Главы завершаются объемными списками литературы по предмету, которая, по существу, и представляет материал, образующий Ядро знаний. Ее изучение рекомендовано для овладения знаниями в соответствующей области знаний.

Структура SWEBOK приведена на рисунке 2.1, а разделы областей знаний – на рисунке 2.2. Особенность руководства по SWEBOK – матричное представление (пересечение) разделов/подразделов и ссылок на литературу, обеспечивающее быстрое определение множества источников знаний в определенной подобласти знаний.

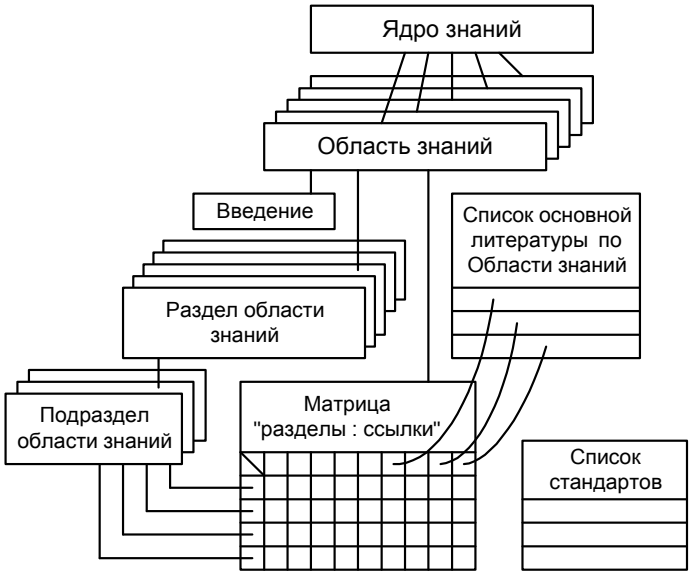


Рис. 2.1. **Структура SWEBOK**

В данной книге не представляется возможным дать развернутую характеристику SWEBOK. Мы ограничимся аннотированным изложением основных положений по каждой из 10 областей знаний и порекомендуем читателям обратиться к доступным литературным источникам [3, 9-14].

Рис. 2.2 . **Разделы и рубрики Руководства по SWEBOK**

Ядро знаний по программной инженерии продолжает развиваться. Текущая версия SWEBOK рассматривается как проект соответствующего международного стандарта, разработку которого осуществляет ISO/IEC JTC1/SC7 WG20[2]. В 2005 году вышла первая версия стандарта - ISO/IEC TR 19759:2005 «Software Engineering - Guide to the Software Engineering Body of Knowledge - SWEBOK».

2.2.2. Область знаний «Программные требования»

Область знаний «Программные требования» (Software Requirements) по существу охватывает все аспекты *инженерии требований* – инженерной дисциплины (в рамках программной инженерии) извлечения, анализа, спецификации, документирования, проверки и управления требованиями к ПО на всех стадиях ЖЦ систем.

Требования – это «*свойства*, которыми должно обладать разработанное или адаптированное ПО, для того чтобы решать поставленные задачи».

С одной стороны, требования - это *множество функций*, которые должно выполнять ПО, чтобы удовлетворить потребности его заказчиков, с другой стороны - это множество ограничений на *способ реализации* функций (например, язык, платформа, эффективные алгоритмы), с третьей – это множество условий выполнения функций ПО в определенной среде эксплуатации (с учетом конфигурации вычислительной среды, режима секретности и др.).

«Многоаспектность» требований связана с необходимостью учитывать не только пожелания непосредственных пользователей ПО, но и потребности сопровождения и развития ПО. В SWEBOK различаются требования к продукту и к процессу, функциональные и нефункциональные требования, а также системные требования, касающиеся взаимосвязанных программных и аппаратных подсистем. Требования могут иметь количественное представление (например, «количество запросов в сек. должно составлять…», «средний показатель ошибок не должен превышать …% от объема вводимой информации» и т.п.). Значительная часть требований относится непосредственно к характеристикам и подхарактеристикам качества: безотказность, надежность и др.

Область знаний включает следующие разделы:

- основы требований[3] (Requirements Fundamentals),
- процесс инженерии требований (Requirements Process),
- выявление (извлечение) требований (Requirements Elicitation),
- анализ требований (Requirements Analysis),
- спецификация требований (Requirements Specification),
- проверка требований (Requirements Validation),
- практические соображения (Practical Considerations).

Раздел области знаний ***Процесс инженерии требований*** базируется на общем процессном подходе к работе с требованиями, в соответствии с которым каждое требование с момента его выявления должно идентифицироваться, определяться как элемент конфигурации и попадать в сферу непрерывного управления и *уточнения* с учетом мнений всех заинтересованных сторон и обеспечением компромисса между требованиями. Процесс инженерии требований нуждается в поддержании, управлении, оценке качества (зрелости) и совершенствовании и для сво-

[2] Структура комитетов ISO по стандартам описана в главе 5.

[3] «Основы» в областях знаний мы здесь не всегда выделяем в виде раздела.

его выполнения требует определенных ресурсов проекта, включая квалифицированный персонал – участников процесса (пользователей, аналитиков и др.).

Раздел ***Выявление требований*** касается процесса извлечения информации из разных источников (договоров, материалов аналитиков по задачам и функциям системы и др.), проведения организационных мероприятий (наблюдений за выполнением бизнес-процессов, собеседований, собраний и др.) для определения спектра требований на разработку, согласования требований с заказчиком, исполнителем и всеми заинтересованными сторонами. Основа надлежащего выявления требования – налаженные коммуникации между всеми участниками проекта.

Раздел ***Анализ требований*** касается изучения заявленных потребностей и целей пользователей, их классификации и преобразования к требованиям к системе, установления и разрешения конфликтов между требованиями, определения приоритетов, границ системы и принципов взаимодействия со средой функционирования. Требования классифицируются как функциональные и нефункциональные, внешние (по отношению к системе и ПО) и внутренние. Функциональные требования характеризуют функции системы и ее ПО, способы поведения ПО в процессе выполнения функций и методы преобразования входных данных в выходные (результаты). Нефункциональные требования определяют условия и среду выполнения функций (например, относительно защиты и доступа к БД, секретности, взаимодействия функциональных компонентов и др.). Моделирование требований и их декомпозиция до уровня программных компонентов завершается на этапе проектирования архитектуры ПО и отражается в специальном документе, по которому проводится *согласование требований* для достижения взаимопонимания между заказчиком и разработчиком.

Спецификация требований к ПО в SWEBOK трактуется как процесс формализованного описания функциональных и нефункциональных требований к ПО и характеристикам его качества в *документе* спецификации требований, структура и содержание которого регламентируется стандартом, утвержденным для использования в проекте, например, IEEE Std. 830:1998 «Рекомендованные приемы спецификации требований к ПО».

Проверка требований заключается в проведении процедуры верификации требований, сформулированных в документе спецификации требований. Ее цель – убедиться в том, что все требования заказчиков учтены и правильно поняты разработчиками. *Верификация требований* – это процесс проверки соответствия требований потребностям, их непротиворечивости, полноты и реализуемости, соответствия стандартам, а также их *пригодности для проведения проверки и подтверждения* соответствия с помощью приемочных тестов. Верификация проводится методами обзора или прототипирования, т.е. быстрого моделирования (отработки) отдельных требований на конкретном инструменте.

В разделе ***Практические соображения*** акцент делается на непрерывности и итеративности процесса инженерии требований, включая мониторинг источников требований, управление изменениями в документе требований, трассирование программных требований к системным, а также принятых архитектурно-программных решений к программным требованиям. Даются также практические рекомендации по определению объема функциональности ПО на основе концепции FSM (Functional Size Measurement), отраженной в стандарте ISO/IEC 14143:1998 «Information technology - Software measurement - Functional size measurement».

2.2.3. Область знаний «Проектирование программного обеспечения»

Проектирование ПО (Software Design) – процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы. Конечный результат проектирования – *проект* (или *дизайн*[4]). В иностранной литературе по программной инженерии слово *Design* определяет оба понятия – и процесс проектирования, и результат проектирования. Базовые понятия в области проектирования – цели, ограничения, возможные альтернативы, представления и решения. В соответствии с процессным походом деятельность по проектированию организована в виде последовательных процессов «проектирования архитектуры системы» и «проектирования ПО», включая архитектурно-функциональное проектирование и детальное проектирование отдельных компонентов. Результат этого процесса – множество моделей и артефактов, фиксирующих основные решения.

При проектировании используются принципы и техники абстракции, связывания (coupling) и сцепления (cohesion), декомпозиции и модульности, инкапсуляции и сокрытия информации, разделения интерфейса и деталей реализации, необходимой достаточности, полноты и простоты.

Область знаний включает следующие разделы:

- основы проектирования ПО (Software Design Fundamentals),
- ключевые вопросы проектирования ПО (Key Issue in Software Design),
- структура и архитектура ПО (Software Structure and Architecture),
- анализ и оценка качества проекта ПО (Software Design Quality Analysis and Evaluation),
- нотации проектирования ПО (Software Design Notations),
- стратегия и методы проектирования ПО (Software Design Strategies and Methods).

Раздел ***Ключевые вопросы проектирования ПО*** определяет фундаментальные проблемы, поднимаемые вне зависимости от используемых методологий проектирования: обеспечение производительности, декомпозиция ПО на функциональные компоненты и их интеграция в систему, поведенческие аспекты (параллелизмы) в работе ПО, распределение компонентов в среде выполнения и их взаимодействие между собой, организация данных и потоков управления, обработка событий и исключительных ситуаций, эргономические аспекты и др.

В разделе ***Структура и архитектуры ПО*** проект рассматривается как «многоаспектный» артефакт, произведенный в ходе процесса проектирования и отражающий множество независимых ортогональных взглядов на ПО (поведенческий, функциональный, структурный и др.).

Современная практика проектирования использует понятие архитектурных стилей и шаблонов проектирования.

Архитектурный стиль – это множество ограничений, определяющих семейство архитектур, удовлетворяющих этим ограничениям; или метамодель, определяющая высокоуровневую организацию ПО (например, клиент-серверные архитектуры, послойные архитектуры и др.).

---

[4] См., например, [1]. В отечественной программной инженерии слово «проект» также трактуется двояко – как «проект = дизайн», и как «проект = вид организации работ» (в английском – Project). Поэтому, возможно, автор работы [1] прав, воспользовавшись словом «дизайн».

Шаблон проектирования (паттерн, pattern) — это эффективный способ решения характерных задач проектирования, образец типового решения общей проблемы в заданном контексте. Паттерн не есть готовое проектное решение, которое может быть непосредственно преобразовано в код, скорее это описание того, как решить задачу таким образом, чтобы решение можно было использовать многократно в различных ситуациях.

Архитектурные стили рассматриваются как паттерны, описывающие макроархитектуру ПО. Для описания микроархитектуры ПО (деталей ПО на более низких, локальных уровнях) используют другие паттерны: *структурные* («контейнер», «адаптер», «компоновщик» и др.), *креативные* («строитель», «фабрика», «прототип», «одиночка» и др.), *поведенческие* («интерпретатор», «посредник», «наблюдатель» и др.). Главная польза паттернов состоит в том, что каждый из них описывает решение целого класса абстрактных проблем, имеет свое имя, независим от применяемого языка программирования, облегчает взаимодействие между разработчиками, использующими унифицированную терминологию.

Еще один из возможных подходов к повторному использованию архитектурных решений и компонентов - формирование *линий* (*семейств*) *продуктов* (product lines) на основе общего дизайна. В объектно-ориентированном программировании аналогичным по смыслу понятием являются *фреймворки* (каркасы), частично завершенные программные подсистемы, которые могут быть надлежащим образом дополнены в соответствии со спецификой решаемой задачи.

В разделе ***Анализ и оценка качества проекта ПО*** рассматриваются проблемы обеспечения и оценивания качества, непосредственно касающиеся проекта ПО. Делается различие между атрибутами качества, наблюдаемыми во время работы ПО (эффективность, безопасность, удобство применения и др.), и атрибутами качества, присущими проекту ПО как таковому (концептуальная целостность, корректность, полнота и др.).

Для анализа качества проекта рекомендуются *методики* статического анализа, моделирования и прототипирования, а также *меры и метрики* для оценки проекта по различным аспектам – размера, структуры, функций и качества проектирования. Очевидно, что спектр метрик зависит от выбранного подхода к проектированию. В SWEBOK они классифицируются как *функционально-ориентированные* (структурные) метрики проекта (на базе анализа иерархических диаграмм представления проекта) и *объектно-ориентированные* (на базе диаграмм классов).

В разделе ***Нотации проектирования*** представлены две категории нотаций артефактов проекта – структурные и поведенческие.

Структурные нотации, как правило, графические, используются для представления структурных аспектов проекта - компонентов и их взаимосвязей, элементов архитектуры и их интерфейсов. К ним относятся формальные языки спецификаций и проектирования: ADL (Architecture Description Language), UML (Unified Modeling Language), IDL (Interface Description Language), диаграммы сущность-связь - ERD (Entity–Relation Diagrams), диаграммы классов и компонентов, диаграммы Джексона, структурные схемы, схемы классов (CRC Cards) и др.

Поведенческие нотации используются для описания динамических аспектов функционирования систем и их компонентов. К ним относятся диаграммы потоков данных, диаграммы действий (потоков управления), диаграммы взаимодействия

объектов, диаграммы переходов состояний, таблицы решений, блок-схемы, формальные языки спецификации, псевдокод, язык проектирования PDL и др.

Раздел ***Стратегии и методы проектирования ПО*** представляет общие стратегии проектирования и специализированные методы, предлагающие собственные нотации для описания различных аспектов проекта.

К *общим стратегиям* относятся: «разделяй и властвуй», «пошаговое уточнение», «снизу-вверх», «сверху–вниз», «абстракция данных» и «сокрытие информации», «применение паттернов и языков паттернов», «применение итеративного и инкрементного подходов» и др.

Функционально–ориентированные (*структурные*) *методы* базируются на структурном анализе, структурных картах, диаграммах потоков данных и др. Они ориентированы на идентификацию функций и их уточнение сверху–вниз и последующую разработку диаграмм потоков данных и описание процессов.

В методах *объектно–ориентированного проектирования* ключевую роль играет наследование, полиморфизм и инкапсуляция, а также абстрактные структуры данных и представления объектов.

Методы *проектирования от данных* во главу угла при проектировании ставят структуры данных, которыми должны манипулировать программы, и базируются на применении методов Джексона, Варнье-Орра и др. для задания входных и выходных данных структурными диаграммами.

Методы *компонентного проектирования* ориентированы на построение программ-компонентов с четко описанными интерфейсами и зависимостями, которые могут разрабатываться автономно (с ориентацией на их повторное использование) и интегрироваться для совместного функционирования в системе, взаимодействуя посредством интерфейсов. Компонентное проектирование образует базис для других видов проектирования и программирования, например, сервисно-ориентированного.

В SWEBOK упоминаются также другие подходы к проектированию, включая *формальные* методы проектирования и методы *трансформации*.

2.2.4. Область знаний «Конструирование программного обеспечения»

Конструирование ПО (Software Construction) – создание работающего ПО путем комбинирования методов кодирования, верификации, автономного и интеграционного тестирования и отладки компонентов. Границы между конструированием, проектированием и тестированием ПО существенно варьируются в зависимости от моделей ЖЦ и методологий разработки ПО.

Область знаний включает следующие разделы:

- основы конструирования ПО (Software Construction Fundamentals),
- управление конструирование ПО (Managing Construction),
- практические соображения (Practical Considerations).

Основу конструирования составляют концепции *минимизации сложности* ПО, *ожидания изменений* (готовности к изменениям), *облегчения проверки*, *соблюдения стандартов* (внешних и внутренних).

Управление конструированием базируется на моделях конструирования, планировании деятельности по конструированию и измерении артефактов процесса конструирования.

Модели конструирования описывают спектр выполняемых действий, их последовательность (взаимосвязь) и результаты. Виды моделей определяются применимыми стандартами ЖЦ, методологиями и приемами, например, Экстремальное программирование или RUP[5].

Планирование состоит в определении порядка создания и интеграции компонентов, распределении задач между исполнителями и действий по обеспечению качества ПО. Планирование регламентируется выбранной для проекта моделью конструирования.

Измерение в конструировании ориентировано на количественную оценку объема кода (разработанного, модифицированного, повторно используемого, удаленного), сложности кода; оценку статистических данных, касающихся верификации (инспекции) кода, интенсивности выявления и устранения дефектов в коде, а также трудоемкости конструирования. Данные измерений используются для управления конструированием с целью совершенствования самого процесса конструирования и повышения качества кода.

Практические соображения в данной области знаний касаются следующих *проблем конструирования:*

1) применение элементов *проектирования в конструировании* (особенно детального проектирования) и рекомендаций по использованию подходов, предложенных в области знаний «Проектирование ПО»;

2) применение различных *языков программирования и нотаций*.

В SWEBOK представлены такие типы языков конструирования, как *конфигурационный язык* (configuration language), позволяющий задавать параметры выполнения программной системы; *инструментальный язык* (toolkit language) – язык конструирования из повторно используемых компонентов в определенной инструментальной среде, а также *язык программирования* (programming language) – наиболее гибкий тип языков конструирования.

Определены такие виды *нотаций* современных языков программирования:

– *лингвистические нотации*, основанные на строковых инструкциях и предложениях для представления сложных программных конструкций;

– *формальные нотации*, основанные на точных, однозначных и формальных (математических) определениях;

– *визуальные нотации*, которые, в отличие от ориентированных на текст лингвистических и формальных нотаций, основаны на непосредственном визуальном представлении элементов ПО (как, например, язык UML);

3) кодирование с соблюдением хорошего *стиля кодирования*, включающего соглашения об именовании и структурировании исходного кода, использование классов, перечисляемых типов, переменных, поименованных констант; организацию исходного текста; использование структур управления, обработку ошибок и исключительных ситуаций, организацию доступа к параллельно используемым ресурсам, документирование кода и другие вопросы. В процессе конструирования должны использоваться внешние стандарты применяемых языков программирования (например, Ада 95, C++ и др.), языков описания данных (XML, SQL и др.), средств коммуникации (COM, CORBA и др.), интерфейсов компонентов (POSIX, IDL, APL), сценариев (UML и др.);

---

[5] Современные методологии конструирования (включая упоминаемые в SWEBOK) детально описаны в главе 11.

4) применение элементов *тестирования в конструировании*. Проводится две формы тестирования кода – модульное (автономное) и интеграционное. Виды тестирования определены в отдельной специальной области знаний (см. ниже). Рекомендованы два стандарта тестирования элементов ПО и документирования этого процесса - IEEE Std. 829:1998 «IEEE standard for software test documentation» и IEEE Std. 1008:1987 «IEEE Standard for software unit testing»;

5) реализация *повторного использования* ПО, включая выбор элементов, подлежащих повторному использованию, оценку потенциальной возможности повторного использования кода и тестов, ведение информации и отчетности по повторному использованию и других процедур, регламентируемых стандартом IEEE Std.1517:1999 «IEEE Standard for Information Technology – Software Lifecycle Processes – Reuse Process»;

6) *обеспечение качества кода* путем применения не только модульного и интеграционного тестирования, но и разработки, управляемой тестами, пошагового кодирования (итеративного кодирования с тестированием), технических обзоров, статического анализа и др.;

7) *интеграция* отдельно сконструированных процедур, классов, компонентов и подсистем.

2.2.5. Область знаний «Тестирование программного обеспечения»

Тестирование ПО (Software Testing) заключается в проверке работы кода в *динамике* на *конечном* (ограниченном) наборе тестов, охватывающих *выбранное подмножество* сценариев использования ПО и тестовых данных, и сравнении полученных результатов с *ожидаемыми* (ключевые слова – выделены *курсивом*).

Область знаний включает следующие разделы:

- основы тестирования (Software Testing Fundamentals),
- уровни тестирования (Test Levels),
- приемы тестирования (Test Techniques),
- метрики тестирования (Test-related Measures),
- процесс тестирования (Test Process).

Раздел ***Основы тестирования*** охватывает терминологию, основные проблемы тестирования и связь с другими областями знаний в SWEBOK. Базовые понятия в области тестирования определяются по стандарту IEEE Std. 610:1990 «Standard Glossary of Software Engineering Terminology». Рассматриваются ключевые проблемы, связанные с формированием критериев отбора (и адекватности) тестов, эффективности тестирования, идентификации дефектов, выбора «оракула» (который принимает решение о том, прошел или не прошел тест), тестируемости и др.

В разделе ***Уровни тестирования*** указаны *уровни объектов* ПО, для которых формируются тесты, и *цели тестирования* этих объектов. Выделены следующие уровни тестирования:

– *тестирование отдельных элементов* (*unit testing*), которое заключается в автономной проверке отдельных, изолированных и независимых частей ПО;

– *интеграционное тестирование*, ориентированно на проверку связей и способов взаимодействия (интерфейсов) отдельных компонентов;

– *тестирование системы* предназначено для проверки правильности функционирования системы в целом и выполнения сформулированных нефунк-

циональных требований (по безопасности, надежности и др.), а также правильности внешних интерфейсов системы со средой окружения.

В разделе выделены такие наиболее распространенные и обоснованные цели и, соответственно, *виды тестирования*: функциональное тестирование, регрессионное тестирование, тестирование эффективности, нагрузочное (стрессовое) тестирование, альфа и бета-тестирование, тестирование конфигурации, тестирование производительности, надежности и др. К виду тестирования отнесены также методы проверки поведения системы на этапе испытаний ПО и его приемки.

В разделе ***Приемы (техники) тестирования*** представлена следующая классификация приемов тестирования:

– приемы, базирующиеся *на интуиции и опыте исполнителя* (тестирование по опыту (ad hoc), исследовательское тестирование),

– приемы, базирующиеся на *анализе спецификации* (по формальным спецификациям, на основе эквивалентного разделения программы на части, анализа граничных значений, таблиц решений, анализа переходов состояний и др.),

– приемы, *базирующиеся на анализе кода* (покрытие условий и решений в блок-схеме, анализ «жизненного цикла» переменных и др.),

– специальные приемы *обнаружения дефектов* (обнаружение дефектов на основе анализа рисков, «подсев» дефектов, мутационное тестирование),

– приемы, базирующиеся *на сценариях и условиях использования* (по операционному профилю, по специализированным методикам испытаний системы - на надежность, безопасность и др.),

– приемы, ориентированные на определенный *тип и архитектурную природу* ПО (приемы объектно-ориентированного, компонентно-ориентированного, Web-ориентированного тестирования, тестирования на соответствие протоколам, тестирование систем реального времени и др.).

Измерения с помощью ***метрик, связанных с тестированием***, используются для анализа качества ПО, как инструмент оптимизации планирования и выполнения тестов, а также для управления процессом тестирования.

Для оценки программ в ходе тестирования рекомендуются метрики программ – размера, структурной сложности, частоты обращения к модулям; метрики (модели) дефектов и отказов - плотности дефектов, интенсивности отказов, роста надежности и др. Для оценки эффективности выполненных тестов рекомендуются метрики покрытия/глубины тестирования, оценки результатов мутаций и подсева дефектов, сравнительного анализа различных приемов тестирования.

В разделе ***Процесс тестирования*** указывается, что концепции, стратегии, техники и измерения тестирования должны быть объединены в *единый процесс тестирования* как деятельности по обеспечению качества на основе учета четырех элементов и связанных с ними факторов: людей, инструментов, регламентов и количественных оценок (измерений). Базовый стандарт по процессам ЖЦ ISO/IEC 12207 не выделяет всю деятельность по тестированию в качестве *единого* самостоятельного процесса, однако рассматривает соответствующие принципы работ по тестированию как неотъемлемую часть процессов ЖЦ.

Составной частью формализации процесса тестирования и основой для сертификации и оценивания организации по моделям зрелости и стандартам качества (например, CMMI) является *документация тестирования*. При ее составлении SWEBOK рекомендует руководствоваться стандартом IEEE Std. 829:1998 «Standard

for Software Test Documentation» по документированию планов тестирования, спецификации процедур тестирования, отдельных тестов и др.

Существенным моментом в процессе тестирования является принятие решения о *завершении тестирования* (необходимом объеме тестирования). Тщательные измерения покрытия кода или охвата функциональности должны сочетаться с оценками *стоимости тестирования*, а также *рисков отказов* и стоимости устранения их последствий при эксплуатации системы.

Практические соображения относительно процесса тестирования, охватываемые данной областью знаний, касаются повторного использования тестов и шаблонов тестирования, а также отдельных работ по тестированию: планирования, генерации тестовых сценариев, создания тестовой среды, регламента выполнения тестов, анализа результатов тестирования и отчетности.

2.2.6. Область знаний «Сопровождение программного обеспечения»

Сопровождение ПО (Software Maintenance) – совокупность действий по *эффективной* (с точки зрения затрат) *поддержке* функционирования ПО в период эксплуатации, включая внесение изменений при обнаружении ошибок в работе ПО, адаптацию ПО к новым условиям среды, повышение производительности и др.

Хотя фаза сопровождения в ЖЦ ПО обычно начинается сразу после приемки/передачи продукта и действует в течение периода гарантии (или технической поддержки), предварительные работы начинаются до выпуска ПО и включают планирование деятельности по сопровождению системы и организацию перехода к ее полнофункциональному использованию. После выпуска ПО выполняется обучение обслуживающего персонала и модификация ПО, при условии сохранения целостности продукта.

Область знаний включает следующие разделы:

- основы сопровождения (Software Maintenance Fundamentals),
- ключевые вопросы сопровождения ПО (Key Issue in Software Maintenance),
- процесс сопровождения (Maintenance Process),
- приемы сопровождения (Techniques for Maintenance).

В разделе ***Основы сопровождения*** представлены базовые концепции и терминология, составляющие основу для понимания роли и содержания работ по сопровождению ПО, и сопоставлены положения стандартов IEEE Std. 1219:1998 «IEEE Standard for Software Maintenance», ISO/IEC 14764:1999 «Information technology – Software maintenance» и ISO/IEC 12207.

Введены четыре *категории сопровождения*: корректирующее, адаптирующее, совершенствующее и профилактическое сопровождение.

Сопровождение в современной концепции, по сути, представляет собой *эволюционную разработку* системы. На работы по сопровождению приходится *большая* часть ресурсов ЖЦ ПО, причем, более 80% усилий связаны не столько с устранением отказов, сколько с другими работами. Это работы по улучшению дизайна, реализации расширений функциональных возможностей, созданию интерфейсов взаимодействия с другими (внешними) системами, адаптации к другим аппаратным платформам, обеспечению функционирования в среде обновленной телекоммуникационной инфраструктуры и т.п. вплоть до вывода ПО из эксплуатации.

Специалисты по сопровождению получают знания о продукте от разработчиков ПС. Организации или лица, на которые возлагается сопровождение системы, привлекаются к выполнению проекта задолго до его закрытия и непосредственно контактируют с разработчиками ПО, что позволяет снизить затраты на сопровождение системы после ввода в эксплуатацию.

В разделе ***Ключевые вопросы сопровождения ПО*** рассматриваются *технические*, *управленческие*, *стоимостные* и *измерительные* аспекты сопровождения.

«Техника» сопровождения охватывает вопросы быстрого изучения сопровождаемого ПО и локализации ошибок, использования тестирования, анализа последствий внесения изменений, сопровождаемости ПО (как показателя качества, обеспечиваемого при разработке ПО).

Сущность управленческих проблем заключается в поиске компромиссов в политике разработки и выпуска ПО (в срок и в рамках бюджета), с одной стороны, и политике дальнейшего сопровождения ПО, с другой стороны; в выборе сопровождающей организации, подборе персонала сопровождения, организации процесса сопровождения и др.

Вопросы оценки стоимости сопровождения решаются путем применения *параметрических моделей* и методов *экспертного оценивания* исходя из опыта по сопровождению ПО.

Для выполнения измерений при сопровождении SWEBOK рекомендует *общие метрики* (для всего ЖЦ ПО): размер, усилия, график работы, характеристики качества, а также *специализированные метрики* для оценки анализируемости, изменяемости, стабильности и тестируемости ПО - факторов, влияющих на уровень затрат по сопровождению ПО.

Процесс сопровождения должен определяться множеством выполняемых действий, входов и выходов и соответствовать требованиям IEEE Std. 1219:1998 или ISO/IEC 14764:1999[6].

Рассматриваются действия, характерные для процесса сопровождения:

– передача ПО от разработчика организации (службе) сопровождения;

– принятие/отклонение запросов на модификацию;

– извещение персонала сопровождения и отслеживание статуса запросов на модификацию и отчетов об ошибках;

– анализ влияния - анализ возможных последствий изменений, вносимых в существующую систему;

– поддержка ПО – консультационные услуги по запросам пользователей;

– ведение контрактов и обязательств об уровне услуг, предоставляемых сопровождающей организацией.

Кроме того, дополнительно определяются действия по планированию сопровождения, управлению конфигурацией, верификации и валидации ПО, оценке качества ПО, а также обучению персонала сопровождения.

Практические ***приемы сопровождения*** включают использование браузеров кода и других инструментальных средств для изучения ПО, а также таких методов эволюции ПО, как реинженерия, реверсная инженерия и рефакторинг.

Реинженерия – тщательное изучение функций и последующая радикальная переделка ПО путем его реорганизации, реструктуризации, перепрограммирования

[6] модель процесса (порядок выполнения действий) в этих стандартах отличаются.

или настройки на другую платформу или среду. Цель реинженерии – не столько обеспечение удобства его сопровождения, сколько замена устаревшего ПО.

Реверсная (*обратная*) *инженерия* –- восстановление спецификации (графов вызовов, потоков данных и др.) по исходному коду (например, после внесения в него множества изменений) с целью идентификации программных компонентов и связей между ними, формирования представления о ПО и обновления документации (в частности, для обеспечения последующей реинженерии).

Рефакторинг - трансформация ПО с сохранением архитектурных и технологических решений и без изменения его поведения, предпринимаемая с целью улучшения структурных характеристик и качественных показателей (включая удобство сопровождения). Может рассматриваться как одна из форм реверсной инженерии.

2.2.7. Область знаний «Управление конфигурацией программного обеспечения»

Управление конфигурацией или *конфигурационный менеджмент* (Configuration Management) – деятельность, связанная с контролем эволюции и целостности программного продукта в ходе выполнения проекта ПО системы путем идентификации всех артефактов[7], отслеживания их изменений и ведения отчетности о состоянии конфигурации ПО системы.

Работы по управлению конфигурацией ПО (далее SCM, от Software Configuration Management) включают тщательное планирование процесса SCM[8], определение характеристик элементов конфигурации, учет указанных характеристик и контроль за их изменениями, составление отчета о внесенных изменениях в конфигурацию и статусе их реализации; аудит конфигурации и проверку соответствия внесенных изменений заданным требованиям.

Область знаний включает следующие разделы:

- управление процессом (Management of the SCM Process),
- идентификация конфигурации ПО (Software Configuration Identification),
- контроль конфигурации ПО (Software Configuration Control),
- учет состояния конфигурации (Software Configuration Status Accounting),
- аудит конфигурации ПО (Software Configuration Auditing),
- управление выпуском и поставка ПО (Software Release Management and Delivery).

Управление процессом *конфигурационного менеджмента* осуществляется с учетом организационного контекста[9], особенностей и ограничений аппаратно-программной архитектуры системы и ПО (включая размер и сложность) и принятой модели процесса, устанавливаемой применяемыми стандартами и руководствами по SCM (например, ISO/IEC 12207, ISO/IEC 15504, CMMI).

---

[7] Имеются в виду не только версии кода ПО, но и проектные элементы (схемы, диаграммы, расчеты и др.), а также документы проекта.

[8] Для ПО систем SQM должно согласовываться с управлением конфигурацией системы, включая аппаратное обеспечение (или аппаратно-программное обеспечение).

[9] Особенности оргструктуры проекта, наличие других процессов ЖЦ в модели процессов, наличие инфраструктуры для управления конфигурацией и др.

Планирование процесса SCM охватывает распределение обязанностей и полномочий и способы отчетности; определение ресурсов и составление графика работ; подбор (разработку) инструментов (для ведения SCM-библиотек, запросов на изменения, управления кодом, сборки программных продуктов, отчетности и др.), а также построение плана управления конфигурацией (например, в соответствии с IEEE Std.828:1998 «Standard for Software Configuration Management Plans»).

Для *контроля выполнения процесса SCM* используются метрики, состав которых определяется потребностями проекта и набором характеристик, значения которых могут быть получены с помощью применяемых инструментов управления конфигурацией (количество внесенных изменений в одном элементе, количество отвергнутых изменений, количество версий и др.).

Работы по ***идентификации конфигурации*** включают: определение контролируемых элементов (исходного и исполняемого кода, библиотек компонентов, проектной документации, графиков работ и др.); определение схемы идентификации элементов и их версий; документирование функциональных и физических характеристик элементов; описание применяемых инструментов и приемов управления элементами конфигурации; определение базовой версии ПО; описание взаимосвязей между элементами, версиями и др.

Контроль конфигурации подразумевает управление изменениями в течение ЖЦ ПО. Включает анализ запросов на изменения и определение того, какие именно изменения должны быть сделаны и какие полномочия необходимы для их утверждения или отклонения, в чем суть поддержки реализации изменений и каков порядок формального утверждения «отхода» от проектных требований. Принятые (утвержденные) запросы на изменения реализуются с помощью определенных процедур и в определенном порядке с учетом взаимосвязи элементов (одновременно или последовательно). «Закрытие» изменений предполагает аудит конфигурации.

Учет состояния конфигурации подразумевает ведение всей информации об элементах конфигурации и проделанных изменениях, автоматизированный контроль и анализ внесенных изменений, и генерацию отчетности. Отчетность может использоваться как основа для проведения различных количественных оценок в интересах руководства или процесса разработки, или для повышения эффективности процесса управления конфигурацией ПО.

Аудит конфигурации – это деятельность по оценке соответствия продуктов и процессов принятым стандартам, инструкциям, планам и процедурам. В ходе аудита определяется степень, в которой элемент конфигурации удовлетворяет заданным функциональным и физическим характеристикам. Различают неформальный аудит (в контрольных точках проекта) и формальный аудит двух типов: функциональный и физический аудит конфигурации, который завершается фиксацией *базовой версии* продукта. Цель функционального аудита - убедиться в том, что контролируемый программный элемент полностью соответствует заданным спецификациям, а физического – убедиться, что проект (дизайн) и документация точно согласуются с самим программным продуктом. Может также проводиться выборочный внутренний аудит элементов базовых версий.

Управление выпуском версий и поставкой ПО включает отслеживание версий элементов конфигурации; обеспечение оперативного доступа к информации относительно элементов конфигурации и системы, к которым эти элементы относятся; сборку (объединение) компонентов; создание новых версий системы на ос-

нове существующей путем внесения изменений в конфигурацию; согласование версии продукта с требованиями. Управление выпуском охватывает идентификацию, упаковку и передачу элементов продукта и документации заказчику.

2.2.8. Область знаний «Управление инженерией программного обеспечения»

Управление (*менеджмент*) *инженерией ПО* (Software Engineering Management) – приложение общих методов управления к инженерной деятельности по созданию ПО, осуществляемой в рамках *проектов* ПО и регламентированной *процессами ЖЦ* ПО[10].

Как любое управление, менеджмент ПО подразумевает планирование, координацию действий, мониторинг и измерение, контроль и принятие управленческих решений, а также отчетность по управлению проектом. Отличительные особенности управления программной инженерией связаны с учетом таких факторов, как: нестабильность требований потребителей, итеративность выполняемых процессов, высокий уровень новизны и сложности разработок (не позволяющий в полной мере использовать наработки и опыт), быстрота обновления технологий и т.п.

Распределение и координация взаимодействия людских, финансовых и технических ресурсов при реализации задач проекта возлагается на менеджера проекта. В решении общих вопросов управления он должен руководствоваться таким источником знаний, как Руководство к PMBOK, а в решении специальных вопросов управления программными проектами – информацией из данной области знаний, а также положениями ISO/IEC 12207, где управление проектом рассматривается как необходимый *организационный процесс* ЖЦ, выполняемый в соответствии с политиками, целями и принятыми стандартами организации-разработчика ПО.

Область знаний включает следующие разделы:

- инициирование и определение рамок проекта (Initiation and Scope Definition),
- планирование проекта (Software Project Planning),
- выполнение проекта (Software Project Enactment),
- обзор и оценка (Review and Evaluation),
- закрытие (Closure),
- измерения в инженерии ПО (Software Engineering Measurement).

Раздел ***Инициирование и определение рамок проекта*** касается вопросов, связанных с принятием решений о начале программного проекта. Это вопросы эффективного выявления, анализа, специфицирования и проверки требований к ПО с использованием различных методов извлечения требований; определения целей, содержания и ограничений проекта; оценки осуществимости проекта с различных точек зрения (технической, технологической, финансовой, политической и др.); определения процедур пересмотра требований и анализа зависимостей и рисков для оценки влияния изменения требований.

Планирование программного проекта представляет собой *итеративный* процесс, базирующийся на анализе требований и содержания проекта, а также условий его осуществимости.

---

[10] Фактически управление программной инженерией объединяет в себе управление процессами ЖЦ и управление проектами, выполняемыми с помощью процессов.

Первый шаг планирования предполагает выбор *модели разработки* (спиральной, эволюционной и т.п.) и *применяемых процессов* ЖЦ, а также методов и инструментов, используемых при управлении проектом, в частности, для *декомпозиции проекта* в виде *задач* с ассоциированными входами, выходами и условиями завершения. Далее по каждой задаче специфицируются результаты выполнения (артефакты) и производится оценка трудоемкости и стоимости работ с помощью аналого-сопоставительных методов (по накопленным историческим данным) или экспертным путем. Идентифицируются связи и зависимости между задачами и потенциально критические аспекты проекта (что может быть сделано, например, с помощью метода *анализа критического пути*). Для задач определяются ожидаемые сроки выполнения и расписание (график) работ (например, в форме *PERT-диаграмм*). Требования к ресурсам трансформируются в прогнозируемые затраты на проект. Все эти действия проводятся итеративно до тех пор, пока не будет достигнут консенсус между соответствующими заинтересованными лицами – в первую очередь, менеджером и формируемой группой проекта, после чего осуществляется распределение обязанностей/ответственности людей, а также технических и финансовых ресурсов (с отражением, например, в *диаграмме Ганта*).

К данному разделу знаний SWEBOK отнесены также вопросы *управления рисками* и *управления качеством* в проекте, поскольку планирование проекта должно осуществляться с учетом дополнительных затрат ресурсов, в том числе и на эту деятельность. Управление рисками заключается в регулярной идентификации и анализе рисков, оценке их критичности и упорядочению по приоритетам и разработке мероприятий по устранению или смягчению рисков, начиная с наиболее критичных для проекта. Управление качеством заключается в планировании и осуществлении мероприятий, нацеленных на достижение характеристик качества программного продукта, указанных в спецификации требований.

Планироваться должен также процесс *управления* разработанным *планом проекта*, что особенно важно в условиях перманентных изменений требований и окружения проекта. Актуальность плана должна систематически контролироваться и, при необходимости, восстанавливаться.

Основой успешного ***Выполнения проекта*** является следование плану проекта, соблюдение процессов, представленных в плане, и управленческая деятельность по систематическому оцениванию и измерениям, мониторингу, контролю и отчетности по проекту. Существенным моментом для выполнения проекта является у*правление контрактами с поставщиками*, включающее подготовку и выполнение соглашений с поставщиками, мониторинг деятельности поставщиков, приемку поставляемых продуктов и интеграцию этих продуктов в рамках проектных работ.

Процесс выполнения проекта сопровождается *регулярными измерениями* (усилий, расходованных средств, выполненных объемов работ и др.), а результаты измерений *оцениваются* по отношению к плану проекта в ходе *мониторинга проекта*. Соблюдение плана, а также выход проекта за ограничения, проверяется регулярно, через предопределенные интервалы времени, и сопровождается оценками (и переоценками) известных рисков и выявлением новых, и проверкой удовлетворения требований к качеству. Отчетность по результатам мониторинга обеспечивает базис для принятия решений и *регулирования*[11] проекта, т.е. выполнения корректи-

[11] Используемый в Руководстве к SWEBOK термин *control* лучше переводить как *регулирование*, а *controls* - как *управляющие воздействия*, *рычаги регулирования*.

рующих действий, что, в свою очередь, может привести к изменению плана, отдельных работ, документов и других активов проекта. Выполнение проекта сопровождается также *регламентными отчетами* по состоянию проекта, направляемыми всем заинтересованным лицам.

Формальный ***Обзор и оценка проекта*** выполняются в контрольных точках при достижении определенных вех проекта для оценивания *общего прогресса* в достижении установленных целей и удовлетворении требований заинтересованных лиц, а также для оценивания *эффективности* процессов, работы персонала и инструментов и методов, которые применялись в заданном промежутке времени.

После выполнения всех планов, окончания процессов и подтверждения достижения *критериев завершения* проекта наступает ***Закрытие проекта***. На этой стадии оценивается успешность проекта, принимается и утверждается решение о его закрытии и передаче продукта заказчику, и выполняются действия по архивированию проектных данных и обновлению измерений, анализу результатов, полученных в процессе работы над проектом, и улучшению процессов.

Измерения в инженерии ПО направлены на получение количественных оценок отдельных характеристик процессов, ресурсов и артефактов проекта с целью их использования при управлении инженерией ПО. Ключевые термины, а также методы измерений, определены в стандарте ISO/IEC 15939:2002 «Software Engineering - Software Measurement Process», который описывает информационную модель измерений и процесс, определяющий действия/работы и задачи, необходимые для реализации процесса измерений. В полном соответствии с этим стандартом в данном разделе рассматриваются следующие *вопросы:* формулирование требований в отношении измерений (уровень, цели и объекты измерений, *потребители результатов* измерений); выбор *метрик*, спектр и объемов *собираемых данных*; определение *процедур анализа* и ведения отчетности по измерениям; *планирование процесса* измерений и выделения ресурсов; организация измерений в подразделениях и *выполнение процесса* измерений (интеграция действий по измерениям в выполняемые процессы); анализ результатов измерений, *составление отчета* о полученном *информационном продукте* и его передача потребителю измерений; оценка хода измерений и совершенствование процесса измерений.

2.2.9. Область знаний «Процесс программной инженерии»

Процесс программной инженерии (Software Engineering Process, SEP), как область знаний, представлен в Руководстве к SWEBOK на двух уровнях описания.

Прежде всего (на нижнем уровне), SEP – это множество взаимосвязанных видов деятельности по *созданию программных систем* в рамках проектов, выполняемых на основе интегрированных процессов ЖЦ, адаптированных к конкретным проектам. В этом, привычном для разработчиков, понимании процесс программной инженерии рассматривается практически повсюду в Руководстве к SWEBOK.

Однако данная область знаний охватывает и другой аспект процесса программной инженерии, определяя его (на верхнем, мета уровне) как множество взаимосвязанных видов деятельности по определению, реализации, оценке, измерению, управлению и совершенствованию собственно *процессов ЖЦ*, формирующих *базовый процесс программной инженерии* в организации (безотносительно к проектам). Именно этот аспект SEP имеется в виду при рассмотрении данной области знаний SWEBOK.

Область знаний включает следующие разделы:

- реализация и изменение процесса (Process Implementation and Change),
- определение процесса (Process Definition),
- оценивание процесса (Process Assessment),
- измерения процесса и продукта (Process and Product Measurement),

Раздел ***Реализация и изменение процесса*** описывает инфраструктуру, действия, модели и практические соображения по реализации SEP и его изменению. Изложение вопросов раздела базируется на стандартах ISO/IEC 12207 и ISO/IEC 15504 «Information technology - Software process assessment».

Инфраструктура SEP может организовываться по типу *Группы процесса программной инженерии SEPG* (от Software Engineering Process Group) или по типу *экспериментальной фабрики EF* (от Experience Factory)[12]. SEPG образуют специалисты организации, в обязанности которых (возможно наряду с работой по проектам) входит внедрение и совершенствование процессов ЖЦ. В отличие от SEPG, экспериментальная фабрика никак не связана с выполнением проектов в организации, представляет собой отдельное подразделение (или другую организацию) и всецело ориентирована на институциализацию накопленного в масштабе организации коллективного опыта и извлеченных уроков, путем разработки, обновления и внедрения в проекты «пакетов опыта» (experience packages). Пакеты формируются из типовых активов процесса и могут включать, например, руководства, модели и методики, курсы обучения и др. Подразделения, работающие по проектам, в свою очередь, предлагают на рассмотрение EF свои наработки по внедренным и выполняемым процессам, что может послужить толчком к их усовершенствованию.

Помимо создания инфраструктуры *управление процессами* в области программного обеспечения включает еще три действия, выполняемые в рамках итеративного цикла: 1) *планирование* внедрения нового процесса или совершенствования существующего, а также разработку плана реализации или изменения процесса; 2) собственно *осуществление* реализации или изменения процесса; и 3) *оценку* реализации процесса и отдачи от его внедрения.

В SWEBOK упомянуты две распространенные *модели реализации и изменения* SEP – модель *QIP* (от Quality Improvement Paradigm) и модель *IDEAL* (от Initiating-Diagnosing-Establishing-Acting-Learning)[13] , а также два подхода к *оценке реализации и изменения процесса*, первый из которых состоит в оценке самого процесса, второй - в оценке результатов (наработок) процесса[14].

Определение процесса может быть представлено процедурой, рекомендацией или стандартом и основываться на таких важных факторах, как природа работ (например, разработка или сопровождение), прикладная область, а также модель ЖЦ и уровень зрелости самой организации.

Модель жизненного цикла задает высокоуровневое определение фаз (стадий) разработки ПО, включая основные работы и их взаимосвязи[15]. Определение процесса программной инженерии обычно детальнее, чем модель, однако не описывает порядка выполнения работ во времени. Это определение может быть «выстроено» из определений выбранных процессов ЖЦ в соответствии с любой моделью.

---

[12] Чаще говорят о SEPG, которая может быть или не быть организована по типу EF.

[13] Механизмы работы EF, а также модели QIP и IDEAL рассматриваются в главе 11.

[14] Эти подходы рассматриваются в главе 12.

[15] Модели ЖЦ (каскадная, спиральная и другие) представлены в приложении 1.

Можно согласиться с С.Орликом [1], что совокупность модели ЖЦ, процессов ЖЦ и приемов их выполнения определяют *методологию* поддержки ЖЦ ПО.

Основным источником знаний по всем процессам ЖЦ ПО являются стандарт ISO/IEC 12207 (а также руководство к нему - ISO/IEC 15271:1998) и IEEE Std. 1074:1997 «IEEE Standard for Developing Software Life Cycle Processes» (стандарт по разработке процессов ЖЦ). Кроме того, существуют стандарты по отдельным процессам ЖЦ, например, IEEE Std. 1219:1998 «IEEE Standard for Software Maintenance» или ISO/IEC 14764:1999 «Information technology - Software Maintenance», IEEE Std. 1540:2001 «IEEE Standard for Software Life Cycle Processes - Risk Management», IEEE Std. 1517:1999 «Software Reuse Processes», ISO/IEC 15939:2002 «Software Engineering - Software Measurement Process» и другие[16].

Процессы программной инженерии могут также определяться с учетом потребностей системы менеджмента качества, внедренной в организации и сертифицированной по стандарту ISO 9001:2001, который формулирует требования к процессам управления качеством[17].

Процессы могут определяться на различных уровнях абстракции и с использованием различных *нотаций*: диаграмм потоков данных; перечислений целей и результатов процессов (как, например, в ISO/IEC 15504); набора процессов и их декомпозиции в виде действий и задач, определенного на естественном языке (как, например, в ISO/IEC 12207); диаграмм переходов состояний, SADT, IDEF0 и др.

Для применения в конкретных проектах предопределенные (стандартизованные) процессы должны адаптироваться в соответствии с организационным контекстом, размером проекта, существующими регулирующими требованиями, отраслевой и ведомственной практикой и корпоративной культурой. Рекомендации по адаптации содержатся в ISO/IEC 12207 и ISO/IEC 15504.

Используемые в проектах методологии могут поддерживать автоматизированное определение процессов, обеспечивая их описание с использованием тех или иных нотаций, или предоставлять руководства по определению процессов.

Оценивание процесса проводится с использованием соответствующих моделей и методов.

Типовая модель оценивания (SPICE) определена в стандарте ISO/IEC 15504. Там же содержатся требования по соответствию других моделей (например, SW CMM, CMMI, Bootstrap и др.) этой типовой модели. На практике используются две архитектуры моделей оценивания: непрерывное оценивание и поэтапное оценивание. Выбор архитектуры обусловлен целями и возможностями организации.

Методы оценивания определяют порядок и способы получения количественных параметров оцениваемого процесса, которые характеризуют мощность этого процесса, и направлены либо на совершенствование процессов, либо на оценку готовности (зрелости) организации для выполнения определенного контракта[18].

Измерения процессов и продуктов этих процессов могут проводиться для идентификации сильных и слабых сторон процессов (в определенном контексте их применения) и инициирования работ по их изменению, а также для оценивания процессов после того, как они реализованы и/или изменены.

---

[16] Перечень действующих стандартов представлен в приложении 3.

[17] Процессы ЖЦ, внедряемые в соответствии с ISO 9001, описаны в главе 12.

[18] Модели и методы оценивания процессов описаны в главе 12.

Кроме того, измерения составляют основу для оценивания ситуации и принятия решений при управлении проектом, для оценивания качества, размера, структуры и других характеристик программных продуктов, а также эффективности применяемых технологий.

Ключевые понятия, термины и методы измерений в программной инженерии определены в стандарте ISO/IEC 15939:2002 «Software Engineering - Software Measurement Process» наряду с определением самого базового процесса измерения характеристик процессов и продуктов[19].

Подходы к измерению процессов классифицируются как *аналитические* и *эталонные* (benchmarking) и используются совместно, поскольку основываются на различных типах взаимодополняющей информации.

Аналитические подходы включают экспериментальное исследование процесса в специальной среде (наблюдение за процессом, моделирование поведения процесса и отклонений), обзор определения процесса (например, его сопоставление со стандартом ISO/IEC 12207), ортогональную классификацию дефектов (изъянов в процессе), анализ коренных причин (Root Case Analysis), статистический контроль процесса.

Подходы «от эталонов» базируются на идентификации «совершенной» организации процесса, приемов и инструментов, и последующем использовании опыта (например, наработок таких организаций, как SEI и PMI).

2.2.10. Область знаний «Инструменты и методы программной инженерии»

Программные инструменты предназначены для автоматизированной поддержки выполнения процессов ЖЦ ПО (с охватом отдельных процессов и методов или всего ЖЦ). Методы, как и инструменты, варьируются по содержанию (в зависимости от охватываемой области приложения) и применяются на отдельных фазах жизненного цикла или в течение всего ЖЦ.

Область знаний включает два раздела:

- инструменты программной инженерии (Software Engineering Tools),
- методы программной инженерии (Software Engineering Methods).

Раздел ***Инструменты программной инженерии*** описывает классы инструментов, используемых практически во всех областях знаний SWEBOK. Дополнительно рассматриваются вопросы интеграции инструментов и создания платформ разработки, применения метаинструментов и подходов к оцениванию инструментов. Краткая характеристика инструментов в предложенной SWEBOK классификации представлена в таблице 2.1.

Методы программной инженерии классифицируются как:

– *эвристические* (неформальные) методы: структурные, объектно-ориентированные, ориентированные на данные;

– *формальные* (математически обоснованные) методы: языки и нотации специфицирования, методы уточнения (трансформации), методы подтверждения/ доказательства,

– *методы прототипирования*: стили прототипирования, цели прототипирования, приемы оценки результатов прототипирования.

---

[19] Подходы к измерениям и процесс измерения подробно описан в главе 4.

Таблица 2.1. **Классификация инструментов по SWEBOK**

Класс	Типы	Описание назначения и примечания
Инструменты работы с требованиями	Средства моделирования	Управление программными требованиями – извлечение, анализ, специфицирование и проверка
	Средства трассировки	Отслеживание проекций требований на всех этапах ЖЦ
Инструменты проектирования	В SWEBOK не классифицированы	Можно классифицировать по базовым нотациям (SADT, UML, DSL…), по целевым задачам (бизнес-проектирование, проектирование БД…)
Инструменты конструирования	Редакторы программ	Создание и модификация исходного кода и, возможно, документации (в IDE – интегрир. средах)
	Компиляторы и генераторы кода	Трансляторы с командной строкой. Однако есть тенденция интеграции с редакторами в IDE
	Интерпретаторы	Исполнение программ посредством эмуляции
	Отладчики	
Инструменты тестирования	Генераторы тестов	Разработка сценариев тестирования
	Средства выполнения тестов	Исполнение тестовых сценариев в контролируемом окружении
	Инструменты оценки тестов	Оценка результатов выполнения тестов
	Средства управления тестами	Поддержка всех аспектов процесса тестирования
	Инструменты анализа производительности (как инструменты *целевого анализа*)	Оценка поведения программы с точки зрения производительности. Другие инструменты *целевого анализа* – для функционального тестирования, тестирования безопасности, тестирования пользовательского интерфейса, нагрузочного тестирования и др.
Инструменты сопровождения	Средства визуализации	Инструменты, облегчающие понимание
	Инструменты реинженерии	Восстановление спецификации и архитектуры ПО по коду
Инструменты управления конфигурацией	Инструменты отслеживания	Отслеживание дефектов, расширений и проблем
	Инструменты управления версиями	
	Инструменты сборки и выпуска	Управление задачами сборки и выпуска продуктов, включая инсталляцию
Управление инженерией	Инструменты планирования и отслеживания работ	Календарное планирование работ по проекту, оценка трудоемкости и прогнозирование стоимости
	Инструменты управления рисками	Идентификация рисков, оценка последствий, мониторинг рисков
	Средства количественной оценки	Ведение измерений, поддержка Программы измерений
Инструменты поддержки процессов	Инструменты моделирования процессов	Более широкий класс инструментов моделирования, используемых, в частности, для описания и разработки модели процессов
	Средства управления процессами	Инструменты для управления проектами и управления конфигурациями

Класс	Типы	Описание назначения и примечания
Инструменты поддержки процессов	Интегрированные CASE-среды и ролевые платформы разработки	Охват многих фаз ЖЦ и поддержка «смежных» работ по управлению требованиями, управлению конфигурациями и изменениями, тестированию и оценке качества
	Процессо-ориентированные среды разработки	Ведение всей информации по процессам ЖЦ, предоставление ее (а также сопутствующих рекомендаций) пользователю
Инструменты обеспечения качества	Инструменты инспектирования	Поддержка обзоров и аудитов
	Инструменты статического анализа	Анализ программных артефактов, данных, потоков работ и зависимостей и проверка соответствия их свойств заданным характеристикам
Дополнительные аспекты	Приемы (виды) интеграции инструментов	Интегрированные среды разработки (IDE), библиотеки программ/компонентов, программные платформы (Java, J2EE и Microsoft .NET), платформы распределенных вычислений (CORBA и WebServices)
	Метаинструменты	Средства генерации других инструментов (компилятор компиляторов и др.)
	Приемы оценки инструментов	Подходы к оценке ввиду эволюционирования инструментов программной инженерии

2.2.11. Область знаний «Качество программного обеспечения»

Качество ПО (Software Quality) – совокупность свойств программного продукта, обеспечивающих его способность удовлетворять *установленным* или *предполагаемым* требованиям заказчика (пользователя). В это определение понятия вкладывается разный смысл в зависимости от контекста его применения и аспектов рассмотрения. Можно различать, например, рыночное качество ПО (market-driven quality), потребительское качество (customer-driven quality); качество ПО с позиций его сопровождения, обеспечения безопасности и т.п.; качество внутренне, внешнее и эксплуатационное. Залог повышения качества при процессном подходе – применение процессов проверки (верификации, тестирования, аудита и др.), нацеленных на своевременное обнаружение и устранение дефектов в ПО и изъянов в самих процессах ЖЦ. Даная область знаний SWEBOK охватывает вопросы *статической* проверки продуктов и процессов. Вопросы тестирования, как способа *динамической* проверки работы ПО, в данной области знаний SWEBOK не рассматриваются.

Область знаний включает следующие разделы:

- основы качества ПО (Software Quality Fundamentals),
- процессы управления качеством (Software Quality Management Processes),
- практические соображения (Practical Considerations).

В разделе ***Основы качества ПО*** подчеркивается, что все вопросы качества должны рассматриваться в контексте *требований к программному продукту*, которые не только определяют необходимые характеристики качества ПО, но и обусловливают выбор методов их количественной оценки и критериев приемки ПО.

Инженеры по программному обеспечению должны воспринимать вопросы качества как часть своей *профессиональной культуры*. Требования к характеристикам качества всегда являются результатом определенного *компромисса* (как по но-

менклатуре и значениям характеристик, так и по *стоимости* обеспечения и *ценности* этих характеристик для заказчика). Стоимость качества дифференцируется как стоимость предупреждения дефектов, стоимость оценки качества, цены внутренних сбоев и внешних потерь. Большинство *альтернативных* компромиссных решений по качеству должно предлагаться и приниматься в процессе работы с требованиями, однако вопросы, касающиеся качества, могут подниматься на протяжении всего ЖЦ ПО и в рамках любых процессов ЖЦ.

Способы построения таксономий и *моделей качества* различны - иерархии, графические образы и т.п. *Базовый стандарт по качеству* ISO/IEC 9126-1:2001 «Software Engineering - Product Quality, Part 1: Quality Model» предлагает рассматривать качество на трех уровнях его обеспечения в артефактах проекта (внутреннее качество, внешнее качество и качество ПО в эксплуатации) и рекомендует иерархическую модель качества для каждого из этих уровней. Оценивается качество в соответствии со стандартом ISO/IEC 14598:1999 «Information Technology - Software Product Evaluation».

Непосредственное отношение к качеству создаваемого продукта имеет *управление качеством* (SQM, от Software Quality Management) и *качество процессов* программной инженерии. В области управления (менеджмента) качества важнейшими являются стандарты TickIT и ISO 90003:2004 «Software and Systems Engineering - Guidelines for the Application of ISO 9001:2000 to Computer Software», а в области качества (мощности, зрелости) процессов ЖЦ – стандарты CMMI и ISO/IEC 15504:2004 «Information Technology - Software Process Assessment» (модель SPICE, от Software Process Improvement and Capability dEtermination). Эти стандарты взаимно дополняют друг друга, причем сертификация процессов по ISO 9001 помогает в достижении старших уровней зрелости по CMMI.

Понимание термина *продукт* в данной области знаний расширено включением *любых артефактов*, создаваемых в результате выполнения всех процессов ЖЦ, используемых для создания конечного программного продукта (например, спецификации системных требований и требований для программных компонентов, моделей, кода, тестовой документации и др.). Признаком высокой зрелости организации является наличие и использование собственных (исторических) данных относительно требований к качеству отдельных артефактов и результатов оценки их фактического достижения. Это дает возможность оценивать соответствие заданным характеристикам качества не только конечного продукта, но и промежуточных результатов/продуктов ЖЦ в рамках всех процессов программной инженерии.

Качество ПО может *повышаться* в ходе итеративного процесса его постоянного улучшения, требующего контроля, координации и обратной связи при управлении одновременно выполняемыми процессами: разработки, обнаружения, устранения и предотвращения дефектов, а также совершенствования процессов ЖЦ.

Раздел ***Процессы управления качеством ПО*** рассматривает во взаимосвязи процессы *обеспечения гарантии качества* (SQA, от Software Quality Assurance), *верификации* и *валидации*, *обзора* (*совместного просмотра*) и *аудита*, а также собственно *процесс управления качеством* (SQM), обеспечивающий функции контроля и оценки по всем аспектам процессов, продуктов и ресурсов, включая требования к процессам, измерения процессов и их результатов и установление обратной связи.

SQA, как отмечается в руководстве к SWEBOK, концентрируется *на процессах*. Роль SQA состоит в том, чтобы обеспечить планирование процессов, дальней-

шее исполнение процессов на основе заданного плана и проведение необходимых измерений процессов. Верификация и валидация (V&V, от Verification and Validation) применяется к *артефактам* проекта в его контрольных точках с целью проверки и подтверждения их правильности. Действия по V&V регламентируются базовыми стандартами IEEE 1012:1998 «Software Verification and Validation» и IEEE 1059:1993 «IEEE Guide for Software Verification and Validation Plans». Деятельность по обзорам и аудитам продуктов включает пять типов действий: управленческие обзоры, технические обзоры, инспекции, сквозной контроль и аудиты, и регламентируется стандартом IEEE 1028:1997 «IEEE Standard for Software Reviews».

Практические соображение в данной области касаются, прежде всего, определения *факторов*, влияющих на выбор стратегии SQM. Один из них – *критичность области применения* ПО. Комплексной характеристикой качества такого ПО является *гарантоспособность* (dependability) - совокупность свойств ПО, обеспечивающих его отказоустойчивость (fault tolerance), безопасность эксплуатации (safety), защищенность информации (security), а также удобство и простоту использования (usability). Другой важный фактор – уровень необходимой *целостности* ПО, связываемый с оценкой возможных последствий отказа ПО и вероятности его возникновения. Обеспечение целостности ПО требует привлечения приемов анализа опасностей окружению и угроз целостности информации.

В этом разделе высказываются также практические соображения относительно описания *характеристик дефектов* (рекомендуемый стандарт IEEE 1044:1993 «IEEE Standard for the Classification of Software Anomalies»), сбора и регистрации данных о дефектах, отчетов об устранении дефектов и т.п.

Дается классификация приемов SQM, включая статические и динамические, коллективные и индивидуальные, а также аналитические (формальные и неформальные) приемы. Подчеркивается, что тестирование может и должно применяться для решения таких задач SQM:

- оценка и тестирование инструментов, используемых в проекте (согласно ISO/IEC 14102:1995 «Information Technology - Guideline for the Evaluation and Selection of CASE Tools»),
- тестирование соответствия компонентов ПО и COTS-продуктов (от commercial of-the-shelf, готовый к использованию продукт, продукт «с полки») потребностям их использования в создаваемом продукте (согласно ISO/IEC 12119: 1994 «Information Technology–Software Packages - Quality Requirements and Testing»).

Эта область знаний охватывает также такие аспекты, как количественная оценка качества ПО с использованием метрик и оценка стоимости процесса SQM.

2.3. Ядро знаний по управлению проектами (PMBOK)

2.3.1. Структура руководства по PMBOK

Руководство к ядру профессиональных знаний в области управления проектами *PMBOK* (от *Project Management Body of Knowledge*) – стандарт по управлению проектами, разработанный общественной организацией – Институт управления проектами (PMI, Project Management Institute, www.pmi.org) [15]. Первый вариант руководства PMBOK был опубликован PMI в 1987 году с целью документирования и стандартизации общепринятых и широко применяемых практических приемов управления проектами в промышленности. Как и SWEBOK, руководство PMBOK

не раз пересматривалось и дополнялось по мере совершенствования знаний и опыта управления проектами в различных отраслях.

Версия PMBOK 2000 года оформлена как стандарт IEEE Std. 1490: 2003 «IEEE Guide Adoption of PMI Standard A Guide to the Project Management Body of Knowledge» и представляет собой описание базовой лексики и согласованной структуры взаимосвязанных *процессов* и *ключевых областей знаний*, олицетворяющих передовую практику управления проектами в любых отраслях не только промышленности, но и создания ПО. Текущая третья версия PMBOK (2004 года)[20] принята как национальный стандарт США ANSI/PMI 99-001-2004 и рассматривается как проект нового международного стандарта по управлению проектами [16]. Основное отличие текущей версии PMBOK от предыдущих – четкое декларирование «процессного» подхода и реструктуризация всего материала руководства по отдельным областям знаний в соответствии с этим подходом [17].

PMBOK определяет 39 процессов ЖЦ проекта, объединенных в *5 базовых групп процессов* и *9* ключевых *областей знаний*, типичных практически для любых проектов. *Группы процессов* таковы: 1. Инициация. 2. Планирование. 3. Исполнение. 4. Мониторинг и управление. 5. Завершение.

Ключевые *области знаний*:

1. Управление интеграцией проекта.
2. Управление содержанием (рамками) проекта.
3. Управление продолжительностью (сроками) проекта.
4. Управление стоимостью проекта.
5. Управление качеством проекта.
6. Управление человеческими ресурсами.
7. Управление коммуникациями проекта.
8. Управление рисками проекта.
9. Управление закупками (поставками) проекта.

В каждой фазе проекта в рамках каждого направления деятельности (отвечающего соответствующей области знаний) выполняются (итеративно) некоторые или все процессы той или иной группы (таблица 2.2).

Процессы связаны между собой (по входу и выходу) и взаимодействуют, как показано на рисунке 2.3 применительно к группам процессов.

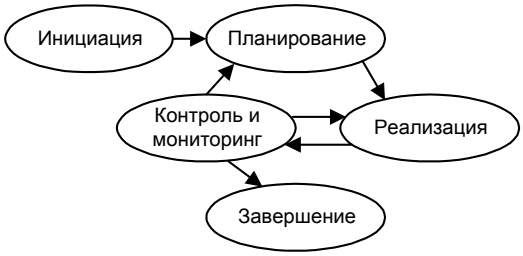


Рис. 2.3. **Схема взаимодействия групп процессов**

[20] Русский перевод PMBOK 2004 года можно найти в Интернет по адресу http://petukhov.zaklad.ru/pmbok2004_rus.pdf (ссылка актуальна на 01.08.2006)

Таблица 2.2. **Процессы по направлениям деятельности (по областям знаний) в PMBOK**

Группы / Области знаний	Инициация	Планирование	Исполнение	Мониторинг и управление	Завершение
Управление интеграцией проекта	Разработка Устава проекта	Разработка плана управления проектом	Руководство исполнением проекта	Мониторинг и управление работами. Управление изменениями	Закрытие проекта
Определение содержания проекта		Планирование содержания (рамок) Определение содержания (рамок) проекта		Подтверждение содержания Управление содержанием (изменениями)	
Управление продолжительностью проекта		Определение состава операций Определение взаимосвязей операций Оценка ресурсов операций Оценка длительности операций Разработка плана-графика		Управление планом-графиком	
Управление стоимостью проекта		Стоимостная оценка Разработка сметы затрат (бюджета расходов)		Управление стоимостью	
Управление качеством проекта		Планирование качества	Процесс обеспечения качества	Процесс контроля качества	
Управление людскими ресурсами		Планирование человеческих ресурсов	Набор команды Развитие команды	Управление командой проекта	
Управление коммуникациями		Планирование коммуникаций	Распространение информации	Отчетность по исполнению. Управление участниками	
Управление рисками проекта		Планирование управления рисками Идентификация рисков Качественный анализ рисков Количественный анализ рисков Планирование реагирования на риски		Мониторинг и управление рисками	
Управление поставками проекта		Планирование поставок и закупок Планирование контрактов	Анализ предложений Выбор продавцов	Администрирование контрактов	Закрытие контракта

Процессы в PMBOK детально описаны в терминах Входов, Выходов, применяемых Методологий и Инструментов.

Далее приводится аннотированное описание областей знаний PMBOK.

2.3.2. Области знаний PMBOK

Область знаний ***Управление интеграцией проекта*** включает процессы, необходимые для выявления, определения, комбинирования, унификации и координации всех других процессов и операций по управлению проектами. В контексте управления проектами *интеграция* подразумевает принятие решений о месте концентрации ресурсов в каждый момент выполнения проекта, предугадывание потенциальных проблем и их своевременное решение, и нахождение компромиссов между пересекающимися целями и альтернативами. Данная область знаний охватывает следующие процессы:

- разработка Устава проекта, формально авторизующего проект или фазу;
- разработка предварительного (общего) описания содержания проекта;
- разработка Плана управления проектом – документирование операций, необходимых для определения, подготовки и интеграции всех вспомогательных планов в один план управления проектами и их координации;
- руководство и управление исполнением проекта – выполнение работы по Плану управления проектом, необходимой для достижения требований, определенных в описании содержания проекта;
- мониторинг и управление работами проекта – мониторинг и управление процессами инициации, планирования, выполнения и завершения проекта для достижения целевых показателей эффективности, намеченных в Плане управления проектом;
- общее управление изменениями – обработка всех запросов на изменения, утверждение этих изменений и управление ими с целью оптимизации результатов поставки и активов организационного процесса;
- закрытие проекта – завершение всех операций по всем группам процессов проекта в целях формального завершения проекта.

Область знаний ***Управление содержанием проекта*** охватывает процессы, обеспечивающие включение в проект всех тех и только тех работ, которые необходимы для успешного выполнения проекта и отвечают его содержанию (целям, требованиям, границам, ограничениям, критериям приемки и др.). Это такие процессы:

- планирование содержания – создание плана управления содержанием проекта, в котором документируется процесс формулирования, верификации и контроля содержания проекта, а также процесс формирования *иерархической структуры работ* (ИСР);
- определение содержания – разработка подробного описания содержания проекта в качестве основы для принятия будущих решений по проекту;
- создание ИСР – разбивка крупных результатов поставки проекта и проектных работ на более мелкие элементы, более удобные для управления;
- подтверждение содержания – формальное принятие завершенных результатов поставки (комплектности поставленных продуктов, а не их качества);
- управление содержанием – управление изменениями содержания проекта.

Область знаний ***Управление сроками проекта*** охватывает следующие процессы, обеспечивающие своевременное завершение проекта:

- определение состава операций – конкретных плановых операций, которые необходимо выполнить для получения различных результатов поставок;
- определение взаимосвязей операций – выявление и документирование зависимостей между плановыми операциями;
- оценка ресурсов операции – оценка типов и количества ресурсов, необходимых для выполнения каждой плановой операции;
- оценка длительности операций — оценка количества периодов, необходимых для выполнения отдельных операций;
- разработка графика — составление графика проекта с учетом последовательностей операций, их длительности, требований к ресурсам и ограничений по срокам;
- управление графиком – управления изменениями графика.

Область знаний ***Управление стоимостью проекта*** объединяет следующие процессы, выполняемые в ходе планирования, разработки сметы и контроля затрат и обеспечивающие завершение проекта в рамках утвержденного бюджета:

- стоимостная оценка – определение примерной стоимости ресурсов, необходимых для выполнения операций проекта;
- разработка сметы расходов – суммирование оценок стоимости отдельных операций или пакетов работ с целью формирования базового плана по стоимости;
- управление стоимостью – воздействие на факторы, вызывающие отклонения по стоимости, и управление изменениями бюджета проекта.

Область знаний ***Управление качеством проекта*** охватывает процессы и операции, выполняемые организацией-исполнителем, которые определяют политику, цели и распределение ответственности в области качества таким образом, чтобы проект удовлетворял тем потребностям, для которых он был предпринят. Управление качеством проекта осуществляется на основе системы менеджмента качества (СМК) при помощи правил и процедур, а также действий, направленных на постоянное улучшение процесса и предпринимаемых на всем протяжении выполнения проекта. Процессы управления качеством таковы:

- планирование качества – определение стандартов качества, которые соответствуют проекту, и способов их соблюдения;
- процесс обеспечения качества – выполнение плановых систематических операций, направленных на достижение качества, обеспечивающих выполнение всех предусмотренных процессов, необходимых для того, чтобы проект соответствовал установленным требованиям;
- процесс контроля качества – анализ определенных результатов с целью определения их соответствия принятым стандартам качества и определение путей устранения причин, приведших к неудовлетворительным результатам.

Область знаний ***Управление человеческими ресурсами проекта*** охватывает процессы по организации и управлению командой проекта. Команда проекта состоит из людей (персонала проекта), каждому из которых назначена определенная роль и ответственность за выполнение проекта. Процессы управления человеческими ресурсами проекта таковы:

- планирование человеческих ресурсов – определение и документальное оформление ролей, ответственности и подотчетности, а также разработка Плана управления обеспечением проекта персоналом;
- набор команды проекта – привлечение человеческих ресурсов, необходимых для выполнения проекта;
- развитие команды проекта – повышение квалификации членов команды проекта и укрепление взаимодействия между ними с целью повышения эффективности исполнения проекта;
- управление командой проекта – контроль за эффективностью работы команды проекта, обеспечение обратной связи, решение проблем и координация изменений, направленных на повышение эффективности исполнения проекта.

Область знаний ***Управление коммуникациями проекта*** охватывает следующие процессы, необходимые для своевременной и надлежащей подготовки, сбора, распространения, хранения, выборки и, в конечном итоге, использования проектной информации:

- планирование коммуникаций – определение потребностей участников проекта в коммуникации и информации;
- распространение информации — своевременное предоставление информации участникам проекта;
- отчетность по исполнению - сбор и распространение информации о выполнении работ, включая отчеты о текущем состоянии, оценку прогресса и результаты прогнозирования;
- управление участниками проекта – управление коммуникациями в целях удовлетворения требований участников проекта и решения возникающих проблем.

Область знаний ***Управление рисками проекта*** охватывает процессы, относящиеся к планированию управления рисками, их идентификации и анализу, реагированию на риски, мониторингу и управлению рисками проекта. Цель управления рисками проекта - повышение вероятности возникновения и степени влияния позитивных событий и снижение вероятности возникновения и степени влияния негативных для проекта событий. Процессы управления рисками проекта следующие:

- планирование управления рисками – определение и выполнение операций по управлению рисками проекта (в соответствии с выбранным подходом);
- идентификация рисков – определения рисков, способных повлиять на проект, и документирование их характеристик;
- качественный анализ рисков – расположение рисков по приоритетам для их дальнейшего анализа или обработки исходя из оценки вероятности возникновения каждого риска и степени влияния на проект;
- количественный анализ рисков – количественный анализ потенциального влияния идентифицированных рисков на общие цели проекта;
- планирование реагирования на риски – разработка возможных вариантов действий, направленных на повышение благоприятных возможностей и снижение угроз для достижения целей проекта;
- мониторинг и управление рисками – отслеживание идентифицированных рисков, мониторинг остаточных рисков, идентификация новых рисков, исполнение планов реагирования на риски и оценка их эффективности на протяжении жизненного цикла проекта.

Область знаний ***Управление поставками проекта*** охватывает процессы закупки или приобретения тех необходимых продуктов, услуг или результатов, которые производятся вне команды проекта. Поставки рассматриваются с двух точек зрения: продавца и покупателя. Согласно условиям контракта организация может выступать в качестве продавца или покупателя продукта, услуги или результатов.

Процессы управления поставками проекта таковы:

- планирование закупок и приобретений – определение того, что необходимо купить или приобрести (заказать), а также когда и на каких условиях;
- планирование контрактов – оформление в документальном виде требований к продуктам, услугам и результатам, которые необходимо приобрести, а также определение потенциальных продавцов;
- запрос информации у продавцов – получение информации, предложений или заявок (в зависимости от поставки) от продавцов;
- выбор продавцов – анализ предложений, отбор потенциальных продавцов и обсуждение условий контракта с продавцом;
- администрирование контрактов – управление контрактом и взаимоотношениями между покупателем и продавцом; анализ и документирование текущей и прошлой деятельности продавца для определения необходимых корректирующих действий и обеспечения основы для будущих отношений с продавцом; управление изменениями, связанными с контрактом, и (при необходимости) управление контрактными взаимоотношениями со сторонним покупателем проекта;
- закрытие контракта — завершение каждого контракта, включая урегулирование всех открытых вопросов, и закрытие каждого контракта, относящегося к проекту или фазе проекта.

PMBOK определяет круг общих знаний по управлению проектами, без привязки к конкретным предметным областям и технологиям, и потому не может применяться самостоятельно. Рабочей группой по стандартам WG7 комитета ISO/IEC JTC1/SC7 предложено модифицировать стандарт ISO/IEC TR 16326:1999 «Guide for the application of ISO/IEC 12207 to project management» (Руководство по применению ISO/IEC 12207 при управлении проектом), для того чтобы обеспечить применимость положений PMBOK в программной инженерии [18].

2.4. Связь ядра знаний в области качества с элементами SWEBOK и PMBOK

Разработка Руководства к SWEBOK преследовала цели способствовать формированию *согласованного видения* дисциплины в международном масштабе; определить место и установить *границы дисциплины* по отношению к другим дисциплинам - информатике, управлению проектами, математике; охарактеризовать *содержание дисциплины*; обеспечить *доступ* к «кладезю» знаний по предмету путем предоставления ссылок на множество литературных источников; создать основу для разработки *курсов обучения* и индивидуальной *сертификации*.

В работе над Руководством к SWEBOK и отдельным областям знаний принимали участие сотни авторов и рецензентов из 42 стран и это не могло не сказаться на однозначности и ясности применяемой терминологии как в пределах отдельных глав, так и «насквозь» по главам. Так, термин «качество» встречается в руководстве 340 раз применительно к процессам, конечному и промежуточным продук-

там, ресурсам и др., а слово «качество» употребляется в самых различных сочетаниях, например, «измерения качества», «качество измерений» и т.п.

Собственно проблематика *качества ПО* выделена в самостоятельную область знаний, хотя разные аспекты обеспечения качества рассматриваются практически в каждой главе Руководства, что усложняет целостное восприятие инженерии качества как сферы деятельности.

На рисунке 2.4 показано, в каких структурных образованиях SWEBOK (главах (областях знаний), разделах, подразделах и рубриках) поднимаются вопросы качества.

Глава 2.Требования к ПО
- 2. Процесс требований
  - 2.4. Качество процесса и улучшение

Глава 3. Проектирование ПО
- 4. Анализ и оценивание качества проекта
  - 4.1. Атрибуты качества
  - 4.2. Приемы анализа и оценивания качества

Глава 4. Конструирование ПО
- 3. Практические соображения
  - 3.5. Качество конструкции

Глава 6. Сопровождение ПО
- 3. Процесс сопровождения
  - 3.2. Деятельность по сопровождению
    - 3.2.5. Качество ПО

Глава 8. Управление инженерией ПО
- 2. Планирование проекта ПО
  - 2.6. Управление качеством

Глава 9. Процесс программной инженерии
- 4. Измерение процесса и продукта
  - 4.1. Измерение процесса
    - 4.1.3. Измерение качества
  - 4.2. Качество результатов измерений

Глава 10. Инструменты и методы программной инженерии
- 1. Инструменты программной инженерии
  - 1.9. Инструменты качества ПО

Глава 12. Качество ПО
- 1. Основы качества ПО
  - 1.2. Ценность (значимость) и стоимость качества
  - 1.3. Модели и характеристики качества
    - 1.3.1. Качество процесса программной инженерии
    - 1.3.2. Качество продукта ПО
  - 1.4. Улучшение качества
- 2. Процессы управления качеством ПО
  - 2.1. Гарантия качества ПО
- 3. Практические соображения
  - 3.1. Требования к качеству ПО
  - 3.3. Приемы управления качеством ПО
  - 3.4. Измерение качества ПО

Рис. 2.4. **Отражение аспектов качества в SWEBOK**

Очевидно, что управление качеством невозможно без надлежащего управления проектом и, прежде всего, управления людьми.

В SWEBOK вопросы управления проектами рассматриваются в области знаний «Управление инженерией ПО», причем таким основополагающим проблемам, как *управление стоимостью* (затратами) и *продолжительностью* проекта, а также *управление рисками* проекта и *управление персоналом*, внимание практически не уделяется. Создатели Руководства ссылаются на общность задач управления программными проектами и любыми другими (не программными) проектами, охватываемыми ядром знаний PMBOK.

Действительно в PMBOK, особенно в текущей версии 2004-го года, все вопросы управления проектами систематизированы и рассматриваются достаточно подробно. Однако, управление программными проектами имеет свою специфику, поскольку применяемые подходы к управлению базируются на выбранной *модели ЖЦ* и используемой *методологии разработки*, включая конкретные *методы и инструменты управления*, которые она (методология) предлагает разработчикам ПО.

Поэтому информация в PMBOK, касающаяся упомянутых аспектов управления, должна быть взята за основу и дополнена с учетом современных представлений о подходах, методах, инструментах, применяемых в инженерии качества.

Анализ SWEBOK и PMBOK, а также программ обучения инженеров по качеству и надежности ПО (software quality and reliability engineers) за рубежом показывает возможность синтеза ядра знаний в области инженерии качества на базе SWEBOK и PMBOK (безусловно, с привлечением элементов системного анализа, теории вероятности, математической статистики, теории прогнозирования и измерений).

Ядро знаний в области инженерии качества схематически представлено на рисунке 2.5.

В виде главных направлений знаний, составляющих ядро, выделены основные элементы парадигмы качества – контроль качества, измерение качества, управление качеством и система качества, а тематические рубрики подобраны таким образом, чтобы отразить весь спектр специальных знаний, необходимых для решения задач в рамках парадигмы качества. Нужно отметить, что материал всех последующих глав книги также сформирован в соответствии с указанными рубриками.

2.5. Парадигмы и стили программирования

2.5.1. Классификация парадигм и стилей программирования

В Руководстве по подготовке учебных программ по программной инженерии говорится [5]: «общее заблуждение по поводу программной инженерии состоит в том, что она понимается, прежде всего, как инженерная деятельность, ориентированная на *процесс* разработки как таковой … В этом контексте, компетентность специалиста в области программной инженерии может быть достигнута в основном за счет приобретения *инженерных навыков*, … при минимуме подготовки в области программирования…». Однако, только «благодаря фундаменту программирования и математики» в программной инженерии возможно создание теоретически обоснованных моделей и методов. Именно вокруг парадигм программирования выстраиваются и развиваются методологические концепции программной инженерии и инженерии качества.

Рис. 2.5. **Связь ядра знаний в области качества ПС с SWEBOK и PMBOK**

Парадигмы программирования на практике реализуются через стили программирования, определяющие соглашения о базовых языковых средствах, приемлемых или не приемлемых для определенных парадигм.

В инженерии качества знание парадигм программирования и языков, представляющих эти парадигмы, – залог выработки правильных подходов к верификации и тестированию программных систем, выбора адекватных метрик и инструментов измерения свойств ПС, а также методов оценки размера и сложности, классификации дефектов и построения таксономии рисков отказа ПС.

Существуют разные подходы к классификации парадигм, например:

– парадигмы *прикладного* (систематического) и *теоретического* программирования [19];

– парадигмы *основные* (Императивного, Функционального, Алгебраического и Логического программирования) и *высшие* (*по уровню*) (Параллельного, Объектно-ориентированного, Агентного программирования) [20];

– парадигмы, поддерживающие *синтезирующее*, *сборочное* и *конкретизирующее* программирование и др. [21];

– парадигмы, отличающиеся способом декомпозиции задачи (*Императивного*, *Декларативного*, *Объектно-ориентированного*, *Сценарного программирования*, *Программирования в ограничениях*) [22];

– парадигмы (стили), отличающиеся глубиной и общностью проработки технических деталей организации процессов компьютерной обработки информации (*Машинно-ориентированное*, *Системное*, *Логическое*, *Трансформационное*, *Высокопроизводительное* /параллельное программирование) [19] и другие.

Текущая версия упомянутого Руководства 2004-го года рекомендует включать в учебные курсы по программной инженерии основы *процедурного программирования* (курс CS1011 «Programming fundamentals»), *объектно-ориентированного программирования* (курс CS102I «The Object-Oriented Paradigm»), а также *событийного программирования* (event-based programming). Это широко используемые стили прикладного программирования императивной и объектно-ориентированной парадигм, представленных множеством распространенных языков программирования. Именно на эти стили программирования в основном и ориентируется инженерия качества на современном уровне ее развития. Однако, для расширения сферы применения ее методов и моделей, а также разработки новых в будущем, целесообразно хотя бы вкратце остановиться на существующих парадигмах программирования.

В работе [23] выделены, как основные, парадигмы *процедурного*, *функционального*, *алгебраического* и *логического* программирования, различающиеся способами представления данных и алгоритмов в программах, дан их сравнительный анализ и обоснована необходимость интеграции в процессе создания высокоинтеллектуальных программных композиций.

Эти парадигмы лежат в основе образования всего разнообразия ныне существующих парадигм. Часть из них схематически представлена на рисунке 2.6 и рассматривается детальнее в этом разделе. Стили, которые далее не упоминаются, указаны в блоках пунктирным контуром.

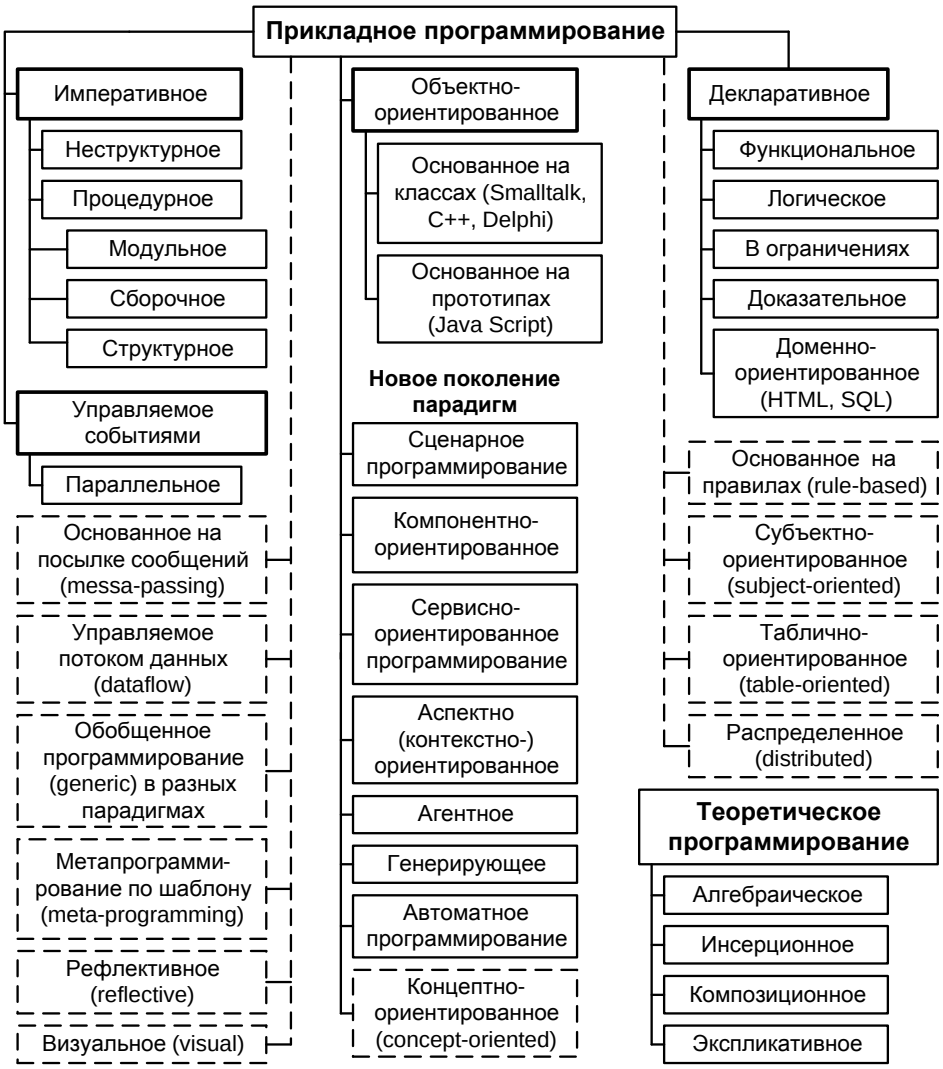


Рис. 2.6. **Основные парадигмы и стили программирования**

2.5.2. Парадигма и основные стили императивного программирования

Императивное программирование основано на машине Тьюринга-Поста - абстрактном вычислительном устройстве, выполняющем последовательность инструкций (команд) программы, которая, таким образом, переходит из одного состояния в другое. Существует понятие «текущего шага исполнения» и «текущего состояния», которое меняется с течением времени.

Поскольку практически все современные компьютеры ориентированы на последовательные вычисления, императивное программирование заведомо выигрывает в эффективности реализации прикладных задач, для которых очень важна скорость исполнения (решения).

Примеры стилей императивного программирования представлены в таблице 2.3.

Таблица 2.3. **Стили императивного программирования**

Стиль	Описание	Примечания
Неструктурное программирование	Весь код программы представлен единым непрерывным блоком. Переходы к нужным секциям программы выполняются с помощью условных операторов и операторов перехода (обычно, goto или jump) без ограничений.	*Примеры языков*: языки скриптов (bat-файлы), Fortran, Basic, Ассемблеры *Проблемы обеспечения качества:* образование «спагетти-кода», сложность верификации кода.
Процедурное программирование	Иногда используется как синоним императивного программирования (состояние процедуры определяется значением множества ее переменных в определенный момент времени). Базируется на концепции *вызова процедур* (подпрограмм, функций, методов). Любая процедура может быть вызвана в любой точке в ходе выполнения программы.	*Примеры языков*: Algol, Ada, Basic, C, Cobol, Fortran, Pascal, PL/1, Matlab, многопарадигменные языки. *Проблемы обеспечения качества:* облегчается повторное использование кода (без дублирования), улучшается прослеживаемость кода и сопровождаемость программ. Есть методы оценки сложности программ (глава 5).
Модульное и сборочное программирование [24, 25]	Модульная организация кода при процедурном программировании. Строго определенные механизмы ввода-вывода данных – через *аргументы* (на входе) и *возвращаемые значения* (на выходе) процедуры. Определение сферы действия переменных. Возможность автономной реализации и хранения кода (в библиотеках модулей), а также сборки программ из модулей.	*Примеры языков*: языки императивного и объектно-ориентированного программирования. *Проблемы обеспечения качества:* облегчается тестирование и повторное использование кода; допускается распределенная разработка программ (разработка автономных модулей отдельными программистами).
Структурное программирование	Направление процедурного программирования, особенности которого – структурирование программы путем ее разделения на секции (блоки) одного из видов: последовательность, ветвление, цикл. Базовые конструкции могут быть вложены друг в друга произвольным образом, но никаких других средств управления последовательностью выполнения операций нет. Ограничение - сокращение (запрет) использования goto; одна точка входа в секцию (блок), но несколько возможных точек выхода. Часто ассоциируется с разработкой «сверху-вниз». Структурирование данных. Использование абстрактных типов данных.	*Примеры языков*: во все современные языки императивного программирования введены элементы и ограничения структурирования, например, Pascal, Ada, Fortran, Cobol, Basic, C. *Проблемы обеспечения качества:* структурирование снижает сложность программы, облегчает ее верификацию и внесение изменений, локализация обработки исключительных ситуаций в отдельных блоках облегчает отладку и тестирование. Разработка с использованием «заглушек» облегчает тестирование и локализацию дефектов.

2.5.3. Парадигма событийно-управляемого программирования

Стили программирования, обычно связываемые с итеративной парадигмой, основываются на концепции организации *«внутреннего» потока управления* вы-

полнением инструкций программы, не связанного с наступлением событий вне ее. Однако, существует множество задач, решаемых в реальном времени (в медицине, ядерной энергетике, управлении полетами и др.), а также задач, решаемых в интерактивном режиме, ожидающих, принимающих и обрабатывающих команды и наборы данных извне. Для эффективной реализации таких задач используются стили событийно-управляемого (или событийного) программирования, основанные на концепции организации *цикла обработки внешних событий*. Вместо того, чтобы ожидать полного завершения ввода и обработки одного набора данных, и лишь потом *последовательно* переходить к ожиданию следующего, в системе организуется цикл постоянного опроса очереди поступающих извне событий (будь то появление файла в каталоге, щелчок на кнопке мыши, наступление заданного момента времени или другое), и запускается *функция-триггер* для обработки очередного события. Событийное программирование заключается в подмене системных функций-триггеров «по умолчанию» нужными функциями, соответствующими каждому возможному событию.

Таким образом, программа в стиле событийного программирования обычно состоит из ряда маленьких прикладных программ, называемых *обработчиками событий*, которые запускаются в ответ на появление этих событий, и *диспетчера*, использующего очередь необработанных событий, вызывающего обработчики событий и имитирующего среду, управляемую прерываниями.

Событийное программирование (event-oriented, event-based или event-driven programming) – наиболее распространенная современная парадигма прикладного программирования, поддерживаемая множеством операционных систем и сред разработки программ (в частности, Windows).

Особенности инженерии качества программ, разработанных с применением событийной парадигмы, заключаются в применении методов тестирования, основанных на операционных (сценарных) профилях программ (глава 7), а также методов прогнозирования размера, основанных на подходе FPA (глава 8).

К стилям событийно-управляемого программирования можно отнести согласованное (параллельное) программирование.

Согласованное программирование и параллельные вычисления. Естественный резерв повышения производительности компьютеров — распараллеливание потока команд или потока данных. Распараллеливание данных подразумевает применение одной операции сразу к нескольким элементам массива данных, а параллелизм задач — разбиение вычислительного процесса на несколько подзадач (процессов), каждый из которых выполняется на своем процессоре.

Достаточно быстро были разработаны супер-ЭВМ, имеющие «векторную» или «матричную» архитектуру процессоров (транспьютеров), автономных, но способных обмениваться между собой результатами своих вычислений через *каналы связи*. А к началу 2006 года главными элементами производственных программ ведущих компаний–разработчиков микропроцессоров стали *многоядерные процессоры* [26, 27].

Может быть обеспечен параллелизм на двух уровнях: параллелизм уровня *микроопераций* и параллелизм уровня *процессов* - простых (послать, принять значение или процессы вычислительной модели) или структурных (последовательных или параллельных). Когда процесс выполняет инструкцию «Принять значение (из канала)», он входит в состояние ожидания, пока канал пуст. Как только в канале

появляется значение, процесс его считывает и продолжает работу. В системе *параллельных процессов* каждый отдельный процесс обрабатывает события.

Активное внедрение новых аппаратных архитектур потребовало смены парадигмы программирования — перехода от последовательного программирования к *согласованному программированию* (concurrent programming), в частности, *параллельному программированию* (parallel programming или параллельным вычислениям - parallel computing), связанному с параллельными потоками, порождением и обработкой асинхронных событий и др.

Согласованное (параллельное) программирование - это программирование в терминах взаимодействия некоторых одновременно работающих вычислителей (нескольких процессоров или многоядерного процессора).

При разработке параллельных программ используются специализированные языки, как, например, язык высокого уровня Occam (неразрывно связанный с транспьютерами), а также современные библиотеки и системы параллельного программирования PVM, LAM, CHMP и др. Три основных подхода к реализации этих систем различаются методами взаимодействия параллельных задач. Первый подход базируется на концепции *обмена сообщениями*, второй — на использовании *разделяемой памяти*, третий опирается на *стандарт POSIX* и объединяет оба подхода.

Наиболее известной реализацией первого подхода является спецификация MPI (Message Passing Interface) для языков C и Fortran, а второго подхода - спецификация OpenMP для Fortran, C, C++. OpenMP — это набор специальных директив компилятору (pragma), а также библиотечных функций и переменных среды. Компилятор, поддерживающий OpenMP, преобразует исходный код программы - вставляет соответствующие вызовы функций для параллельного выполнения областей кода, выделенных с помощью ряда специальных директив компилятору. Поддержка спецификации OpenMP обеспечена во всех компиляторах Intel, начиная с шестой версии, в Microsoft C и C++, начиная с Visual Studio 2005, а также в GCC. В третьем подходе используется спецификация POSIX (Portable Operating System interface for unIX), введенная как международный стандарт ISO/IEC 9945-1:1990. В рамках POSIX можно реализовать параллельные вычисления на основе обмена сообщениями (аналогично MPI) или разделяемой памяти (как в OpenMP). Естественно, в POSIX допустима и любая комбинация этих методов.

В работе [26] отмечается, что многоядерные процессоры – это единственное перспективное направление для дальнейшего развития компьютерной индустрии, поскольку «ресурсы трех слонов, на которых стояла вся индустрия, оказались исчерпанными: тактовая частота не растет; оптимизация исполнения в пределах одного ядра достигла предела, за которым рост сложности становится неоправдан; у дальнейшего наращивания размеров кэш-памяти процессоров остался лишь небольшой зазор для развития. Как следствие, старый добрый мир остался в прошлом… Ведущий эксперт корпорации Microsoft Херб Саттер в статье «Программное обеспечение и революция согласования» (ACM QUEUE, September 2005) называет происходящее революцией по масштабам существенно более мощной, чем распространение объектно-ориентированного программирования в начале 90-х годов. Тогда причиной перемен стало усложнение программных систем, нынешняя же революционная ситуация вызвана необходимостью выполнять программы на многоядерных процессорах». Очевидно, что проблемы инженерии качества таких

программ (основанных на принципах согласования и параллелизма) требуют отдельного рассмотрения и формирования новых направлений исследований.

2.5.4. Парадигма объектно-ориентированного программирования

Из «недр» событийно-управляемого программирования появилось и объектно-ориентированное программирование, оказавшись значительно более мощным и универсальным средством программирования и проектирования.

Объектно-ориентированное программирование (ООП) – парадигма, основанная на представлении предметной области в виде системы взаимосвязанных абстрактных *объектов* и их реализаций.

В парадигме объектно-ориентированного программирования различают два стиля – *программирование, основанное на классах* (class-based programming), и *программирование, основанное на прототипах* (prototype-based programming).

Программирование «на классах». В отличие от стиля процедурного программирования, в котором данные и функции их обработки не связаны, в этом стиле определена новая структура данных - *класс* - по сути дела это тип объекта, в котором кроме данных (свойств, параметров объекта) содержатся функции их обработки (методы, допустимые действия объекта или с объектом).

Объект по отношению к своему классу является *экземпляром*. Класс определяет структуру и функциональность (*поведение*), одинаковую для всех экземпляров данного класса. Один класс отличается от других именем и, обычно, набором поддерживаемых *интерфейсов*. Интерфейсы, в свою очередь, представляют собой набор *сообщений*, которые можно посылать объекту. Экземпляр является носителем данных, то есть обладает *состоянием*, меняющимся в соответствии с поведением, заданным классом. В языках ООП новый экземпляр создается через вызов конструктора класса (возможно, с набором параметров). Получившийся экземпляр имеет структуру и поведение, жестко заданные его классом.

Все объекты могут обмениваться между собой *сообщениями*. При получении объектом сообщения запускается соответствующий ему обработчик (*метод*)**.** Объект имеет ассоциативный *контейнер*, позволяющий получить по сообщению нужный метод для обработки. Если метод для обработки сообщения не найден, сообщение перенаправляется объекту-*предку*.

Важнейшими принципами ООП являются:

наследование - механизм установления отношений «потомок-предок» (возможность порождать один класс от другого с сохранением всех свойств и методов класса-предка),

инкапсуляция (свойство сокрытия реализации (поведения) класса),

абстракция (описание взаимодействия исключительно в терминах сообщений/событий в предметной области),

полиморфизм (возможность подмены в сценарии взаимодействия объектов одного объекта другим объектом со сходной структурой).

Многие современные языки специально созданы для программирования на классах, например, Smalltalk, C++, Java, Python, PHP, Object Pascal (Delphi), VB.NET, Xbase++ и др. Однако следует отметить, что можно применять приемы ООП и для не-объектно-ориентированного языка и наоборот, применение объектно-ориентированного языка еще не означает, что код автоматически становится объектно-ориентированным. В наши дни парадигма объектно-ориентированного

программирования на классах активно используется при проектировании сложных систем и моделировании, например, в языке UML.

Для ПС, создаваемых в представленном стиле программирования, в инженерии качества наработаны и методы тестирования, и метрики, и способы прогнозирования размера и сложности.

Программирование по прототипу[21]. В этом стиле программирования понятие класса отсутствует, а повторное использование (наследование) производится путем клонирования существующего экземпляра объекта — *прототипа*.

В отличие от стиля программирования, основанного на классах и концентрирующего внимание разработчика на таксономии классов и на отношениях между ними, стиль прототипного программирования заостряет внимание на поведении небольшого количества «образцов», которые далее используются как «базовые» объекты для создания других объектов.

Создание нового объекта выполняется одним из двух методов: путем *клонирования* существующего объекта, либо путем создания объекта *«с нуля»*. Для создания объекта с нуля предоставляются синтаксические средства добавления свойств и методов в объект. В дальнейшем, для построенного объекта может быть получена полная копия, клон. В процессе клонирования копия наследует все характеристики своего прототипа, но с этого момента становится самостоятельной и может быть изменена. В некоторых реализациях копии хранят ссылки на объекты-прототипы, *делегируя* им часть своей функциональности; при этом изменение прототипа может затронуть все его копии. В других реализациях новые объекты обретают полную *независимость* от своих прототипов: прототип копируется один-в-один, со всем методами и атрибутами, и копии присваивается новое имя (ссылка). Преимущество данного подхода состоит в том, что создатель копии может ее менять, не опасаясь побочных эффектов среди других потомков своего предка. Кроме того, существенно снижаются вычислительные затраты на диспетчеризацию, так как нет необходимости обходить всю цепочку возможных делегатов в поисках подходящего метода или атрибута. В числе недостатков - трудности с распространением изменений в системе: модификация прототипа не влечет за собой немедленное и автоматическое изменение всех его потомков. Другой недостаток состоит в том, что простейшие реализации этой модели приводят к увеличенному (по сравнению с моделью делегирования) расходу памяти, так как каждый клон будет содержать копию данных своего прототипа до тех пор, пока не будет изменен.

Каноническим примером прототипного языка является язык Self. Прототипный стиль программирования положен также в основу таких языков программирования, как JavaScript, Squeak, Cecil, NewtonScript, Io, MOO, REBOL, Kevo и др.

Прототипно-ориентированные системы сравнительно новы и немногочисленны, поэтому методы программной инженерии и инженерии качества для их разработки пока не определены.

2.5.5. Парадигма и основные стили декларативного программирования

Чаще всего парадигма декларативного (повествовательного, описательного) программирования ассоциируется со стилями и каноническими языками *функционального программирования* (например, Lisp), *логического программирования*

[21] Источник информации – энциклопедия Википедия (http://ru.wikipedia.org/wiki)

(Prolog) или *программирования в ограничениях* (Oz), и противопоставляется парадигме императивного (повелительного, командного) программирования [28].

Однако декларативным является и описание Web-страниц на HTML и XML-документов (в проблематике Web-технологий), и описание структур таблиц БД на SQL (в проблематике баз данных), и т.п. Соответствующий стиль программирования также относят к декларативному, а языки называют *доменными* (специализированными для определенных доменов (областей применения)) или *DSL-языками* (Domain Specific Languages).

В программе, разрабатываемой в парадигме декларативного программирования, четко формулируется *цель* и *результат* ее работы, а *не алгоритм* получения этого результата. Для обработки описанных структур данных используются *стандартные алгоритмы* реализующих систем этих языков. Например, при создании Web-страницы средствами HTML специфицируется, каким образом должна выглядеть страница, а трансляция этой спецификации в пиксельное представление выполняется с помощью процедурного алгоритма браузера. DSL-языки иногда встраиваются в универсальные языки, хотя писать программы в декларативном стиле можно и с помощью императивных языков, инкапсулируя «не декларативные» детали в библиотечные компоненты.

Функциональное программирование. В отличие от императивного программирования, где алгоритмы - это описания последовательно исполняемых *операций*, существует понятие *текущего шага исполнения* (во времени), и *текущего состояния*, изменяющегося с течением этого времени, в функциональном программировании понятие *времени отсутствует*.

Как и другие стили «неимперативного» программирования, функциональное программирование (functional programming) применяется для решения задач, трудно формулируемых в терминах последовательных операций – преимущественно задач, связанных с распознаванием образов, общением на естественном языке, реализацией экспертных систем, автоматизированным доказательством теорем, символьными вычислениями.

Функциональное программирование опирается на теории лямбда-исчисления (Алонзо Черча) и комбинаторной логики (Мозеса Шенфинкеля и Хаскелла Карри). Функциональная программа состоит из совокупности *определений функций*, представляющих собой вызовы других функций, и предложений, управляющих последовательностью вызовов. Описание *зависимости* функций друг от друга (возможно рекурсивной) фактически определяет способ решения задачи (без указания последовательности шагов). Рекурсия является фундаментальной основой построения семантики языков функционального программирования. В общем случае, функциональная программа не содержит оператора присваивания, вычисление любой функции не приводит ни к каким побочным эффектам, отличным от собственно вычисления ее *значения*. Разветвление вычислений основано на механизме обработки аргументов условного предложения, а циклические вычисления реализуются с помощью рекурсии.

Наиболее известны следующие функциональные языки программирования:

– LISP (1958 год) и множество его потомков, наиболее современные из которых - Scheme и Common Lisp,

– ML (1979 год) и его наиболее известные диалекты - Standard ML и Objective CAML,

– Miranda (1985), который дал развитие языку Haskell и др.

Языки функционального программирования в основном используются в научных (академических), а не в коммерческих приложениях, за исключением языка Erlang (для разработки высокопроизводительных приложений в телекоммуникации), языков J and K (для финансового анализа), а также таких доменных языков, как, например, XSLT.

Логическое программирование. В логическом программировании (logic или logical programming) программа представляет собой некоторую *теорию* (описанную на достаточно ограниченном языке), и *утверждение*, которое нужно доказать. Теория задается с помощью *аксиом* и *правил вывода*. Утверждение, которое требуется доказать, вводится в программу в качестве *цели*. Работа программы состоит в поиске доказательства предложенного утверждения в имеющейся базе знаний, представляющей собой совокупность определенных фактов и правил.

Если в функциональном программировании функции однонаправлены - получают аргументы и возвращают результат, в логическом программировании разница между вводом и выводом условна. Можно указать желаемый вывод и получить ввод, который его обеспечит. Другое важное отличие - недетерминизм логических языков. Результат в них не обязательно определяется однозначно. Таким образом, поскольку произвольное целевое утверждение может быть доказано не единственным образом, система, реализующая язык логического программирования (например, Prolog), последовательно предлагает возобновить попытку доказать утверждение по-другому. Для доказательства утверждений используется унификация (сопоставление двух произвольных термов) и метод резолюций.

В логическом программировании, также как и в функциональном программировании, программист остается в неведении о методах вычислений, и последовательности элементарных действий. Большая часть ответственности за их эффективность возлагается на транслятор используемого языка программирования. В настоящее время существует несколько «промышленных» реализаций языка Prolog, причем исполняемый код, порождаемый транслятором, как правило, сопоставим по эффективности с кодом аналогичной программы на императивных языках.

Логическое программирование оказалось эффективно не только в реализации задач искусственного интеллекта, для чего оно в основном и используется, но и для описания сложных систем, например, диспетчерских систем[22].

Программирование в ограничениях [22]. Программирование в ограничениях (constraint programming) - это программирование в терминах *постановок задач*, где постановка задачи – это конечный набор переменных $V = \{v_1, ..., v_n\}$, соответствующих им конечных (перечислимых) множеств значений $DV = \{dv_1,...,dv_n\}$, и набор ограничений $C = \{c_1,...,c_m\}$. Система ограничений может включать в себя уравнения, неравенства, логические функции, а также любые допустимые формальные конструкции, связывающие переменные из набора *V*. Решение задачи состоит в построении набора значений переменных, удовлетворяющего всем ограничениям c_i, где $i = 1,.., m$.

Программирование в ограничениях тесно связано с традиционным логическим программированием. Большинство систем программирования в ограничениях представляют собой интерпретатор языка Prolog со встроенным механизмом для

[22] Известно, что диспетчерская система лондонского аэропорта Хитроу в настоящий момент переписывается на языке Prolog.

решения определенного класса задач удовлетворения ограничениям. Программирование в таких системах называют *логическим программированием в ограничениях* (Constraint Logic Programming или CLP), а большинство языков или библиотек называются CLP(X), где X указывает на класс решаемых задач. Например, CLP(B) означает возможность решать уравнения с булевыми переменными, CLP(Q) – уравнения в рациональных числах, а CLP(R) – в вещественных. Наиболее популярны решатели задач на конечных множествах целых чисел CLP(FD).

Семантически программирование в ограничениях отличается от традиционного логического программирования в первую очередь тем, что исполнение программы рассматривается не как доказательство утверждения, а как нахождение значений переменных. При этом внутренний порядок решения не имеет значения для выполнения отдельных ограничений и система программирования в ограничениях, как правило, стремится оптимизировать порядок доказательства утверждений с целью минимизации отката в случае неуспеха. Любая логическая программа может быть преобразована в эквивалентную программу с ограничениями и наоборот. Иногда логические программы специально конвертируют в программы с ограничениями, поскольку выполнить решение задачи с ограничениями бывает проще, чем произвести логический вывод программы.

Наиболее популярными языками программирования в ограничениях являются языки, базирующиеся на Prolog: B-Prolog, CHIP V5, Ciao Prolog, ECLiPSe, GNU Prolog и др. Программирование в ограничениях может быть реализовано и в рамках императивного программирования путем подключения библиотек функций (например, библиотек функций Java, C++).

Доказательное программирование и синтез программ [29]. Чтобы быть уверенным в том, что программа работает правильно, лучше всего иметь доказательство ее правильности. Можно *доказывать правильность* готовых программ либо строить программы параллельно с доказательством их правильности (*синтезировать* заведомо правильные программы). Практика показала, что во втором случае программы получаются яснее, изящнее и зачастую эффективнее, чем программы, написанные бессистемно.

Условие задачи при синтезирующем программировании записывается в виде *спецификации* задачи. В последующем синтезе программы по спецификациям различают два подхода: *логический* и *аналитический*.

При логическом подходе спецификация трактуется как формулировка теоремы, утверждающей существование решения задачи. Синтез программы состоит в поиске доказательства этой теоремы существования в некотором конструктивном логическом исчислении. Если такое доказательство удается построить, то используются формальные правила, позволяющие преобразовать доказательство в программу, находящую решение задачи. Существует универсальная контролирующая процедура, которая проверяет правильность доказательства и применения правил перехода от доказательства к программе.

При аналитическом подходе спецификация трактуется как уравнение относительно программы. Такое уравнение можно подвергать символическим преобразованиям, при которых символ неизвестной программы «рассыпается» на систему вспомогательных неизвестных, а они, в свою очередь, заменяются либо другими неизвестными, либо конкретными программными конструкциями. Таким образом, по мере преобразования синтезируемая программа постепенно превращается в за-

конченный программный текст. Как и в первом подходе, правильность выполнения символических преобразований может формально проверяться на каждом шагу.

2.5.6. Парадигмы прикладного программирования нового поколения

Сценарная парадигма [22]. Сценарная парадигма предполагает разбиение задачи на отдельные части, каждая из которых решается специализированными программными средствами, сценарий выступает в роли «диспетчера», ответственного за организацию их взаимодействия. Концепция сценарного программирования возникла как естественное развитие языка LISP. Первые сценарные (скриптовые) языки служили средством управления командной оболочкой операционных систем - командный файл на языке операционной системы содержал сценарий управления заданной последовательностью действий по организации взаимодействия элементов системы.

В настоящее время популярность сценарных языков связана с развитием Internet-технологий. Скриптовые языки используются для создания динамических, интерактивных web-станиц, содержание которых модифицируется в зависимости от действий пользователя и состояния остальных страниц и данных. Сценарный язык в малой степени ориентируется на создание конечного продукта с нуля и в большей степени – на использование тех мощностей, которыми обладает операционная система, графическая среда, прикладная сервисная машина и прочие подобные компоненты, взаимодействие которых осуществляется с помощью сценариев.

Сценарные языки для web-разработки в основном созданы в 90-е годы прошлого века и включают в себя элементы различных парадигм программирования. Среди наиболее мощных и популярных скриптовых систем - Perl, Pyton, PHP, ASP.

Компонентно-ориентированное программирование. Компонентная разработка (CBSE, от Component based software engineering или CBD, от Component-based development) – направление в программной инженерии, в основе которого лежит *индустриальный* подход к разработке программных систем - «не с нуля», а путем быстрой сборки из готовых *программных компонентов* [30-32].

Главная задача CBSE – основываясь на знании свойств отдельных интегрируемых компонентов гарантировать предсказуемость свойств системы. CBSE обобщает идеи объектно-ориентированной парадигмы, повторного использования [33], абстрактных архитектур, формальных спецификаций и т.п. и базируется на концепциях *компонента* (component), *интерфейса* (interface), *контракта* (contract), *компонентной модели* (component model), *компонентного каркаса* (component framework), *композиции* (composition) и *сертификации* (certification).

Схематическое описание модели компонентной системы представлено на рисунке 2.7 и кратко прокомментировано далее [34].

Компонент (1) – это программная *реализация* (исполнимый код) функций объекта, предназначенная для выполнения (работы) на физическом или логическом устройстве. Наряду с функциональностью компонент реализует один или более *интерфейсов* (2) в соответствии с определенными обязательствами, описанными в *контракте* (3). Контрактные обязательства гарантируют, что независимо разработанные компоненты руководствуются определенными правилами поведения, могут взаимодействовать друг с другом предсказуемым образом и *разворачиваться* в *стандартных средах* (4) (на *этапе построения* системы или на *этапе ее исполнения* (работы)).

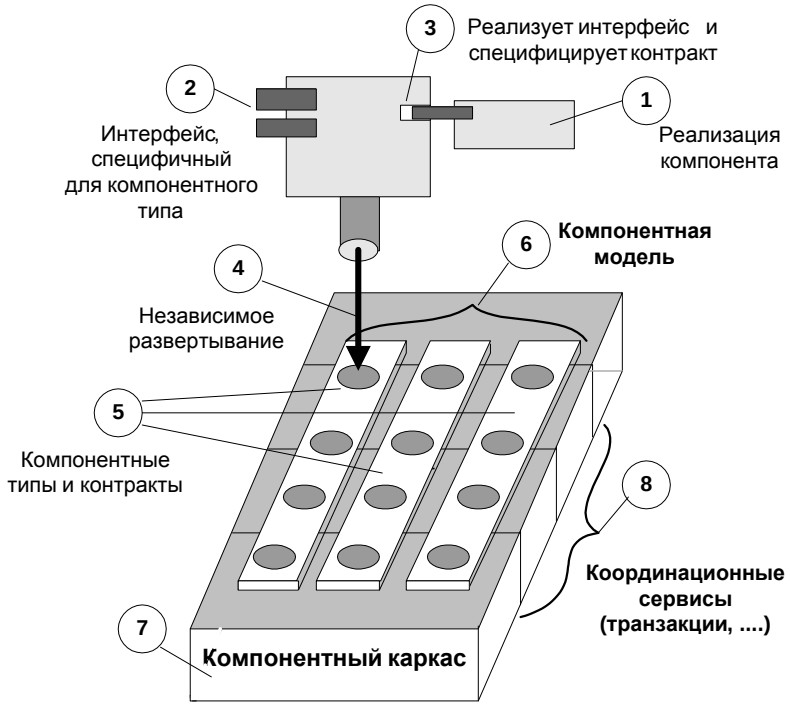


Рис. 2.7. **Модель компонентной системы**

Компонентная система основывается на небольшом количестве разных *типов компонентов* (5), каждый из которых играет специализированную роль в системе и описывается интерфейсом (2). *Компонентная модель* (6) представляет собой множество типов компонентов, их интерфейсов и, дополнительно, спецификацию допустимых *шаблонов* (*паттернов*) *взаимодействия* типов компонентов. *Компонентный каркас* (7) обеспечивает множество *сервисов* (8) периода времени выполнения для поддержки функционирования компонентной модели. Во многих отношениях компонентные каркасы уподобляются специализированным операционным системам, хотя они оперируют на гораздо более высоких уровнях абстракции и имеют гораздо более ограниченный диапазон механизмов координации взаимодействия компонентов.

Компонентный подход дает следующие преимущества при разработке ПС:

- *независимые расширения* ПС благодаря тому, что элементом (единицей) расширения является компонент, а компонентная модель и каркас гарантируют, что расширения не станут причиной непредсказуемого взаимодействия. Независимая разработка и развертывание расширений;

- *создание рыночных компонентов* благодаря тому, что компонентные модели предписывают стандарты, которым должны удовлетворять компоненты для независимой разработки и развертывания в стандартной среде;

- *экономия времени* выхода на рынок благодаря тому, что ключевые архитектурные решения могут быть «встроены» в компонентную модель и каркас;

- *предсказуемость обеспечиваемого качества ПС* благодаря тому, что компонентные модели и каркасы могут проектироваться с учетом приоритетных атрибутов качества для определенной области применения. Компонентные модели оп-

ределяют правила проектирования, которые обязательны для всех компонентов, развертываемых в компонентной системе. По сути, различные «глобальные» свойства системы (например, масштабируемость, безопасность и т.п.) встраиваются непосредственно в компонентную модель. Если компонентная модель налагает ограничения на проектируемые компоненты, то компонентный каркас реализует эти ограничения наряду с предоставлением необходимых сервисов.

Наиболее важный аспект CBSE - предсказуемость *композиции* взаимодействующих компонентов и каркасов. Возможны три основных вида композиций в ПС:

– «компонент-компонент» (взаимодействие по контрактам *прикладного* уровня, определяемое во время разработки (*раннее связывание*) или, что предпочтительнее, во время выполнения (*позднее связывание*)),

– «каркас-компонент» (взаимодействие между компонентом и другими компонентами каркаса по контрактам *системного* уровня с обеспечением управления ресурсами),

– «каркас-каркас» (взаимодействия между компонентами, разворачиваемыми в гетерогенных средах (каркасах) по контрактам *интероперабельного* (interoperation) уровня).

Компонент выступает в двух ипостасях – как *реализация* (разрабатываемая, развертываемая и интегрируемая в систему) и как *архитектурная абстракция* (отражающая правила проектирования, предлагаемые стандартной моделью для всех компонентов).

Компоненты могут изменяться и пополняться новыми функциональными возможностями и интерфейсами. Один компонент может содержать реализацию нескольких разных интерфейсов, а один интерфейс, в свою очередь, может быть реализован в разных компонентах. Замена одного компонента другим не должна приводить к перекомпиляции или перенастройке связей в ПС.

Компоненты конструируются в виде некоторой программной абстракции, включающей:

информационную часть - описание назначения, даты изготовления, условий применения (ОС, платформа и т.п.), возможностей повторного использования и др.;

внешнюю часть – интерфейсы, определяющие взаимодействие компонента с внешней средой и с платформами, на которых он будет работать, и обеспечивающие следующие общие характеристики компонента:

– интероперабельность – способность взаимодействовать с компонентами других сред;

– переносность (мобильность) – способность работать на разных платформах компьютеров;

– интегрируемость – способность к объединению с другими компонентами в сложные ПС;

– не функциональные характеристики - безопасность, надежность и защиту компонента и данных;

внутреннюю часть – фрагмент программного кода или абстрактную структуру - проект компонента, его спецификацию и исходный код – которые реализуют интерфейсы компонента, его функциональность и схемы развертывания.

Спецификация интерфейса может выполняться средствами API (Application Programming Interface) определенного языка программирования (с отражением только *функциональных* возможностей компонента) или на языке спецификации

интерфейса (нейтральном к языку программирования) - Interface Definition Language (IDL) (с отражением также и экстра-функциональных возможностей – спецификаций *поведения* (пред- и пост- условий выполнения операций и их результата), спецификаций *синхронизации* функций во времени, *характеристик качества* предоставляемого сервиса - точность, быстрота отклика и др.).

Современные языки программирования, например, Java, имеют расширения, специально предназначенные для спецификации поведения: iContract, JML (Java Modeling Language), Jass (Java with assertions), Biscotti и JINSLA (Java INterface Specification LAnguage). Могут использоваться также методы (языки) VDM (VDM++), Z (OOZE, Object-Z). Все они обеспечивают описание последовательности выполнения операций безотносительно ко времени. Для описания синхронизации операций в распределенных и параллельных системах могут использоваться, например, Object Calculus, CSP (Communicating Sequential Processes), Piccola, а для спецификации не функциональных атрибутов - NoFun. Специфицировать можно также сложность компонента, используя для этого функции библиотек Apple/SANE™.

В CBSE осуществлен переход от концепции специфицирования собственно компонентов к концепции специфицирования схем (шаблонов) их взаимодействия путем определения *контрактов* – с одной стороны, обязательств компонента (шаблонов взаимодействия, обеспечиваемых компонентом), а с другой стороны, правил взаимодействия (абстрактных шаблонов взаимодействия в соответствии с ролью в системе, которая возлагается на компонент).

Таким образом, компонентные системы основываются на четко определенных *стандартах* и *соглашениях* для разработчиков компонентов (определяемых компонентной моделью) и поддерживающей их *инфраструктуре* (компонентном каркасе), которая реализует сервисы для модели или приводит ее в действие, очевидно упрощая развертывание отдельных компонентов и приложений.

Компонентная модель предлагает:

- стандарты по *типам компонентов* (например, проекты, формы, beans-компоненты, COBRA-компоненты, RMI-компоненты, стандартные классы-оболочки, базы данных, JSP-компоненты, сервлеты, XML-документы, DTD-документы и т.п., определяемые в используемых языках программирования),
- стандарты *схем взаимодействия* (методы локализации, протоколы коммуникации, требуемые атрибуты качества взаимодействия – безопасность, управление транзакциями и др.),
- соглашения о *связывании* компонентов с ресурсами (сервисами, предоставляемыми каркасом или другим компонентом, развернутым в данном каркасе). Модель описывает, какие ресурсы доступны компонентам, и как, и когда компоненты связываются с этими ресурсами. Каркас, в свою очередь, рассматривает компоненты как ресурсы, подлежащие управлению.

Компонентный каркас (подобно операционной системе, объекты действия которой - компоненты) управляет ресурсами, разделяемыми компонентами, и предоставляет механизмы для коммуникации (организации взаимодействия) компонентов. Каркас необязательно существует отдельно от компонентов во время работы системы, его реализация может быть объединена с реализацией компонента, хотя предпочтительнее первое, как, например, каркас для поддержки компонентной модели EJB (Enterprise JavaBeans™). Примером визуализации композиции компо-

нентов может служить каркас VBXs (интерпретатор Microsoft Visual Basic) совместно с COM-разверткой и коммуникационными сервисами, предоставляемыми исходной ОС.

Продолжительность успеха компонентной инженерии будет обусловлена доступностью высококачественных программных компонентов и *доверием* потребителей к качеству компонентов, которые они покупают, что, в свою очередь, может быть обеспечено путем *сертификации* компонентов. Сертификации подлежат отдельные компоненты, каркасы и система, ими образованная. Однако, существует зависимость свойств системы от свойств сертифицируемых компонентов и, чем больше эта зависимость, тем более значимыми будут результаты сертификации компонентов. В пределе, при 100% уверенности (доверии) к каузальной связи, вообще отпадет необходимость сертифицировать систему. Она будет «правильной» по определению (по крайней мере, в контексте определенных свойств, по которым установлена зависимость). В отсутствии 100% уверенности проблема сертификации системы переходит в плоскость *предсказаний* и *обоснований композиций* компонентов в условиях существующей неопределенности.

Таким образом, суть и цель компонентной программной инженерии заключается в *быстрой сборке* программных систем из *отдельных компонентов*, причем компоненты и каркасы обладают *сертифицированными свойствами*, обеспечивающими базис для предсказания свойств системы в целом.

С точки зрения программирования парадигма компонентного программирования – это попытка решить те проблемы, которые возникают при использовании объектно-ориентированной парадигмы. Это, например, организация повторного использования путем распространения библиотек классов (исходного кода) или динамически компонуемых библиотек (dll-библиотек), проблема разработки *распределенных приложений* и др.

Основная идея новой парадигмы - распространение классов в *бинарном* виде (т.е. не в виде исходного кода) и *предоставление доступа* к методам класса через строго определенные *интерфейсы*, что позволяет снять проблему несовместимости компиляторов и обеспечивает смену версий классов без перекомпиляции использующих их приложений [35].

Наиболее известные технологии, реализующие парадигму компонентного программирования, - COM (DCOM, COM+), CORBA, .Net.

Сервисно-ориентированное программирование. Сервисно-ориентированное программирование (или архитектура ПО) (service-oriented architecture) — развитие компонентного подхода к разработке программного обеспечения, основанное на использовании *программ-сервисов* со *стандартизированными интерфейсами*. Компоненты ПС могут быть распределены по разным *узлам сети*, и предлагаться как независимые, слабо связанные, заменяемые сервисы-приложения. Интерфейс компонентов ПС обеспечивает *инкапсуляцию* деталей реализации каждого конкретного компонента от остальных. Таким образом, сервисно-ориентированная архитектура предоставляет гибкий способ комбинирования и повторного использования компонентов для построения сложных распределенных программных систем, в частности, крупных корпоративных программных приложений.

Программные системы, разработанные в соответствии с этой парадигмой, часто реализуются как набор *Web-сервисов*, интегрированных с помощью известных стандартных *протоколов* (SOAP, WSDL и т.п.).

Web-сервисы - это новая перспективная архитектура, обеспечивающая распределенность на новом уровне [36-38]. Вместо покупки компонентов и их встраивания в приложение предлагается покупать *время* их работы и формировать приложение, осуществляющее вызовы методов из компонентов, принадлежащих и поддерживаемых независимыми владельцами. Благодаря Web-сервисам функции любой программы в сети могут стать доступными через Интернет, а результаты обращениям к ним использоваться PHP-, ASP-, JSP-скриптами, JavaBeans, COM-объектами или другими средствами программирования. Простейший пример Web-сервиса - система *Passport* на *Hotmail*, позволяющая внедрить аутентификацию пользователей на собственном сайте.

В основе Web-сервисов лежат *стандарты*, *открытые протоколы* обмена и передачи данных и такой порядок действий:

- лицо, ответственное за Web-сервис, определяет *формат запросов* к своему Web-сервису и его *ответов*;
- любой компьютер в сети *делает запрос* к Web-сервису;
- Web-сервис обрабатывает запрос, выполняет какое-либо действие, а затем *отправляет ответ*.

Разница между Web-сервисами и другими технологиями (например, DCOM, именованные каналы - named pipes, RMI) состоит в том, что они основаны на открытых стандартах, которые широко поддерживаются на всех платформах Unix и Windows.

Формат запросов к Web-сервисам определяет стандартный протокол Simple Object Access Protocol (SOAP), разработанный W3C (www.w3.org). Сообщения между Web-сервисом и его пользователем пакуются в SOAP-конверты (SOAP envelopes). Сообщения содержат либо запрос на осуществление какого-либо действия, либо ответ - результат выполнения этого действия. Конверт и его содержимое кодируются в языке XML и отправляются через HTPP к Web-сервису.

Проблема применения Web-сервисов состоит в том, чтобы их *найти*. В недавнее время IBM, Microsoft и компания Ariba выступили с инициативным проектом системы Universal Description, Discovery and Integration (UDDI), которая призвана обеспечить ведение *общего каталога* всех Web-сервисов в Web-пространстве, предоставляя всем компаниям возможность бесплатно «опубликовать» (зарегистрировать) свой Web-сервис. Этот каталог работает как телефонная книга всех Web-сервисов.

Чтобы найти нужный Web-сервис достаточно обратиться на сайт http://www.uddi.org/, запустить поиск и получить описание сервисов. Определения форматов Web-сервисов представлены на легко читаемом языке WSDL (Web Services Description Language). Если необходимый Web-сервис найден, нужно узнать у компании, которая его предоставляет, цену на доступ к сервису, и, если она приемлема для бюджета, написать JSP-страницу для своего сайта с обращением к запрашиваемому Web-сервису.

Для создания Web-сервисов могут использоваться такие инструменты, как *Open Net* (компании Sun), *.NET* (Microsoft), e-services (HP) и *Web Services* (IBM). Существуют также инструменты с *открытыми исходными кодами* (open source frameworks), например, проекта *Mono Project* (www.go-mono.com), предоставляющего систему компиляции, исполнения кода и библиотек для работы одних и тех же Web-сервисов на всех платформах, включая Unix.

Очевидно, что при реальном использовании сервисно-ориентированной архитектуры (основанной на протоколе SOAP, языке XML и системе UDDI) неизбежно возникнут вопросы надежности приложений, использующих много «промежуточных» звеньев в Интернет (не говоря уже о больших объемах тегов XML, проблемах лицензирования и взимания платежей). Наиболее вероятными ее «почитателями» будут не столько системы управления большим и сложными объектами, сколько различные *информационные системы*.

Аспектно-ориентированное программирование (АОП). Многие из существующих парадигм и языков программирования, включая объектно-ориентированные, процедурные и функциональные, базируются на общих абстракционных и композиционных механизмах и процедурах, в основе которых – *функциональная декомпозиция*. Каждая структурная единица (включая компонент) системы заключается в процедуру, функцию или объект, образуя *функциональный* элемент (свойство) общей системы [39, 40]. Этот элемент хорошо локализован, легко доступен и, при необходимости, может быть сопоставлен с другими. Однако, *не функциональные* свойства системы, как, например, реакция на ошибки (их обработка и выдача диагностических сообщений), обеспечение доступа к памяти (динамический заказ-освобождение), синхронизация параллельно действующих объектов и др., обычно «рассеиваются» по всем элементам системы, «пересекая» структуру системы, вплетаясь в код и запутывая его. Для реализации свойств, отражающих тот или иной нефункциональный *аспект* системы, предложены *аспектные языки*, а стиль программирования с использованием совместно как языков описания компонентов, так и языков описания аспектов, назван аспектно-ориентированным.

Цель аспектно-ориентированного программирования (aspect-oriented programming) – инструментальная поддержка программиста в четком разделении компонентов и аспектов с помощью предоставления механизма, позволяющего абстрагировать и составлять их (компоненты и аспекты) для разработки общей системы.

Реализация приложений, включающих компоненты и аспекты, обеспечивается следующим:

компонентным языком, с помощью которого создаются компоненты,

одним или несколькими *аспектными языками*, с помощью которых реализуются аспекты,

компоновщиком (weaver) аспектов для комбинации языков и интеграции приложения. Компоновка (вплетение аспектов) в систему может производиться либо во время выполнения (runtime weaving) или во время компиляции (compile time weaving).

В АОП, как парадигме, принято рассматривать следующие важнейшие сущности, которые определяются в *модели встраивания аспекта в систему* (JPM, от Join Point model):

Точка присоединения — однозначно определяемое место в программе,

Срез — набор точек присоединения, удовлетворяющий заданному правилу,

Фрагмент вставки — набор инструкций языка программирования, который интегрируется во все точки заданного среза,

Аспект — *пара*: правило, задающее *срез*, и *фрагмент*, подлежащий вставке в точки этого среза. Аспект представляет собой языковую концепцию, схожую с классом, но только более высокого уровня абстракции. Аспекты могут затрагивать

многие классы и используют точки вставки для реализации регулярных действий (связанных с безопасностью, обработкой ошибок и т. п.).

Представление — формализованное правило изменения структуры класса.

Парадигма АОП призвана прийти на смену объектно-ориентированному и компонентному программированию и способствовать существенному повышению качества программ, особенно их сопровождаемости. Она поддерживает *мультипарадигмовую* концепцию программирования, сущность которой состоит в том, что разные аспекты проектируемой ПС могут быть реализованы в разных парадигмах программирования.

В настоящее время существует несколько реализаций аспектно-ориентированного программирования, наиболее известная из которых разработка центра Xerox PARC – *AspectJ* (www.aspectj.org), поддерживающая АОП в рамках языка Java. Новый релиз AspectJ 1.1 встраивается в такие системы разработки, как Eclipse, Sun ONE Studio и Borland JBuilder. Другой исследовательский центр — IBM Research — выпустил версию *HyperJ* (www.alphaworks.ibm.com/tech/hyperj) и систему Cosmos (www.research.ibm.com/AEM/mdsoc.html) с гипертекстовой поддержкой построения требований и диаграмм. Помимо Java новая парадигма АОП поддерживается и в других языках, таких как Си, Си++, Squeak/Smalltalk, Perl, Python, Ruby [41]. Идеи АОП продолжают развиваться в компании *Intentional Software* идеологами этой парадигмы Грегором Кишалесом и Чарльзом Саймони.

АОП тесно связано с *ментальным программированием* (intentional programming), *генерирующим* (порождающим, трансформационным) программированием (generative programming, transformational programming).

Контекстно-ориентированное программирование (то же, что и аспектно-ориентированное программирование). Суть контекстно-ориетированного программирования (context-oriented programming) состоит в применении аспектов, которые, действуя на систему, изменяют ее поведение, причем код аспекта *модифицируется* в зависимости от контекста, в котором он применяется. Модификация может быть выполнена средствами *метапрограммирования*. Таким образом, можно рассматривать идеи, заложенные в АОП, как средство структуризации идей метапрограммирования, а само АОП – как одно из его возможных проявлений [42].

Для эффективной реализации аспектов в разных контекстах их применения разработаны библиотеки расширений языков программирования для определенных предметных областей. Они включают отдельные функции компиляторов, средств оптимизации, редактирования, визуализации понятий, перестройки компонентов компиляторов под новое языковое расширение, средства программирования на основе шаблонов и т.п. Библиотеки с такими возможностями получили название библиотек *генерирующего типа*. Пример - библиотека матриц, предоставляющая средства для вычисления выражений с массивами. Программирование с использованием таких библиотек относится к разряду стилей *обобщенного* (*родового*) *программирования* (generic programming).

Генерирующее* (*порождающее*) *программирование. На современном этапе развития информационных технологий архитектурно-функциональный спектр типов программных компонентов очень широк (от утилит для мэйнфреймов до Web-сервисов). Построение из них программных систем с нужными свойствами требует решения задач классификации и типизации, идентификации архитектуры, спецификации функций и интерфейсов, хранения, поиска, выбора, адаптации, интегра-

ции (конфигурирования) и, наконец, собственно генерации ПС. Существующие методы разработки таких ПС не предполагают автоматизации. Их *автоматизированное* построение возможно в рамках новой парадигмы в программной инженерии – *генерирующее* (*порождающее*) *программирование* (generative programming) [43].

Генерирующее программирование – парадигма программной инженерии, в основе которой лежит двух-фазовый процесс разработки ПС – *инженерия предметной области* (domain engineering) и *инженерия приложений* (application engineering), то есть, сочетание разработки программных компонентов для обеспечения их повторного использования и последующей разработки ПС с применением повторно используемых компонентов. Главным элементом программирования в этой парадигме является не уникальный программный продукт, созданный из повторно используемых компонентов для конкретных применений, а *семейство* (линейка) продуктов (product line). Элементы семейства не создаются с нуля, а генерируются на основе общей *генерирующей модели ПрО* (generative domain model) - модели семейства.

Инженерия ПрО (доменная инженерия) включает:

– *анализ домена:*

- определение границ домена и его связей с другими доменами;
- выявление и формальное описание общностей и отличительных особенностей внутри домена - определение постоянных *общих требований* ко всему семейству программных продуктов, как единому целому, и *специфичных переменных требований* к его компонентам (объектам, аспектам), в совокупности покрывающих все *требования к ПС;*
- классификация и документирование моделей домена;
- формирование терминологических словарей для описания основных понятий в домене и взаимосвязей между его активами;
- оценка моделей и словарей домена в соответствии с выбранной методологией моделирования;

– *архитектурное проектирование домена:*

- создание платформы из повторно используемых компонентов, разрабатываемых или приобретаемых. Используются горизонтальная и вертикальная типизация компонентов. К «горизонтальным» относят общие системные средства: графические пользовательские интерфейсы, СУБД, системные программы, библиотеки расчета матриц, контейнеры, каркасы и т.п. К «вертикальным» - прикладные системы (медицинские, биологические, научные и т.д.), специализированные прикладные методы инженерии предметных областей и др.;
- верификация и тестирование повторно используемых компонентов;
- формирование генерирующей модели домена на базе описанных требований.

Архитектура домена – высокоуровневый проект, в котором формально определены интерфейсы компонентов. Эта архитектура служит каркасом для организации повторного использования компонентов с целью конструирования из них программных продуктов. Архитектура домена подвергается оценке согласованности с моделью домена и стандартами организации, а также соответствия выбранной методологии архитектурного проектирования.

Применение процесса инженерии ПрО дает следующие очевидные преимущества организации-разработчику: выигрыш в производительности труда при

создании ПС, повышение качества продуктов, ускоренный выпуск, снижение потребности в трудовых ресурсах.

Инженерия приложений подразумевает:

- разработку ПС по спецификации требований на базе повторно используемых компонентов и генерирующей модели домена, в результате чего генерируемая ПС приобретает все общие свойства своей платформы и наделяется специфичными (требуемыми) свойствами;
- тестирование ПС.

Примером систем поддержки инженерии ПрО является система DEMRAL (ориентированная на компоненты «горизонтального» типа: библиотеки численного анализа, распознавания речи, графовых вычислений и т.д.) и RSEB (ориентированная на компоненты «вертикального» типа).

Агентно-ориентированное программирование. *Интеллектуальный программный агент* - сущность, способная формулировать цели, обучаться, планировать свои действия и принимать решения в динамически изменяемых обстоятельствах [44, 45]. Появление концепции агентов обусловлено существующим уровнем информационных технологий и назревшей объективной необходимостью обработки больших объемов информации, распределенной в информационных сетях и постоянно обновляемой. Простым примером задачи, которая может эффективно решаться с помощью агентов, является поиск нужных сведений в Интернет, требующий, как правило, больших затрат времени на выборку, анализ и отсеивание лишней информации.

Парадигма агентно-ориентированного программирования формировалась на базе теории динамических систем, теории управления, когнитивной психологии (теории мотивации), концептуального моделирования баз данных и знаний и др.

Проблема построения и использования агентов исследуется в рамках областей искусственного интеллекта и компьютерных наук. Под агентом, как правило, понимают *компьютерную программу*, которой присущи следующие свойства:

– *автономность* – способность работать без внешних вмешательств и осуществлять определенный контроль своих действий и состояний;

– *социальная активность* – способность сотрудничать с другими программами-агентами (и людьми), используя агентно-коммуникационные языки;

– *реактивность* – способность изменять свое поведение в зависимости от состояния внешней среды;

– *проактивность* – умение не только решать текущую задачу поиска, но и выбирать при этом потенциально полезную для пользователя информацию в своей базе данных.

Интеллектуальность агента повышает наличие дополнительных свойств:

– *убеждений* – основанных на знании агента о текущем состоянии окружения и изменениях в нем, к которым должны привести действия агента;

– *желаний* – основанных на отношении агента к будущим состояниям окружения и предпочтении одного состояния над остальными, а также способности различать несовместимые желания и не осуществимые желания;

– *целей* – рассматриваемых как подмножества непротиворечивых желаний агента;

– *намерений* – образованных непротиворечивым подмножеством целей, достижимых агентом при определенном ограничении ресурсов, и средствами их достижения;

– *мобильности* – способности самостоятельно переходить с одной платформы на другую.

В настоящее время большинство агентов жестко специализированы и не обладают одновременно всеми вышеописанными свойствами.

Хотя существует множество подходов к классификации агентов, наиболее устоявшейся является такая их классификация:

– по *архитектуре построения* агентов и их свойствам

- рассудительный агент - делающий выводы и обучающийся, имеющий четкую модель мира, собственную базу знаний и механизмы логического вывода новых знаний,

- реагирующий (реактивный) агент - анализирующий пред- и пост- условия активации модулей, его составляющих, в ответ на изменения во внешнем окружении;

- гибридная архитектура – в которой одна из подсистем – рассудительная (разрабатывает планы и принимает решения), другая – реагирующая (действует по плану, реагирует на события);

– по *функциональному назначению*

- интерфейсные агенты, взаимодействующие с пользователями,

- задачные агенты, привлекаемые к решению определенных задач,

- информационные агенты, непосредственно работающие с источниками данных;

– по способности к *мобильности*

- мобильные (распределенные) агенты, полностью автономные, способные перемещаться от сервера к серверу в поисках информации и нести в себе информацию о своем состоянии,

- стационарные (локальные) агенты, работающие на стороне клиента или на стороне сервера.

Наиболее известные агентные архитектуры – PRS, JAM, TOURINMACHINE, COSY, INTERRAP.

Отдельно разработанные агенты могут образовывать *мультиагентную систему*, взаимодействуя для достижения общих целей. В такой системе каждый агент имеет представление об окружении (модель мира), о себе (ментальную модель) и об агентах, с которыми он взаимодействует (социальная модель), знает цели - реактивные (вызванные внешними событиями), собственные (локальные) и кооперативные, а также условия собственного поведения в определенных ситуациях и условия локального и совместного планирования действий.

Основу агентно-ориентированного программирования составляют:

– формальный язык описания системы моделей (ментальной, социальной);

– язык спецификации информационных, временных, мотивационных и функциональных действий агента в среде работы;

– язык интерпретации спецификаций агента;

– инструменты конвертирования любых программ в соответствующие агентные программы.

Спецификация агента уточняется, интерпретируется и компилируется в вычислительное представление агентной программы, пригодное для выполнения в среде функционирования.

Способы и средства взаимодействия агентов определены как координация, коммуникация, кооперация или коалиция.

Координация агентов – это процесс, с помощью которого агенты обеспечивают последовательное функционирование при согласованности их поведения и без взаимных конфликтов. Координация агентов определяется:

– взаимозависимостью целей всех агентов-членов коалиции, а также возможного влияния агентов друг на друга;

– ограничениями, которые принимаются для группы агентов коалиции в рамках совместного функционирования;

– компетенцией – знаниями условий среды функционирования и степени их использования.

Координационные механизмы основаны на соглашениях, которые определяются с учетом стоимости и полезности (в терминах прибыли от установленного соглашения между агентами).

Главным средством коммуникации агентов является транспортный протокол TCP/IP. Однако он не достаточен для поддержки «социальности» агентов. Более приемлем выработанный в рамках проекта FIPA стандарт языка передачи сообщений - ACL (Agent Communication Language).

К первым коммерческим языкам программирования агентов относится Telescript. На практике агент может быть реализован как компонент Java, COM-объект, Lisp-программа или описание TCL. Для создания мультиагентных систем могут использоваться языки APRIL и MAIL. Все современные инструментальные средства построения мультиагентных систем подразделяются на два класса – *библиотеки* (например, JATLite – дополнительные библиотеки к языку Java) и *среды* (например, AgentBuilder, среда, предоставляющая средства для организации мультиагентной системы, средства спецификации архитектуры агента и поведения агентов, а также средства отладки агентных приложений и наблюдения за поведением агентов).

С развитием глобальной компьютерной сети и ростом объемов информационных ресурсов Интернет роль интеллектуальных агентов и мультиагентных систем в организации эффективного поиска информации возрастает [46, 38].

Автоматное программирование. Это стиль программирования, основанный на применении *конечных автоматов* для описания поведения программ. Автоматы задаются графами переходов, для различения вершин в которых вводится понятие *кодирование состояний*. Особенность автоматного программирования состоит в том, что графы переходов используются при *спецификации*, *проектировании*, *реализации*, *отладке*, *документировании* и *сопровождении* программ [47].

Программирование выполняется «через состояния», а не «через события и переменные», что позволяет лучше понять и специфицировать задачу и ее составные части. Переход от графового представления к текстовому осуществляется формально и изоморфно с применением конструкции switch (в языке C) или ее аналогов (в других языках). Поэтому стиль автоматного программирования часто называют «SWITCH-технологией».

В настоящее время этот стиль развивается в нескольких вариантах, различающихся как классом решаемых задач, так и типом вычислительных устройств, на которых осуществляется программирование. Известны, например, его варианты для систем логического управления, в которых события отсутствуют, входные и выходные воздействия являются двоичными переменными, а операционная система работает в режиме сканирования. Автоматный подход распространен и на событийные системы, называемые также *реактивными*. В них как входные воздействия используются события, в качестве выходных воздействий - произвольные процедуры, а в качестве операционных систем — любые операционные системы реального времени. Для программирования событийных систем с применением автоматов используется процедурный подход, поэтому такой стиль программирования называется «программированием с явным выделением состояний». Известен также подход, основанный на совместном использовании объектного и автоматного стилей и называемый «объектно-ориентированное программирование с явным выделением состояний».

В контексте обеспечения качества, очевидно, что применение автоматов проясняет *поведение* программы, а наличие добротной проектной документации резко упрощает осуществление рефакторинга программы.

2.5.7. Парадигмы теоретического программирования

Наряду с парадигмами прикладного программирования ПС продолжают развиваться и парадигмы теоретического программирования, в основе которых лежат фундаментальные исследования, базирующиеся на математических теориях [48 - 50]. Авторами украинской теоретической школы программирования, созданной В.М. Глушковым, предложены новые парадигмы алгебраического, инсерционного, композиционного, экспликативного программирования, и теоретические подходы в решении ключевых проблем программирования.

Парадигма ***алгебраического программирования*** [51, 52] основывается на теории переписывания термов. В этой парадигме *термы* представляют данные, а системы *переписывающих правил*, выражаемых с помощью системы равенств, - алгоритмы вычислений. Элементарный шаг вычисления включает сопоставление с образцом, проверку условий и подстановку. Порядок выбора переписывающих правил и подтермов данного терма для сопоставления с левыми частями равенств определяется *стратегией* переписывания. По существу, стратегия определяет результат вычислений – терм – с точностью до эквивалентности исходному терму. Собственно стратегия переписывания может быть описана в парадигме более низкого уровня, например, процедурной или функциональной, что приводит к необходимости интеграции парадигм [51]. В настоящее время идея интеграции парадигм (процедурной, функциональной, алгебраической и логической) нашла воплощение в системе алгебраического программирования (APS) [52], в которой используются специализированные структуры данных - *графовые термы* – для представления данных и знаний о предметных областях.

Инсерционное программирование обобщает взгляд на программу, как на алгебраически определенное *преобразование* множества состояний информационной среды, однако вместо пассивной среды (памяти) рассматривается *активная информационная среда*, обладающая наблюдаемым *поведением* [53]. Преобразование поведения этой среды происходит в результате воздействия на ее объекты взаимодействующих *агентов*. В основе инсерционного программирования – модель пове-

дения агентов в средах [54], базирующаяся на понятиях *транзиционной системы* (основного стандарта в поведенческой теории взаимодействующих процессов) и отношения *бисимуляционной эквивалентности* агентов относительно среды (состояние агента отождествляется с его поведением). В отличие от агентного программирования, концентрирующего внимание, в большей степени, на проблемах интеллектуализации агентов, инсерционное программирование охватывает поведенческие аспекты агентов.

Термин «инсерционное программирование» происходит от английского *insert* – вставлять, помещать, погружать. Программа в этой парадигме рассматривается как агент, обладающий поведением, который, *погружаясь* в среду, меняет ее поведение по отношению к внешнему наблюдателю, а также другим агентам, погружаемым в эту среду в дальнейшем. Написать инсерционную программу – значит определить функцию погружения (закон функционирования среды с погруженными в нее агентами), а также начальное состояние среды и агента, погруженного в эту среду. В качестве базовой системы программирования для представления состояний агентов и сред в виде структур данных, а также для программирования функции погружения, используется система алгебраического программирования APS [55]. Инсерционная программа пишется на языке AL+ и имеет три уровня [56]:

1) описание поведения инициализированной многоуровневой среды с погруженными в нее агентами;

2) задание функции, определяющей отношения переходов агентов;

3) описание ядра функции погружения (функции развертывания погружений).

Инсерционными программами моделируются реальные системы с недетерминированным поведением агентов и сред. Реализация систем инсерционного программирования требует применения *программ-симуляторов*, а не интерпретаторов, а также постановки *целей* для получения конкретных результатов.

Наряду с новыми парадигмами программирования разрабатывается общая теория программирования, *программология*, объединяющая идеи логики, конструктивной математики и информатики, *уточняющая* понятия программы и программирования и на единой концептуальной основе предоставляющая общий формальный аппарат для конструирования программ [57-59]. В числе наиважнейших программных понятий и принципов выделяются понятие *композиции* и *принцип композиционности*, который трактует программы как функции, строящиеся из других функций с помощью специальных *операций*, называемых композициями. Принцип композиционности стал основным в ***композиционном программировании*** [60-62]. На основе композиционной *экспликации* (от explication-уточнение, разъяснение) понятия программирования была развита логико-математическая система композиционного построения программ, получившая в последствии название ***экспликативного программирования*** [63]. Экспликативное программирование интегрирует в себе все наиболее существенные парадигмы (стили) программирования (структурное, функциональное, объектно-ориентированное и др.) в рамках концептуально единой экспликативной платформы, основу которой составляет три основных типа объектов: собственно объекты, средства построения из одних объектов других (функции) и программологические средства применения средств построения (композиции). Прагматически ориентированную конкретизацию экспликативного про-

граммирования (с выделенным эталонным ядром из совокупности композиций) представляет собой эталонное программирование [64].

За рамками рассмотрения этого раздела осталось множество стилей программирования (часть из которых указана на рисунке 2.6). Отправной точкой для знакомства с ними могут послужить энциклопедии – *Wikipedia* и *Википедия*.

Литература к главе 2

1. *Орлик Сергей*. Программная инженерия. - http://www.almportal.ru/public/so/book/3-software_engineering.pdf
2. *Jackson M.* Software requirement & specifications.– Wokingham, England: Addison–Wesley, ACM Press Books, 1995. –228 p.
3. *ISO/IEC DTR 19759* Software Engineering -- Guide to the Software Engineering Body of Knowledge– SWEBOK. - http://www.swebok.org/ironman/pdf/ SWEBOK_Guide_2004.pdf.
4. *ACM/IEEE-CS.* Software Engineering Code of Ethics and Professional Practice. - http://www.acm.org/serving/se/code.htm
5. *Software* Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.- http://sites.computer.org/ccse/SE2004Volume.pdf.
6. *ABET* – Criteria for According Computing Programs, 2002 (Computing Accreditation Commission of the Accreditation Board for Engineering and Technology) – http://www.abet.org/criteria.htm
7. *Shaw M.* Software Engineering Education: A Roadmap. The future of Software Engineering, A finkkelstein, ed., ACM press, New York, 2000.
8. http://computer.org/education/cc2001 или http://se.math.spbu.ru/cc2001 (русский вариант).
9. *Лаврищева Е.М.* Проблематика программной инженерии // К.: Знання.–1991.– 19с.
10. *Бабенко Л.П.*, Лаврищева Е.М. Основы программной инженерии. Учебник (укр. язык). – Киев: Знання, 2001. – 269 с.
11. *Майер Б.* Программная инженерия как предмет обучения. Открытые системы. Июль-август, 2001.–С.80–86.
12. *Лаврищева Е.М.* Пути стандартизации программной инженерии как специальности // Материалы междун. научно–практ. конф. “Теория и практика стандартизации образования” часть 1. Минск, 18–19 янв. 2001г.– С. 8–13.
13. *Соммервил И.* Инженерия программного обеспечения. 6 -издание.–Москва–Санкт–Петербург–Киев, 2002.– 623 с.
14. *Лаврищева Е.М.*, Грищенко В.Н. Области знаний программной инженерии SWEBOK и подход к обучению этой дисциплине // УСиМ.-№1.- 2005.- С.38-54.
15. *A guide* to the Project Management Body of Knowledge // PMBOK GUIDE. Third Edition (http://www.pmi.org/emeaelink/pmiE-link10-04.pdf).
16. *ISO/IEC JTC1 N7727*. SC7 Proposed new work item on project management standard (28.02.2005) - http://isotc.iso.org/livelink/livelink/fetch/2000/2122/327993/755080/1054034/2541793/JTC001-N-7727.pdf?nodeid=3961521&vernum=0
17. *PMI PMBoK* 2004 - "Новый завет" библии менеджера проектов. - http://www.cfin.ru/itm/project/pmbok2004.shtml
18. *ISO/IEC JTC1/SC7/WG7* N0820. NP proposal Project Management standard //07N3156 W07N0820_NP_PROPOSAL_PROJECT_MANAGEMENT_STD.DOC
19. *Развитие* парадигм программирования. В курсе «Основы функционального программирования» - http://www.intuit.ru/department/pl/funcpl/15/3.html

20. *Парадигмы* программирования - http://schum.kiev.ua/let/
21. *Оберон* как эсперанто программирования – http://www.computer-museum.ru/histsoft/ober_esp.htm
22. *Ваныкина Г.В.*, Якушин А.В. Рекурсивные вычисления в различных парадигмах программирования - http://mech.math.msu.su/conference/nikolsky-100/Articles/Vanikina.htm
23. *Капитонова Ю.В.*, Летичевский А.А. Об основных парадигмах программирования // Кибернетика и системный анализ.-1994.-№6.-с.3-20.
24. *Лаврищева Е.М.*, Грищенко В.Н. Сборочное программирование.–Киев.– Наукова Думка, 1991.–213с.
25. *Липаев В.В.*, Позин Б.А., Штрик А.А. Технология сборочного программирования.М.: – Радио и связь, 1992. – 275с.
26. *Рыбаков А.*, Золотарев С. Программное обеспечение многоядерных систем - http://old.osp.ru/os/2006/02/012.htm
27. *Суперкомпьютерные* кластерные системы – организация вычислительного процесса / В.Н. Коваль, С.Г.Рябчун, И.В.Сергиенко, А.А.Якуба // Проблеми програмування.-№2-3 (спеціальний випуск за матеріалами конференції УкрПРОГ'2006).-с.197-210.
28. *Дехтяренко И.А*. Декларативное программирование - http://www.softcraft.ru/paradigm/dp/dp01.shtml
29. *Доказательное* программирование и синтез программ - http://mail.spb.fio.ru/archive/group43/c4wu3/proof.htm
30. *Грищенко В.Н.*, Лаврищева Е.М. Методы и средства компонентного программирования // Кибернетика и системный анализ.-2003.– №1.– С. 39 – 55.
31. *Грищенко В.Н.*, Лаврищева Е.М. Компонентно-ориентированное программирование. Состояние, направления и перспективы развития // Проблемы программирования. – 2002. - № 1–2.– С. 80–90.
32. *Лаврищева Е.М.* Парадигма интеграции в программной инженерии // Проблемы программирования. - 2000. – №1–2.– С.351–360
33. *Бабенко Л.П.* Повторное использование в программной инженерии // Кибернетика и системный анализ.-1999.-№2.-С.37-48.
34. www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr008.pdf
35. *Добрынин В.* Технология компонентного программирования – http://www.ict.edu.ru
36. Conney P. Web-services - // www.alistapart.com
37. *Рогушина Ю.В.*, Гладун А.Я. Онтологическая модель интеллектуализации сервис-ориентированных вычислений в распределенной среде Интернет // Проблеми програмування.-№2-3 (спеціальний випуск за матеріалами УкрПРОГ'2006).-2006.-с.526-536.
38. *Андон П.*, Дерецький В. Проблеми побудови сервіс-орієнтованих прикладних інформаційних систем в Semantic Web середовищі на основі агентного підходу // там же.-с.493-502.
39. *Аспектно* ориентированное программирование – http://www.javable.com/columns/aop/workshop/02/index1.pdf
40. *Elrad T.*, Filman R.E. Aspect-oriented programming // Communications of the ACM.–2001.– vol.44.– N 10.– P.33–38
41. *Богатырев Р.* Глядя в будущее // Мир ПК. - 2003.- №1 (www.osp.ru/pcworld/2003/01/142_3.htm)
42. *Чистяков В.* R# - метапрограммирование в .NET // RDSN Magazine.-2004.-№5 (www.rsdn.ru/projects/rsharp/article/rsharp_mag.xml)

43. *Чернецки К.*, Айзенекер У. Порождающее программирование. Методы, инструменты, применение // Издательский дом «Питер».–2005.–730с.
44. *Плескач В.Л.*, Рогушина Ю.В. Агентні технології // Київ.- КНТЕУ.–2005.–337с.
45. *Трахтенгерц Э.А*. Взаимодействие агентов в многоагентных средах //Автоматика и телемеханика.– М.: Наука .–1998.–№8.– с.3-52.
46. *Рогушина Ю.В*. Разработка средств интеллектуализации поиска информации в Интернет // Проблемы программирования.-№1-2 (специальный выпуск по материалам конференции УкрПРОГ'2002). -2002.-с.378-385.
47. *Автоматное* программирование - http://ru.wikipedia.org/wiki
48. *Глушков В.М*. Теория автоматов и формальные преобразования микропрограмм // Кибернетика.-1965.-№5.-С.1-10.
49. *Цейтлин Г.Е*. Теория алгоритмических метаалгебр и ее приложения // Проблемы программирования.-1999.-№1.-С.4-15.
50. *Глушков В.М.*, Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование – 3-е изд., перераб. и доп.- Киев: Наукова думка, 1989.-376с.
51. *Капитонова Ю.В.*, Летичевский А.А. Методы и средства алгебраического программирования // Кибернетика.-1993.-№3.-С.7-12.
52. *Летичевский А.А*., Маринченко В.Г. Объекты в системе алгебраического программирования // Кибернетика и системный анализ. –1997.– N2. – С.160–180.
53. *Инсерционное* программирование / Летичевский А.А., Капитонова Ю.В., Волков В.А., Вышемирский В.В., Летичевский А.А. (мл.) // Кибернетика и системный анализ.– 2003, №1.– 19-32.
54. *Letichevsky A.A.*, Gilbert D.R. A model for interaction of agents and environments //Recent trends in algebra's development technique Language.–2000. – P.311 – 329.
55. *Капитонова Ю.В*., Летичевский А.А., Волков В.А. Дедуктивные средства системы алгебраического программирования // Кибернетика и системный анализ. –2000. – №1.– С.17 – 34.
56. *Letichevsky A.A.*, Gilbert D.R. A General Theory of Action Language // Кибернетика и системный анализ. –1998. – №1.– С.16 – 36.
57. *Редько В.Н.* Композиционная структура программологии // Кибернетика и системный анализ. – 1998. –№4. – С. 47–66.
58. *Редько В.Н.* Основания программологии // Кибернетика и системный анализ. – 2000.– № 1.– С. 35–57.
59. *Редько В.Н*., Брона Ю.Й., Буй Д.Б. Реляційні бази даних: табличні алгебри та SQL–подібні мови //Видав. дім "Академперіодика", Київ, 2001.–195с.
60. *Редько В.Н.* Композиции программ и композиционное программирование // Программирование.-1978.-№5.-С.3-24.
61. *Редько В.Н.* Основания композиционного программирования // Программирование.-1979.-№3.-С.3-13.
62. *Никитченко Н.С.* Композиционно-номинативный подход к уточнению понятия программы // Проблемы программирования. –1999.– №1.– С. 16–31.
63. *Редько В.Н.* Экспликативное программирование: ретроспективы и перспективы // Проблемы программирования.– 1998. – №2.–С. 22 – 41.
64. *Редько В.Н.*, Гришко Н.В., Редько И.В. Эталонное программирование: ретроспективы и перспективы // Проблемы программирования.– 2000. – №1-2.–С. 13– 28.

Глава 3. МОДЕЛИ И МЕТРИКИ КАЧЕСТВА

3.1. Метрики качества программных систем

3.1.1. Метрика как основа измерения

Атрибуты программной системы, характеризующие ее качество, измеряются с использованием метрик качества.

Метрика – это комбинация конкретного *метода измерения* (способа получения значений) атрибута сущности и *шкалы измерения* (средства, используемого для структурирования получаемых значений).

Метрика определяет *меру*[1] атрибута – переменную, которой присваивается значение в результате измерения (рисунок 3.1). Например, сущность – «отчет об обнаруженных дефектах», атрибут – «список дефектов», метод измерения – «подсчет количества дефектов в списке», шкала – «целые числа больше нуля», мера атрибута – «общее число дефектов», имя метрики (обычно одноименное мере) – «общее число дефектов».

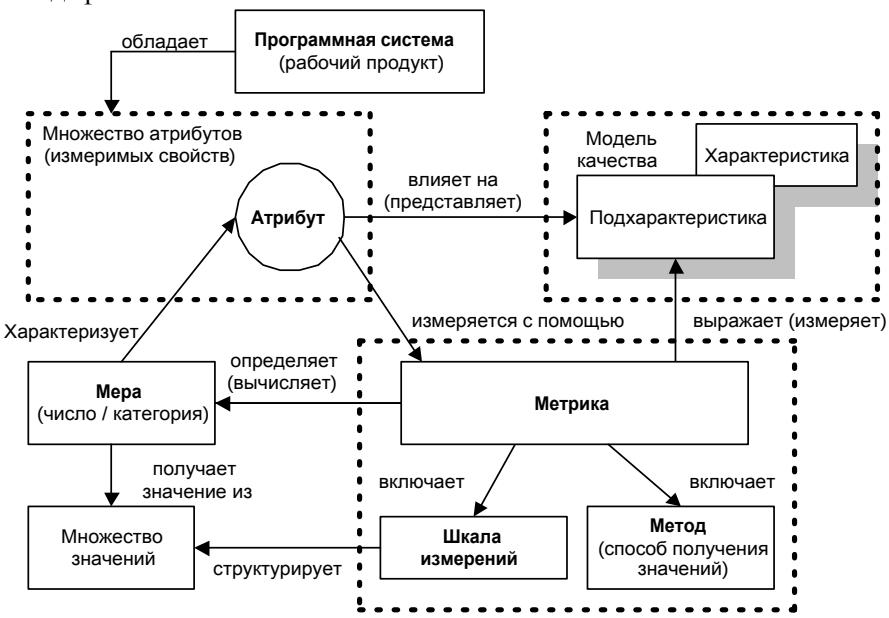


Рис. 3.1. **Метрика в системе измерения качества**

Мера атрибута может быть *непосредственной*, если она не зависит от мер других атрибутов, либо *косвенной*, полученной по мерам других атрибутов.

По определению стандарта ISO/IEC 9126-2 *метрика качества программной системы* представляет собой «модель измерения атрибута, связываемого с характеристикой качества ПС. Служит индикатором одного или многих атрибутов. Ее можно увидеть, например, в левой части большинства уравнений X = A*B, где X имеет не ту же шкалу, что A или B» [1].

[1] Термин «мера» - эквивалент употребляемого в международных стандартах термина «measure» (measure – мера, измерять, единица измерения). Понятия меры и метрики в программной инженерии отличаются от используемых в функциональном анализе.

Метрика называется *базовой*, если в ее основе лежит элементарный метод (примитив) измерения атрибута. По определению того же стандарта «базовая метрика сама по себе не является индикатором характеристики или подхарактеристики качества. Ее можно увидеть, например, в правой части большинства уравнений X = A*B. A и B – базовые метрики». То есть базовые метрики используются только в составе модели измерения атрибута.

Для того чтобы правильно пользоваться результатами измерений, для каждой меры нужно идентифицировать ***шкалу измерения***.

Стандарт ISO/IEC 9126-2 рекомендует применять 5 видов шкалы измерения значений (упорядоченных от менее строгих к более строгим):

- *номинальная шкала*. Это классификационная шкала, выполняющая категоризацию свойств оцениваемого объекта. Категории не упорядочены. Например, дефекты могут классифицироваться на дефекты интерфейса, логики, объявления данных и др. Языки – Fortran, C++, Java и др.;
- *порядковая шкала*. Позволяет упорядочивать характеристики по возрастанию или убыванию путем сравнения их с базовыми значениями. Например, для уровня серьезности последствий события шкала может включать значения «низкий», «средний», «высокий», «критический». Для уровней CMM – 1, 2, 3, 4, 5. Расстояние между значениями по шкале не играет роли. Характеристики, имеющие номинальную или порядковую шкалу измерения, называются качественными (или категорийными). Все остальные - количественными.
- *интервальная шкала*. Отмечает существенные различия свойств объекта, «дистанцию» между ними (например, календарные даты или значения плотности дефектов - 1.5 дефекта/KSLOC, 3.5 дефекта/KSLOC и т.д.). Используется в арифметических операциях и операциях сравнения (в данном примере разница равна 2 дефекта/KSLOC). Нулевое значение не допустимо;
- *относительная шкала*. Значения по этой шкале различаются по отношению к выбранной единице (например, времени, изменяющемся от 0 до бесконечности, или стоимости). Применяя эту шкалу можно рассчитать, например, время между отказами, размер программного компонента в SLOC и др. Считается наиболее предпочтительной шкалой измерений. Позволяет применять широкий спектр инструментов измерения (гистограмм, диаграмм Парето и др.);
- *абсолютная шкала*. Это специальный случай относительной шкалы. В этой шкале указывается абсолютное значение величины. Например: «размер программы равен 2К», «число обнаруженных ошибок равно 20».

Измеренное значение метрики само по себе не несет информации об уровне удовлетворения требований к качеству. Для этих целей шкала должна быть разделена на области (ранги), соответствующие различным степеням удовлетворения требований. Примеры деления шкалы:

- деление значений по двум категориям - удовлетворительные и неудовлетворительные значения;
- деление шкалы по четырем категориям, ограниченным тремя уровнями значений - текущим, худшим и плановым (рисунок 3.2).

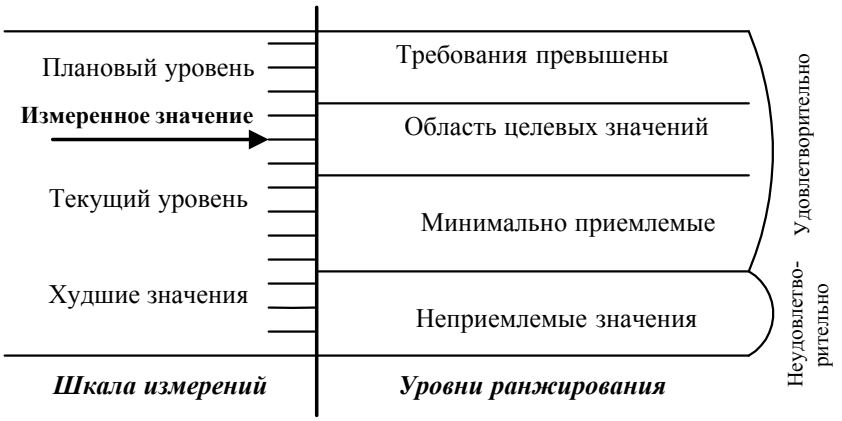


Рис. 3.2. **Уровни ранжирования метрик**

По мере накопления практики измерений и знаний об измеряемых атрибутах шкалы их измерения могут эволюционировать от менее информативных (номинальной и порядковой) к более информативным (относительной или абсолютной).

3.1.2. Классификация мер качества

Для разработки процедур сбора данных, интерпретации мер и их нормализации с целью сравнения, нужно различать следующие типы мер, определяемых метриками.

- ***меры размера***. Представляют размер ПС в разных единицах измерения, например:

– *функциональный размер* (учет функциональных возможностей ПС. Значение представлено в условных единицах функциональности (глава 8));

– *размер программы* (учет числа строк исходного кода, количества модулей, количества операторов на языке программирования);

– *объем ресурсов*, используемых работающей программой (учет объема оперативной памяти, дисковой памяти или сетевого трафика, загруженности процессора - количества обрабатываемых инструкций в секунду и др.);

- ***меры времени***. Представляют периоды *реального времени* (в секундах, минутах или часах), *процессорного времени* (в секундах, минутах или часах работы процессора) или *календарного времени* (в рабочих часах, календарных днях, месяцах, годах), например:

– *время функционирования системы* с непрерывной или дискретной работой компонентов программного обеспечения (учет *истекшего времени* (при работе программного обеспечения в системе в течение установленного периода времени), учет *времени с момента включения* (при непрерывном использовании встроенного программного обеспечения системы реального времени), учет *нормализованного времени* (при совместной работе нескольких компьютеров, обменивающихся данными);

– *время выполнения* (время, необходимое работающему программному компоненту системы для завершения решения определенной задачи в определенных условиях);

– *время использования* (время, затрачиваемое определенным пользователем на решение задач с помощью ПС), например: *время сеанса работы* (от начала до завершения), *время решения задачи* и др.;

- ***меры усилий***. Представляют полезное (продуктивное) время, связанное с определенной задачей проекта, например:

– *производительность труда (при индивидуальной работе);*

– *трудоемкость (при коллективной работе);*

- ***меры интервалов между событиями***. Представляют интервалы времени между наступлением событий, происходящих в определенный период наблюдения, например, *время между последовательными отказами*. Вместо этой меры может использоваться *частота* наступления событий;
- ***счетные меры***. Представляют собой *статические счетчики* (для учета определенных элементов в рабочих продуктах ПС (документах)) или *кинетические (динамические) счетчики* (для учета событий или действий человека), например:

– *количество обнаруженных ошибок* (учет ошибок в ходе инспекции, тестирования, функционирования или сопровождения ПС);

– *структурная сложность программы* (количество путей в программе, цикломатическая сложность и др.);

– *число несовместимых элементов* (учет *ошибок согласования* (требований, стандартов, форматов и др.), учет ответов типа «да»/«нет» в вопросниках);

– *число изменений* (учет элементов конфигурации, в которых произошли изменения):

– *число обнаруженных отказов* (учет отказов при тестировании, функционировании или сопровождении ПС);

– *число попыток* (учет попыток корректировки дефектов или ошибок);

– *эргономические счетчики* – число нажатий клавиш, щелчков на кнопках и др.

– *счетчики-оценки (-очки, -баллы)* (учет результатов, представленных в вопросниках, контрольных листах и др.).

При выполнении измерений базовые меры размера, времени и счета могут использоваться в различных комбинациях. Они служат основой для нормализации и обеспечивают возможность сопоставления метрик. Примеры стилей композиции мер представлены в таблице 3.1.

Таблица 3.1. **Стили композиции мер**

Стиль композиции мер	**Описание**	**Примеры**
Стиль нормализации по размеру		
(Время) / (Размер)	Подходит для представления временной эффективности или производственных усилий на единицу размера	• Время выполнения на инструкцию исходного кода • Время исправления на инструкцию исходного кода
(Количество) / (Размер)	Подходит для представления плотности	• Плотность ошибок • Тестовое покрытие

Стиль композиции мер	Описание	Примеры
(Размер) / (Размер)	Подходит для представления отношения размера определенного фрагмента к размеру целого	• Цикломатическое число модуля • Процентиль[2] объема модулей, предрасположенных к ошибкам, к общему объему
Стиль нормализации по времени		
(Размер) / (Время)	Подходит для представления трендов величин во времени	• Число строк разработанного кода в месяц
(Количество) / (Время)	Подходит для представления частоты в единицу времени	• Число транзакций в секунду • Число измененных инструкций исходного кода в месяц
(Время) / (Время)	Подходит для представления отношения времени наблюдаемых событий к общему времени наблюдения	• Отношение полезного времени работы ко времени функционирования системы
Стиль нормализации по количеству		
(Размер) / (Количество)	Подходит для представления сферы охвата (диапазона изменений)	• Среднее покрытие тестами выполняемых инструкций программы
(Время) / (Количество)	Подходит для представления интервала времени	• Среднее время между отказами • Среднее время исправления одной ошибки
(Количество) / (Количество)	Подходит для представления отношения числа определенных событий к их общему числу	• Число успешных попыток пользователя выполнить действие к общему числу попыток

3.1.3. Классификация метрик качества

Для удобства применения общих приемов измерений метрики обычно классифицируют как:

- *объективные/субъективные*. Объективные метрики включают подсчеты элементов, которые могут быть независимо проверены (число строк кода, число ошибок, сложность и др.). Они снижают влияние личного мнения на вычисление и анализ метрик. Субъективные метрики основываются на индивидуальном или коллективном понимании или предпочтении определенных характеристик или условий (уровень сложности проблем, стоимостные коэффициенты и др.);
- *примитивные/вычисляемые*. Примитивные (базовые) метрики можно непосредственно наблюдать (размер программы в KSLOC, число дефектов, найденных при тестировании и др.). Вычисляемые метрики не могут непосредственно наблюдаться, но могут вычисляться по примитивным метрикам (число дефектов, приходящихся на SLOC, трудоемкость и др.);
- *динамические/статические*. Динамическим метрикам свойственен компонент времени. Значения изменяются с течением времени, начиная с момента сбо-

[2] Процентиль – единица относительной градации на участке шкалы от 1 до 100.

ра данных (например, число ошибок в месяц). Статические метрики инвариантны ко времени (число обнаруженных дефектов, общая трудоемкость работ и др.);

- *предсказывающие/объясняющие*. Значения предсказывающих (прогнозирующих метрик) могут быть получены заранее (например, оцениваемая интенсивность отказов). Значения объясняющих метрик появляются пост-фактум (реальная интенсивность отказов).

По отношению к виду объекта измерения (работающая программа или совокупность документов) меры и соответствующие метрики подразделяются на внешние, внутренние и метрики использования ПС.

Внешние метрики используют меры работающего на компьютере программного продукта, полученные в результате измерения его поведения в ходе тестирования и функционирования [1].

Внешние метрики разрабатываются с целью:

- демонстрации качества программного продукта, представленного характеристиками и подхарактеристиками качества, на стадии тестирования и эксплуатации;
- использования для подтверждения (валидации) того, что программный продукт удовлетворяет внешним требованиям к качеству;
- предсказания реального эксплуатационного качества;
- определения степени, в которой программный продукт будет удовлетворять установленным и предполагаемым требованиям пользователей в ходе реальной эксплуатации.

Можно сказать, что совокупность внешних метрик предназначена для оценивания *внешнего качества* - степени, в которой продукт удовлетворяет установленным (заявленным) и подразумеваемым потребностям при использовании в определенных условиях.

Разработка внешних метрик основывается на выполнении следующих измерений:

- поведения программного продукта при тестировании и функционировании в сочетании с другими программными продуктами, аппаратным обеспечением или системой обработки информации в целом;
- поведения пользователя (сценариев использования ПС).

Под измерением (оценкой) поведения понимается оценка масштабов возможных последствий неадекватного поведения, которые угрожают жизни или здоровью людей, природным ресурсам, могут привести к разрушению данных, несогласованности или недостоверности информации, потере безопасности, деградации сервиса (услуг), экономическим потерям и др.

Примерами внешних метрик для такой характеристики качества как надежность, могут быть среднее время между отказами, число устраненных дефектов при тестировании, интенсивность отказов и др.

Внутренние метрики обеспечивают возможность пользователям, разработчикам, тестировщикам и менеджерам оценивать качество промежуточных и конечных продуктов ПС непосредственно по их свойствам, без выполнения на компьютере [2].

Внутренние метрики разрабатываются таким образом, чтобы они могли:

- представлять (отражать) качество не выполняющихся на компьютере промежуточных и конечных программных продуктов по тем характеристикам и

подхарактеристикам качества, которые определены в модели качества ПС (раздел 3.2);

- служить руководством к действию при планировании и улучшении процессов, которые воздействуют на промежуточные и конечные продукты;
- использоваться при верификации того, что промежуточные и конечные продукты удовлетворяют требованиям к внутреннему качеству ПС, предусмотренным планами совершенствования процессов;
- предсказывать внешние метрики качества.

Можно сказать, что совокупность внутренних метрик предназначена для оценивания *внутреннего качества* - множества атрибутов продукта, которое определяет его способность удовлетворять установленным или реальным потребностям при использовании в определенных условиях.

Разработка внутренних метрик основывается на выполнении измерений статических атрибутов, которые определены и могут быть оценены по тексту исходного кода, графическому или табличному представлению потоков управления и данных, структур перехода состояний или по документам ПС.

Примерами внутренних метрик для надежности могут быть число ошибок, найденных при инспекции кода, число устраненных дефектов в результате инспекции кода, прогнозируемое число оставшихся ошибок и др.

Метрики качества в использовании (метрики эксплуатационного качества) измеряют степень, в которой программный продукт, установленный и эксплуатируемый в определенной среде, удовлетворяет потребности пользователей в эффективном, продуктивном и безопасном решении задач [3].

Метрики качества в использовании помогают оценить не свойства самой ПС, а видимые результаты ее эксплуатации - эксплуатационное качество.

Очевидно, что для правильного измерения эксплуатационного качества важно учитывать *контекст* применения ПС – особенности категорий ее пользователей, специфику решаемых ими задач, а также физические и социальные факторы среды их работы.

Примерами эксплуатационных метрик качества могут быть точность и полнота достижения целей пользователей, производительность труда, ресурсы, потраченные в связи с достижением эффективного решения задач, мнение пользователей относительно свойств ПС и др.

Внутренние, внешние и эксплуатационные метрики качества взаимосвязаны. Достижение эксплуатационного качества зависит от удовлетворения критериев внешнего качества, основанных на внешних мерах и метриках качества, которые, в свою очередь, зависят от удовлетворения соответствующих критериев внутреннего качества, основанных на внутренних мерах и метриках, связанных с внешними.

Обычно требования пользователей к качеству специфицируются с помощью внешних метрик и эксплуатационных метрик качества, а внутренние метрики выбираются таким образом, чтобы они могли использоваться для предсказания значений внешних метрик.

Построить строгую теоретическую модель, устанавливающую взаимосвязь внешних и внутренних метрик, сложно, поэтому, как правило, строится гипотетическая модель, взаимосвязь метрик в которой моделируется статистически в ходе использования метрик.

3.2. Модели качества программных систем

3.2.1. Обобщенная модель качества

Модель качества программной системы представляет множество взаимосвязанных характеристик, образующих базис для спецификации требований к качеству и оценивания качества.

Как эталон при решении вопросов, связанных с качеством, характеризуемым внутренними и внешними атрибутами ПС, стандарт ISO/IEC 9126-1 предлагает двухуровневую иерархическую модель (рисунок 3.3) [4, 5].

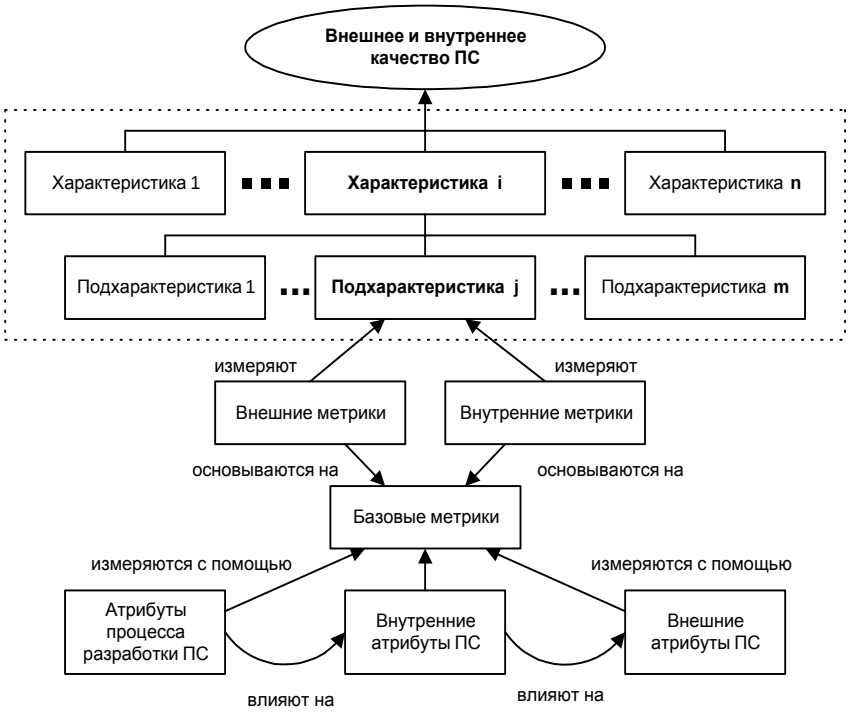


Рис.3.3. **Обобщенная модель качества**

Эта модель качества может использоваться для:

- идентификации требований к качеству ПС (внешнему и внутреннему);
- подтверждения полноты определения требований к качеству;
- идентификации целей проекта;
- идентификации целей тестирования;
- идентификации пользовательских критериев приемки завершенного программного продукта.

В соответствии с моделью качества, предложенной в стандарте ISO/IEC 9126-1, свойства (или атрибуты) ПС, характеризующие ее качество, укладываются в шесть категорий, каждая из которых ассоциируется с одной из характеристик качества, приведенных в таблице 3.2[3].

[3] Приводятся наименования характеристик, установленные стандартом ДСТУ 2850 [5], и их эквиваленты по стандарту ISO/IEC 9126-1 [4].

Таблица 3.2. **Характеристики качества ПС**

№	Наименование характеристики	Определение характеризуемых свойств ПС
1	**Функциональность** (functionality)	Свойства ПС, обусловливающие ее способность выполнять функции, соответствующие установленным и предполагаемым потребностям, при использовании в указанных условиях
2	**Надежность** (reliability)	Свойства ПС, обусловливающие ее способность сохранять уровень функционирования при работе в указанных условиях
3	**Удобство применения** (usability)	Свойства ПС, обусловливающие ее способность быть легко понимаемой, осваиваемой, удобной и привлекательной для пользователя при использовании в указанных условиях
4	**Эффективность** (efficiency)	Свойства ПС, обусловливающие ее способность обеспечивать рациональное использование выделенных ресурсов при работе в установленных условиях
5	**Сопровождаемость** (maintainability)	Свойства ПС, обусловливающие возможность ее эффективной модификации. Модификация может включать корректировку, усовершенствование или адаптацию ПС к изменениям среды, требований и функциональных спецификаций.
6	**Переносность** (portability)	Свойства ПС, обусловливающие ее способность быть переносимой из одной среды в другую

Характеристики качества ПС подразделяются на подхарактеристики. Их перечень представлен в таблице 3.3[4].

Таблица 3.3. **Подхарактеристики характеристик качества ПС**

№	Наименование подхарактеристики	Определение характеризуемых свойств ПС
1	**Подхарактеристики функциональности**	
1.1	**Функциональная пригодность** (suitability)	Свойства ПС, обусловливающие ее способность предоставлять надлежащее множество функций для решения специфицированных задач и достижения целей пользователя
1.2	**Точность** (accuracy)	Свойства ПС, обусловливающие ее способность обеспечивать правильные или согласованные результаты или воздействия с необходимой степенью точности
1.3	**Способность к взаимодействию** (interoperability)	Свойства ПС, обусловливающие ее способность взаимодействовать с одной или более специфицированными системами
1.4	**Защищенность** (security)	Свойства ПС, обусловливающие ее способность обеспечивать защиту информации от несанкционированного доступа лиц или систем на чтение и модификацию, и ее доступность для лиц и систем, обладающих правами доступа

[4] Данный перечень подхарактеристик соответствует стандарту ISO/IEC 9126-1 и отличается от приведенного в стандарте ДСТУ 2850, который гармонизирован с более ранней версией стандарта ISO/IEC 9126 (версией 1991 года).

№	Наименование подхарактеристики	Определение характеризуемых свойств ПС
2	**Подхарактеристики надежности**	
2.1	**Завершенность** (maturity)	Свойства ПС, обусловливающие ее способность избежать отказа из-за содержащихся в ней дефектов
2.2	**Отказоустойчивость** (fault tolerance)	Свойства ПС, обусловливающие ее способность поддерживать установленный уровень функционирования в условиях проявления дефектов в ПС, ошибок в данных или нарушений специфицированного интерфейса
2.3	**Восстановимость** (recoverability)	Свойства ПС, обусловливающие ее способность возобновлять функционирование на заданном уровне и восстанавливать поврежденные программы и данные
3	**Подхарактеристики удобства применения**	
3.1	**Понятность** (understandability)	Свойства ПС, обеспечивающие пользователю возможности понять, действительно ли она может удовлетворить его потребности, как она может использоваться для решения определенных задач и каковы условия ее использования
3.2	**Изучаемость** (learnability)	Свойства ПС, обеспечивающие пользователю возможность освоить приемы ее применения
3.3	**Управляемость** (operability)	Свойства ПС, обеспечивающие пользователю возможность управлять или контролировать ее действия
3.4	**Привлекательность** (attractiveness)	Свойства ПС, обеспечивающие ее притягательность для пользователя
4	**Подхарактеристики эффективности**	
4.1	**Временная эффективность** (time behaviour)	Свойства ПС, обусловливающие ее способность обеспечивать надлежащее время отклика (ответа) и обработки заданий, а также уровень пропускной способности при выполнении функций в установленных условиях применения
4.2	**Используемость ресурсов** (resource utilization)	Свойства ПС, обусловливающие ее способность использовать надлежащие ресурсы в нужные периоды времени при выполнении своих функций в установленных условиях применения
5	**Подхарактеристики сопровождаемости**	
5.1	**Анализируемость** (analyzability)	Свойства ПС, обусловливающие возможность диагностирования ее недостатков или причин отказов, а также идентификации частей, которые должны модифицироваться
5.2	**Изменяемость** (changeability)	Свойства ПС, обусловливающие возможность выполнения установленных видов модификации
5.3	**Стабильность** (stability)	Свойства ПС, обусловливающие ее способность минимизировать неожиданные эффекты модификаций
5.4	**Тестируемость** (testability)	Свойства ПС, обеспечивающие ее способность содействовать проверке модифицированного ПО
6	**Подхарактеристики переносности**	
6.1	**Адаптируемость** (adaptability)	Свойства ПС, обусловливающие ее способность адаптироваться для применения в различных специфицированных средах без использования действий или средств, отличных от тех, которые специально предназначены для этих целей

№	Наименование под характеристики	Определение характеризуемых свойств ПС
6.2	**Удобство установки** (installability)	Свойства ПС, обусловливающие ее способность к инсталляции в специфицированной среде
6.3	**Способность к сосуществованию** (co-existence)	Свойства ПС, обусловливающие ее способность сосуществовать с другими независимыми ПС в общей среде, разделяя общие ресурсы
6.4	**Замещающая способность** (replaceability)	Свойства ПС, обусловливающие ее способность использоваться вместо других специфицированных ПС в среде их применения

Кроме представленных в таблице подхарактеристик, *с каждой* из шести характеристик качества связана подхарактеристика «*Соответствие нормам и правилам* (compliance)». Это означает, что каждая из указанных характеристик должна отвечать требованиям соответствующих стандартов (по обеспечению надежности, сопровождению и др.) для определенного класса ПС.

В отличие от модели, применимой для описания внутренних и внешних атрибутов качества, для описания эксплуатационного качества используется одноуровневая модель, распределяющая атрибуты эксплуатационного качества по четырем характеристикам (таблица 3.4).

Таблица 3.4. **Характеристики эксплуатационного качества ПС**

№	Наименование характеристики	Определение характеристики
1	**Результативность** (effectiveness)	Степень, в которой пользователями достигаются заданные цели по точности и полноте решения задач в установленном контексте использования ПС
2	**Продуктивность** (productivity)	Степень, в которой расходуются ресурсы на достижение пользователями заданной эффективности решения задач в установленном контексте использования ПС
3	**Безопасность** (safety)	Уровень риска нанесения ущерба (материального и морального) - вреда здоровью людей, бизнесу, имуществу, окружающей среде при использовании ПС в установленном контексте
4	**Удовлетворенность** (satisfaction)	Степень, в которой пользователь удовлетворен ПС в определенном контексте ее использования. На степень удовлетворенности влияют отношение пользователя к свойствам ПС, представленным характеристиками качества для внешних и внутренних атрибутов ПС, а также его отношение к эффективности, продуктивности и безопасности

Измерения нужны на всех уровнях модели качества, поскольку удовлетворение критериев внутреннего качества обычно не достаточно для обеспечения гарантии достижения критериев внешнего качества, а удовлетворение критериев внешнего качества (используемых для измерения подхарактеристик) обычно не достаточно для обеспечения гарантии удовлетворения критериев эксплуатационного качества. Например, надежность может быть измерена и оценена внешне, путем наблюдения числа отказов в заданный период времени выполнения в ходе наблюдений за работой ПС, а также «измерена» внутренне, путем проведения инспекции

детальных проектных спецификаций и исходного кода с целью предсказания уровня устойчивости к аномалиям.

Ресурсы для оценивания качества должны распределяться между различными уровнями (видами) измерения в зависимости от целей оценивания качества, а также особенностей продукта и процесса разработки.

В представленной модели все множество атрибутов ПС, характеризующих ее качество, образует иерархическую структуру характеристик и подхарактеристик. Наивысший уровень этой структуры состоит из характеристик качества, а самый низкий - из атрибутов качества. Это не в полной мере иерархия, поскольку некоторые атрибуты могут вносить свой вклад в более чем одну характеристику, также как на одну характеристику качества может влиять больше чем один атрибут ПС.

Может быть прослежена также обратная связь в модели. С одной стороны, внешние подхарактеристики качества (такие как функциональная пригодность, точность, отказоустойчивость или временная эффективность) влияют на наблюдаемое качество в использовании. С другой стороны, недостаток качества в использовании (например, если пользователь не может закончить начатую задачу) может быть прослежен обратно к внешнему качеству (например, к функциональной пригодности или управляемости) и затем к соответствующим внутренним атрибутам, которые должны быть изменены для устранения установленного недостатка.

3.2.2. Метрики в обобщенной модели качества

Метрики в обобщенной модели классифицируются по подхарактеристикам внутреннего качества, внешнего качества и характеристикам эксплуатационного качества и описаны в частях 2, 3 и 4 стандарта ISO/IEC 9126, соответственно.

Описание метрик выполнено по единой схеме с указанием следующей информации:

- *имя метрики*. Соответствующие метрики внутреннего и внешнего качества имеют одинаковые имена;
- *назначение метрики*. Сформулировано в виде вопроса, на который дает ответ применение метрики;
- *метод применения*. Содержит правила получения данных и схему их применения в метрике;
- *формула и элементы данных*. Указывается формула вычисления (или формулы) и поясняются участвующие в ней элементы данных;
- *интерпретация измеренных данных*. Диапазон значений и предпочтительное значение;
- *тип шкалы метрики*. Один из типов шкалы - номинальная, порядковая, интервальная, относительная или абсолютная;
- *тип меры*. Один из типов меры – мера размера, времени, счетная мера;
- *исходные данные*. Источник данных, используемый в измерении;
- *процесс ЖЦ*. Указывается наименование процесса ЖЦ ПС, в котором рекомендовано применение метрики при проведении измерения;
- *получатель*. Указываются потребители результатов измерения.

Примеры внешних и внутренних метрик качества для некоторых характеристик представлены в приложении 2.

Уровень определения требований к эксплуатационному, внешнему и внутреннему качеству, множество и тип объектов измерения и измеряемых атрибутов,

выбор эксплуатационных, внешних и внутренних метрик, а также порядок применения процедур измерения и оценивания качества зависит от стадии ЖЦ ПС (рисунок 3.4).

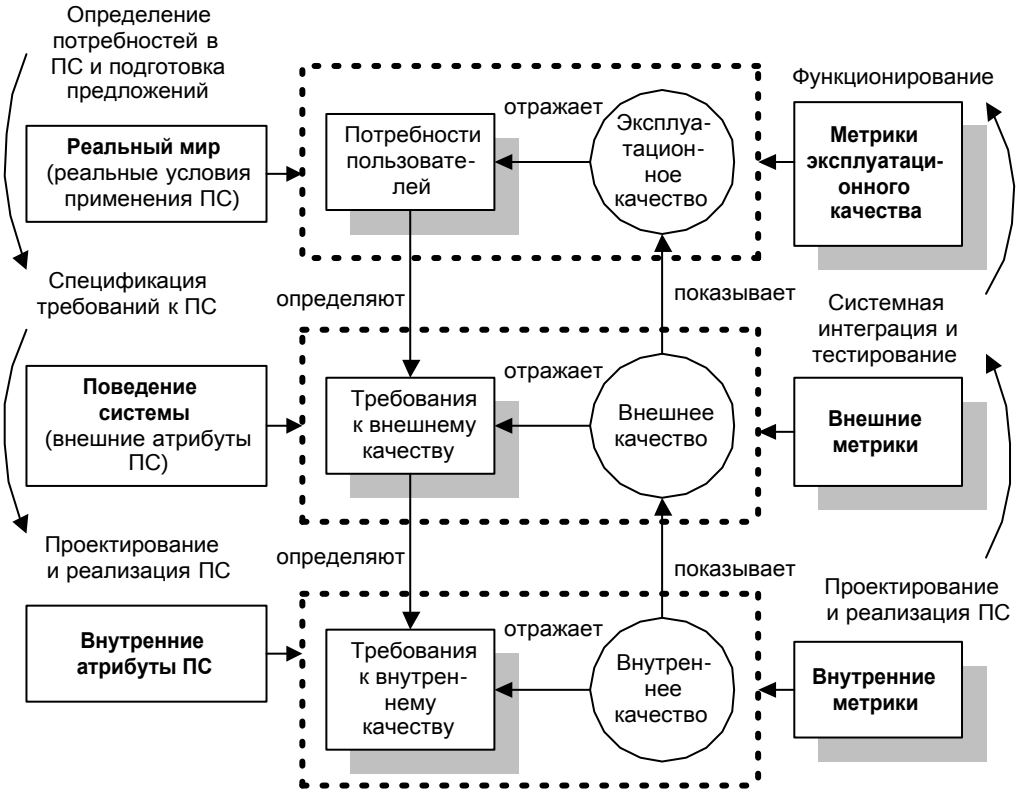


Рис.3.4. **Качество в ЖЦ ПС**

Цели использования внутренних и внешних метрик обобщенной модели на различных стадиях ЖЦ подытожены в таблице 3.5.

Метрики качества не исчерпываются указанным в данной версии стандарта множеством. Специализированные базовые метрики размера можно найти, например, в ISO/IEC 14143 [6], а метрики временной эффективности – в ISO/IEC 14756 [7]. Широкий спектр метрик представлен на сайте R.S.Pressman&Associates, inc. http://www.rspa.com/spi/metrics-process.html.

3.2.3. Другие иерархические модели качества программных систем

Существует множество моделей качества, большинство которых являются иерархическими по природе.

Одна из первых иерархических моделей качества ПС была предложена Мак-Коллом в 1977 году [8]. Наивысший уровень деления в модели отражает три аспекта качества ПС: функциональность, модифицируемость и способность к адаптации и взаимодействию с аппаратным обеспечением.

Эти три основные целевые характеристики качества далее подразделяются на 11 факторов качества и затем на 23 критерия качества.

Таблица 3.5. **Использование метрик на стадиях ЖЦ**

Использование внутренних метрик	Использование внешних метрик
Стадии подготовки предложений и определения требований к ПС	
Используются для: • спецификации плана повышения качества проекта и/или промежуточных продуктов • описания количественных подцелей внутреннего качества проекта и/или промежуточных продуктов ПС по характеристикам, указанным в модели. *Источники данных:* потребности пользователей, описания предложений, спецификации требований	*Используются для:* • спецификации требований к качеству с точки зрения пользователя • описания количественных целей внешнего качества по характеристикам, указанным в модели. *Источники данных:* потребности пользователей, описания предложений, спецификации требований
Стадии проектирования, кодирования и автономного тестирования ПС	
Используются для: • оценивания достигнутого уровня реализации плана повышения качества по характеристикам, указанным в модели • выявления отклонений от подцелей внутреннего качества • предсказания значений внешних метрик и внешнего качества *Источники данных:* результаты обзоров, статического анализа и измерений спецификаций, проекта, кода и др.	*Используются для:* • предсказания качества при использовании (эксплуатационного качества) *Источники данных:* результаты анализа сценариев использования ПС, исполняемых прототипов приложений в ходе их тестирования, отчеты о результатах тестирования
Стадии интеграции, системного тестирования, инсталляции, приемки ПС	
Используются для: • оценивания достигнутого уровня реализации плана повышения качества проекта и/или промежуточных продуктов *Источники данных:* результаты обзоров, статического анализа и измерений спецификаций, проекта, программного кода и др.	*Используются для:* • представления характеристик качества ПС, определенных в модели • подтверждения того, что достигнутое качество удовлетворяет пользователя • предсказания эксплуатационного качества *Источники данных:* сценарии использования ПС при тестировании и эксплуатации, а также совместно с другими программными системами
Стадия эксплуатации ПС	
Используются для: • исследования взаимосвязи внутренних и внешних метрик *Источники данных:* результаты обзоров, статического анализа и измерений спецификаций, проекта, программного кода и др.	*Используются для:* • представления характеристик эксплуатационного качества ПС, определенных в модели • подтверждения того, что имеющийся уровень качества все еще удовлетворяет реальные потребности пользователя *Источники данных:* сценарии использования ПС и сценарии ее поведения в ходе эксплуатации, отчеты о претензиях пользователей

Критерии качества представляют собой целевые числовые уровни факторов качества, оцениваемые по шкале от 0 до 10. Для оценивания факторов используются следующие метрики качества [9]:

- удобство проверки на соответствие стандартам,
- точность управления и вычислений,
- степень стандартности интерфейсов,
- функциональная полнота,
- однородность используемых правил проектирования и документации,
- степень стандартности форматов данных,
- устойчивость к ошибкам,
- эффективность работы,
- расширяемость,
- широта области потенциального использования,
- независимость от аппаратной платформы,
- полнота протоколирования ошибок и других событий,
- модульность,
- удобство работы,
- защищенность,
- самодокументированность,
- простота работы,
- независимость от программной платформы,
- возможность соотнесения проекта с требованиями,
- удобство обучения.

Отдельный критерий качества не обязательно связывается только с единственным фактором качества, следовательно, образуется скорее сетевая, чем строго иерархическая структура.

В 1978 году Боэм предложил иерархическую модель, подобную по природе модели МакКолла [10]. Начальные разделения по иерархии в двух моделях схожи, хотя проблема установления интерфейса и адаптируемости отнесена не к верхнему, а к промежуточному уровню модели. Нижний уровень содержит основополагающие критерии качества. В пределах этих уровней устанавливается в целом 19 критериев качества. Критерии в этой модели не независимы и их взаимодействие друг с другом часто вызывает конфликты.

К сожалению, адекватность этих моделей главным образом предполагается исходя из здравого смысла, а не доказывается на основе эмпирических оценок. Тем не менее, эти модели послужили базисом для дальнейшей работы по моделированию качества ПС.

В 1987 году Ваттсом [11] была предложена модель, развивающая модель МакКолла, а за ней в 1988 году появилась модель Дьюча и Виллиса [12]. Эта модель описывает 15 пользовательских и 27 технических атрибутов, которые могут связываться с факторами качества и критериями качества, соответственно.

Наконец, в 1991 году Международная организация по стандартизации (ISO) предложила концепцию качества ПС и иерархическую модель качества, содержащую 6 характеристик качества: функциональность, надежность, эффективность, удобство применения, сопровождаемость и переносимость (стандарт ISO 9126, 1991 год). Слабость этой модели состоит в том, что не все подхарактеристики оп-

ределены численно и таким образом вопрос их применения открыт для различных интерпретаций. Эти недостатки устранены в новой модели ISO/IEC 9126, которая обсуждалась выше (раздел 3.2).

Существуют трудности в применении иерархических моделей качества [13].

Во-первых, многие из предлагаемых целевых требований к качеству (характеристик качества) было бы естественнее включать в функциональные требования к ПС и не прибегать для их оценки к методам метрического анализа. Например, характеристика «способность к взаимодействию» может выступать в качестве целевого требования к интерфейсу ПС, а ее достижение - проверяться путем тестирования. Таким же образом можно поступить с характеристикой «переносимость» и рядом других.

Во-вторых, иерархические модели являются статическими по природе, то есть они не описывают, как проецировать метрики от их текущих значений на определенном этапе разработки к новым значениям на последующих стадиях проекта, что очень важно, например, для определения риска проекта.

Эти модели вообще не дают менеджерам проекта никаких указаний о том, как использовать метрики и атрибуты ПС для идентификации и классификации риска, управления процессом разработки и т.д.

Как альтернативу классическим иерархическим моделям, Джилб [14] предложил модель, которая хотя и базируется на иерархии характеристик и подхарактеристик качества, но имеет важные отличия:

- модель локально адаптируема (в масштабе организации);
- характеристики ресурсов включены в модель наряду с характеристиками качества;
- модель связана с эволюционной моделью процесса разработки.

В рамках этой модели Джилб выделил четыре характеристики качества и четыре характеристики ресурсов, хотя к ним могут быть сделаны дополнения в ходе процесса настройки модели. Предложенные характеристики качества таковы: работоспособность, готовность, адаптируемость и удобство использования, а характеристики ресурсов - время, деньги, люди и инструменты.

Еще одна известная модель качества - конструктивная модель качества (COQUAMO), которая была разработана Китченхем. Эта модель использует некоторые факторы качества из классических моделей Джилба, МакКолла и Боэма и затем помещает их в рамки соответствующих категорий конкретного проекта [15].

Отряд иерархических моделей продолжил пополняться и в последующие годы. Так в 1996 году Дромей предложил гибкую модель, которая позволяет в большей степени учесть особенности разрабатываемой ПС [16]. Для построения модели нужно:

1) выбрать множество атрибутов качества верхнего уровня, которые будут использовать для оценивания;

2) построить список всех компонентов или модулей системы;

3) идентифицировать определяющие (главные) свойства каждого компонента, оказывающие наибольшее влияние на свойства конечного продукта;

4) определить, каким образом каждое свойство отражается на атрибутах качества;

5) оценить правильность модели;

6) установить слабые места и устранить их (возвращаясь в цикле на один из предыдущих шагов 1 - 5).

Модель строится таким образом, чтобы улучшить понимание взаимосвязи между атрибутами (характеристиками) и суб-атрибутами (подхарактеристиками) качества и точнее определить свойства программного продукта, которые влияют на атрибуты качества.

В рамках программы ESPRIT разработан метод и инструмент Squid (Software quality in Development) для управления, оценки и предсказания качества продукта и процесса разработки ПС, который может настраиваться на нужды отдельной организации и базироваться на своей собственной базе данных [17]. Squid специфицирует модель качества в терминах характеристик качества, которые уточняются до тех пор, пока не станут непосредственно измеримыми (атрибутами). Как и в модели по стандарту ISO/IEC 9126-1, характеристики и атрибуты качества могут быть внутренними и внешними. Пользователю дается возможность определить, каким образом внутренние характеристики влияют на внешние, связывая их в модель данных.

Достоинство модели состоит в том, что она учитывает большинство факторов, влияющих на качество ПС (глава 1), помогая избежать конфликтов между различными требованиями к качеству, дает возможность указывать множество мер для функционально различающихся компонентов ПС (вычислительных, интерфейсных, информационных (баз данных)) и устанавливать различные правила выполнения измерений.

Своеобразную модель качества COQUALMO (COnstructive QUALity MOdel) недавно предложил Боэм, продолжая развитие модели COCOMO для определения трудоемкости и стоимости разработки ПС [18]. Эта модель ориентирована на достижение целей качества в сочетании с определением затрат на достижение этих целей. Модель позволяет определить «цену ошибки», своевременное устранение которой приводит к повышению качества.

В большинстве случаев при разработке моделей приоритетным, с точки зрения участников разработки проекта ПС, является взгляд менеджера проекта и его интересы в управлении риском проекта (по качеству, стоимости и продолжительности). Именно эта точка зрения учтена, например, при разработке модели в SATC (Software Assurance Technology Center) [13]. Предложенная модель поддерживает классификацию рисков и последующую полную оценку риска проекта. Она описывает, каким образом проецируются друг на друга метрики на разных стадиях проекта, что дает возможность определять риск различных атрибутов ПС, интересующих менеджера.

3.2.4. Не иерархические модели качества

Стандарт ISO/IEC 9126 включает большое количество мер, однако они не отражают причинно-следственных взаимосвязей, существующих между характеристиками, подхарактеристиками, измеряемыми атрибутами качества и множеством их прямых и косвенных мер.

Параллельно с разработкой иерархических моделей проводились исследования способов учета и представления каузальных взаимосвязей в проблематике качества ПС.

В 1999 году А.Абраном и Л.Буглионе была предложена ***модель QEST*** (Quality Factor + Economic, Social and Technical dimensions) [19]. Это трехмерная структура-оболочка, которая позволяет одновременно представлять зависимость значений параметров продукта, отражающих три аспекта его измерения (три точки зрения на продукт).

Три измерения модели образуют геометрическую фигуру - правильный четырехгранник (тетраэдр), стороны которого представляют нормализованные значения измеренных величин по каждому из измерений, а вершина - нормализованное оптимальное целевое значение для каждого измерения (рисунок 3.5).

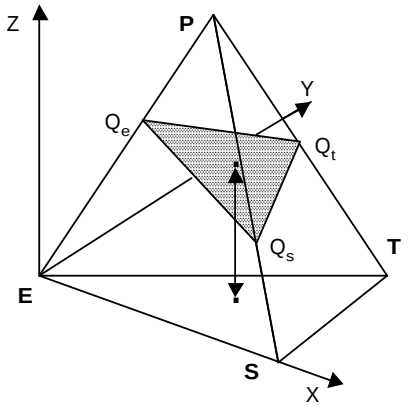


Рис. 3.5. **Модель QEST для оценивания качества ПС в ходе проекта**

Модель адаптирована авторами для совместного представления количественных и не количественных (качественных) мер разрабатываемого программного продукта, интегрированных в единый (интегральный) показатель его качества, полученный экспертным путем на основе опроса всех заинтересованных в нем лиц.

Три измерения модели отражают:

1. Экономический аспект (E). Точка зрения менеджеров на стоимость и график разработки (поставки) продукта.
2. Социальный аспект (S). Точка зрения пользователей на характеристики эксплуатационного качества (качества в использовании) продукта.
3. Технический аспект (T). Точка зрения разработчиков на обеспечение, измерение и оценивание достижения требований к качеству продукта.

Вершина (P) тетраэдра описывает максимальный уровень качества продукта (эффективности проекта).

Стороны правильного тетраэдра (пирамиды) равны единице. Все три измерения нормализованы на интервале {0,1}. Значения по каждому измерению представляют собой взвешенные суммы списка *n* значений нормализованных коэффициентов, отражающих каждую из трех точек зрения экспертов на продукт (номинальные оценки). Эти значения откладываются на сторонах пирамиды {Q_e, Q_s, Q_t}. Полученные точки соединяются линиями. Образуется сечение пирамиды, «отделяющий» лучшие и худшие оценки качества продукта. В ходе выполнения проекта формируются «реальные срезы» эффективности проекта.

Эффективность проекта измеряется с помощью любого из трех геометрических объектов:

- *расстояние от сечения до основания пирамиды*. Чем больше расстояние от текущего среда до основания, тем выше эффективность проекта;
- *площадь сечения*. Чем меньше площадь, тем выше эффективность;
- *объем нижней части пирамиды*. Чем больше объем под сечением, тем выше эффективность проекта.

В 2001 году А.Абраном и Н.Кецеси была предложена ***схема GDQA*** (Graphical Dynamic Quality Assessment) [20].

В этой схеме изначально определяется иерархическая модель качества ПС, включающая декомпозицию артефактов в категории процессов, продуктов и ресурсов проекта (вплоть до выделения измеримых свойств). Затем выполняется их приоритезация, «взвешивание» и нормализация для последующего сопоставления. Далее устанавливается взаимосвязь факторов качества, находящихся на разных уровнях, поскольку многие атрибуты качества нижнего уровня могут вносить вклад в несколько характеристик верхнего уровня иерархии. После этого производится отражение атрибутов на соответствующие меры и связанные с ними метрики качества (в данном подходе они называются функциями), а затем определяются входные переменные, значения которых можно установить, анализируя проектные документы, код, отчеты о тестировании и др.

На рисунке 3.6 показан пример применения схемы GDQA для подхарактеристики «завершенность» характеристики качества «надежность», а в таблице 3.6 описаны измеряемые величины и метрики (функции), представленные на рисунке. Кружки на схеме указывают, для каких метрик используются те или иные собираемые данные.

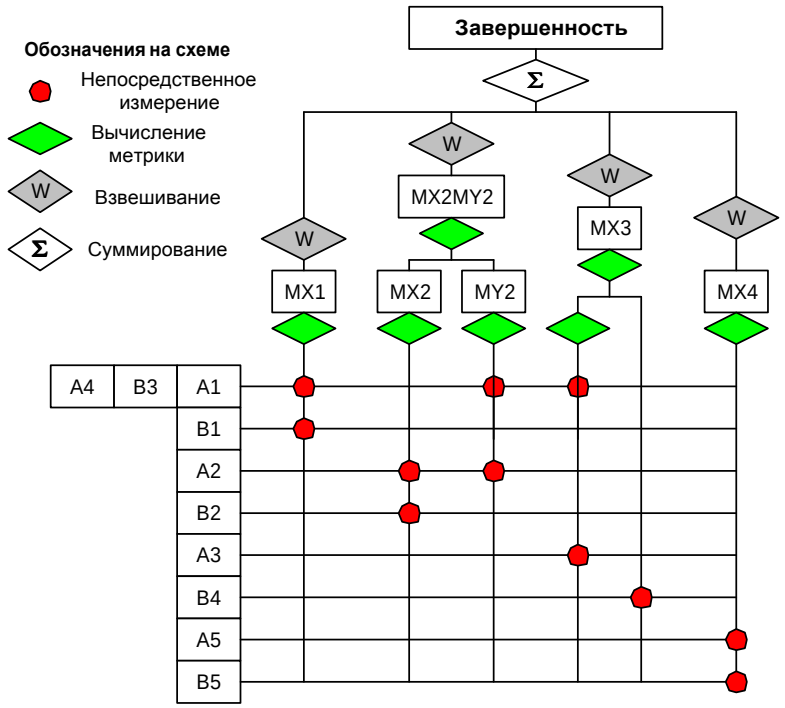


Рис. 3.6. **Схема GDQA для измерения «завершенности»**

Таблица 3.6. **Внутренние метрики «завершенности» ПС**

Название метрики	**Обозначение**	**Формула**	**Собираемые данные**	
			Имя	**Описание**
Интенсивность обнаружения ошибок при проверке	MX1	MX1 = A1/B1	A1	Число ошибок, обнаруженных при проверке
			B1	Ожидаемое число ошибок, которые будут обнаружены при проверке
Количество устраненных дефектов	MX2Y2	MX2Y2 = MX2/MY2 MX2 = A2/B2 MY2 = A2/B3	A2	Число откорректированных дефектов в проекте, коде…
			B2	Оцениваемое число дефектов, которые будут обнаружены в ПС (при инспекции, тестировании)
			B3	Число ошибок, обнаруженных при проверке
Плотность оставшихся дефектов	MX3	MX3 = (A3 - A4) / B4	A3	Ожидаемое число ошибок, которые будут обнаружены при проверке
			A4	Число ошибок, обнаруженных при проверке
			B4	Оцениваемый размер ПС
Адекватность тестирования	MX4	MX4 = A5/B4	A5	Число тестовых ситуаций в плане тестирования
			B4	Число тестовых ситуаций, которые должны быть предусмотрены для обеспечения адекватности тестового покрытия

3.2.5. Байесовский подход к моделированию качества

Общепризнанным «мерилом» качества ПС служит внутренняя метрика надежности – *количество обнаруженных дефектов* в рабочих продуктах ПС, а также внешние метрики:

- ожидаемое количество *скрытых дефектов* в ПС,
- реальное количество *выявленных дефектов* в ПС,
- количество *устраненных (откорректированных) дефектов*,
- *плотность дефектов*.

С этих позиций основные процессы ЖЦ ПС (*ПРо*) можно ассоциировать с процессами *внесения дефектов*, а процессы проверки (ПРп) – с процессами *выявления* и *устранения дефектов* в рабочих продуктах ПС (*РП*) (рисунок 3.7).

Для прогнозирования качества конечного программного продукта важно знать, какие факторы, и каким образом влияют на качество на *каждой стадии разработки* [21]. Однако существует большой уровень *неопределенности* относительно этого влияния. Потому на практике широко используется интуитивный подход и вероятностные рассуждения, основанные на собственном опыте менеджера проекта. Очевидно, что по мере накопления опыта возникает потребность коррекции суждений.

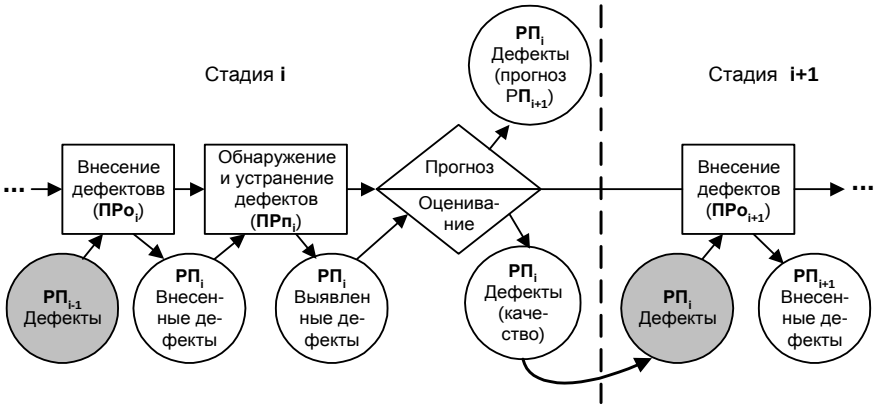


Рис. 3.7. **Распространение дефектов в рабочих продуктах ПС**

Средства для построения логически непротиворечивой схемы суждений с возможностью их пересмотра в свете новых данных предоставляет аппарат теории вероятности. Ключевую роль в нем, с позиций принятия решений в условиях неопределенности, играют *формула полной вероятности*, *правило распространения (умножение) вероятностей*, *формула Байеса* и, в целом, байесовский подход, который может быть положен в основу управления проектами [22].

Рассмотрим простой пример применения байесовского подхода в проблематике инженерии качества.

Пусть при разработке программных приложений (ПрП) в проекте возможны такие группы несовместимых событий:

A_1 – разработано ПрП *высокого качества*, A_2 – разработано ПрП *низкого качества*;

B_1 – разработку выполнил специалист *высокой квалификации*, B_2 – разработку выполнил специалист *низкой квалификации*.

Пусть известны априорные вероятности событий $P(B_1)=0.3$, $P(B_2)=0.7$ и условные вероятности $P(A_1|B_1)$ =0.6, $P(A_1|B_2)$ = 0.2 разработки ПрП высокого качества. Тогда по *формуле полной вероятности*

$$P(A_1) = \sum_{i=1}^{2} P(A_1 \mid \mathrm{B}_i) \cdot P(B_i) = 0.6 \cdot 0.3 + 0.2 \cdot 0.7 = 0.32 .$$

Таким образом, используя формулу полной вероятности, можно строить прогнозы достижения высокого качества ПрП в зависимости от разных факторов влияния.

В инженерии качества возможная и другая постановка вопроса, например, какой была квалификация разработчика, обусловившая фактически наблюдаемое высокое качество ПрП (с вероятностью $P(A_1)$=1)? Эта постановка вопроса типичная для задач *диагностики* причин по их видимым последствиям. Для выполнения диагностики выдвигаются и проверяются гипотезы.

Пусть E – наблюдаемое событие (свидетельство), которое заключается в том, что построенное ПрП имеет низкое качество с вероятностью $P(E)$=1. Для диагностики причин низкого качества ПрП целесообразно проверить две взаимно независимые гипотезы:

H_1 - разработчик имел высокую квалификацию

H_2 - разработчик имел низкую квалификацию.

Пусть известны априорные вероятности обеих гипотез $P(H_1) = 0.5$, $P(H_2) = 0.5$ и условные вероятности $P(E|H_1)=0.2$, $P(E|H_2)=0.6$. Тогда по *формуле Байеса*

$$P(H_1 \mid \mathrm{E}) = \frac{P(E \mid H_1) \cdot P(H_1)}{\sum_{i=1}^{2} P(E \mid H_i) \cdot P(H_i)} = \frac{0.2 \cdot 0.5}{0.2 \cdot 0.5 + 0.6 \cdot 0.5} = 0.25$$

$$P(H_2 \mid \mathrm{E}) = \frac{P(E \mid H_2) \cdot P(H_2)}{\sum_{i=1}^{2} P(E \mid H_i) \cdot P(H_i)} = \frac{0.6 \cdot 0.5}{0.2 \cdot 0.5 + 0.6 \cdot 0.5} = 0.75$$

Из расчетов видно, что с появлением очевидного свидетельства относительно события E доверие к гипотезе H_1 снижается, а к H_2 – повышается. То есть правомерно допустить, что разработка выполнена специалистом низкой квалификации и должны быть приняты соответствующие решения по управлению качеством.

Обобщенное выражение формулы Байеса на случай множества гипотез и множества свидетельств $E_1, E_2, .E_m$ имеет вид

$$P(H_i \mid E_1 E_2 ... E_{\mathrm{m}}) = \frac{P(E_1 E_2 ... E_{\mathrm{m}} \mid H_i) \cdot P(H_i)}{\sum_{k=1}^{n} P(E_1 E_2 ... E_{\mathrm{m}} \mid H_k) \cdot P(H_k)} \qquad \mathrm{i} = 1,.n.$$

Для вычисления знаменателя нужно знать условные вероятности всех возможных комбинаций свидетельств и гипотез, что делает формулу Байеса малопригодной для практического применения. Но, если можно допустить *условную независимость* свидетельств, формула Байеса упрощается.

События $E_1, E_2, .E_m$ является условно независимыми, если их совместная вероятность при условии определенной гипотезы H_i равняется произведению условных вероятностей этих событий при условии H_i, то есть

$$P(E_1 E_2 ... E_m \mid H_i) = \prod_{j=1}^{m} P(E_j \mid H_i) \cdot P(H_i).$$

Тогда в общем виде

$$P(H_i \mid E_1 E_2 ... E_{\mathrm{m}}) = \frac{\prod_{j=1}^{m} P(E_j \mid H_i) \cdot P(H_i)}{\sum_{k=1}^{n} \prod_{j=1}^{m} P(E_j \mid H_k) \cdot P(H_k)} \qquad \mathrm{i} = 1., n$$

В практических расчетах условной вероятности событий применяется итерационная процедура последовательного поэтапного распространения вероятностей, с суммированием отдельных свидетельств E_j и их влияния на условную вероятность $P(H_i|E_j)$ по мере поступления каждого свидетельства. Сначала устанавливается априорная вероятность события H_i и $P(E_j|H_i)$ для определенного свидетельства E_j. После вычисления апостериорной вероятности события H_i, то есть $P(H_i|E_j)$, она рассматривается как новая априорная вероятность $P(H_i) = P(H_i|E_j)$. Потом выбирается новое свидетельство E_j, вычисляется $P(E_j|H_i)$ и $P(H_i|E_j)$ и процесс повторяется.

Даже в предположении условной независимости событий в программном проекте, непосредственное применение формулы Байеса для решения практических

задач инженерии качества усложнено из-за очень больших объемов расчетов условных вероятностей. Лишь с появлением *байесовских сетей доверия* (английский эквивалент *Bayesian Belief Network*) байесовский подход к управлению качеством приобрел практический смысл.

Аппарат байесовских сетей постепенно находит все более широкое использование: в диагностике, прогнозировании, распознавании образов, поисковых и экспертных системах разного назначения [23, 24].

3.2.6. Графические модели качества. Байесовские сети

В основе теории байесовских сетей лежит формула Байеса и *правило сети*, которое является обобщением правила умножения вероятности и средством вычисления совместного распределения вероятностей случайных событий.

Пусть A_1, A_2, $.A_n$ – случайные величины (СВ), ассоциируемые с наступлением случайных событий. Тогда по правилу сети

$$P(A_1 A_2 ... A_n) = P(A_1 \mid A_2 ... A_n) \cdot P(A_2 \mid A_3 ... A_n) \cdot ... \cdot P(A_{n-1} \mid A_n) \cdot P(A_n).$$

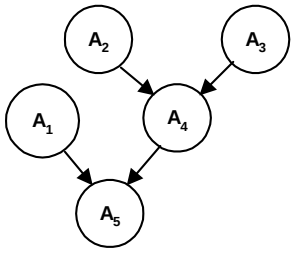
Байесовская сеть - это ориентированный ациклический граф, каждая вершина которого представляет собой событие, описанное случайной величиной (переменной) A_i, которая может находиться в нескольких состояниях (принимать несколько значений).

С каждой вершиной априори связываются параметры, соответствующие закону распределения случайной величины. Дуга между любыми двумя вершинами A_i и A_j $(i \neq j)$ устанавливает причинно-следственную связь "A_i *вызывает* A_j". В сети справедливо отношение *условной независимости вершин*: вершина-потомок A_j не зависит от вершин-предков, которые находятся выше вершины -родителя (то есть A_i). Это позволяет упростить правило сети.

Пусть $a_1., a_n$ - множество *всех возможных конфигураций* значений множества СВ $A_1,A_2.,A_n$ в вершинах сети, а запись *Предки*(A_i) означает множество предков вершины A_i. Тогда совместное распределение вероятностей для $A_1,A_2.,A_n$ на множестве всех конфигураций

$$P(a_1, a_2 .., a_n) = \prod_{i=1}^{n} P(a_i \mid Предки(a_i))$$

На рисунке 3.8 показан пример байесовской сети с пятью вершинами A_1, A_2, A_3, A_4, A_5, переменные в которых могут принимать лишь по два значения – *T* и *F*.



$P(A_4=T \mid A_2=T, A_3=T) = 0.4,\ P(A_4=F \mid A_2=T, A_3=T) = 0.6$

$P(A_4=T \mid A_2=T, A_3=F) = 0.3,\ P(A_4=F \mid A_2=T, A_3=F) = 0.7$

$P(A_4=T \mid A_2=F, A_3=T) = 0.6,\ P(A_4=F \mid A_2=F, A_3=T) = 0.4$

$P(A_4=T \mid A_2=F, A_3=F) = 0.5,\ P(A_4=F \mid A_2=F, A_3=F) = 0.5$

$P(A_5=T \mid A_1=T, A_4=T) = 0.7,\ P(A_5=F \mid A_1=T, A_4=T) = 0.3$

$P(A_1=T) = 0.8,\ P(A_1=F) = 0.2$ $\quad P(A_5=T \mid A_1=T, A_4=F) = 0.3,\ P(A_5=F \mid A_1=T, A_4=F) = 0.7$

$P(A_2=T) = 0.6,\ P(A_2=F) = 0.4$ $\quad P(A_5=T \mid A_1=F, A_4=T) = 0.8,\ P(A_5=F \mid A_1=F, A_4=T) = 0.2$

$P(A_3=T) = 0.7,\ P(A_3=F) = 0.3$ $\quad P(A_5=T \mid A_1=F, A_4=F) = 0.3,\ P(A_5=F \mid A_1=F, A_4=F) = 0.7$

Рис. 3.8. **Пример сети с априорными вероятностями значений переменных**

Тогда по правилу сети и с учетом условной независимости A_5 от A_2 и A_3 имеем $P(A_5A_4A_3A_2A_1) = P(A_5 \mid A_4A_1) \cdot P(A_4 \mid A_2A_3) \cdot P(A_1) \cdot P(A_2) \cdot P(A_3)$.

Если значение $A_1=A_2=A_3=T$ (с вероятностью $P(A_i)=1$, $i=1.,3$), а значение A_4 не известно, тогда вероятность того, что $A_5=T$, равна

$$P(A_5 = T \mid A_4 = T, A_3 = T, A_2 = T, A_1 = T) =$$
$$= P(A_5 = T \mid A_4 = T, A_1 = T) \cdot P(A_4 = T \mid A_2 = T, A_3 = T) +$$
$$+ P(A_5 = T \mid A_4 = F, A_1 = T) \cdot P(A_4 = F \mid A_2 = T, A_3 = T) \cdot = 0.7 \cdot 0.4 + 0.3 \cdot 0.6 = 0.46$$

Если известно, что и значение $A_4=T$, тогда вероятность того, что $A_5=T$,

$$P(A_5 = T \mid A_4 = T, A_3 = T, A_2 = T, A_1 = T) = P(A_5 = T \mid A_4 = T, A_1 = T) = 0.7$$

Байесовская сеть предоставляет полное совместное распределение вероятности всех возможных событий, вычисляя априорную вероятность каждой комбинации значений всех переменных, которые представляют эти события. Когда значение какой-то переменной становится известным (очевидным фактом), первичное совместное распределение вероятности пересчитывается с учетом этого факта, наблюдается переходный процесс замены априорных вероятностей значений каждой переменной апостериорными, то есть пересматриваются вероятности событий, моделируемых байесовской сетью.

Таким образом, посредством байесовской сети можно формулировать предположение о существовании зависимости между разными переменными, а затем последовательно “распространять” получаемые объективные данные наблюдений по сети.

Параметры каждой вершины сети должны быть представлены *таблицей вероятностей вершины (TBB)* (для дискретных СВ), *функцией распределения вероятности* (для непрерывных СВ) или *детерминированной функцией* от значений СВ в вершинах-родителях.

Пусть, например, сеть имеет три дискретные вершины (рисунок 3.9).

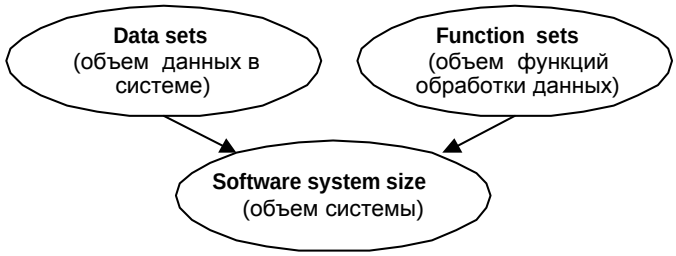


Рис. 3.9. **Байесовская сеть с тремя вершинами**

Дискретные значения СВ в вершинах определены по порядковой шкале: “*High*” (большой), “*Medium*” (средний), “*Low*” (малый) (таблица 3.7 – для вершин – родителей, таблица 3.8 – для дочерней вершины).

Таблица 3.7. **Таблицы вероятностей вершин-родителей**

Data sets		**Function sets**	
Значение	**Вероятность**	**Значение**	**Вероятность**
High	0,3	High	0,3
Medium	0,5	Medium	0,3
Low	0,2	Low	0,4

Таблица 3.8. **Таблица вероятностей дочерней вершины**

Вершины-Родители		**Software system size**		
Function sets	**Data sets**	High	Medium	Low
High	High	0,6	0,3	0,1
	Medium	0,5	0,3	0,2
	Low	0,4	0,4	0,2
Medium	High	0,4	0,4	0,2
	Medium	0,3	0,4	0,3
	Low	0,3	0,3	0,4
Low	High	0,2	0,3	0,5
	Medium	0,2	0,2	0,6
	Low	0,1	0,2	0,7

Выбор типа вершины определяется, прежде всего, тем, качественные или количественные характеристики соответствующего фактора качества нужно описать в вершине. Кроме того, во внимание принимаются типы всех вершин, являющихся непосредственными предками или потомками данной вершины.

Пусть, например, нужно определить типы вершин в сети, моделирующей количество выявленных дефектов во время проверки (рисунок 3.10).

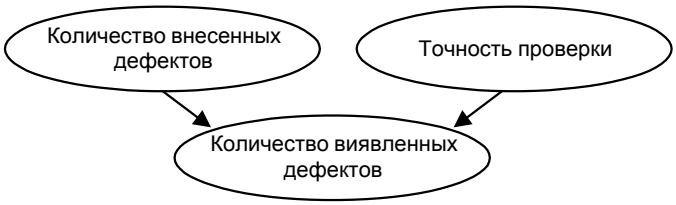


Рис.3.10. **Фрагмент байесовской сети**

Примеры вариантов выбора шкал значений показаны в таблице 3.9.

Таблица 3.9. **Варианты шкал значений переменных в вершинах сети**

Название вершины	**Шкала**	**Значение**	**Объяснение**
Вариант 1			
Количество внесенных дефектов	Порядковая	{“очень мало”, “мало”, “среднее количество”, “много”, “очень много”} Безусловные вероятности значений	Заполнение таблиц вероятности вершин выполняется вручную.
Точность проверки	То же	{“высокая”, “средняя”, “низкая” } Безусловные вероятности значений	
Количество выявленных дефектов	То же	{“очень мало”, “мало”, “среднее количество”, “много”, “очень много”} Условные вероятности значений	
Вариант 2			
Количество внесенных дефектов	Интервальная	(0-5] (5-10] (10-20] (20-50] (50-100]	Бета-функция распределения вероятности
Точность проверки	То же	(0-1] (1-2] (2-3] (3-4]	
Количество виявленных дефектов	То же	*Beta* ((5 - 1) / 4 * *точность проверки* + (5 - (5 - 1)) (1 - 5 / 4) * *точность проверки* + (1 - (1 - 5)), 0, *внесенные дефекты*)	

Вариант 3			
Количество внесенных дефектов	Относительная	0, 1, 2, 3, 4, 5	Биномиальное распределение вероятности
Точность проверки	Интервальная	(0–1] (1-2] (2-3] (3-4] (4-5]	
Количество выявленных дефектов	Относительная	*Binomial* (*внесенные дефекты*, *точность _проверки* / 5)	

Очевидно, что «глубина» и разветвленность построенных сетей зависит, во-первых, от системы знаний человека-строителя о предмете исследования, во-вторых, от степени полезности углубленной детализации сети для исследования предмета, и, в третьих, от наличия объективных (фактических) данных о причинно-следственных связях величин в вершинах сети.

Основные преимущества использования графических байесовских моделей качества – поддержка *прогнозирования* дефектов и *диагностики* наиболее вероятных причин (источников) их возникновения.

Одна из базовых графических моделей прогнозирования дефектов показана на рисунке 3.11. Это модель самого верхнего уровня и может быть детализирована в исходных вершинах (без родителей). Модель дефектов является результатом определения множества факторов качества, анализа причинно-следственных связей между ними, комбинирования качественных (экспертных) и количественных оценок их влияния на плотность дефектов, а также учета ограничений выбранного *доступного* инструмента моделирования (подробнее об инструментах построения байесовских сетей – в главе 5).

Рис. 3.11. **Модель прогнозирования плотности дефектов в ПрП**

Семантическое описание вершин байесовской сети представлено в таблице 3.10.

Модель позволяет в начале текущего этапа проекта прогнозировать (с определенной степенью уверенности), какой будет плотность дефектов в ПрП, если не изменятся условия его разработки.

Таблица 3.10. **Описание вершин байесовской сети моделирования дефектов**

Название вершины	Описание вершин сети и зависимостей переменных
Остаток дефектов	Разница между плотностью внесенных дефектов в текущую версию ПрП и плотностью устраненных (откорректированных) дефектов
Внесенные дефекты	Сумма плотности новых внесенных дефектов и плотности остатка дефектов (от предыдущего этапа)
Внесенные дефекты (новые)	Плотность внесенных новых дефектов. Зависит от сложности решаемых задач в ПрО и совершенства (качества) процесса разработки
Остаток старых дефектов	Плотность дефектов, которые, согласно прогнозу, остались не выявленными в ПрП на прошлом этапе разработки (остаток минус фактически устраненные дефекты)
Устраненные дефекты	Плотность устраненных дефектов. Зависит от плотности выявленных дефектов в ПрП, а также точности их устранения (нормируемой на [0,1]). Биномиальный закон распределения значений переменной в вершине
Выявленные дефекты	Плотность выявленных дефектов. Зависит от плотности внесенных дефектов и качества проверки (нормируемой на [0,1]). Биномиальный закон распределения значений
Точность коррекции	Способность разработчика точно устранить дефект при коррекции. Чем больше значение, тем выше способность
Качество проверки	Способность верификатора найти дефекты в ПрП. Чем больше значение, тем выше способность
Качество разработки	Способность разработчиков предотвратить внесение дефектов при разработке. Чем больше значение, тем выше способность
Сложность проблемы	Ассоциируется с риском проекта в категории «сложность реализации требований» к ПрП. Чем выше оценка риска, тем больше сложность проблемы

Аппарат байесовских сетей позволяет строить достаточно сложные модели (сотни вершин), которые не могут быть реализованы другими методами.

Хотя байесовские сети известны давно, широкое практическое применение они начали находить лишь недавно, с появлением эффективных алгоритмов распространения вероятности по сети, а также инструментальных средств построения байесовских сетей (глава 5).

3.3. Построение метрик и моделей качества

3.3.1. Проектирование метрик качества

Для создания модели качества ПС необходимо:

1. Определить цели измерения и выполнять все последующие действия с учетом факторов, влияющих на качество.
2. Идентифицировать компоненты ПС, качество которых должно моделироваться отдельно.
3. Выделить и классифицировать наиболее существенные, несущие ответственность за качество свойства (атрибуты) для каждого компонента.
4. Сформулировать аксиомы для связывания свойств ПС с характеристиками качества.
5. Определить систему внутренних и внешних метрик качества ПС, адекватную перечню характеристик качества и атрибутов ПС.

6. Проверить модель качества.

Процесс, выполняемый при разработке собственных или выборе существующих метрик качества ПС, отвечающих целевым требованиям к качеству ПС (или ее компонента), включает следующие шаги [25]:

Шаг 1. Определение понятий. Определения всех используемых понятий (сущностей и измеряемых атрибутов) должны соответствовать указанным в применяемых стандартах или приводиться в документе, описывающем принятую модель качества ПС. Нельзя допускать неоднозначности толкования терминов различными категориями лиц, использующих метрики (пользователями, заказчиками, разработчиками).

Шаг 2. Определение внутренней структуры (модели) каждой метрики. Значения базовых метрик измеряются непосредственно и модель таких метрик - это наименование соответствующей переменной. Значения сложных метрик представляют собой математические модели, использующие базовые или другие сложные метрики. Предпочтителен прагматический подход к моделированию метрик - метрики должны отражать наиболее важные аспекты измеряемого атрибута и не быть слишком сложными. Модели метрик могут быть заимствованы из стандартов, научной литературы, из опыта других организаций и т.д. и должны накапливаться и при необходимости адаптироваться к нуждам конкретных измерений. Для разработки новых (оригинальных) метрик полезно привлекать экспертов в проблемной области проекта и инженерии качества ПС.

Шаг 3. Формулирование метода вычисления метрики (критерия оценивания). Модель каждой метрики декомпозируется до уровня метрик-примитивов (базовых метрик) и далее для этих примитивов определяется механизм получения значения (критерий оценивания). Пример метрики-примитива со сложным механизмом получения значения – SLOC (число строк исходного кода).

Метрики-примитивы и критерии их оценивания образуют первый уровень необходимых собираемых данных.

Методы определения значений метрик, рекомендованные стандартом ДСТУ 2850, таковы [5]:

- *измерительный*. Метод основан на получении информации о свойствах и характеристиках ПС с использованием измерительных технических и программных средств (размер загрузочного модуля, время выполнения ветви программы и др.);
- *регистрационный*. Метод основан на получении информации во время испытаний или при непосредственном использовании ПС по назначению, когда регистрируются и подсчитываются определенные события (моменты времени сбоев, число отказов и др.);
- *расчетный*. Метод основан на использовании теоретических и эмпирических зависимостей на ранних стадиях разработки, а также статистических данных, накапливаемых при испытаниях, эксплуатации и сопровождении. Этим методом прогнозируются характеристики и подхарактеристики на ранних стадиях разработки (точность, устойчивость, надежность и др.). Расчетный метод используется и для определения фактических значений характеристик по результатам тестирования;
- *экспертный*. Метод заключается в определении значений характеристик группой специалистов-экспертов. Применяется в том случае, если задача определения значений не может быть решена иными методами. Для проведения экспертизы

устанавливаются контролируемые признаки, включаемые в виде вопросов в аналитические опросные анкеты экспертов.

Шаг 4. Определение критерия «хорошего» значения метрики. Когда известно, *что* измерять и *как* измерять, нужно определить, как интерпретировать результаты измерений, и какими должны быть пределы их изменений.

Критерием истины во многих случаях служит мнение заказчика и пользователя. Могут также использоваться опубликованные сведения о приемлемых значениях тех или иных метрик, полученные в отраслевой производственной практике значения и др. Так, например, известно, что цикломатическая сложность модулей должна быть ≤ 10. Наиболее достоверный источник информации - собственные исторические данные об измерении схожих проектов.

Шаг 5. Документирование метрик. На этом шаге нужно принять решение о формате отчета, о цикле извлечения и регистрации данных (частоте измерений), о механизмах учета данных (твердая копия, электронная копия), о порядке их распределения (направления) различным участникам процесса измерения, об ограничениях (определяющих готовность метрики к использованию) и др.

Шаг 6. Определение дополнительных квалификаторов метрик. Метрика считается хорошо спроектированной, если она является обобщением различных взглядов и уровней измерения и если ее можно применять, например, на уровне класса продуктов, одного продукта или компонентов этого продукта ПС. В качестве дополнительных квалификаторов может указываться контекст применения и другие ограничения.

3.3.2. Подготовка к использованию метрик качества в измерениях

Процесс подготовки к использованию метрик качества ПС для выполнения измерений включает выполнение следующих шагов, рекомендованных стандартом ISO/IEC 9126-2:

Шаг 1. Идентификация собираемых и измеряемых элементов данных. При подготовке к измерению данных нужно:

- выполнить их декомпозицию на элементы - количественные описания фактов, используемых для вычисления значений метрик;
- определить время, частоту и продолжительность сбора значений (мер) соответствующих элементов данных;
- разработать формы учета и отчета о результатах оценивания, обзора и/или тестирования продуктов.

Шаг 2. Идентификация ограничений и контекста использования метрик. Прежде всего, необходимо установить *ограничения* на использование метрик при тестировании.

Большинство внешних метрик используют измеряемые значения, полученные в ходе тестирования. Поэтому на их достоверность влияет тщательность (глубина) тестирования. По аналогии, на достоверность значений внутренних метрик влияет тщательность (глубина) применения инспекций и других методов верификации. Нужно устанавливать степень этого влияния.

Эффективность проверок (тестирования или верификации) может определяться, например, в таких единицах, как количество «человеко-часов обзоров», потраченных на проверку определенного числа строк исполнительного кода, или «количество выполненных тестов». А уровни их значений, в свою очередь, могут уста-

навливаться исходя из того, как много пользователей и как часто будут использовать те или иные компоненты (функции) ПС.

Нужно также устанавливать ограничения на степень соответствия полученных результатов проверок спецификациям исходных требований, поскольку сами спецификации могут быть не всегда правильными. Поэтому перед планированием тестирования на соответствие спецификациям необходимо выполнять их верификацию.

Для правильного применения метрик помимо ограничений необходимо также установить *контексты* их использования. Без указания контекста, описывающего ситуацию и условия, в которых будут собираться данные, и применяться метрики, меры могут интерпретироваться не адекватно. Например, такие метрики, как «время на освоение» или «время на выполнение операции» существенно зависят от опыта оператора. Поэтому, наряду с указанием значения метрики нужно указывать условия (контекст), в которых значение получено.

Могут быть полезны и некоторые другие примеры информации, сбор которой важен для правильной интерпретации метрик:

- виды процессов, действий и задач, в ходе выполнения которых собираются данные;
- время, продолжительность сбора данных (применения метрик);
- персонал, участвующий в сборе данных (применении метрик);
- среда тестирования ПС и ее отличия от среды эксплуатации;
- условия и виды тестирования;
- пользовательские профили (категория, опыт и др.);
- сведения о видах процедур сбора данных (автоматизированные, ручные, вопросники и др.);
- уровень проверки правильности данных, что важно для оценки степени точности данных (самопроверка, инспекция и др.).

Шаг 3. Идентификация свойств метрик. Точность и корректность оценивания качества зависит от свойств, которыми обладает каждая метрика, участвующая в оценивании. Свойства метрик соответствуют требованиям к измерениям, предъявляемым стандартом ISO/IEC 14598-1 [26], и требованиям к оцениванию, предъявляемым ISO/IEC 14598-5 [27]. Наличие необходимых свойств должно проверяться.

Стандарт ISO/IEC 9126-2 определяет следующие свойства метрик:

- *надежность* (степень свободы от случайных ошибок оценщика);
- *показательность* (наглядность);
- *готовность к использованию* (документальные свидетельства возможности использовать метрику);
- *экономичность* (документальные свидетельства оправданности затрат на применение метрики);
- *корректность* (объективность, беспристрастность, точность отражения мнения оценщика);
- *значимость результатов* (предоставление значимых результатов измерения поведения ПС или характеристик ее качества).

Шаг 4. Идентификация метода визуализации и интерпретации. Существует множество методов и инструментов, которые могут быть использованы для

визуализации и анализа результатов измерений и должны быть идентифицированы при подготовке к измерениям, например:

- гистограммы и диаграммы Парето (для определения частоты появления событий);
- графики «время – значение» (для предсказания изменений (тренда));
- графики «метрика X - метрика Y» (для выполнения корреляционного анализа);

Подробнее эти и другие методы описаны в главе 5.

Шаг 5. Идентификация метода демонстрации правильности метрик. Нужно идентифицировать методы, которые будут применяться для демонстрации правильности выбранных метрик, например:

демонстрация коррелируемости. Объяснение отклонения значений характеристик отклонением значений соответствующих метрик;

демонстрация трассируемости. Подтверждение того, что изменение значений характеристик качества сопровождается изменением значений метрик (в одном и том же направлении);

демонстрация согласованности. Подтверждение того, что если значения характеристик качества продуктов 1, 2, …, n связаны отношением Q1>Q2>…>Qn, то и значения соответствующих метрик связаны отношением M(Q1)>M(Q2)> …>M(Qn);

демонстрация предсказуемости. Подтверждение того, что погрешность предсказания характеристики качества с помощью метрики будет допустимой;

демонстрация дифференцирующих способностей. Подтверждение того, что применение метрики позволит отличать компоненты ПС с приемлемыми значениями характеристик качества от компонентов с неприемлемыми значениями характеристик.

3.4. Применение метрик и моделей качества

3.4.1. Спецификация требований к качеству

Как уже отмечалось, для идентификации согласованных требований к качеству важно выбрать подходящие внутренние и внешние метрики качества и определить ожидаемые диапазоны значений (адекватные критерии оценки качества), что позволит детализировать характеристики и подхарактеристики качества, требуемые моделью качества ПС. Выбираемые внутренние и внешние метрики должны быть попарно связанными, что позволит определить модель дальнейшего улучшения качества в ходе разработки ПС. Каждая внутренняя метрика должна соответствовать улучшаемому атрибуту ПС, а внешняя - определять степень достижения улучшений.

Для построения модели и определения требований к качеству можно использовать целеориентированный подход, который создает концептуальный базис для определения множества целей и их преобразования в перечень конкретных вопросов, которые применяются для спецификации мер и метрик, пригодных для получения информации, необходимой для ответа на вопросы. Подробно данный подход рассматривается в главе 4. Можно привести следующий пример его применения для определения целей качества:

Цели (Goals)

***G1*:** Повысить надежность программного продукта

Вопросы (Questions):

1. Внешние вопросы, касающиеся того, как проверить достижение цели в ходе тестирования или эксплуатации.

Q1.1: как долго продукт работает без отказа?

Q1.2: как часто обнаруживаются ошибочные входные данные и предотвращается отказ?

2. Внутренние вопросы, касающиеся того, как проверить достижимость цели качества в ходе разработки.

Q1.3: как много ошибок обнаруживается при проведении инспекции?

Q1.4: какие типы ошибок наиболее часты?

Метрики (Metrics):

1. *Внешние метрики* (External Metrics).

EM1: среднее время между отказами.

EM2: коэффициент обнаружения ошибок ввода.

2. *Внутренние метрики* (Internal Metrics).

IM1: плотность дефектов, обнаруженных при инспекции.

IM2: число входов, для которых реализована функция обнаружения ошибочных данных.

Множество требований к качеству ПС могут вступать в конфликты, которые должны разрешаться на ранних стадиях разработки. Для выявления и устранения такого рода конфликтов Дьютч и Виллис предложили использовать «матрицу конфликтов» [12]. Фрагмент такой матрицы, используемой для обозначения конфликтующих характеристик и подхарактеристик качества, представлен на рисунке 3.12.

Характеристики, на которые оказывается влияние / Характеристики, по которым определяется рост качества		Функциональность				Надежность			Удобство применения				Эффективность	
		Функциональная пригодность	Точность	Способность к взаимодействию	Защищенность	Завершенность	Отказоустойчивость	Восстанавливаемость	Понятность	Изучаемость	Управляемость	Привлекательность	Временная эффективность	Используемость ресурсов
Функциональность	Функциональная пригодность								+		+			
	Точность		+											
	Способность к взаимодействию					-							-	-
	Защищенность				-					-	-		-	
Надежность	Завершенность										+		-	-
	Отказоустойчивость					-					+		-	-
	Восстанавливаемость				-									-

Рис. 3.12. **Фрагмент матрицы конфликтов характеристик качества**

Положительное воздействие одной характеристики на другую отмечается в матрице символом «+», а отрицательное - символом «-».

Одно из ключевых свойств матрицы конфликтов - необязательная симметрия взаимосвязей конфликтующих характеристик. Например, если в качестве приоритетного требования к качеству пользователем выбрана надежность, - ее обеспечение может нанести урон рациональности (эффективности по времени и ресурсам), хотя обратное отрицательное влияние (рациональности на надежность) может отсутствовать.

Может быть и другое представление матрицы конфликтов требований. Например, в методологии Quality Function Deployment (QFD) («развертывание» функции качества) предлагается идентифицировать и разрешать конфликты с использованием треугольной матрицы, символизирующей «крышу» так называемого «дома качества». При этом предполагается, что конфликтующие взаимосвязи полностью симметричны. Подробнее методология QFD представлена в разделе 3.5.

3.4.2. Другие аспекты использования метрик

Оценивание и предсказание характеристик ПС на ранних стадиях ЖЦ – два наиболее полезных аспекта использования метрик[5].

Предсказания могут выполняться с использованием байесовских сетей (как это было показано в п.3.3.6), *регрессионного анализа* или *корреляционного анализа*.

Если нужно предсказать значение в будущем определенной характеристики (атрибута), используя текущее значение этой характеристики (атрибута), - применяется регрессионный анализ множества данных, наблюдаемых в определенный период времени. Так можно, например, по значениям среднего времени между отказами ПС при тестировании определить среднее время между ее отказами при эксплуатации.

Если нужно определять будущие значения характеристики (атрибута), используя текущие измеренные значения другого атрибута, т. е. определять внешнюю меру по внутренней мере, непосредственно не связанной с данной внешней мерой, - строятся корреляционные зависимости и выполняется корреляционный анализ. Например, можно использовать значения сложности модулей, полученные при кодировании, для предсказания трудоемкости внесения изменений и тестирования при сопровождении.

Если нужно оценить текущие значения атрибута, которые не могут быть непосредственно измерены, или есть какая-либо другая мера, имеющая строгую корреляционную зависимость с целевой мерой, - также полезен корреляционный анализ. Например, поскольку количество оставшихся ошибок в программном продукте невозможно измерить, - его можно *оценить*, зная количество обнаруженных ошибок и тенденции в процессе обнаружения.

Существуют и другие аспекты полезного применения метрик:

- *оценка качества на промежуточных этапах разработки ПС*. Для управления качеством полезно использовать метрики *промежуточных продуктов* в определенные моменты выполнения процессов разработки и сопровождения ПС, а также оценки эффективности *процессов ЖЦ ПС*, особенно если требования к каче-

[5] Процедура *оценки* качества разработанной программной системы рассматривается в главе 12.

ству продуктов этих процессов заранее установлены путем указания ожидаемых измеряемых значений;

- *трассирование эффективности проекта ПС*. Для управления проектом ПС полезно применять метрики, предназначенные для оценки ключевых характеристик промежуточных продуктов проекта в определенные моменты разработки. В этих случаях меры могут интерпретироваться с точки зрения управления для принятия адекватных решений;
- *идентификация, наблюдение и предупреждение риска*. Это особенно важно для критических программных систем, характеристики и подхарактеристики качества, а также атрибуты которых, идентифицируются как факторы риска. В этих случаях меры могут интерпретироваться с точки зрения заказчика или конечного пользователя;
- *оценивание характеристик ПС при ее приобретении*. Полезно применять метрики для оценивания продуктов ПС при их приобретении в случае наличия альтернатив. Это должны быть метрики, отражающие взгляд пользователя на модель качества при приемочных испытаниях ПС – кандидатов;
- *идентификация расхождений между требуемым и достигнутым уровнем качества*. С одной стороны, можно идентифицировать расхождения между внутренними требованиями к качеству и качеством, достигнутым *до начала тестирования*, а с другой, - между внешними требованиями к качеству и качеством, достигнутым *в процессе тестирования*. До начала тестирования внутренние метрики служат в качестве индикаторов внешнего качества, а затем, в ходе тестирования, внешние метрики применяются для представления реально достигнутого уровня качества;
- *анализ проблем с качеством*. Метрики могут использоваться для отождествления источника наиболее вероятных проблем с каждой из подхарактеристик качества. Периодическое выполнение такого рода анализа в процессе разработки позволяет контролировать факт появления проблем, частоту их появления и принятие решений по проблемам;
- *идентификация компонентов ПС, предрасположенных к появлению проблем*. Метрики можно использовать отдельно для каждого компонента ПС, выполняя идентификацию «проблемных» компонентов путем построения распределений значений метрик и корреляционных зависимостей, анализ которых позволяет установить отклонения значений от ожидаемых, выход за границы диапазонов или превышение пороговых значений.

Для извлечения максимальной пользы от применения анализа метрик ПС должна быть создана система сбора и хранения данных о качестве продуктов на уровне организации – репозиторий наблюдаемых и вычисляемых результатов внешних и внутренних измерений. Это поможет накапливать опыт и повышать точность анализа метрик.

3.5. Парадигма «встраивания» качества в программной инженерии

3.5.1. Базовые идеи в основе модели QFD

В 60-годы в Японии была предложена методология обеспечения качества при создании промышленной продукции, получившая название методологии «развер-

тывания» качества (QFD, от Quality Function Deployment). Позже она была адаптирована к производству программной продукции.

Методология QFD ориентирована на последовательную трансляцию пользовательских требований к создаваемому продукту в технические требования на всех стадиях его разработки и производства.

Основная цель QFD состоит в преодолении трех основных недостатков, свойственных традиционным подходам к разработке: *пренебрежение мнением* (голосом) заказчика, *потеря информации* и *несогласованность требований* ввиду их реализации различными исполнителями с использованием различных методов. Фактически QFD изменила парадигму производства качественной промышленной продукции: традиционная идея повышения качества за счет контроля создаваемой продукции была заменена идеей производства бездефектной продукции за счет встраивания элементов обеспечения качества в процессы производства.

QFD, используя серию матриц, декомпозирует спецификации продукта или описания проблем до уровня определения предписаний конкретных действий. Эти предписания определяют минимальный уровень усилий, которые необходимо потратить для удовлетворения требований заказчика.

QFD кладет в основу бригадный подход к работе и помогает производственным бригадам систематически достигать согласия по вопросам: «что делать?», «каким образом лучше делать?», «в каком порядке делать?», «какие ресурсы использовать?». QFD предлагает также четкую технологию проведения совместных обсуждений и решения проблем всеми заинтересованными сторонами.

3.5.2. Компоненты модели QFD. Матрица «Дом Качества»

Самая распространенная матрица, используемая QFD, - это «Дом Качества» (рисунок 3.13).

Символический Дом Качества содержит следующие компоненты:

- *целевое утверждение*. Первый шаг процесса QFD состоит в достижении консенсуса относительно общей цели потребителя (заказчика) продукта и его поставщика (разработчика). Целевое утверждение обычно представляется в форме вопроса и определяет, чего нужно попытаться достичь. Хорошо сформулированная цель позволяет бригаде (команде проекта) сосредоточиться на пользовательских требованиях, представленных в виде единой задачи;
- *голос заказчика (потребителя)*. После того, как сформулировано пробное целевое утверждение, может быть проведено совещание для проверки полноты учета (охвата) мнения заказчика. Мнение заказчика выражается в виде перечня конкретных требований с указанием желательных атрибутов и достоинств будущего продукта. Эти требования определяют все те блоки «ЧТО», которые должны быть созданы. QFD предлагает метод учета Голоса Заказчика путем идентификации индивидуальных характеристик продукта, услуги или проблемы. В совещании должны принимать участие группа QFD, представители заказчика и руководства поставщика, а также представители заинтересованных подразделений (например, службы маркетинга). Каждый блок «ЧТО» должен представлять единственное требование. Формулировка требований должна ограничиваться пятью - десятью словами. После того, как все требования заказчика будут учтены, они уточняются, сортируются и вносятся в список;

Рис. 3.13. **Матрица «Дом Качества»**

- *рейтинги важности*. QFD предлагает систематизированный метод определения степени важности требований заказчика. Рейтинги важности служат и как факторы взвешивания, и как множители к другим значениям в матрице, позволяя делать определенные статистические заключения. Шкалы рейтингов важности и методы опроса участников голосования могут быть различными, использующими числовые значения или символьные обозначения (для удобства участников голосования из разных стран), но во всех случаях, чем выше рейтинговое значение, тем больше уровень важности;
- *конкурентная оценка заказчика*. На этом шаге процесса QFD оценивается степень восприятия продукта заказчиком по сравнению с другими заказчиками-конкурентами. В качестве базы для сравнения используются данные, полученные от различных заказчиков и представленные в графическом виде. Это дает возможность оценить, насколько хорошо конкуренты воспринимают требования, выдвинутые заказчиком, и помогает:

- убедиться в том, что список требований важен для сообщества пользователей;

- учесть дополнительные требования заказчиков;
- установить достоинства и недостатки продукта;
- определить слабые места продуктов, используемых конкурентами.

Для оценки используется та же шкала, что и при ранжировании важности;

- *голос поставщика (разработчика)*. На этом шаге процесса QFD рассматриваются все вопросы, касающиеся блока «КАК», то есть выясняются способы достижения требований, образующих блок «ЧТО». В блоке «КАК» указываются необходимые процессы, средства и методы. Акцент переносится с идентификации проблемы на ее решение. Процесс поиска решений представляет собой совещание с использованием приемов мозгового штурма;
- *адресные (локальные) цели*. После того как составлен перечень возможных решений в блоке «КАК», необходимо установить наилучшее из них. Адресные цели используются для того, чтобы помочь определить, квантифицируемы ли предложения в блоке «КАК», то есть, можно ли их оценить количественно. Адресные цели указывают, каким образом предложенное решение может способствовать повышению или снижению чего-либо. Для представления адресных целей используют символьные обозначения (например, стрелки «вверх» или «вниз») либо числовые значения. Если цель не может быть оценена количественно, она должна быть удалена из списка;
- *корреляционная матрица*. Эта матрица символизирует собой крышу Дома Качества (рисунок 3.14). Она показывает положительные и отрицательные связи между решениями в блоке «КАК».

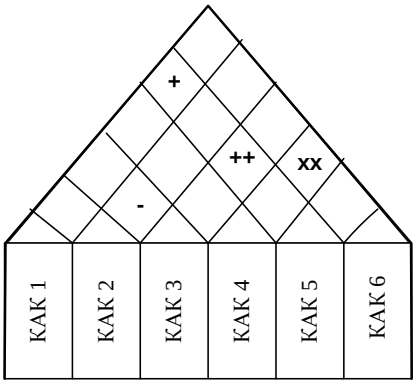


Рис. 3.14. **Проверка совместимости принятых решений**

Определяя, какие решения сочетаются друг с другом и где могут возникнуть конфликты, корреляционная матрица помогает идентифицировать, какие ресурсы могут использоваться сразу для нескольких целей. Она указывает также, куда необходимо направить дополнительные усилия или применительно к чему провести дополнительные исследования.

Корреляционная матрица использует следующую символику:

« ++ » - для указания очень хорошего взаимодействия;

« + » - для указания положительной связи;

« - » - для указания отрицательной связи;

« xx » - для указания очень плохого взаимодействия (конфликта).

Корреляционная матрица строится после того, как все решения «КАК» внесены в матрицу QFD и идентифицированы адресные цели. Проводится попарное сравнение решений, и присваиваются указанные выше обозначения.

Положительная связь указывает, что решения дополняют друг друга, а отрицательная - что решения отрицательно сказываются друг на друге и необходимо разрешение конфликта;

- *конкурентная оценка технических решений*. Эта оценка подобна конкурентной оценке заказчика, но касается технических деталей продукта или услуг, а не требований заказчика. Данные для оценки предоставляются инженерным и техническим персоналом. В результате оценки определяются целевые значения - количественные меры для каждого решения из блока «КАК» (на основе отраслевых, ведомственных и других стандартов). При выборе базы для оценок сравниваются решения конкурентов-производителей продуктов и определяется тот уровень технических решений в продукте, который обеспечит его конкурентоспособность на рынке;
- *вероятностные коэффициенты*. Служат для взвешивания каждого решения по вероятности его успеха. Низкий коэффициент вероятности может указывать на то, что оцениваемое техническое решение не будет конкурентоспособным. Это значит, что должны быть применены или разработаны новые технологии, методы и приемы. Для оценивания обычно используются числовые шкалы. Чем выше значение по шкале - тем выше вероятность правильности выбранного решения;
- *матрица взаимосвязи*. Это центральная матрица в Доме Качества. Используется для анализа того, насколько каждое решение из блока «КАК» удовлетворяет каждое требование из блока «ЧТО». Если устанавливается связь между элементами «КАК» и «ЧТО» - это означает, что решение удовлетворяет определенному требованию или устраняет определенную проблему. Если на вопрос, «может ли решение помочь в достижении требования?», оценщики получают ответ «нет» - в соответствующей позиции матрицы проставляется «0». Если ответ – «да», связь далее должна быть категоризирована как «сильная», «средняя» или «слабая» с указанием численных значений. Обработка выполняется, начиная с левого столбца матрицы. Переход к следующему столбцу «КАК» производится только после завершения обработки данного столбца. Для вычисления оценки в каждой ячейке матрицы рейтинг важности умножается на число, указанное в ячейке для соответствующего требования;
- *абсолютная и относительная оценка*. Абсолютная оценка представляет собой сумму взвешенных значений связи для каждого решения из блока «КАК». Относительная оценка - это ранг. Абсолютные оценки вычисляются путем суммирования оценок в матрице связей по каждому столбцу решений из блока «КАК» и отражают важность каждого решения по отношению к требованиям.

3.5.3. Методология QFD для программных систем

В последнее десятилетие методология QFD, изначально разработанная для промышленного производства, была перенесена в среду разработки программного обеспечения систем и получила название SQFD (Software Quality Function Deployment) – «развертывание» качества в программном обеспечении. Сейчас она с успехом применяется фирмами DEC, AT&T, Hewlett-Packard, IBM и Texas Instruments при разработке программных систем различных классов.

SQFD концентрирует внимание на повышении качества процесса разработки программных систем путем реализации технологий повышения качества в ходе определения исходных требований в ЖЦ разработки.

SQFD адаптируется к любой существующей методологии программной инженерии, применяемой для выявления и количественного обоснования критических требований заказчика. Ее применение приводит к повышению производительности труда аналитиков и программистов, сокращению количества изменений, вносимых в проект, снижению числа ошибок, распространяющихся с одной стадии ЖЦ на другую, и т.д. Создаваемые на ее основе системы качества требуют меньшего сопровождения и позволяют экономить на этом денежные средства.

SQFD адаптирует к условиям разработки ПС наиболее часто используемую матрицу QFD - Дом Качества. Базовая модель SQFD представлена на рисунке 3.15.

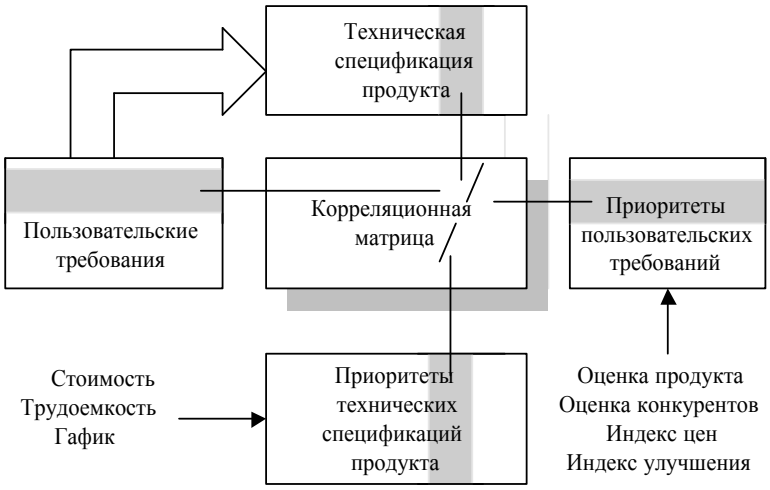


Рис. 3.15. **Базовая модель SQFD**

В этой модели присутствуют такие компоненты:

- *требования пользователя*. Извлекаемые при анализе проблемной области пользовательские требования (к которым относятся требования конечных пользователей, менеджеров, разработчиков системы и др.) фиксируются слева по оси ***y***. Представляют собой простые утверждения в терминологии пользователя. Сопровождаются подробными определениями, образующими словарь SQFD;
- *технические спецификации продукта*. Пользовательские требования преобразуются в технические, измеримые спецификации требований к программному продукту и фиксируются над осью ***x***. Одному пользовательскому требованию может соответствовать несколько технических спецификаций;
- *корреляционная матрица*. Пользователям предлагается идентифицировать уровень коррелируемости между различными пользовательскими требованиями и техническими спецификациями продукта (сильная, средняя, слабая корреляция) и заполнить корреляционную матрицу. При этом должны быть учтены мнения всех категорий пользователей;
- *приоритеты требований пользователя*. На основе пользовательских данных выстраиваются приоритеты установленных требований и перечисляются

справа по оси ***y***. В ходе определения приоритетов может использоваться дополнительная информация о конкурирующих продуктах;

- *приоритеты технических спецификаций продукта*. Приоритеты технических спецификаций определяются путем суммирования произведений приоритетов пользовательских требований и корреляционных значений и фиксируются под осью ***x***. Веса полученной строки приоритетов технических спецификаций затем обычно преобразуются в процентное представление суммарных весов строки приоритетов. Могут использоваться дополнительные данные об адресных (целевых) мерах технических спецификаций с учетом оценок стоимости, сложности и графика работ;
- *конечный продукт*. Конечный продукт процесса SQFD будет, как минимум, включать измеримые технические спецификации ПС, указатели их важности в процентном соотношении и адресные (частные) меры.

Методология SQFD (как и QFD) имеет как достоинства, так и недостатки.

Методология переносит акцент в разработке на начальные стадии, что позволяет сосредоточиться на тщательном планировании и снизить изменяемость требований. Она обеспечивает «встраивание» качества в процесс проектирования, что приводит к предупреждению распространения дефектов на более поздние стадии ЖЦ, а это, в свою очередь, сокращает продолжительность и стоимость проекта, повышает производительность труда и качество конечного продукта.

В числе недостатков SQFD - существенная стоимость и сложность методологии. Для успешного применения она требует полной поддержки со стороны руководства и автоматизации процессов принятия решений.

Такие фирмы, как DEC, AT&T, Hewlett-Packard, IBM разрабатывают свои собственные модели SQFD и стратегии сбора и обработки данных.

Более подробная информация о методологии SQFD, ее достоинствах и недостатках, а также примерах применения содержится в [28, 29, 30]. Для широкого применения модель SQFD нуждается в стандартизации.

Литература к главе 3

1. *ISO/IEC 9126-2:2003.* Software Engineering - Product Quality - Part 2: External Metrics.
2. *ISO/IEC 9126-3:2003.* Software Engineering - Product Quality - Part 3:Internal Metrics.
3. *ISO/IEC 9126-4:2004.* Software Engineering – Software Product Quality - Part 4: Quality In Use Metrics.
4. *ISO/IEC 9126-1:2001.* Software Engineering - Product quality - Part 1: Quality model.
5. *ДСТУ 2850-94.* Програмні засоби ЕОМ. Показники і методи оцінювання якості.
6. *ISO/IEC 14143-1:1998.* Information technologies - Software measurement - Functional size measurement - Part 1: Definition of concepts.
7. *ISO/IEC 14756:1999.* Information technologies – Measurement and rating of performance of computer-based software systems.
8. *McCall J.A.*, Richards P.K., Walters G.F. Concepts and definitions of software quality // Factors in Software Quality, NTIS. - v. 1. - nov. 1977.
9. *В. В. Кулямин*, О. Л. Петренко. Место тестирования среди методов оценки качества ПО. – http:/software-testing.ru/lib/ispras/testing_in_various_models.htm
10. *Boehm B.W.*, J. R. Brown, H. Kaspar, M. Lipow, G. MacLeod, and M. J. Merritt. Characteristics of Software Quality // TRW series of Software Technology. North Holland. Amsterdam. – 1978.

11. *Watts R.* Measuring Software Quality // Manchester: NCC Publications. – 1987.
12. *Abel D.E., Rout T.P.* Determining and Specifying the Quality Attributes of Software Products // Austral. Comput. J. - V. 25. - N. 3. - aug. - 1993. - P.105-112.
13. *Hyatt L.*, Rosenberg L. A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality // 8th Annual Soft. Technology Conf., Utah, april. – 1996.
14. *Gilb T.* Principles of Software Engineering Management, Addison-Wesley.- 1988. – 442 p.
15. *Musson T.* The evaluation of Instructional Software Quality // Software Quality Management III, v.1: Managing Quality Systems, Ed. M.Ross. - Computational Mechanics Publications. - 1994. - P. 311 – 321.
16. *R.Geoff Dromey*. A model for software product quality. // IEEE Trans. On Softw. Eng. – V. 21. – N.2. – febr. – 1995. – P. 146-162.
17. *Boegh J.* et al. A Method for Software Quality Planning, Control and Evaluation // IEEE Software. - march/april. - 1999. - P. 69 – 77.
18. *Devani-Chulani S.* Incorporating Bayessian Analysis to Improve the Accuracy of COCOMO II and its Quality Model Extension // Univ. of South. Calif., Los Angeles, CA.- 1997.
19. *Buglione L., Abran A.*, Geometrical and statistical Foundations of a Three-dimensional Model of Performance // International Journal of Advanced in Engineering Software. - 30. - 1999.
20. *Kececi N. Abran A.* Analysing. Measuring and assessing software quality in a logic based graphical model // Qualita2001 (4th International Congress on Quality and Reliability), Annecy, France, March 22-23, 2001.
21. *Коваль Г.І.* Байєсівські мережі як засіб оцінювання та прогнозування якості програмного забезпечення // Проблеми програмування. - 2005. - № 2. -С. 15 – 23
22. *У. Моррис*. Наука об управлении. Байесовский подход. – М.: Мир.- 1971. –304 с.
23. *Капитонова Ю.В.*, Мищенко Н.М., Фелижанко О.Д., Щеголева Н.Н. Об использовании байесовских сетей для мониторинга пользователей компьютеров // Кибернетика и системный анализ. –2004. -№ 6. –С. 3-14.
24. *Веревка О.В*., Кузьмина К.И., Семик Т.М. Байесовское прогнозирование эффективности деятельности оператора по социопсихофизиологическим характеристикам структуры личности // Проблеми програмування. – 2005. - № 1. – С. 69 - 84.
25. *Westfall L.L.* Software Metrics that Meet Your Information Needs // Software Measurement Services, Phano. – 1996.
26. *ISO/IEC 14598-1:1999.* Information technologies - Software product evaluation - Part 1: General overview.
27. *ISO/IEC 14598-5:1998.* Information technologies - Software product evaluation - Part 5: Process for evaluators.
28. *Haag S.*, Raja M.K., Schkade L.L. Quality Function Deployment. Usage in Software Development // Comm. of the ACM. – 1996.- V. 39. - N.1.
29. *Ouyang S.* et al. Quality Function Deployment // Department of Computer Science Univ. of Calgary, Canada. – 1997. - http://sern.ucalgary.ca/courses/seng/613/F97/ grp2/report.htm
30. *Krogstie J.* Using Quality Function Deployment in Software Requirements Specification. – http://www.ifi.uib.no/konf/refsq99/papers/krogstie.pdf

Глава 4. ИЗМЕРЕНИЯ В ПРОГРАММНОЙ ИНЖЕНЕРИИ

4.1. Измерение как процесс жизненного цикла

4.1.1. Цели, задачи и объекты измерения

Процесс измерения входит в группу организационных процессов ЖЦ и непосредственно связан с процессом верхнего уровня – процессом управления. Он может выполняться «в интересах», прежде всего процессов управления проектом, качеством, риском и процесса совершенствования. Ответственность за определение, внедрение, сопровождение, оценивание и улучшение процесса измерения возлагается на менеджеров (руководителей) уровня организации или проекта.

Перспективы успеха в выполнении и совершенствовании всех видов деятельности в процессах ЖЦ существенно расширяются, когда решения принимаются на основе анализа фактической, количественной информации. Эта информация может быть получена только с помощью наблюдения и измерения процессов, продуктов и ресурсов, вовлеченных в процессы. Для того чтобы измерение было экономически оправданным, оно должно быть хорошо спроектировано и нацелено на поддержку деловых целей организации.

Существует четыре причины, обусловливающие измерение процессов, продуктов и ресурсов в ЖЦ – это необходимость их охарактеризовать, оценить, предсказать или совершенствовать.

Измерения характеристик процессов, продуктов и ресурсов обеспечивает, прежде всего, понимание их сути и создание базиса для выполнения сравнительного анализа.

Измерение с целью последующего оценивания помогает определять состояние выполнения планов, а меры служат датчиками, используемыми для контроля. Кроме того, оценивание важно для определения возможности достижения целей качества и влияния усовершенствований технологии и процесса на продукты и процессы.

Измерение с целью предсказания направлено на достижение понимания взаимосвязей между процессами и продуктами и построение моделей этих взаимосвязей, что дает возможность использовать значения наблюдаемых атрибутов для предсказания значений других (не наблюдаемых) атрибутов. Это делается для утверждения достижимых целей по стоимости, плану-графику и качеству и рационального распределения ресурсов. Предсказывающие меры служат также основой для определения тенденций в проектах и процессах, что позволяет обновлять оценки стоимости, времени и качества исходя из текущих объективных данных. Проекции и оценки, основанные на исторических данных, также помогают анализировать риски и делать выбор альтернативных вариантов проекта.

Измерения с целью совершенствования предполагают сбор количественной информации, которая помогает распознавать проблемы и препятствия на пути улучшения качества продукции и выполнения процесса, а также планировать и контролировать затраты усилий на осуществление усовершенствований. Измерения текущего состояния процесса создают сравнительный базис, благодаря которому можно судить о том, выполняются ли действия по усовершенствованию надлежащим образом, и какими могут быть побочные эффекты усовершенствований.

Удачно выбранные меры помогают также информировать всех участников процессов о целях измерений и пояснять необходимость совершенствования.

По определению Н. Фентона «измерение – это процесс, в ходе которого атрибутам сущностей реального мира присваиваются числовые или символьные значения, что позволяет охарактеризовать атрибуты с помощью ясно определенных правил» [1]. Таким образом, измерение предполагает наличие сущностей (объектов интереса), атрибутов (характеристик) сущностей и правил для назначения значений атрибутам.

Есть несколько видов *сущностей*, представляющих интерес с позиции измерения, например:

- продукты
- процессы
- ресурсы
- артефакты
- действия
- агенты
- организации
- среды
- ограничения

Атрибуты определяют свойства сущностей. Искусство измерения состоит в принятии решений о том, какие атрибуты необходимо использовать, чтобы дать правильное представление о соответствующих сущностях.

В области программной инженерии предложено ряд таксономий для классификации сущностей. Н. Фентон, например, связывает понятие сущности с продуктом, процессом или ресурсом [2], а Д. Армитеж и другие используют схему классификации, основанную на артефактах, действиях и агентах [3]. Обе схемы имеют как достоинства, так и недостатки, и обе оказываются слабо пригодными, когда речь идет об элементах среды.

В таблице 4.1 показаны некоторые примеры сущностей наряду с атрибутами и мерами, которые могли бы использоваться для определения количественных значений атрибутов сущности.

Таблица 4.1. **Примеры измеряемых атрибутов сущностей**

Класс сущностей	**Атрибуты**	**Возможные меры**
Меры продукта		
Система	Размер	• Число модулей • Число петель в диаграмме потоков данных • Число условных единиц функциональности (FP) • Число физических строк исходного кода (SLOC) • Количество памяти в байтах или словах (требуемых или выделенных)
	Плотность дефектов	• Число дефектов в SLOC • Число дефектов в FP
Компонент	Длина	• Число физических строк исходного кода • Число логических исходных операторов
	Объем повторного использования	Отношение числа неизмененных физических строк, исключая комментарии и пустые строки, к общему числу строк
Модуль	Сложность	Число линейно независимых путевых потоков
Документ	Длина	Число страниц
Строка кода	Тип оператора	Имена типа
	Язык программирования	Название языка

Класс сущностей	Атрибуты	Возможные меры
Дефект	Тип	Имена типа
	Место расположения	Название действия, при котором внесен
	Серьезность	Упорядоченное множество классов серьезности
	Усилия на устранение	Человеко/часы
	Возраст (для открытых дефектов)	Количество времени (в днях) с момента получения отчета о дефекте
Меры процесса		
Процесс разработки	Истекшее время	• Календарные дни • Рабочие дни
	Контрольные сроки	Календарные даты
	Усилия на разработку	Человеко/часы, дни или месяцы
	Объем локализации дефектов	% общего числа дефектов, найденных на той стадии, на которой внесены
	Соответствие стандартам	% задач, согласующихся со стандартными процедурами или директивами
	Эффективность	Число успешно пройденных тестов, деленное на число выполненных тестов
Детальное проектирование	Истекшее время	• Календарные дни • Рабочие дни
	Качество проекта	Плотность дефектов: число дефектов проекта, найденных в проекте, деленное на меру размера продукта (FP, KSLOC)
Тестирование	Объем	Число запланированных тестов
	Прогресс	• Число выполненных тестов • Число успешно пройденных тестов
Сопровождение	Стоимость	• Объем финансирования/год • Человеко/часы, потраченные на выполнение запроса об изменении
Портфель запросов об изменении	Размер	• Число запросов об изменении, ожидающих обслуживания • Оцененная трудоемкость (человеко/часы) выполнения запросов
Меры ресурса		
Назначенный штат	Размер бригады	• Количество назначенных людей
	Опыт	• Количество лет опыта в проблемной области • Количество лет опыта программирования
CASE-инструменты	Тип	Название типа
	Используемость	• Да/нет (двоичная классификация)
Время	Дата начала, ожидаемая дата окончания, истекшее время	• Календарные даты • Дни

Нельзя ожидать, что все, что нужно будет измерить, может быть выражено в относительных шкалах. Начать можно с номинальных и порядковых шкал, постепенно переходя к более совершенным шкалам по мере развития практики измерений.

Выбирая между объективными и субъективными измерениями (метриками), нужно, насколько возможно, стремиться к объективным измерениям, однако не стоит пренебрегать и субъективными измерениями, если информация, которую они обеспечивают, помогает охарактеризовать, оценить, предсказать или усовершенствовать процессы или продукты. Важно постоянно совершенствовать процессы согласования мнений при проведении субъективных измерений и протоколирование их результатов.

4.1.2. Модель процесса измерения

Действия и задачи, которые должны выполняться в процессе измерения, определены в стандарте ISO/IEC 15939 [4] и соответствуют требованиям к процессу стандартов ISO/IEC 12207 и ISO/IEC 15504.

Исходя из положений стандарта ISO/IEC 15939, цель процесса измерения состоит в том, чтобы «собирать, анализировать и сообщать о данных, имеющих отношение к разрабатываемым продуктам и выполняемым (оперативным) процессам внутри подразделения организации, для того чтобы поддерживать эффективное управление процессами и объективно демонстрировать качество продуктов. В результате успешного выполнения процесса измерения:

- будут учреждены и поддержаны на высшем руководящем уровне соглашения (обязательства) по проведению измерений;
- будут идентифицированы информационные потребности в процессах инженерии и управления;
- будет идентифицирован и/или разработан надлежащий набор мер исходя из информационных потребностей;
- будут идентифицированы виды деятельности по измерению;
- идентифицированные виды деятельности по измерению будут спланированы;
- необходимые данные будут собираться, сохраняться, анализироваться, а результаты интерпретироваться;
- информационные продукты будут использоваться для поддержки принятия решений и предоставления объективного базиса для коммуникации;
- процесс измерения и меры будет оцениваться; и
- усовершенствования будут сообщаться инициатору и спонсору процесса измерения».

Это определение процесса измерения гораздо шире предложенного Н. Фентоном (п.4.1.1) и включает действия по инициации и подготовке процесса измерений, достижению необходимых договоренностей и согласований с другими процессами и созданию инфраструктуры процесса измерений.

Стандарт ISO/IEC 15939 определяет модель процесса и устанавливает требования к результатам выполнения действий и задач, не конкретизируя детали их выполнения.

Модель процесса измерения представлена на рисунке 4.1.

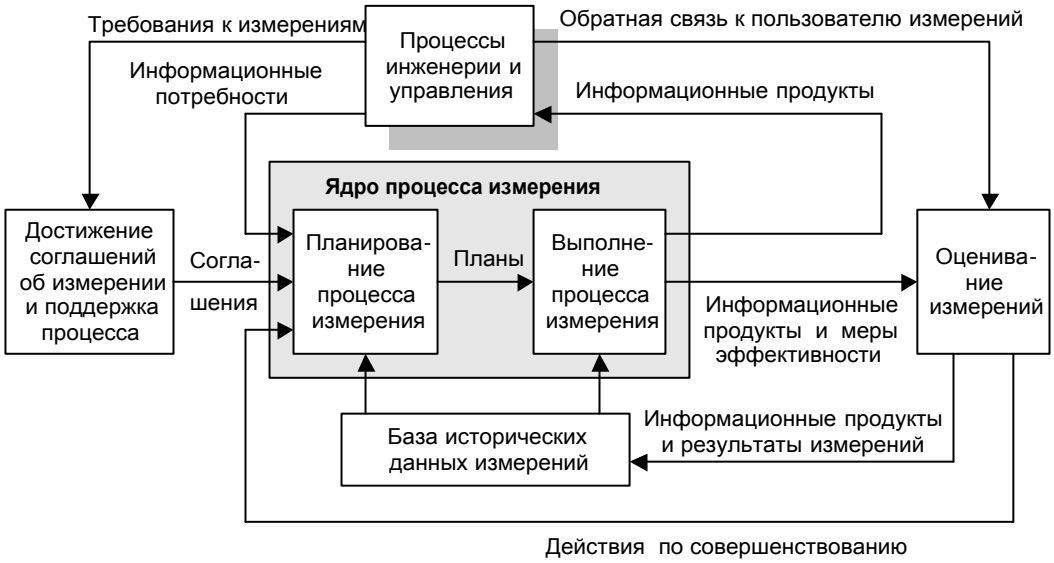


Рис. 4.1. **Модель процесса измерения**

Процесс измерения включает четыре вида деятельности, упорядоченных в виде итеративного цикла, обеспечивающего непрерывную обратную связь и совершенствование процесса измерения. Задачи, охватываемые каждым из видов деятельности, также итеративны.

Ядро процесса составляют два вида деятельности: *планирование процесса измерения* и *выполнение процесса измерения*. Эта деятельность в большей степени касается интересов потребителя информационных продуктов измерения. Два других вида деятельности – *учреждение (достижение соглашений) и поддержка процесса измерения* и *оценивание измерений* – обеспечивают фундамент для выполнения основных действий по измерению, а также обратную связь для их совершенствования. Эти виды деятельности касаются интересов лиц, ответственных за выполнение процесса измерения.

Действия, составляющие ядро процесса, непосредственно связаны с *информационными потребностями* организации, побуждающими к выполнению измерений и применению *информационных продуктов измерений* для оценивания, прогнозирования или других деловых целей организации.

Общий перечень действий и задач процесса измерения приведен ниже в четырех рамках (рисунок 4.2), соответствующих четырем видам деятельности (действий). Действия указаны в том порядке, в котором они обычно выполняются. Порядок, в котором перечислены задачи для каждого действия, не обязательно соответствует порядку их выполнения. Для каждой задачи (на третьем уровне вложенности) приводятся базовые приемы выполнения.

Связь между информационными потребностями и конкретными мерами измеряемых атрибутов сущностей описывается в виде *информационной модели измерения*. Стандарт ISO/IEC 15939 не приводит эталонной модели и, значит, не ограничивает потребителей в выборе известных или построении собственных информационных моделей.

1. Внедрить и сопровождать (поддерживать) программу измерения

1.1. Утвердить требования по измерению

1.1.1. Идентифицируются границы измерения.

1.1.2. Утверждаются обязательства по проведению измерений.

1.1.3. Обязательства доводятся до соответствующей организационной единицы.

1.2. Распределить ресурсы

1.2.1. Устанавливается личная ответственность за выполнение процесса измерения внутри организационной единицы.

1.2.2. Назначенные лица обеспечиваются ресурсами для планирования процесса измерения.

2. Спланировать процесс измерения

2.1.Охарактеризовать организационную единицу

2.1.1. Явно описываются особенности организационной единицы, имеющие отношение к выбору мер и интерпретации информационных продуктов.

2.2. Идентифицировать информационные потребности

2.2.1. Идентифицируются информационные потребности для измерения.

2.2.2. Устанавливаются приоритеты идентифицированных потребностей.

2.2.3. Выбираются информационные потребности, которые будут дальше рассматриваться.

2.2.4. Выбранные информационные потребности документируются и сообщаются заинтересованным лицам.

2.3. Выбрать меры

2.3.1. Идентифицируются меры-кандидаты, удовлетворяющие выбранным информационным потребностям.

2.3.2. Выбираются меры из числа мер-кандидатов.

2.3.3. Выбранные меры документируются с указанием имени (названия), единицы измерения, формального определения, метода сбора данных и связи с информационными потребностями.

2.4. Определить процедуры сбора данных, анализа и составления отчетов

2.4.1. Определяются процедуры сбора данных, включая их хранение и проверку.

2.4.2. Определяются процедуры для анализа данных и составления отчетов об информационных продуктах.

2.4.3. Определяются процедуры управления конфигурацией.

2.5. Определить критерии для оценивания информационных продуктов и процесса измерения

2.5.1. Определяются критерии для оценивания информационных продуктов.

2.5.2. Определяются критерии для оценивания процесса измерения.

2.6. *Просмотреть, одобрить и выделить ресурсы для решения задач измерения*

2.6.1. Просматриваются и утверждаются результаты планирования измерения.

2.6.2. Обеспечивается возможность использовать ресурсы для осуществления запланированных задач измерения.

2.7. Развернуть (внедрить) поддерживающие (вспомогательные) технологии

2.7.1. Оцениваются существующие доступные поддерживающие технологии и из них выбираются подходящие.

2.7.2. Выбранные поддерживающие технологии приобретаются и внедряются

3. Выполнить процесс измерения

3.1. Интегрировать процедуры

3.1.1. Генерация и сбор данных интегрируются в соответствующие процессы.

3.1.2. Процедуры сбора данных доводятся до ведома поставщиков данных.

3.1.3. Анализ данных и составление отчетов интегрируются в процессы.

3.2: Сбор данных

3.2.1. Данные собираются.

3.2.2. Собираемые данные сохраняются, включая любую контекстную информацию, необходимую для проверки, понимания или оценивания данных.

3.2.3. Собираемые данные проверяются.

3.3. Проанализировать данные и разработать информационные продукты

3.3.1. Собираемые данные анализируются.

3.3.2. Результаты анализа данных интерпретируются.

3.3.3. Информационные продукты просматриваются.

3.4. Сообщить о результатах

3.4.1. Информационные продукты документируются.

3.4.2. Информационные продукты доводятся до ведома их потребителей.

4. Оценить измерение

4.1. Оценить информационные продукты и процесс измерения

4.1.1. Информационные продукты оцениваются на соответствие специфицированным критериям оценки и делаются заключения о их достоинствах и недостатках.

4.1.2. Процесс измерения оценивается на соответствие специфицированным критериям оценки и делаются заключения о достоинствах процесса измерения.

4.1.3. Извлеченные уроки сохраняются в «Хранилище опыта измерения».

4.2. Идентифицировать потенциальные усовершенствования

4.2.1. Идентифицируются потенциальные усовершенствования к информационным продуктам.

4.2.2. Идентифицируются потенциальные усовершенствования к процессу измерения.

4.2.3. Потенциальные усовершенствования доводятся до ведома заинтересованных сторон.

Рис. 4.2. **Перечень действий и задач в стандарте ISO/IEC 15939**

Существуют разнообразные модели и механизмы трансформации информационных потребностей в измеримые атрибуты и множество пригодных для измерения мер и метрик. К ним можно отнести, например, модель, используемую в ранее упомянутой методологии «развертывания качества» (QFD) (глава 3), или модели, которые могут быть построены в рамках парадигмы «цель – вопрос – метрика» (GQM).

Подходы к измерениям, основанные на парадигме GQM, базируются на допущении, что для того чтобы организация могла выполнять целенаправленные измерения, она должна, прежде всего, конкретизировать собственные цели и цели проектов, а далее - проследить эти цели к данным, которые необходимы для оперативного определения этих целей, и наконец, обеспечить схему для интерпретации данных относительно установленных целей.

Целеориентированный подход к измерениям был изначально определен для оценивания дефектов по множеству проектов в среде NASA [5]. Теперь он широко

используется в парадигмах повышения качества для постановки целей. Результат применения GQM - спецификация схемы измерения, предназначенной для исследование определенного множества проблем на пути достижения деловых целей и содержащей множество правил для интерпретации данных измерения.

4.2. Методология измерения в парадигме «Цель – Вопрос – Мера»

4.2.1. Принципы измерения и последовательность действий

Рассматриваемая методология измерения опирается на один из подходов в парадигме GQM и охватывает ядро процесса измерения в модели процесса, представленной в стандарте ISO/IEC 15939.

Акцент в целеориентированном (иначе говоря, управляемом целями) измерении делается на планировании и осуществлении сбора и анализа информации, которая помогает достичь деловых целей, а также на поддержке механизмов обратной связи (трассируемости) от мер назад к деловым целям.

Управляемый целями процесс измерения базируется на 3 принципах и включает 10 этапов работ.

Три принципа измерения таковы:

- Цели измерения извлекаются из деловых целей.
- Контекст измерения обеспечивается построением ментальных моделей.
- Неформальные цели преобразуются в информационные структуры измерения.

Этапы процесса измерения таковы:

1. Определить деловые цели.
2. Идентифицировать объекты исследования или изучаемые проблемы.
3. Определить подцели.
4. Идентифицировать сущности и атрибуты, связанные с подцелями.
5. Формализовать цели измерения.
6. Сформулировать вопросы, требующие ответа в количественной форме, и определить пригодные для получения ответов на вопросы наглядные средства отображения зависимостей (диаграммы), которые будут использоваться для того, чтобы помочь оценить достижение целей измерения.
7. Определить элементы данных, собираемых для конструирования диаграмм.
8. Дать определения используемых мер и сделать эти определения оперативными, основанными на конкретных метриках (методах определения значений и шкалах).
9. Идентифицировать действия, которые будут предприняты для выполнения измерений (определения мер).
10. Подготовить план реализации измерений.

На рисунке 4.3 представлена модель измерений в рассматриваемом подходе, на которой в кружках указаны шаги процесса [6].

Основным механизмом преобразования деловых целей в описание проблем, вопросов и мер являются *ментальные модели* оперативных процессов инженерии или управления, отображающие взаимосвязи, существующие между элементами, ассоциируемыми с процессами.

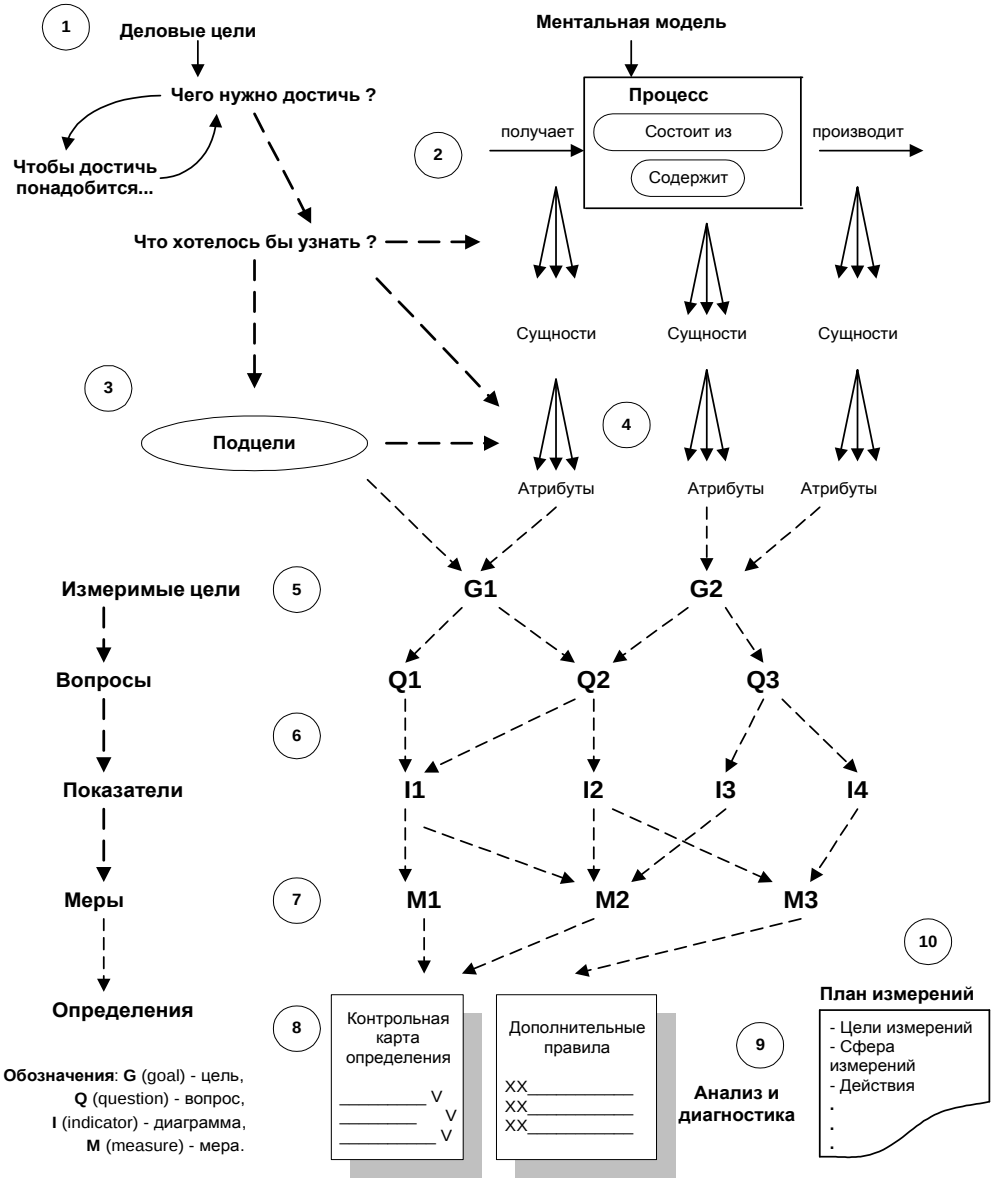


Рис. 4.3. **Модель измерений, управляемых целями**

Существует 4 типа элементов в представлениях оперативных процессов:

- то, что процессы получают извне (входы и ресурсы) и используют либо расходуют;
- то, что процессы производят (результаты) – основные продукты, промежуточные продукты и эффекты;
- то, из чего процессы состоят (действия, граф-схемы выполнения процессов и действующие лица) - структура процессов;
- то, что процессы содержат или сохраняют (внутренние артефакты, например: наработки, незавершенная работа, используемые методы и инструменты).

Все элементы в процессе являются сущностями, имеющими атрибуты, прямо или косвенно относящимися к деловым целям. Ментальные модели процесса должны акцентировать внимание на тех элементах, которые оказывают существенное влияние на процесс или дают глубокое представление о нем.

На рисунке 4.4 приведена таксономия сущностей – кандидатов на измерение. Она может помочь в построении полезных ментальных моделей, а также вопросников и других инструментов, используемых для идентификации сущностей и атрибутов, значениями которых можно управлять, для того чтобы достичь поставленных деловых целей.

Процесс

Получает	**Состоит из**	**Содержит**	**Производит**
Ресурсы Люди Средства Инструменты Деньги	**Процессы** Задачи Шаги Действия Подпроцессы Преобразования Обзоры Инспекции	**Материалы** Незавершенная работа Инструменты Данные Знания Опыт	**Продукты** Требования Спецификации Планы Архитектуры Эскизный проект Технический проект Рабочий проект Код Интегрированный код Протестирован. код Тестовые наборы Результаты тестирования Протестированные компоненты Запросы об изменен. Документация Дефекты Отчеты о дефектах Данные Приобретаемые материалы
Предметы Потребления Время Топливо Энергия Усилия Сырье	**Контроллеры** Контроллеры потока Сенсоры сигнальные Процессоры Шлюзы Клапаны Вентили		
Руководящие материалы Политика Процедуры Цели Ограничения Правила Законы Инструкции Нормы Средства Обучения	**Маршруты потоков…** ресурсов продуктов данных управления		**Побочные продукты** Знания Опыт Навыки Улучшения процесса Отходы и остатки Тепло и свет Данные Удовлетворенные заказчики
Основные продукты и промежуточные продукты от других процессов	**Резервные запасы и регуляторы** Очереди Стеки Резервуары Аккумуляторы Конвейеры		

Рис. 4.4. **Потенциально измеримые элементы (сущности) в модели процесса**

Кроме сущностей, используемых в модели оперативного процесса, возможно, также, придется измерять или характеризовать сущности, находящиеся вне моде-

лей процесса. Это могут быть сущности, связанные с факторами среды, оказывающей влияние на выполнение процесса (включая категории заказчиков, тип выпускаемой продукции, конъюнктуру рынка и др.). Количественная информация об этих внешних факторах поможет *распознать проблемы*, стоящие на пути к успеху, и *оценить возможности совершенствования* процесса - два вида деятельности, обычно присутствующих при достижении почти всех деловых целей.

4.2.2. Этап 1: определение деловых целей

Первый шаг процесса в парадигме целеориентированных измерений состоит в идентификации деловых целей, стимулирующих деятельность организации.

Хотя деловые цели в различных подразделениях зависят от уровня в иерархической структуре организации, на котором они находятся, и специфики их деятельности, - все цели, как правило, взаимосвязаны и часто одна с другой конкурируют. Множество целей от высших глобальных целей организации до локальных целей достижения успеха по определенным аспектам деятельности конкретных отделов и групп образуют иерархии и могут быть декомпозированы с любого уровня.

Управляемый целями процесс измерений достаточно гибок. Он может быть инициирован на любом организационном уровне. Если начать процесс измерения с определения целей высокого уровня, как, например, «уменьшить продолжительность разработки программных продуктов» или «повысить степень удовлетворенности заказчика получаемым программным продуктом», можно быть уверенным в том, что результирующие меры будут трассируемыми назад к деловым целям на этом уровне. Однако в этом случае может понадобиться итерация работ на этапах 2 и 3 (и возможно на этапе 4) процесса для того, чтобы стимулировать декомпозицию и анализ целей до такого уровня, на котором можно будет определить конкретные эффективные меры. Если начать процесс измерения на более низком уровне, можно избежать «лишней» работы на некоторых этапах, но при этом повысится вероятность того, что выбранные цели не будут протрассированы к целям высших уровней. Возможно лучший совет, который можно дать относительно выбора деловых целей, состоит в том, что начинать нужно с деловых целей тех лиц, чья поддержка наиболее важна для внедрения и осуществления процесса измерений.

Выполнение данного этапа работ очень важно для всего целеориентированного процесса измерений, поскольку обеспечивает гарантии, что все существенные моменты в деятельности организации учтены, цели выбраны правильно и измерения будут эффективно применены. Стимулом для выбора целей может стать, например, необходимость управлять рисками в определенных сферах деятельности организации, идеи совершенствования оперативных процессов или технологий, потребности стратегического планирования и пр.

Для идентификации и приоритезации целей полезно использовать методы мозгового штурма, интервьюирования коммерческих директоров и других ответственных исполнителей на всех уровнях иерархической организационной структуры.

Продуктом первого этапа работ является приоритезированный набор деловых целей организации. В интересах дела хорошо было бы получить одобрение со стороны руководителей (менеджеров) более высокого уровня, чтобы гарантировать, что приоритеты расставлены правильно и ничто важное не упущено или не интерпретировано неправильно.

4.2.3. Этап 2: идентификация объектов изучения

После того, как стали ясны деловые цели того организационного уровня, для которого выполняются измерения, нужно определить объекты, изучение которых поможет понять, оценить, предсказать, или улучшить действия, связанные с достижением целей. Задавая такие вопросы, как, например, «*какие именно действия выполняются*» и «*что, конкретно, хотелось бы улучшить или чего достичь?*», и формулируя такие утверждения, как, например, «*чтобы сделать это, нужно будет* …», можно начать идентифицировать необходимую количественную информацию. Формулировать вопросы нужно до тех пор, пока цели верхнего уровня не будут детализированы то такой степени, когда определяться сущности (объекты интереса), подлежащие усовершенствованию, или проблемы, подлежащие решению.

В качестве вспомогательного инструмента для построения вопросов и отождествления сущностей можно использовать перечни «сущность-вопрос», составленные по форме, представленной на рисунке 4.5.

Сущности, представляющие интерес	**Вопросы, связанные с деловой целью**
Основные и промежуточные продукты	
…	…
Входы и ресурсы	
…	…
Внутренние артефакты (незавершенная работа, задолженность, материалы и др.)	
…	…
Действия и маршрутные потоки	
…	…

Рис. 4.5. **Форма для составления перечней «сущность – вопрос»**

Последовательность решения задач на этапе 2 такова:

1. Начать с одной из целей верхнего уровня, которые должны быть определены на этапе 1.

2. Установить круг лиц, проблемы в работе которых нужно исследовать.

3. Построить ментальные модели исследуемых процессов. Руководствоваться нужно вопросом «*чего нужно достичь?*» и проблемой, решением которой нужно заняться, чтобы достичь *этого*.

4. Составить список важных деталей (сущностей) в процессах, убедившись в том, что отражение нашел каждый из четырех видов сущностей в процессе (см. рисунок 4.4). Можно также перечислить некоторые из сущностей среды, находящиеся за рамками рассматриваемых процессов.

5. Для каждой сущности составить список вопросов, ответы на которые смогут помочь планировать и управлять достижением стоящих деловых целей. Например: *Насколько* (то, о чем идет речь) *большое? Сколько его там? Как много компонентов? Насколько быстро происходит? Сколько это продолжается? Сколько это стоит?*

6. Посмотреть на процесс в целом, чтобы понять, не пропущено ли что-либо. С помощью выяснения таких вопросов, как, например: *Стабильный ли процесс?*

Как он выполняется в настоящее время? Что ограничивает возможности его улучшения? Что определяет качество? Что определяет успех? Что можно контролировать? Чего хотят заказчики? Что ограничивает выполнение? Что может идти неправильно? Что могло бы использоваться в качестве сигнализатора для своевременного предупреждения? Насколько велика задолженность? Каково происхождение задолженности? и, что особенно важно, *как об этом узнать?* - можно обнаружить дополнительные сущности, атрибуты которых возможно стоит измерить.

7. Повторить задачи 1 - 6 для всех других целей. Сначала лучше четко идентифицировать элементы данных, касающиеся одной или двух основных деловых целей, а лишь затем переключиться на дополнительные цели. Это поможет обеспечить управляемость процесса измерений, поскольку впоследствии образуется большой набор мер, который нужно будет сокращать, оставляя только меры, наиболее значимые и пригодные для измерения как можно большего числа сущностей. Нет универсальных рекомендаций относительно деления целей на основные и дополнительные (второстепенные). Важно помнить, что как первая цель, так и все последующие ориентированы на повышение степени удовлетворенности заказчика получаемым программным продуктом.

В таблице 4.2 перечислены примеры сущностей и вопросов, которые могли бы интересовать менеджера проекта, пытающегося повысить степень удовлетворенности заказчика получаемым программным продуктом.

Таблица 4.2. **Пример перечня «сущности-вопросы»**

Сущности, интересующие менеджера	**Вопросы, связанные с удовлетворением заказчика**
1. Основные и промежуточные продукты	
1.1. Документы	1.1.1. Читабельны ли разрабатываемые документы?
	1.1.2. Можно ли отследить возможности системы от одного документа к другому?
	1.1.3. Являются ли документы лаконичными и полными?
	1.1.4. Корректна ли терминология?
1.2. Исходный код и откомпилированные продукты	1.2.1. Согласуется ли исходный код с документами?
	1.2.2. Свободен ли исходный код от ошибок?
	1.2.3. Соответствует ли исходный код стандартам программирования?
	1.2.4. Адекватно ли время отклика системы?
	1.2.5. Удовлетворителен ли человеко-машинный интерфейс?
1.3. Планы	1.3.1. Согласуются ли планы с ограничениями заказчика?
	1.3.2. Актуальны ли планы?
	1.3.3. Доводятся ли планы и изменения до заказчика?
1.4. Финансирование	1.4.1. Согласуется ли финансирование с планами?
	1.4.2. Согласуются ли финансовые сметы с ограничениями заказчика?

Сущности, интересующие менеджера	Вопросы, связанные с удовлетворением заказчика
2. Входы и ресурсы	
2.1. Сотрудники	2.1.1. Способны ли люди квалифицированно разработать то, что удовлетворило бы заказчика?
	2.1.2. Существует ли текучесть кадров, препятствующая обеспечению качества продукта?
2.2. Соисполнители	2.2.1. Согласуются ли практические приемы, которыми пользуются соисполнители, с той деятельностью, которую они должны выполнять в проекте?
2.3. Компьютеры	2.3.1. Удовлетворяет ли целевая система требованиям к эффективности?
	2.3.2. Надежна ли целевая система?
2.4. Запросы об изменении	2.4.1. Содержат ли запросы об изменениях, подаваемые заказчиком, информацию о том, что измерения должны делаться вовремя и быть эффективными?
3. Внутренние артефакты (незавершенная работа, задолженность, хранимые материалы и др.)	
3.1. Запросы об изменениях (незавершенные работы)	3.1.1. Насколько велика задолженность по отработке запросов об изменениях, поданных заказчиком?
	3.1.2. Где возникли задолженности?
4. Действия и маршрутные потоки	
4.1. Разработка	4.1.1. Видит ли заказчик прогресс в разработке?
4.2. Тестирование	4.2.1. Адекватны ли процедуры тестирования для оперативного использования системы?
	4.2.3. Одобрил ли заказчик процедуры тестирования и результаты тестирования?
4.3. Устранение ошибок	4.3.1. Соизмеримо ли время устранения ошибок с ограничениями заказчика?
	4.3.2. Поставлено ли изменение на контроль?
	4.3.3. Реализуются ли высокоприоритетные изменения своевременно?
	4.3.4. Видит ли заказчик состояние и продвижение его запросов об изменениях?

4.2.4. Этап 3: определение подцелей

Третий шаг в управляемом целями процессе измерения состоит в том, чтобы трансформировать цели верхнего уровня в подцели, связанные с конкретными действиями, выполняемыми в исследуемом оперативном процессе.

Вопросы, порожденные ментальной моделью оперативного процесса, указывают прямо или косвенно на сущности и атрибуты, связываемые с достижением целей. Теперь нужно перегруппировать эти вопросы таким образом, чтобы очертить проблемы, к которым они могут иметь отношение. Например, первые четыре вопроса в таблице 4.2 (вопросы 1.1.1. – 1.1.4) касаются документации. Они уже сгруппированы таким образом, что охватывают проблему, связанную с разрабатываемой документацией.

Поскольку перечень «сущности-вопросы» просматривается сверху вниз, можно видеть, что новую группу образуют следующие пять вопросов, имеющих отношение к качеству и эффективности работы программного продукта (1.2.1-1.2.5). В эту группу могут быть включены также вопросы 2.3.1, 2.3.2, 4.2.1 и 4.3.2, поскольку тоже касаются данной проблемы.

По мере того, как идентифицируются темы и проблемы, представляющие интерес, собираются относящиеся к ним вопросы и формируется новый список, отсортированный по проблемам (таблица 4.3). Один и тот же вопрос может входить в разные группы, если он касается разных областей проблем.

Таблица 4.3. **Группирование вопросов для определения подцелей**

Группы	**Вопросы в группе, касающиеся удовлетворенности заказчика**
Группа 1 (документы)	• Читабельны ли разрабатываемые документы? • Отслеживаются ли возможности системы от одного документа к другому? • Являются ли документы лаконичными и полными? • Корректна ли терминология?
Группа 2 (программный продукт)	• Согласуется ли исходный код с документами? • Свободен ли исходный код от ошибок? • Соответствует ли исходный код стандартам программирования? • Адекватно ли время отклика системы? • Удовлетворителен ли человеко-машинный интерфейс? • Удовлетворяет ли целевая система требованиям к эффективности? • Надежна ли целевая система? • Поставлено ли изменение на контроль? • Адекватны ли процедуры тестирования для оперативного использования системы?
Группа 3 (управление проектом)	• Согласуются ли планы с ограничениями заказчика? • Актуальны ли планы? • Согласуется ли финансирование с планами? • Согласуются ли финансовые сметы с ограничениями заказчика?
Группа 4 (управление изменениями)	• Содержат ли запросы об изменениях, подаваемые заказчиком, требования о том, что измерения должны делаться вовремя и быть эффективными? • Насколько велика задолженность по отработке запросов об изменениях, поданных заказчиком? • Соизмеримо ли время устранения ошибок с ограничениями заказчика? • Где возникли задолженности? • Поставлено ли изменение на контроль? • Реализуются ли высокоприоритетные изменения своевременно?
Группа 5 (взаимодействие)	• Доводятся ли планы и изменения до заказчика? • Видит ли заказчик прогресс в разработке? • Одобрил ли заказчик процедуры тестирования и результаты тестирования? • Видит ли заказчик состояние и продвижение его запросов об изменениях?
Другие	• Способны ли люди квалифицированно разработать то, что удовлетворило бы заказчика? • Существует ли текучесть кадров, препятствующая обеспечению качества продукта? • Согласуются ли практические приемы, которыми пользуются соисполнители, с той деятельностью, которую они должны выполнять в проекте?

Группирование проблем и вопросов далее естественным образом переходит в формулирование подцелей-кандидатов (таблица 4.4).

Таблица 4.4. **Подцели с позиции цели повышения степени удовлетворения заказчика (взгляд менеджера)**

Обозначение подцели	Формулировка подцели
Подцель 1	Улучшить читабельность и трассируемость документов
Подцель 2	Улучшить надежность и эффективность разработанного кода
Подцель 3	Улучшить мониторинг планов и финансирования
Подцель 4	Повысить эффективность процесса управления изменениями
Подцель 5	Улучшить взаимодействие с заказчиком

Если список получается длинным и может впоследствии вызвать появление большого количества мер, - проблемам можно присвоить приоритеты и выполнить упорядочение. Это поможет в дальнейшем сосредоточить внимание, прежде всего, на главных проблемах.

4.2.5. Этап 4: идентификация сущностей и атрибутов

После того, как построен перечень обоснованных подцелей наряду со списками соответствующих проблем и вопросов, эти списки используются для уточнения ментальной модели и связанных с ней сущностей и атрибутов.

Работа на этапе 4 начинается с создания предварительного эскиза ментальной модели оперативного процесса инженерии или управления. Затем перечисляются вопросы, на которые важно получить ответы. Эти вопросы обычно ассоциируются с подцелями, имеющими наивысший приоритет.

Выполнение эскиза и построение списка – итеративный процесс – эскизы порождают вопросы, а вопросы приводят к уточнению эскизов. Подобные итерации, вероятно, появятся и на более поздних этапах, поскольку развитие ментальной модели будет продолжаться.

Каждый вопрос в списке нужно рассматривать отдельно, идентифицируя сущности, которые в нем содержаться, а затем перечислить подходящие атрибуты, ассоциирующиеся с каждой сущностью. Подходящими считаются такие атрибуты, идентификация которых поможет ответить на вопрос или установить контекст интерпретации ответов. Нужные атрибуты обычно цитируются в вопросе, явно или неявно. Распознавание сущностей может повлечь за собой также появление новых вопросов и атрибутов.

Список сущностей и атрибутов для каждой сущности – основной результат работы на данном этапе. Атрибуты становятся кандидатами на измерение. Полезно помнить о различии между *атрибутами*, характеризующими сущность, и *мерами*, связываемыми с соответствующими метриками и шкалами для измерения атрибутов. При составлении списков сущностей и атрибутов нужно стремиться к тому, чтобы меры, которые далее будут выбираться, были уникальными. Форма для учета сущностей и атрибутов по каждому вопросу представлена на рисунке 4.6, а примеры сущностей и атрибутов для группы вопросов, определяющих подцель 4 – на рисунке 4.7.

Вопрос
- _____

Сущность
- _____

Атрибуты
- _____
- _____

…

Рис. 4.6. **Форма учета сущностей и атрибутов**

Подцель 4: **Повысить эффективность процесса управления изменениями**

Вопрос 1:
- Содержат ли запросы об изменениях, подаваемые заказчиком, информацию о том, что изменения должны делаться вовремя и быть эффективными?

Сущность:
- *Портфель запросов об изменениях, полученных от заказчиков*

Атрибуты:
- *Размер портфеля* (например, количество полученных запросов об изменении)
- *Адекватность* (например, % запросов об изменении, все поля в которых заполнены правильно)
- *Распределение дефектов* (например, для каждого обязательного поля в структуре запроса: 1) % запросов, в которых это поле не заполнено; 2) % запросов, в которых поле заполнено неправильно)

Вопрос 2:
- Насколько велика задолженность по отработке запросов об изменениях, поданных заказчиком?

Сущность:
- *Портфель неотработанных запросов об изменениях*

Атрибуты:
- *Размер портфеля* (количество запросов, полученных, но еще не отработанных)
- *Размер очереди запросов*, пребывающих в ожидании действий на каждой стадии процесса управления изменениями
- *Общее оцененное количество усилий*, необходимых для разгрузки портфеля

Вопрос 3:
- Соизмеримо ли время устранения ошибок с ограничениями заказчика?

Сущность:
- *Процесс управления изменениями*

Атрибуты:
- *Ожидания пользователя* относительно длительности цикла отработки запросов
- *Распределение частоты* периодов времени от момента получения запроса до инсталляции обновленного фрагмента продукта у заказчика
- *Среднее количество времени*, которое тратится на отработку запроса на каждом шаге процесса реализации изменений

Рис. 4.7. **Пример описания сущностей и атрибутов**

4.2.6. Этап 5: формализация целей измерения

До этого момента в центре внимания были деловые цели и все, что влияет на их достижение. Пятый шаг состоит в трансформации проблем и вопросов в ясно определенные и установленные *цели измерения*.

Работы на этапах 1 - 4 подготовили базис для эффективного применения собственно подхода GQM в его оригинальном виде, разработанного в 1988 году авторами парадигмы измерений «цель-вопрос-метрика»[1] - Бейсли и Ромбахом [7, 8, 9].

Отличие рассматриваемого здесь подхода от оригинального состоит в том, что в нем к измерению добавлено промежуточное звено между «вопросом» и «метрикой» (мерой) – «*диаграмма*». Диаграмма в данном случае – обобщенное название для любых средств наглядного отображения (графиков, карт, схем, аналитических описаний) зависимостей между атрибутами, применение которых должно помочь в ответе на вопрос и определении подходящих мер.

Задача данного этапа – установить цели измерения и подготовить необходимые структурированные утверждения о них.

Цели измерения могут иметь активную или пассивную форму.

Активные цели измерения направлены на контролирование процессов или побуждение изменений в продуктах, процессах, ресурсах или средах. Это такой вид целей, который обычно присутствует в деятельности по управлению проектом и усовершенствованию процесса. Примеры активных целей:

- достичь строгого соблюдения графика проекта,
- повысить надежность продукта,
- снизить текучку кадров.

Пассивные цели измерения, с другой стороны, побуждают к изучению и обучению. Они служат основой совершенствования представлений об оперативном процессе до такого уровня, на котором могут быть формализованы удачно выбранные активные цели измерения. Пассивные цели часто формулируются путем определения характеристик объектов, представляющих интерес в контексте некоторой модели продуктивности или качества. Примеры пассивных целей:

- понять текущий процесс разработки,
- установить первопричины,
- понять связь между атрибутами, чтобы можно было разработать модели для предсказания и оценивания.

Активные цели обычно связываются с оцениванием и совершенствованием, а пассивные – с определением характеристик и предсказанием. Пассивными целями нельзя пренебрегать, сосредотачиваясь исключительно на активных целях.

Независимо о того, активные или пассивные цели измерения рассматриваются, их описания должны быть хорошо структурированы и включать четыре компонента:

- объект интереса (сущность);
- намерение;
- позиция;
- условия (описание среды и ограничений).

---

[1] Хотя в названии парадигмы GQM фигурирует термин «метрика», в действительности элементом парадигмы является мера. Этот термин был определен ранее в главе 3.

Форма описания структурированных целей представлена на рисунке 4.8.

Объект интереса: _____
(процесс, продукт, ресурс, задача, действие, [сущность] и др.)

Намерение: _____ (охарактеризовать, проанализировать, оценить и др.)
(что сделать)

_____ ([сущность], [аспект], [атрибуты] и др.)
(с чем)

для того чтобы _____ (понять, обосновать, предсказать, спланировать, сравнить и др.)

Позиция:
Проверить _____ (поведение, стабильность, качество, прогресс, [атрибуты] и др.)
с позиций _____ (разработчика, менеджера, руководителя, заказчика и др.)

Условия: _____(факторы среды или другие параметры для определения контекста)

Рис. 4.8. **Форма структурированного описания цели**

Объект интереса – это любой предмет, реальный или абстрактный, который нужно описать или о котором нужно больше узнать (продукт, процесс, ресурс, агент, артефакт, действие, среда и др.). Объект - это *сущность*, но не любая, а та конкретная сущность, которая должна быть описана в измеримых величинах.

Намерение при выполнении измерений может быть обусловлено желанием понять, предсказать, спланировать, проконтролировать, сравнить, оценить или улучшить продуктивность или качество объекта с учетом таких аспектов, как стоимость, размер, надежность, тестовое покрытие, ответная реакция, согласованность процессов и др. Намерения любой деятельности по измерению должны устанавливаться четко.

Позиция – компонент структурированного описания цели, указывающий того, кто заинтересован в результатах измерения, и уточняющий его намерения по измерению.

Описание *условий* определяет контекст, в котором должны интерпретироваться результаты измерения. Если контекст недостаточно ясен, он может быть не понят всеми теми, кто использует собранные данные. Контекст определяет среду, в которой выполняются измерения, и включает все факторы, влияющие на измеряемый объект, или подверженные его влиянию. Контекст формируют параметры приложений, человеческий фактор, ограничения ресурсов, факторы процесса, методы, специфика организации-заказчика и др. То есть контекст охватывает все существенные ограничения, связанные с объектом измерения, а также ограничения на сферу и продолжительность применения самого процесса измерения.

Для того чтобы информация об условиях измерений приносила пользу, нужно сосредоточить внимание на двух аспектах:

1. В чем *сходство* данного продукта или процесса с уже знакомыми персоналу. На основе сходства формируется контекст выполнения сравнений.

2. В чем *отличие* данного продукта или процесса от тех, с которыми персонал уже знаком. На основе отличий формируется контекст исследования, понимания и объяснения отличий.

После того, как описаны цели измерения, нужно убедиться в их трассируемости к подцелям и деловым целям, которые мотивировали каждую цель измерения. Тогда, если позже будут возникать вопросы по поводу намерения измерения, можно будет вернуться к истокам цели и принять реализационные решения, согласующиеся с деловыми целями. Если определено много мер или подцелей, для установления их взаимосвязи могут пригодиться матричные изображения или другие методы и инструменты отображения, например, применяемые в подходе QFD.

4.2.7. Этап 6: формулирование вопросов и выбор диаграмм

Структурированные цели измерения образуют фундамент для выполнения измерений в парадигме «цель-вопрос-метрика». Фактически, первый шаг процесса измерения, определенный в парадигме GQM, уже выполнен – *измеримые цели* установлены в результате выполнения шагов 1 - 5. Нужно заметить, что применение подхода GQM начиная с более высоких уровней определения целей (например, непосредственно с деловых целей) было бы не эффективно, поскольку сложно четко определить значимые сущности, намерения, позиции и среду измерения. Вопросы, поставленные на очень высоком уровне, имеют слишком размытую формулировку, за которой нельзя распознать множество мер, необходимых для понимания основных атрибутов процессов и продуктов, которые нужно измерять.

После определения измеримых целей можно формулировать *вопросы*, предполагающие ответы в численном выражении, и готовить диаграммы, способствующие получению ответов в числовой форме. Стоит напомнить, что понятие «диаграмма» и дополнительный шаг в парадигме GQM введен для того, чтобы подчеркнуть важность отображения и объяснения результатов измерений. Знание о том, как данные измерений будут отображаться, помогает точно установить, что именно должно измеряться. В условиях быстрого изменения сред точные вычисления и детальный статистический анализ зачастую оказываются менее полезны, чем графические изображения и ответы на простые меткие вопросы.

Действия на этапе 6 лучше пояснить на примере. Предположим, что измеримая цель определена так, как на рисунке 4.9.

Объект интереса: Процесс разработки на предприятии X

Намерение: Проанализировать дефекты, внесенные в ПО при его разработке, для того чтобы установить возможности по снижению стоимости и повышению качества продукта

Позиция: *проверить* внесение, обнаружение и устранение дефектов
с позиций группы улучшения процесса

Условия: Приложения для телефонных станций. СММ Уровень 2. 300 сотрудников. 100 разработчиков. Код на C++ и Ассемблере ПЭВМ. Проверять только проекты, прошедшие автономное тестирование в январе.

Рис. 4.9. **Пример описания цели измерения**

С этой целью могут быть связаны следующие вопросы:

- В каких видах рабочих продуктов были найдены дефекты?
- На какой стадии дефекты были внесены?
- Есть ли типы дефектов, зафиксированных впервые?
- Насколько мы близки к достижению подцели улучшения работы на стадии автономного тестирования?

Некоторые диаграммы, ассоциируемые с указанными вопросами, представлены на рисунках 4.10 – 4.12.

Рисунок 4.10 показывает, где были найдены дефекты различных типов. Можно увидеть, например, что наибольшее количество дефектов проекта было найдено во время системного тестирования. А дефекты в требованиях были обнаружены практически вовремя – там, где были допущены. Видно также, что почти 10% общего количества проблем не было обнаружено вплоть до испытаний в целевой среде и что среди них достаточно большая доля ошибок в документации.

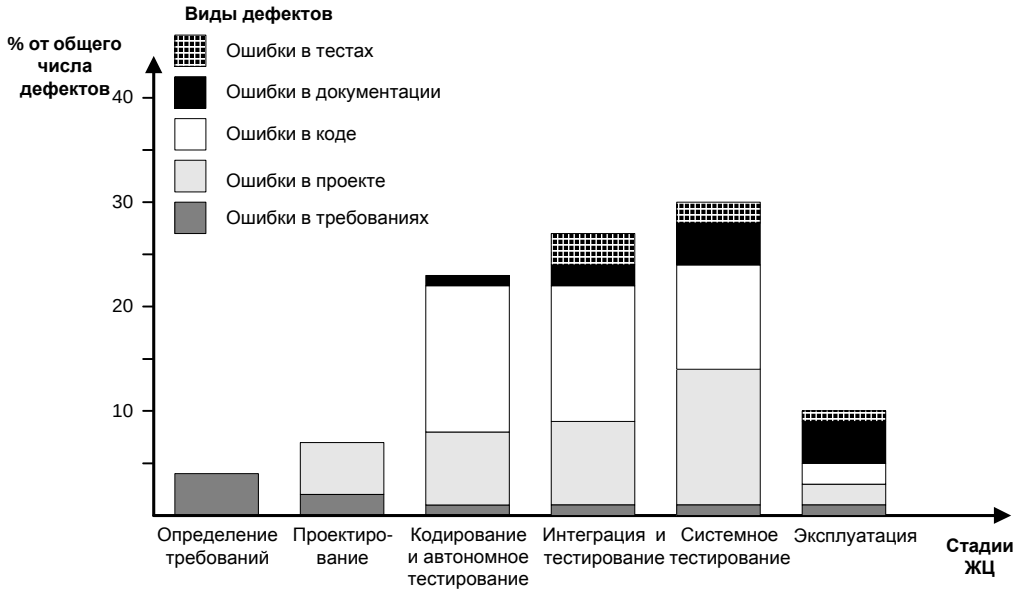


Рис.4.10. **Диаграмма распределения дефектов в разрезе стадий ЖЦ**

Рисунок 4.11 показывает текущие взаимосвязи между тем, где ошибки были допущены, и тем, где были обнаружены дефекты.

Рисунок 4.12 проливает дополнительный свет на анализ потока дефектов, изображенного на рисунке 4.11.

Распределения внесения ошибок и обнаружения дефектов изображаются в виде траекторий и добавляется информация о затратах на устранение дефектов. Имея распределение затрат, даже если оно недостаточно точное, можно просчитать экономические последствия работы в текущих условиях и эффект от предлагаемых усовершенствований. Это часто помогает принимать более обоснованные решения.

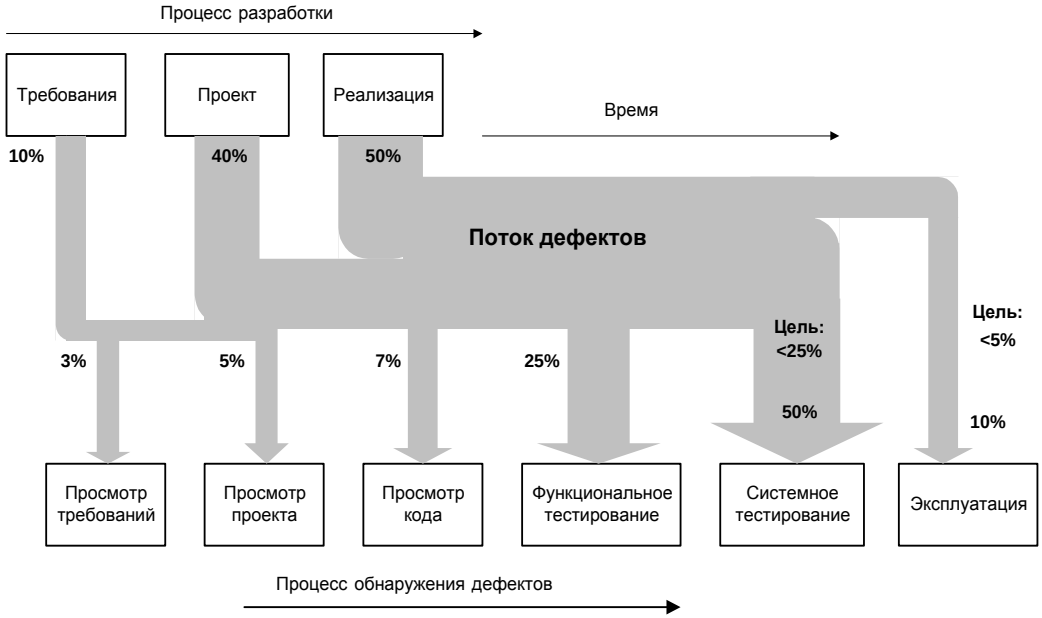


Рис. 4.11. **Связь процессов внесения ошибок и обнаружения дефектов**

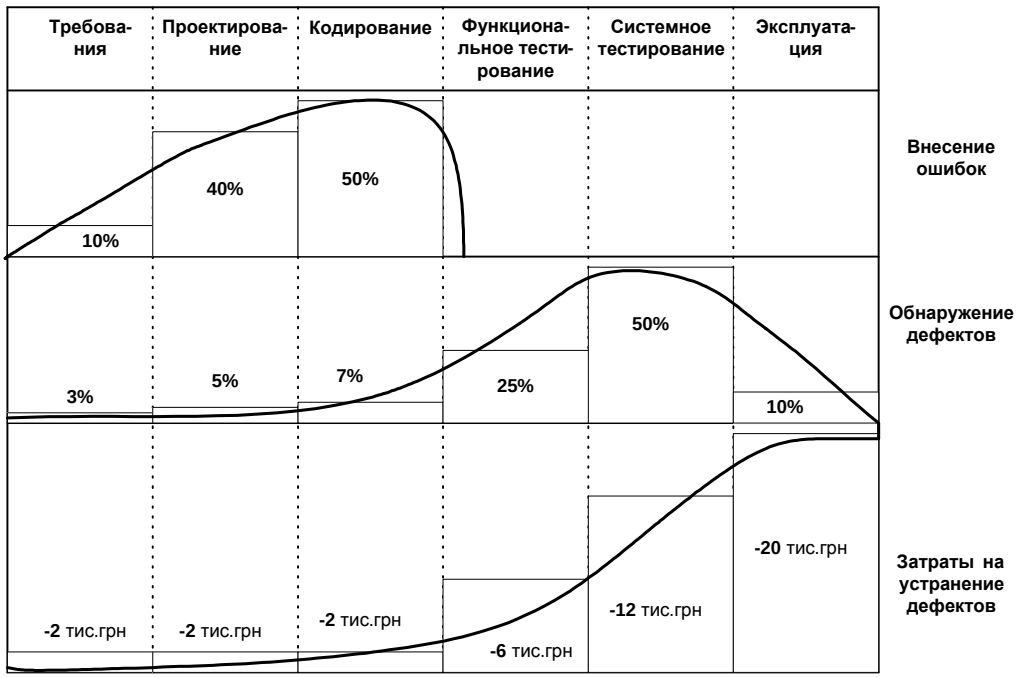


Рис.4.12. **Кривые внесения ошибок, устранения дефектов и затрат**

Как видно, создание полезных диаграмм – творческий процесс. Неправильно выбранные диаграммы могут ввести в заблуждение, как их создателей, так и тех, для кого они предназначаются. Поэтому до завершения этапа 6 нужно критически пересмотреть предложенные средства отображения зависимостей, чтобы убедиться

в том, что изображения, которые они представляют, действительно могут ответить на поставленные вопросы.

4.2.8. Этап 7: идентификация элементов данных

Определив, какие диаграммы могут быть полезны, можно приступить к идентификации элементов данных, которые нужно собрать, для того чтобы построить графические изображения. Правильное выполнение работ на этапах 1-6 обеспечивает гарантии, что собираемые данные будут отвечать конкретным целям управления процессом. Это не данные ради данных.

На этом и следующем этапе процесса измерения нужно сделать следующее:

1. Идентифицировать элементы данных.
2. Определить, каким образом данные будут собираться.

Для того чтобы решить первую задачу, нужно просто составить список всех элементов данных, которые понадобятся для построения диаграмм. Это поможет установить их приоритеты и определить, какие меры могут использоваться одновременно для нескольких целей (рисунок 4.13). Меры, используемые неоднократно, получают более высокий приоритет.

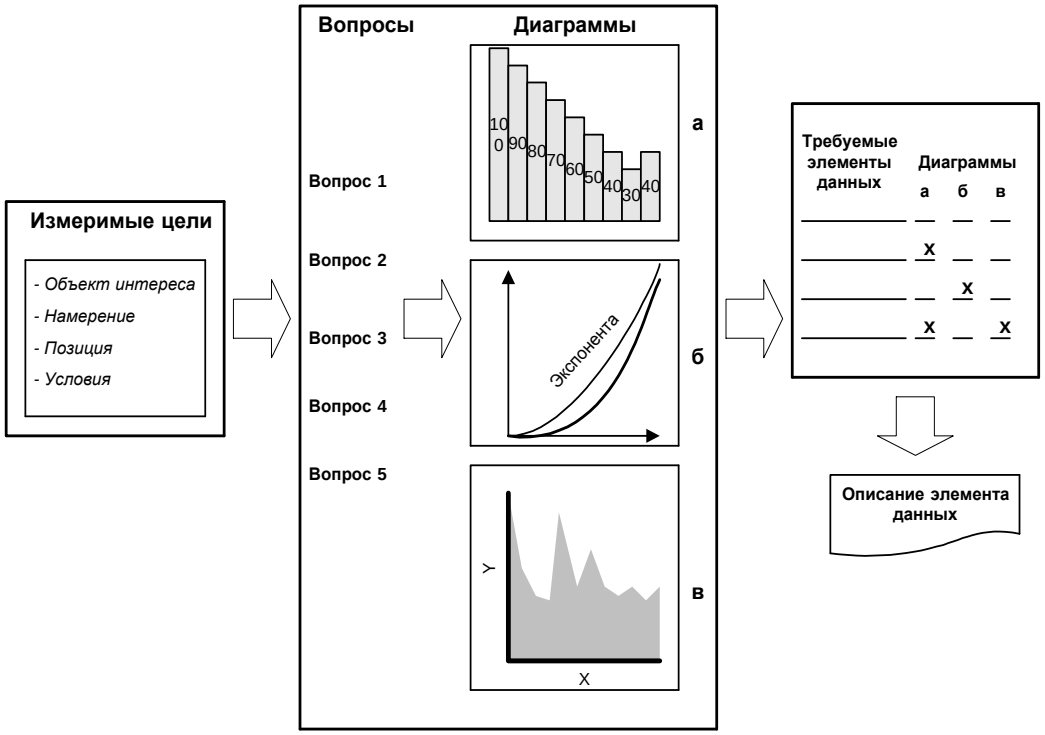


Рис. 4.13. **Последовательность решения задач в парадигме GQM**

4.2.9. Этап 8: разработка схем определения используемых мер

Для того чтобы измерения имели успех, нужно использовать хорошо структурированные схемы, помогающие дать определения мерам, применять их и распространять всем заинтересованным лицам. Определение меры должно быть не просто правильным, но и обеспечивать полное понимание того, что именно измеренные величины представляют. Только в этом случае будет обеспечена полнота

сбора данных, их последующая корректная интерпретация и применение результатов измерений. Определение должно включать все множество правил, используемых для выполнения измерений и протоколирования их результатов. Хотя на взгляд разных категорий пользователей определение может показаться избыточным, нужно учитывать, что не важное для одних может оказать важным для других.

Определения должны удовлетворять двум важным критериям:

- *информативность*. Должно быть понятно, можно ли из оперативного определения узнать, что измеряется, как измеряется, а также какие элементы включаются в измерение, а какие из него исключаются.
- *повторяемость*. Должно быть понятно, можно ли воспользоваться определением повторно, провести измерение и получить практически тот же результат.

Хотя обеспечение информативности определения требует дополнительных затрат, польза станет очевидной в случае повторения измерений кем-либо другим - достаточно будет попросить измеряющего повторить *это* измерение.

Удобно представлять определения в виде *операционных контрольных листков*, сопровождаемых формами, содержащими дополнительную разъясняющую и уточняющую информацию, не включенную в листок. Основное назначение контрольного листка состоит в том, чтобы указать, *какие именно элементы* включаются в конкретное измерение или из него исключаются. Дополнительные формы описывают практические приемы включения и исключения элементов из измерения. Такие описания должны быть частью схемы определения, поскольку они влияют на выбор способа интерпретации результатов измерения.

На рисунке 4.14 представлен фрагмент заполненного контрольного листка для учета дефектов в конкретном процессе измерения.

Организация может создавать собственные схемы определения элементов данных и формы для их описания или адаптировать существующие (например, используемые другими организациями, найденные в литературе или по Интернету). В любом случае, должны быть гарантии, что построены схемы и формы для определения *всех* данных, которые нужно собирать, и описаны *все* методы сбора данных. Незнание методов, применяемых для сбора данных в конкретном измерении, может сказаться либо на результатах, либо на их интерпретации.

Важное преимущество контрольных листков (и дополнительных форм) состоит в том, что они могут использоваться в качестве механизмов идентификации и согласования отличающихся информационных потребностей разных пользователей измерений или мнений лиц, выполняющих измерения.

4.2.10. Этап 9: идентификация задач и инфраструктуры сбора данных

Для того чтобы можно было подготовить эффективный план измерений установленных элементов данных, нужно получить информацию о текущем состоянии исследуемого оперативного процесса и использовании мер. Основа данного этапа работ - анализ, диагностика и технологическая подготовка сбора данных.

Анализ заключается в сборе фактов, которые помогают понять, с чего нужно начать измерение. Он предполагает выявление мер, уже используемых организацией в настоящее время, и применяемых методов сбора данных.

Состояние проблем	**Включать**	**Не включать**	**Количество**
Открыты	**X**		**X**
Признаны			
Оценены			
Решены			
Закрыты	**X**		**X**
Тип проблем	**Включать**	**Не включать**	**Количество**
Дефекты в ПО			
Дефекты требований	**X**		**X**
Дефекты проекта	**X**		**X**
Дефекты кода	**X**		**X**
Дефекты документов	**X**		**X**
Дефекты тестов		**X**	
Другие дефекты		**X**	
Другие проблемы			
Проблемы аппаратуры		**X**	
Проблемы ОС		**X**	
Ошибки пользователя		**X**	
Ошибки в операциях		**X**	
Новые требования		**X**	
Уникальность	**Включать**	**Не включать**	**Количество**
Впервые	**X**		**X**
Повторно		**X**	
Не известно		**X**	
Критичность	**Включать**	**Не включать**	**Количество**
1 уровень (высший)	**X**		**X**
2 уровень	**X**		**X**
3 уровень	**X**		**X**
4 уровень	**X**		**X**
Не известно	**X**		
…			

Рис. 4.14. **Фрагмент контрольного листка для учета дефектов**

Эта информация определяет начальную точку для осуществления управляемых целями измерений по построенной схеме определений.

Анализируя существующие меры и практические приемы измерения, полезно задаваться такими вопросами, как, например, следующие:

- Какие элементы данных требуются для управляемых целями измерений?
- Какие элементы данных собираются в настоящее время?
- Каким образом они собираются?
- Какие процессы служат источниками данных?
- Каким образом элементы данных фиксируются и сохраняются?

Если организация применяет несколько однотипных мер (как, например, меры размера – KSLOC и FP), нужно их точно именовать, не ограничиваясь общим описанием того, что нужно измерять. Это поможет сосредоточиться на поиске всех возможных источников информации. А их может оказаться больше, чем изначально казалось. Например, источниками данных о проблемах могут быть не только отчеты о тестировании и требования об изменениях, поступающие от заказчика, но

и отчеты о просмотре или инспекции требований, проекта, кода и др. Хотя эти данные могут фиксироваться раздельно и содержаться в разных хранилищах, для полноты анализа все они должны учитываться.

Диагностика предполагает оценивание элементов данных, которые уже собираются в организации, определение того, насколько хорошо они отражают потребности управляемых целями измерений, и формирование предложений по использованию данных, их адаптации к информационным потребностям или адаптации потребностей к данным.

Если анализ состоит в поиске фактов, диагностика – в оценивании и составлении мнения. Установление диагноза связано с выявлением альтернативных оценок и мнений и определением стадий принятия решений. Выполняя диагностику можно задаваться, например, такими вопросами:

- Какие существующие меры и процессы могут использоваться, для того чтобы удовлетворить требования к элементам данных?
- Какие элементы определений измерения или практические приемы должны быть заменены или модифицированы?
- Какие новые или дополнительные действия нужны?

Нужно оценить пригодность и готовность существующих данных и идентифицировать текущие и потенциальные источники информации.

Технологическая подготовка заключается в применении результатов анализа и диагностики для выработки конкретных шагов по измерению. Она предшествует составлению плана измерений и предполагает идентификацию задач, выбор ответственных исполнителей и выделение ресурсов. Начинается с определения элементов инфраструктуры, на которых будет основываться план измерения.

Во время технологической подготовки нужно сделать следующее:

- идентифицировать *источники данных* в уже действующих процессах программной инженерии;
- определить *методы*, которые будут использоваться для сбора и регистрации данных;
- идентифицировать (и специфицировать) *инструментальные средства*, которые могут понадобиться для сбора, регистрации и хранения данных;
- определить требования, касающиеся *контрольных точек* в ЖЦ и частоты измерения;
- подробно документировать *процедуры сбора данных, а именно:*
  - установить ответственных за сбор данных;
  - определить, где, как и когда собирать и регистрировать данные;
  - создать эскизы регистрационных форм сбора данных;
- определить, *кто будет использовать* данные;
- определить, как данные будут анализироваться и распространяться;
- *подготовить методики* по описанию данных и процессу их сбора.

Нужно также проанализировать требования к *хранилищам данных* и правам доступа. А это, в свою очередь, связано с выяснением:

- необходимости сохранения исторического опыта;
- кто будет собирать, сохранять, сопровождать данные;

- организационных уровней, которые будет обслуживать хранилище (обслуживание более чем одного организационного уровня часто приводит к потребности в более чем одной базе данных);
- степени детализации данных;
- процедур, которые должны использоваться для динамичного ведения и верификации данных в базе данных;
- количества персонала, который должен будет иметь доступ к данным (с разграничением полномочий);
- необходимости в визуализации определений данных, для того чтобы потребители могли ассоциировать данные с описательной информацией, нужной для правильного использования данных.

При выполнении технологической подготовки может помочь контрольный листок в виде таблицы (таблица 4.5).

Таблица 4.5. **Контрольный листок технологической подготовки**

№ п/п	Задачи	Отметка о выполнении (%)			
		Элемент данных			
		1	2	…	n
1	Определить элементы данных				
2	Определить частоту сбора данных и контрольные точки в процессе, в которых будут выполняться измерения				
3	Определить затраты времени, необходимые для переноса результатов, собранных в каждой контрольной точке, в базу данных или их доставку потребителям				
4	Разработать формы для сбора и фиксации данных				
5	Разработать процедуры для сбора и фиксации данных				
6	Определить, как данные хранятся, и как к ним будет обеспечиваться доступ. Установить, кто несет ответственность за проектирование БД и за ввод и ведение данных				
7	Определить, кто будет собирать данные и иметь к ним доступ. Назначить ответственных за эти действия				
8	Определить, как данные будут анализироваться				
9	Определить, как будет составляться отчет о собранных данных				
10	Идентифицировать служебные инструменты, которые нужно разработать или приобрести для того, чтобы помочь автоматизировать процесс				
11	Подготовить методики по выполнению сбора данных				

4.2.11. Этап 10: подготовка плана сбора данных

Когда известно, с чего нужно начать измерения (выполнен анализ), как представляемые меры будут отображать деловые цели (проведена диагностика) и какие задачи выполнять (обеспечена технологическая готовность), можно приступать к подготовке *плана измерения*.

Для того чтобы правильно установить и структурировать ключевые аспекты измерения, которые должен охватывать план, можно использовать приведенное

ниже примерное содержание плана (эталон). Даже при наличии альтернативных структур плана, его содержание должно удовлетворять эталону (т.е. включать все указанные аспекты измерений).

ПЛАН ИЗМЕРЕНИЯ

(эталон)

1. Цели измерения

Раздел содержит перечень основных целей выполнения конкретного измерения и объясняет, почему они важны для организации, а также суммирует ожидаемые результаты измерения.

2. Описание

Раздел содержит описание истоков составления плана, целей и границ выполняемых действий и объясняет, каким образом мероприятия в плане связаны с другими процессами и действиями. Включает подразделы.

2.1. Обоснование

Подраздел содержит краткую историю событий, которые привели к появлению плана. Указывается происхождение плана, описывается работа, которая ему предшествовала, и ее участники. Запланированные действия связываются с другими выполняемыми действиями по измерению в организации (если они проводились ранее).

2.2. Цели

В подразделе перечисляют и объясняют цели, которые мотивируют запланированные действия. Включается три вида целей: (а) деловые цели, (б) цели измерения и (в) цели этого плана.

Деловые цели обусловливают важность программы измерения и уровень поддержки, которая обеспечивается высшим руководством.

Цели измерения гораздо более детальны и конкретны. Они обусловливают методы, которые будут применяться для сбора, хранения и использования результатов измерения. Каждая цель измерения должна быть идентифицирована и связана с одной или более деловыми целями.

Цели данного плана ориентированы на выполнение определенных операций. Они конкретизируют результаты, которые должны быть достигнуты, и признаки достижения. Часто наиболее эффективный способ представления этих целей – это установление критериев завершения или критериев успеха.

2.3. Сфера применения

Указывают связь мероприятий, охватываемых планом, с целями измерения, для осуществления которых он предназначен, а также ограничения по их выполнению (например, только новые проекты, только определенные подразделения, только соисполнители). Указывают, кого касаются приемы, процессы и методы измерения, и кто будет использовать результаты. Определяют период времени действия этого плана.

2.4. Связь с другими действиями по усовершенствованию процесса

Описывают действия по усовершенствованию процесса в организации, с которыми связаны мероприятия в плане. Поясняют, каким образом действия связаны с любыми целями или усилиями, которые организация, предпринимает для повышения зрелости (например, по методологии CMM) или сертификации (по ISO 9000). Описывают, как действия по измерению связаны с работой других функциональных групп и процессов в организации, как, например, оценивание стоимости, учет трудозатрат, документирование и обеспечение гарантии качества.

3. Выполнение

Раздел содержит описание стратегий осуществления запланированных мероприятий, включая стратегии обучения персонала, стратегии инкорпорации шагов по измерению в существующие приемы работы, стратегии использования результатов измерений для осуществления обратной связи в целях непрерывного улучшения самого процесса измерения и др. Включает подразделы.

3.1. Действия, продукты и задачи

В подразделе описывают программу работ по измерениям. Усилия распределяют по хорошо прослеживаемым действиям, продуктам и задачам, которые могут использоваться в качестве основы для планирования, учета, управления и контроля. Для каждого действия, продукта или задачи устанавливают ориентиры (нормативы). Определяют все ограничения и зависимости, которые воздействуют либо на график, либо на выделение ресурсов. Там, где возможно, устанавливают условия входа и выхода, которые будут определять начало и завершение задачи.

3.2. График

В подразделе указывают, когда должны быть завершены каждое действие, продукт или задача. Для отображения последовательностей и зависимостей между действиями удобно использовать сетевые графики, диаграммы процессов или подходящие альтернативные средства отображения, которые позволяют обозначать ключевые действия, события и результаты, для того чтобы отслеживать соответствие выполнения процесса планам.

3.3. Ресурсы

В подразделе указывают ресурсы применительно к распределению работ - персонал, деньги, инструменты, планы коллективной работы, компьютерные ресурсы и т.д.

3.4. Ответственности

В подразделе указывают лица или группы, которые будут нести ответственность за планирование, утверждение, финансирование, управление и выполнение программы работ. Определяют ответственных за приобретение инструментов, обучение, создание и ведение баз данных.

3.5. Измерение и мониторинг (контроль)

В подразделе указывают, как будет измеряться и анализироваться продвижение в проведении измерений и какие способы информирования заинтересованных лиц будут использованы. Определяют точки в процессе измерения, в которых будет выполняться перепланирование, и описывают, какой должна быть реакция на значительные отклонения от графика, существенные изменения в нем или пересмотр объемов финансирования.

3.6. Допущения

В подразделе описывают ключевые допущения, на которых основан этот план. К ключевым допущениям (предположениям) относятся такие, невыполнение которых может породить риски успешного осуществления процесса измерения.

3.7. Управление риском

В подразделе описывают, как будет осуществляться идентификация рисков и оперативное планирование их оценивания и отслеживания в областях риска, связываемых с выполнением программы работ по измерениям, охватываемой этим планом. Указывают действия, которые будут приниматься для контроля допущений, и механизмы реагирования в случае несоответствия допущениям.

4. Действия по поддержке процесса

В разделе описывают действия, которые будут предприниматься для поддержки и использования результатов измерений, выполненных в соответствии с требованиями раздела 3. Указывают ресурсы и ответственных лиц за обеспечение непрерывной эволюции процесса. Описывают практические приемы, которые будут использоваться для оценки и контроля эффективности измерений и оценивания их значимости для деятельности организации.

4.3. Измерения при управлении проектами и процессами

4.3.1. Измерения и деловые цели организации

Для того чтобы деятельность по измерению была экономически эффективна, она должна быть направлена на поддержку деловых целей организации, и предоставлять эффективную лаконичную информацию для принятия решений. Деловые цели и соответствующие им направления деятельности организации можно классифицировать следующим образом: управление проектами, управления процессами и собственно разработка ПС (выполнение процессов в рамках проектов).

Цель управления проектом – обеспечить соблюдение достигнутых договоренностей с заказчиком относительно стоимости, плана графика и качества поставляемого программного продукта. Ключевые проблемы управления – это те проблемы, появление которых подвергает опасности эти договоренности. Руководство проектом ПС заинтересовано, прежде всего, в построении реальных планов и контроле состояния и продвижения разработки относительно этих планов и принятых перед заказчиком обязательств.

Цель управления процессом – с одной стороны, гарантировать, что процессы внутри организации определены, внедрены и стабильно выполняются в соответствии с ожиданиями, а с другой стороны, - стремиться к такому их усовершенствованию, чтобы способствовать достижению деловых целей проектов. Без основных действий по управлению процессом, менеджеры проектов будут сталкиваться со значительным риском, как заключения договоров, так и выполнения обязательств.

Цели собственно разработки (инженерии) программного продукта - гарантировать его конкурентоспособность и эксплуатационное качество (прежде всего удовлетворенность заказчика продуктом).

Рисунок 4.15 иллюстрирует взаимосвязи между задачами управления процессом и управления проектом и показывает, каким образом действия по измерению связаны с решением этих задач.

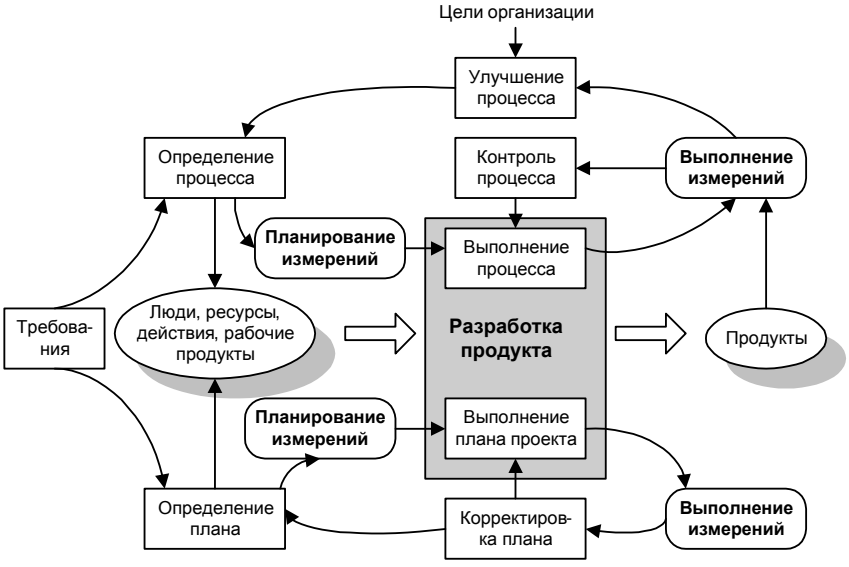


Рис. 4.15. **Место измерений в управлении проектом и процессом**

Как показывает рисунок, участники проекта разрабатывают программные продукты, основываясь на трех основных ингредиентах: требованиях, плане проекта и адаптированном процессе программной инженерии. Использование результатов измерений руководством проекта обусловлено целями:

- идентификации и спецификации требований,
- подготовки плана, конкретизирующего механизмы достижения целей,
- выполнения плана,
- отслеживания состояния и продвижения проекта по отношению к целям, указанным в плане проекта.

С целью управления процессом используются как те же самые, так и другие, схемы и результаты измерений для контроля и совершенствования самого процесса программной инженерии. Эти же схемы могут использоваться и для измерения конечного продукта с целью оценки его качества. Это означает, что для организации целесообразно выработать общую стратегию определения, внедрения, адаптации и поддержки процесса измерений, который мог бы предоставлять данные для обеих функций управления, а также оценки качества продукта.

Процесс измерения должен быть *интегрирован* с другими процессами ЖЦ ПС, поддерживая стратегии управления проектами, усовершенствования процессов и повышения качества программных продуктов. Его нужно *непрерывно развивать*, контролируя эффективность мер, метрик и действий по измерению.

4.3.2. Подходы к измерениям, базирующиеся на парадигме GQM

Хотя парадигма GQM хорошо известна за рубежом и достаточно широко используется в практике измерений, ей присущ ряд недостатков [10]. Во-первых, она не обеспечивает *повторяемости* измерений – нет гарантии, что два лица, определив одни и те же цели измерения, придут к одним и тем же вопросам и метрикам. Во-вторых, она не устанавливает *критериев завершения* формулирования вопросов и перехода к определению метрик, и, в-третьих, она не применима, если ответы на вопросы нужно искать *вне рамок* организации-разработчика ПС.

Поэтому, для практического применения в Программах измерений, на базе парадигмы GQM разрабатываются методологии, восполняющие отмеченные пробелы. Одна из них, лежащая в основе 10 шагового процесса измерений, была представлена в разделе 4.2.

Другой пример развития GQM – методология V-GQM (от Validation GQM) [11]. В отличие от оригинальной парадигмы GQM, в V-GQM процесс не завершается после анализа собранных данных. Методология предлагает три дополнительных шага – *валидацию* (проверку и утверждение метрик), *анализ ответов* на поставленные вопросы и *уточнение целей*. Модель измерения по методологии V-GQM представлена на рисунке 4.16.

Первый дополнительный шаг состоит в валидации построенных метрик и их классификации (с точки зрения пригодности для ответа на вопросы) на категории: «достаточные», «мало значащие», «шире необходимого», «обобщаемые». На основе этой классификации выполняется последующий анализ метрик. Например, если метрика классифицирована как «шире необходимого», может возникнуть вопрос, правильно ли то, что сбор соответствующих данных не был предусмотрен в плане измерений?

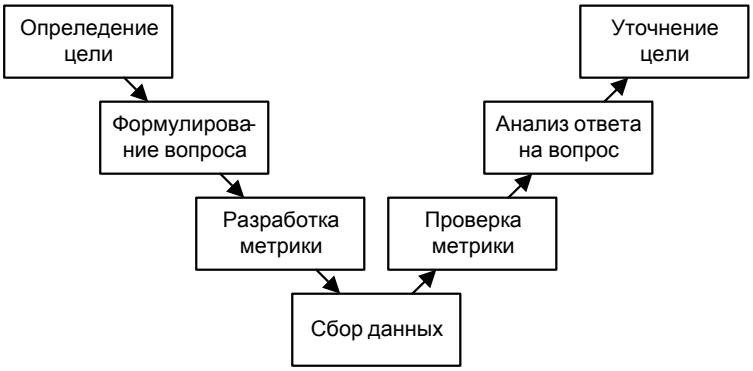


Рис. 4.16. **Шаги процесса измерения в методологии V-GQM**

Цель второго шага - анализ вопросов – их переосмысление в свете полученной информации о значимости метрик и извлечение новых вопросов (либо отказ от несущественных). Во внимание принимается также трудоемкость ответа на вопросы и польза от сбора данных.

Уточнение целей - последний шаг в V-GQM и первый в следующем цикле GQM. Могут быть поставлены новые цели, если установлены любые новые факты, требующие дальнейшего изучения. Этот шаг предполагает также пересмотр перечня вопросов и отказ от тех из них, которые не соответствуют целям.

В работе [12] представлена модель обобщенного процесса *измерения-улучшения*, авторы которой подчеркивают предназначение измерений - улучшение процессов (проектов) в организации (рисунок 4.17).

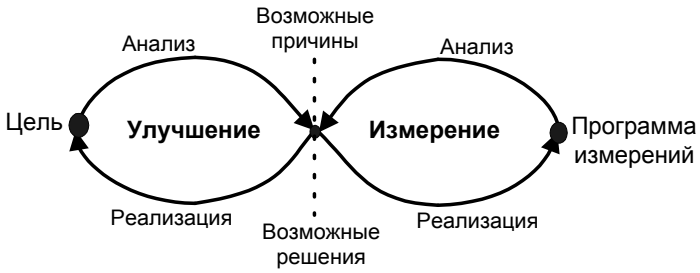


Рис. 4.17. **«Петельная» модель «улучшений, основанных на измерении»**

Процесс начинается с самой левой точки «петли» - определения целей совершенствования процессов, доходит по дуге «анализ целей» до центральной точки – принятия решения. Если информации для принятия решения достаточно, происходит возврат по дуге «реализация целей» в исходную точку. В противном случае – выполняется цикл «реализация измерений» (по правой нижней дуге) и «анализ результатов измерений». Этот цикл (выполняемый по GQM) может повторяться неоднократно до принятия решения о реализации улучшений процессов в организации.

По данным статистики, несмотря на существование множества методологий измерений, продолжительность большинства реально осуществляемых программ измерения не превышает и двух лет, и эти программы считаются неудачными. К.Деккерс, например, сообщает, что около 80 % программ измерений, проанализи-

рованных в 1998, были неудачными, а некоторые из них даже вредны из-за злоупотребления измерениями [13].

Тем не менее, есть и обширный положительный опыт выполнения программ измерений [14]. Его анализ позволяет дать такие рекомендации по их внедрению:

- начинать нужно с малого количества самых простых метрик, увеличивая объем измерений постепенно, по мере накопления опыта в организации;
- устанавливать четкие цели и планы измерений. Программу измерений оформить в виде самостоятельного проекта по совершенствованию процесса измерений (в соответствии с требованиями стандарта ДСТУ ISO/IEC 15504-7 [15]);
- создать среду, обеспечивающую безопасность процесса сбора данных и правильность самих данных. Разработчики должны иметь стимулы собирать корректные данные;
- уполномочить и предложить разработчикам использовать информацию измерений для анализа собственных действий;
- своевременно доводить информацию до всех заинтересованных сторон, участвующих в программе измерений.

4.3.3. Моделирование качества системы с использованием GQM

Несмотря на разнообразие существующих моделей качества (см. главу 3), они не находят широкого применения, потому что не отвечают трем важным требованиям:

– простота, ясность и применимость к небольшому количеству данных,
– гибкость (учет разных типов ПС и областей их применения, категорий пользователей, стадий ЖЦ) и многократное (повторное) использование,
– пригодность для измерений.

Фиксированные модели качества (например, эталонная модель в стандарте ISO/IEC 9126) определяют *постоянный* набор характеристик качества, не учитывая потребности и особенности каждой организации, программного проекта и категории пользователей (заказчиков, менеджеров, разработчиков). Возможность их *повторного* использования ограничивается подобными (схожими) проектами, хотя количественные показатели подобия (сходства), как правило, отсутствуют. Еще один их недостаток – они ориентированы на сбор данных о рабочих продуктах ПС, но не о процессах и ресурсах. Кроме того, они *не прозрачны*, логика структурной декомпозиции характеристик качества не очевидна.

Свойство *пригодности* модели для измерений предполагает возможность ее применения для выполнения поэтапных оцениваний, основанных на измерениях (в количественной или качественной форме). Между тем, объекты измерений в фиксированных моделях различны на разных стадиях ЖЦ и остается непонятным, как связывать результаты оценивания одних и тех же характеристик разных объектов? Например, каким образом подхарактеристика «анализируемость» для документации (на проектных стадиях) связана с этой же подхарактеристикой для кода (на стадии кодирования и отладки). Не ясно, каким образом использовать результаты измерений для прогнозирования качества конечного программного продукта.

Недостатком прозрачности и гибкости страдают и математические регрессионные модели (например, упомянутая в главе 3 модель COQUALMO). Их также сложно применять на ранних стадиях проекта из-за отсутствия данных, определяющих коэффициенты регрессии.

Подробные недостатки частично устраняют подходы, предлагающие разработку собственных моделей качества, достаточно прозрачных и гибких (например, GQM или Squid [16]). В таких подходах обеспечивается возможность сочетания предложенных пользователями характеристик продуктов, процессов и ресурсов, а также дается схема декомпозиции характеристик верхнего уровня на подхарактеристики и метрики. Однако, как и для фиксированных моделей, остается неясным, как взаимоувязывать результаты измерений (снизу-вверх) и прогнозировать качество. Кроме того, применение этих подходов не обеспечивает накопление опыта для последующего использования. Предполагается, что программа измерений выполняется каждый раз «с нуля», что является существенным ограничением для быстро эволюционирующих ПС с часто обновляемыми версиями или для семейств программных продуктов (так называемых product lines).

Сочетание количественных и качественных данных и распространение (продвижение) оценок по сети обеспечивают байесовские модели (см. главу 3). Однако их повторное использование в разных проектах также проблематично.

Общей проблемой, свойственной всем упомянутым классам моделей качества, является поиск ответа на два вопроса:

– как достичь согласия всех сторон, заинтересованных в моделировании качества, относительно спектра и взаимосвязи характеристик качества?

– как сделать опыт моделирования качества повторно используемым в различных проектах и организациях?

Подход к решению проблемы (под названием Prometheus) предложили Т.Пантер, А.Трендович и П.Кайзер в работе [17]. Этот подход объединяет метод моделирование качества, используемый в Squid, концепцию байесовских сетей и методы спецификации целей, вопросов и мер в парадигме GQM. Он сочетает количественные (основанные на измерении) и качественные (основанные на мнении аналитиков) подходы и учитывает различные контексты моделирования качества: взгляды заказчиков, менеджеров, разработчиков на разные категории оцениваемых объектов (процессов, продуктов, ресурсов).

Применение подхода включает три стадии работ: спецификацию модели качества, ее использование для оценивания качества и пакетирование результатов оценивания, обеспечивающее накопление опыта применения и повторного использования модели.

Общая схема подхода соответствует Squid, для спецификации модели используется парадигма GQM. Для отражения взаимосвязей характеристик (количественных и качественных) применяются байесовские сети.

Литература к главе 4

1. *Fenton N.*, Whitty R. Software Quality Assurance and Measurement. A Worldwide Perspective. // London: International Thomson Computer Press. - 1995.
2. *Fenton N.* Software Metrics: A Rigorous Approach. // London: Chapman & Hall. - 1991.
3. *Armitage J.W.*, Kellner M. A Conceptual Schema for Process Definitions and Models // Proceedings of the 3rd International Conference on the Software Process, Reston, Va., Oct. 10-11, 1994. - IEEE Computer Society Press. -1994. P. 53 – 165.
4. *ISO/IEC 15939:2002*. Software Engineering - Software Measurement Process.
5. *Goal*-Question-Metric // sel.gsfc.nasa.gov/website/exp-factory/gqm.htm

6. *Park R.E.*, Goethert W.B., Florac W.A. Goal-Driven Software Measurement - A Guidebook // Technical Report CMU/SEI-96-HB-002 – Software Eng. Inst., Pittsburgh, PA 15213. -1996. - 189 p.
7. *Basili V.R.*, Rombach H.D. The TAME Project: Towards Improvement-Oriented Software Environments // IEEE Trans. on Softw. Eng. - Vol. 14. - No. - June 1988. – P. 758-773.
8. *Basili V.R.* Using Measurement for Quality Control and Process Improvement // Second Annual SEPG Workshop, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. -June 21-22. - 1989.
9. *Rombach H.D.*, Ulery B.T. Improving Software Maintenance Through Measurement // Proceedings of the IEEE. - Vol. 77. - No. 4, April 1989. – P. 581 - 595.
10. *Fuggetta A.* et al. Applying GQM in an industrial software factory, ACM Transactions on Software Engineering and Methodologies, 1997.
11. *Olsson T.*, Runeson P. V-GQM: A Feed-Back Approach to Validation of a GQM Study, Metrics '01 - International Software Metrics Symposium, 2001.
12. *Niessink F.*, van Vliet H. A Pastry Cook's View on Software Measurement // In: Software Measurement - Research and Practice of Software Metrics, Reiner Dumke and Alain Abran (ed.), Deutscher Universitaetsverlag, Wiesbaden, Germany. – 1999. - P. 109-125.
13. *Dekkers C. A.* et al. The Dangers of Using Software Metrics to (Mis)manage, -IT Pro. - March – April, 2002.
14. *Iversen J.*, Mathiassen L. Lessons from Implementing a Software Metrics Program // Proceedings of the 33rd Hawaii International Conference on System Sciences. – 2000.
15. *ДСТУ ISO/IEC TR 15504-7:2002*. Інформаційні технології – Оцінювання процесів життєвого циклу програмних засобів - Частина 7: Настанови з удосконалення процесу.
16. *Kitchenham S.*, Linkman A.,Pasquini, V. Nanni. The SQUID approach to defining quality model // Software Quality Journal.- №6.-1997. – P. 211-233.
17. *Punter T.*, Trendowicz A., Kaiser P. Evaluating Evolutionary Software Systems // PROFES 2002 (4th International Conference on Product Focused Software Process Improvement).

Глава 5. КОНТРОЛЬ И ГАРАНТИЯ КАЧЕСТВА

5.1. Обеспечение гарантии качества в жизненном цикле

5.1.1. Процесс SQA в архитектуре процессов жизненного цикла

В архитектуре процессов ЖЦ ПС представлены два процесса, в наименовании которых содержится термин «качество» - это поддерживающий процесс «обеспечение гарантии качества» (SQA) и организационный процесс «управление качеством» (или «менеджмент качества» (от «quality management»)).

Первый из процессов связан с двумя видами деятельности – внедрением стандартов качества и соответствующих процедур в разработку ПС и оценкой приверженности этим стандартам и процедурам. Эти виды деятельности издавна сопутствуют разработке ПС и требования к ним включены практически во все комплексы национальных и отраслевых стандартов по разработке программного обеспечения, вышедших в 80-е и 90-е годы за рубежом (например, стандарты IEEE Std. 730 [1], NASA Std. 2201 [2], MIL Std. 498 [3] и др.). С 1995 года деятельность по SQA регламентируется международным стандартом ISO/IEC 12207 [4] и оформлена в виде отдельного процесса в архитектуре процессов ЖЦ.

Второй из процессов – «управления качеством» - включен в состав процессов ЖЦ с 1998 года (проекты стандартов ISO/IEC 12207 (1998 – 2001 годы), ISO/IEC TR 15504 (1998 – 2002 годы)). Включение этого процесса в архитектуру (наряду с процессами «управление проектом», «управление риском», «измерение» и др.) свидетельствует о том, что многими организациями-разработчиками ПС накоплен достаточный опыт выполнения и управления процессами программной инженерии (иными словами, преодолен барьер 2-го уровня зрелости по модели CMM). Хотя процесс SQA остается основополагающим процессом *гарантирования* качества ПС, для соблюдения общего целеориентированного подхода к выполнению действий в ЖЦ он интегрируется с процессом управления качеством, который и обеспечивает мониторинг достижения установленных и предполагаемых требований потребителя к качеству ПС.

Исходя из положений стандарта ISO/IEC 12207, *цели процесса гарантирования качества* должны быть таковыми, чтобы их достижение давало уверенность в том, что продукты ПС и процессы, применяемые для получения этих продуктов, согласуются с предъявляемыми к ним требованиями и соответствуют утвержденным планам. «В результате успешного осуществления процесса:

- будет определена стратегия выполнения действий и решения задач в рамках процесса SQA[1].

Эта стратегия должна поддерживаться стандартами для каждого процесса и продукта, процедурами и инфраструктурой, внедряемыми и сопровождаемыми на уровне организации или проектов;

- сведения о действиях и задачах по обеспечению качества будут фиксироваться и сопровождаться.

Должны быть определены учетно-отчетные документы по качеству, которые будут демонстрировать соответствие процесса и рабочих продуктов стандартам качества;

- соответствие программных продуктов, процессов и действий применяе-

[1] Комментарии к положению стандарта даны петитом и не являются частью цитаты.

мым стандартам, процедурам и требованиям будут объективно верифицироваться.

Верификация соответствия должна обеспечить необходимый уровень доверия к тому, что действия в процессах ЖЦ и порождаемые ими рабочие продукты отвечают установленным стандартам;

- будут идентифицироваться проблемы или разногласия с требованиями договора.

Должна быть организована отчетность о выполнении, отклонениях и тенденциях в деятельности по процессам ЖЦ перед соответствующей аудиторией. Любые отклонения нужно анализировать, корректировать и предотвращать в будущем их появление.

Процесс гарантирования качества должен координироваться с другими процессами поддержки, такими как Верификация, Валидация, Совместный просмотр, Аудит и Управление решением проблем, и может использовать их результаты».

Цели процесса гарантирования качества должны соответствовать *высшим целям - удовлетворения требований потребителя* к качеству ПС. Контроль соответствия целей процесса SQA высшим целям качества ПС осуществляет процесс управления качеством.

Согласно ISO/IEC 12207 *назначение процесса управления качеством* состоит в мониторинге качества ПС и гарантировании того, что ПС будет удовлетворять потребителя. «В результате успешного осуществления процесса:

- цели качества, основанные на выявленных и предполагаемых требованиях потребителя к качеству, будут установлены для различных контрольных точек ЖЦ.

Устанавливаются цели эксплуатационного, внешнего и внутреннего качества ПС. Их достижение контролируется в определенных контрольных точках ЖЦ, которые распределяются по ЖЦ таким образом, чтобы можно было количественно оценивать достигнутый уровень качества рабочих продуктов и параметры процессов, а также осуществлять регулирование процессов и прогнозирование качества конечного продукта;

- будет разработана общая стратегия достижения установленных требований.

Стратегия должна вырабатываться на уровне проекта и организации для достижения установленных целей по качеству путем определения метрик, которые будут применяться для измерения результатов деятельности в проекте, и критериев приемки, которые помогут оценить, действительно ли соответствующие цели качества достигнуты;

- идентифицированные действия относительно контроля и обеспечения качества будут выполняться, а их выполнение – подтверждаться.

Для каждой цели по качеству нужно идентифицировать действия по контролю и обеспечению качества, выполнение которых будет способствовать достижению цели. Эти действия встраиваются как *в основные процессы* ЖЦ (для анализа требований к ПС, придания продуктам свойств, обеспечивающих качество, тестирования ПС и др.), так и *в поддерживающие процессы* (SQA, V&V и др.) на уровне проекта и организации в целом;

- если цели не будут достигаться, будут приниматься надлежащие меры.

Везде в проекте и, во всяком случае, в установленных контрольных точках в ЖЦ должны применяться определенные метрики качества для *измерения* и *оценивания* того, достигнуты ли соответствующие «промежуточные» цели по качеству. Если определенные цели по качеству не достигаются, нужно принимать корректирующие или предупредительные меры на уровне проекта или организации. К корректирующим мерам относятся либо исправление продукта, полученного в результате выполнения определенного действия в проекте, либо изменение запланированного множества действий, либо то и другое. К предупредительным мерам относится модификация либо спецификаций продукта, либо проекта, либо того и другого для предотвращения возможности не достичь цели».

Таким образом, в современной модели процессов ЖЦ процесс SQA:

а) непосредственно связан с процессом управления качеством;

б) может интегрироваться с поддерживающими процессами V&V, Совместного просмотра, Аудита и (частично) Управления решением проблем либо инкорпорировать действия этих процессов;

в) опосредованно (через процесс управления качеством) связан с процессами анализа требований к ПС, измерения, управления проектом и др.

Место процесса SQA в архитектуре процессов ЖЦ показано на рисунке 5.1. Соединительные линии обозначают связь процессов по управлению или контролю. Стрелками на линиях обозначено направление информационной связи процесса SQA с другими процессами, а пунктирной линией – связь SQA с управлением проектом, которая может отсутствовать, если внедрен процесс управления качеством.

Рис. 5.1. **Место SQA в архитектуре процессов ЖЦ**

5.1.2. Задачи обеспечения гарантии качества

Существует две категории объектов обеспечения гарантии качества и связанных с ними задач: рабочие продукты и процессы ЖЦ.

Гарантируя *качество рабочих продуктов* нужно убедиться в том, что:

- все планы, требуемые по условиям договора, надлежащим образом документируются, согласовываются с договором, взаимно непротиворечивы и выполнимы;
- рабочие продукты и связанная с ними документация согласуются с условиями договора и отвечают требованиям планов;
- предназначенные для поставки продукты полностью удовлетворяют предъявляемым к ним требованиям и приемлемы для заказчика (потребителя).

Гарантируя *качество процессов* нужно убедиться в том, что:

- *все* процессы ЖЦ ПС, используемые в рамках проекта, согласуются с договором и следуют планам;
- применяемые приемы программной инженерии, среда разработки, среда тестирования и среда применения ПС согласуются с договором;
- требования договора с головным исполнителем доводятся до соисполнителей (при их наличии) и что создаваемые соисполнителями рабочие продукты удовлетворяют требованиям договора с головным исполнителем;
- заказчик и другие заинтересованные стороны обеспечиваются необходимой поддержкой в соответствии с договором, устными договоренностями и планами;
- метрики продуктов и процесса и приемы измерения (при наличии процесса измерения) соответствуют утвержденным стандартам и процедурам;
- назначенный штат исполнителей проекта имеет опыт и знания, необходимые для достижения требований проекта, и получает любую необходимую помощь в обучении.

Перечень решаемых задач может детализироваться и уточняться в зависимости от исходных требований к ПС, поставленных целей качества и условий среды разработки и применения ПС. На широту спектра задач планирования, управления и контроля качества влияют, в частности, следующие факторы:

- состав компонентов ПС (разрабатываемое или приобретаемое программное обеспечение);
- необходимость следования специализированным стандартам разработки ПС (например, отраслевым стандартам);
- наличие стандартов по качеству и соответствующего исторического опыта;
- наличие методов и специальных автоматизированных инструментов (или интегрированных сред) разработки и сопровождения ПС, а также оценивания и повышения качества;
- обеспечение всех процессов в проекте ресурсами (финансовыми, кадровыми, временными, техническими);
- категории пользователей ПС и обеспечение необходимого уровня их подготовки к использованию системы;
- сфера применения ПС (уровень необходимой целостности и безопасности системы).

Ответственность за формирование перечня задач гарантирования качества ПС и их решение возлагается на *независимую группу качества* (группу SQA), которая действует в соответствии с утвержденным планом обеспечения гарантии качества (планом качества). Непредвзятость в работе группы качества (на уровне организации или проектов) обеспечивается путем предоставления ей ресурсов, полномочий и организационной свободы от лиц, непосредственно ответственных за разработку программного продукта или выполнение процесса.

Группа качества, работающая в рамках проекта, принимает участие в формировании планов, подборе стандартов, методологий, методов и инструментов, которые должны использоваться в проекте ПС и удовлетворять проектным ограничениям и общей политике организации. Участие группы SQA в этих работах способствует получению гарантий, что разработанные планы, а также стандарты и процеду-

ры (или базовые определения процессов) отвечают нуждам проекта и применяются при проведении обзоров и аудиторских проверок в ходе ЖЦ.

Нужно отметить, что в зависимости от выбранной модели процессов ЖЦ для конкретного проекта, включающей определенное подмножество процессов, действий и задач, на группу качества могут возлагаться обязанности по выполнению не только собственно процесса SQA, но и других процессов ЖЦ, в том числе процессов Измерения, V&V, Аудита и Управления решением проблем. В этом случае она совмещает сбор информации о процессе и продуктах с ее расширенным анализом, установлением причинно-следственных связей в проекте и выработкой рекомендаций для руководства проектом. В группу качества могут, например, включаться инженер по качеству, инженер по надежности, инженер по безопасности, представители группы измерения и группы тестирования. При функционировании на уровне организации группа качества осуществляет также обратную связь к базовым процессам ЖЦ организации и контактирует с группой инженерии процесса разработки с целью усовершенствования процессов. Кроме того, организация-разработчик может пользоваться услугами органов независимой верификации и валидации (IV&V, от Independent V&V), аудита и сертификации систем качества.

Для успешного функционирования группы качества важно обеспечить надлежащий уровень документирования стандартов и процедур, поскольку действия SQA по мониторингу процессов и оценке продуктов основываются на определенных правилах установления меры соответствия проекта стандартам и регламентированным процедурам.

Руководство NASA GB A201 выделяет, например, следующие категории стандартов, обеспечивающих нормативную информационную поддержку SQA [5]:

- *стандарты документирования*. Определяют форму и содержание документации по планированию и контролю (управлению) процессов, а также документации на продукт, и обеспечивают непротиворечивость документов в проекте;
- *стандарты проектирования*. Определяют форму и содержание проекта. Обеспечивают правила и методы для преобразования требований к ПС в проект программного обеспечения и для его представления в документации проекта;
- *стандарты кодирования и интерфейсов*. Определяют язык, на котором должен быть написан код, и любые ограничения на использование особенностей языка. Указывают принятые структуры языка, соглашения о стиле, правила представления структур данных и интерфейсов в определенной парадигме программирования.

Этот список можно дополнить другими категориями стандартов, например:

- стандарты спецификации требований;
- стандарты сопровождения;
- стандарты управления конфигурацией;
- стандарты измерения (объема, сложности и других атрибутов ПС);
- стандарты оценивания качества;
- стандарты оценивания процессов;
- стандарты планирования отдельных процессов (в частности, планирования управления проектом, качеством, риском, конфигурацией, безопасностью).

Перечень действующих стандартов по упомянутым категориям представлен в приложении 3.

Все процессы, выполняемые в ЖЦ проекта, должны иметь документированные определения, а также описания процедур и методов выполнения отдельных действий. Современные методы и средства, которые могут использоваться группой качества при выполнении процесса SQA и других процессов поддержки, кратко описаны в разделе 5.4, а также в главах 6, 7 и 10.

Перечень стандартов, применяемых процедур и методов должен определяться при планировании создания ПС и фиксироваться в *плане управления проектом* (SPMP, от Software Project Management Plan), который, в частности, содержит раздел, касающийся процессов контроля разработки ПС.

Результатом осуществления SQA являются отчеты руководству, включающие описание обнаруженных недостатков и рекомендаций по приведению разработки в соответствие со стандартами и/или процедурами.

5.1.3. План качества

Осуществление процесса SQA регламентируется разработанным и документально оформленным, реализуемым и сопровождаемым (а при необходимости и актуализируемым) Планом выполнения действий и решения задач по процессу гарантирования качества для ЖЦ проекта (сокращенно, *Планом качества* или SQAP (от Software Quality Assurance Plan)). План описывает, каким образом организация-разработчик гарантирует обеспечение качества ПС. План должен включать следующее:

- стандарты, методологии, процедуры и инструменты для выполнения действий по гарантированию качества (или ссылки на них в официальной документации организации);
- процедуры для проверки (обзора) договора и условия их координации;
- процедуры для идентификации, сбора, накопления, сопровождения и размещения отчетов о качестве;
- ресурсы, план-график работ и ответственности за проведение действий по гарантированию качества;
- описание отдельных (избранных) действий и задач из других поддерживающих процессов, таких как Верификация, Валидация, Совместный просмотр, Аудит и Управление решением проблем. Хотя многие аспекты качества ПС описываются в различных планах, например, в Плане управления конфигурацией, Плане верификации и валидации, Плане обеспечения безопасности функционирования ПС и др., без единого *Плана качества* эти отдельные планы могут оказаться взаимно не согласованными, а некоторые аспекты качества ПС упущенными.

Масштаб, сфера применения и содержание SQAP должны соответствовать размеру и сложности программной системы и риску, который может возникнуть в связи с отказами системы.

План может существовать в виде отдельного документа или быть частью плана обеспечения гарантии качества крупной системы, включающей ПС в качестве подсистемы. Он может ссылаться на другие планы в проекте, например, план управления риском, V&V и др. (рисунок 5.2).

Наряду с другими рабочими продуктами проекта План качества должен находиться в сфере управления конфигурацией и подвергаться проверке со стороны руководства проекта.

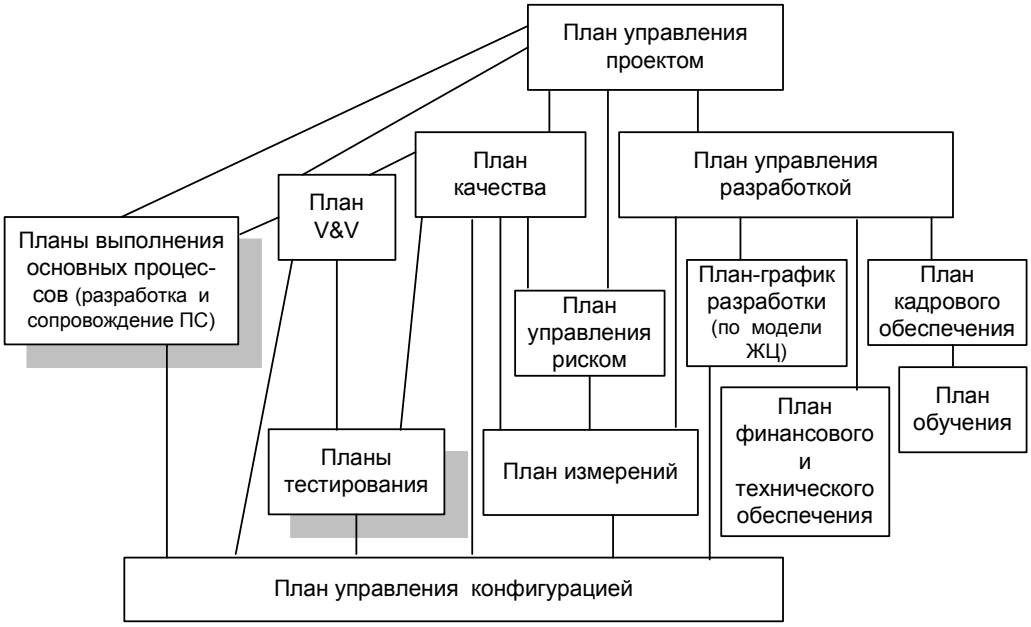


Рис. 5.2. **Иерархия планов при разработке ПС**

Структура и содержание SQAP регламентируется IEEE Std. 730 (рисунок 5.3).

План обеспечения гарантии качества ПС

1. Введение
   - 1.1. Назначение
   - 1.2. Сфера применения
   - 1.3. Определения и сокращения
   - 1.4. Ссылки
2. Управление
   - 2.1. Организация
   - 2.2. Задачи
   - 2.3. Ответственности
3. Документация
4. Стандарты, процедуры, соглашения и метрики
5. Обзоры и аудиторские проверки
6. Испытания (тестирование)
7. Сообщения о проблемах и корректирующие действия
8. Инструменты, технологии и методологии
9. Контроль кода
10. Контроль носителя
11. Контроль поставщика
12. Сбор, ведение и хранение отчетов (записей)
13. Обучение
14. Управление риском

Рис. 5.3. **Структура Плана качества ПС**

План может быть дополнен другими разделами, обеспечивающими охват требований конкретного проекта ПС. Если проект предполагает разработку нескольких программных продуктов, - может быть создано несколько планов SQAP, отражающих специфику каждого из них.

Разъяснения по составлению SQAP содержатся в руководстве IEEE Std. 730-1 [6] и вкратце приводятся ниже применительно к отдельным пунктам плана.

Организация SQA. Описывается структура SQA, основные организационные элементы SQA и связи между ними, характер организационной независимости группы SQA от организации-разработчика (группы разработки).

Задачи управления SQA. Описываются основные задачи, которые должны решаться при проведении SQA:

- определяются границы фрагмента ЖЦ ПС, охватываемого SQA;
- перечисляются задачи, выполняемые в рамках SQA. Эти задачи подробно описываются в разделе «Обучение» SQAP;
- устанавливаются взаимосвязи между задачами SQA и действиями по V&V в контрольных точках обзоров проекта.

Ответственности SQA. Идентифицируются организационные элементы, отвечающие за решение каждой задачи SQA. Указываются лица, отвечающие за план SQA, а также лицо, в целом несущее ответственность за качество ПС.

Документация. Составляется перечень документов, находящихся под контролем SQA. Он должен в основном совпадать со списком, представленным в плане управления проектом ПС. Определяется, каким образом группа SQA будет проверять адекватность каждого документа. Как минимум, перечень контролируемых документов должен включать: спецификацию требований к ПС; описание проекта ПС; план верификации и валидации ПС; отчет о верификации и валидации ПС; документацию пользователя; план управления конфигурацией ПС и др.

Стандарты, процедуры, соглашения и метрики. Описываются все стандарты, процедуры, соглашения и метрики, которые будут использоваться в процессе разработки, и определяются этапы ЖЦ ПС, к которым они будут применены. Указывается, каким образом будет гарантироваться согласованность с каждым стандартом, процедурой, соглашением и метрикой.

Список стандартов, процедур, соглашений и метрик, подлежащих применению на различных этапах ЖЦ, может включать стандарты документирования, кодирования, комментирования, процедуры тестирования, метрики и др.

Обзоры и аудиторские проверки. Описываются все виды проверок, проводимых в контрольных точках процесса разработки с целью обеспечения гарантии качества. Устанавливаются порядок и методы проведения каждого обзора технических аспектов разработки (технического обзора) и аспектов управления разработкой (управленческого обзора), аудиторской проверки работ по проекту. Согласно IEEE Std. 730-1 в минимальном объеме перечень проверок должен включать:

- обзор требований к ПС;
- обзор предварительного проекта[2];

---

[2] Стадии «предварительное проектирование» соответствует в отечественных стандартах стадия «эскизное проектирование», а стадии «детальное проектирование» - стадия «техническое проектирование» и, частично, «рабочее проектирование».

- обзор детального проекта (критический обзор);
- обзор плана верификации и валидации;
- функциональную аудиторскую проверку;
- физическую аудиторскую проверку;
- внутренние аудиторские проверки;
- управленческие обзоры;
- обзоры плана управления конфигурацией.

Фактическое множество проверок определяется совместно руководством проекта и группой SQA.

Испытания. Описываются любые необходимые испытания (тестирование) ПС, не включенные в план верификации и валидации. Устанавливается порядок их проведения.

Сообщения о проблемах и корректирующие действия. Описывается, каким образом проблемы, выявленные в ходе разработки и непосредственно касающиеся качества продукта, будут регистрироваться, отслеживаться и решаться. В частности:

- распределяется ответственность за сообщение о проблемах и наблюдение за их решением;
- устанавливается ответственность за обеспечение гарантии, что все проблемы, непосредственно касающиеся качества ПС, решены.

Инструменты, технологии и методологии. Оговариваются все специальные программные инструменты, технологии и методологии, которые будут использоваться для поддержки действий по SQA, и назначаются ответственные лица за их внедрение и применение. Краткий обзор инструментов инженера по качеству представлен в разделе 5.4.

Контроль кода. Описывается, как исходный и объектный код будут контролироваться в ходе разработки проекта.

Контроль среды. Описываются методы и средства, используемые для идентификации носителя каждого программного продукта и документации, включая хранение, копирование и восстановление.

Контроль поставщика. Описываются средства, используемые для обеспечения гарантии, что программное обеспечение, предоставляемое поставщиком, будет отвечать установленным требованиям проекта. В частности:

- определяются методы, применяя которые можно удостовериться, что поставщики получили адекватные и полные требования;
- определяются методы, используемые для обеспечения гарантии пригодности для проекта ранее разработанного программного обеспечения;
- описываются процедуры, которые должны использоваться для обеспечения гарантии, что методы SQA поставщиков согласуются с настоящим планом SQA.

Управление риском. Указываются методы и процедуры, применяемые для идентификации, оценки, мониторинга и контроля областей риска, особенно касающихся аспектов качества ПС (надежности, безопасности функционирования и др.).

Полнота и правильность плана SQA должны проверяться и подтверждаться руководством проекта (или организации).

5.1.4. Деятельность группы качества по мониторингу процессов

Одна из наиболее важных функций SQA в процессо-ориентированной программной инженерии состоит в мониторинге и периодическом анализе выполнения процесса программной инженерии, включающего множество процессов ЖЦ, адаптированных к нуждам проекта.

Ниже кратко описаны функции SQA применительно к поддерживающим процессам ЖЦ – процессу управления конфигурацией, V&V и тестирования в интерпретации руководства NASA GB A201 по обеспечению гарантии процессов [5].

SQA обеспечивает гарантии, что деятельность по *управлению конфигурацией* программного обеспечения (SCM, от Software Configuration Management) выполняется в соответствии с планами SCM, стандартами и процедурами. SQA проверяет планы SCM на соответствие принципам (политике) и требованиям к SCM и обеспечивает отслеживание несоответствий. SQA проводит аудит функций SCM для определения приверженности стандартам и процедурам и готовит отчеты о результатах проверки.

К SCM-действиям, мониторинг и аудит которых выполняет SQA, относятся контроль базовой версии, идентификация конфигурации, контроль конфигурации, учет состояния конфигурации и установление подлинности (аутентификация) конфигурации. SQA также производит контроль и аудит библиотеки программного обеспечения.

SQA гарантирует что:

- базовые версии образованы и последовательно поддерживаются с целью использования для последующего развития и контроля базовой версии;
- тщательно выполняется идентификация конфигурации в части, касающейся присвоения имен и номеров компьютерным программам, отдельным программным единицам (модулям, компонентам) и связанным с ними программным документам;
- обеспечивается совместимость конфигурации программного обеспечения, используемой на стадиях тестирования, приемки и поставки, с соответствующей документацией;
- тщательно выполняется учет состояния конфигурации, включая регистрацию и сообщение о данных, отражающих подробности идентификации конфигурации, предлагаемые изменения в идентификации конфигурации и состояния реализации утвержденных изменений;
- путем обзоров и аудиторских проверок конфигурации устанавливается ее подлинность, то есть соответствие характеристик программных продуктов спецификациям требований и документам проекта;
- библиотеки поддержки разработки обеспечивают надлежащее ведение программного кода, документации, среды и соответствующих данных в их различных формах и версиях со времени начального утверждения или принятия и до тех пор, пока они не будут включены в заключительную версию продукта;
- должным образом выполняются одобренные изменения к базовой версии программных продуктов. Неправомочные изменения не допускаются.

Применительно к процессам *Верификации и Валидации* SQA обеспечивает гарантию качества действий по V&V, выполняя мониторинг технических обзоров, инспекций, сквозного просмотра, а также тестирования (подробно эти процессы

рассматриваются в главах 6 и 7). Роль SQA в обзорах, инспекциях, сквозном просмотре и тестировании состоит в том, чтобы наблюдать, а при необходимости, участвовать и проверять, действительно ли все действия в этих процессах четко определены, документально оформлены, запланированы и должным образом выполняются.

Формальные обзоры проводятся по концу каждой стадии жизненного цикла с целью идентификации проблем и определения, удовлетворяет ли промежуточный рабочий продукт всем предъявляемым к нему требованиям. Примеры формальных обзоров - Обзор Предварительного проекта (архитектуры) (PDR, Preliminary Design Review), Обзор Детального проекта (CDR, Critical Design Review) и Обзор готовности тестов (TRR, Test Readiness Review). По результатам обзора принимается решение о готовности (не готовности) к переходу на следующую стадию ЖЦ. Хотя решение о переходе - это решение руководства, группа SQA гарантирует соблюдение планов процессов управления и разработки и готовность продукта к передаче на следующую стадию. Она несет ответственность за достоверность информации, получаемой руководством, и принимает участие в принятии решения.

При проведении инспекций и сквозных просмотров SQA гарантирует, как минимум, что процесс должным образом завершен, и что необходимые предписания выполнены.

В ходе *формального тестирования (испытаний)* ПС SQA гарантирует его выполнение в соответствии с планами и процедурами тестирования. SQA проводит обзор документации тестирования (планов тестирования, проектов и спецификаций тестов, процедур и сценариев тестирования, отчетов по тестированию) на полноту и соответствие стандартам. SQA выполняет мониторинг тестирования и обеспечивает отслеживание несоответствий. Путем мониторинга тестирования SQA гарантирует завершенность ПС и ее готовность к поставке.

Цели SQA при мониторинге формального тестирования ПС состоят в том, чтобы обеспечить гарантии, что:

- процедуры тестирования соответствуют планам тестирования;
- процедуры тестирования поддаются проверке;
- тестируется правильная или «рекламируемая» версия ПС (путем мониторинга деятельности по SCM);
- процедуры тестирования строго соблюдаются;
- несоответствия, встречающиеся в ходе тестирования (то есть любые инциденты, не ожидаемые в процедурах тестирования), регистрируются;
- отчеты о тестировании точны и полны;
- регрессионное тестирование проводится с целью обеспечения гарантии, что несоответствия исправлены;
- решение по всем несоответствиям принимается до поставки ПС.

5.1.5. Деятельность группы качества на стадиях жизненного цикла

В дополнение к общим действиям, описанным в п. 5.1.2 и 5.1.4, существуют специфические для отдельных стадий ЖЦ действия по SQA, которые должны выполняться в ходе разработки ПС по завершении соответствующих стадий.

На стадии *принятия концепции ПС* группа SQA должна вовлекаться как в написание, так и в обзор планов управления с тем, чтобы гарантировать, что процессы, процедуры и стандарты, идентифицированные в планах, адекватны своему на-

значению, четко определены и могут быть проверены. В ходе этого этапа группа SQA обеспечивает информацией, касающейся гарантии качества, раздел плана управления проектом ПС.

На стадии *определения требований к ПС* группа SQA гарантирует, что требования к ПС полны, тестируемы и представлены должным образом в виде функциональных и нефункциональных (технических) требований к ПС и интерфейсных требований проекта системы.

Действия по SQA в ходе *предварительного проектирования* включают:

- обеспечение гарантии соблюдения утвержденных стандартов для проекта ПС, обозначенных в плане управления проектом ПС;
- обеспечение гарантии, что все требования к ПС распределены по компонентам программного обеспечения;
- обеспечение гарантии, что инструменты верификации подготовлены и содержатся в актуальном состоянии;
- обеспечение гарантии, что документы по управлению интерфейсом в проекте системы согласованы между собой и со стандартами по форме и содержанию;
- обзоры PDR-документации и обеспечение гарантии, что все работы этапа завершены;
- обеспечение гарантии, что утвержденный проект помещен в сферу управления конфигурацией.

Действия SQA в ходе *детального проектирования* включают:

- обеспечение гарантии, что соблюдаются утвержденные стандарты по проектированию;
- обеспечение гарантии, что выделенные в ходе предварительного проектирования модули включены в детальный проект;
- обеспечение гарантии, что результаты инспекций предварительного проекта учтены в детальном проекте;
- обзоры документации CDR и обеспечение гарантии, что все работы этапа завершены.

Действия SQA на стадии *реализации* включают аудиторские проверки:

- результатов действий по кодированию и проектированию, включая соблюдение графика, содержащегося в плане разработки ПС;
- состояния всех поставляемых компонентов ПС;
- действий по управлению конфигурацией и содержимого библиотеки разработки;
- системы отчетности о несоответствиях и корректирующих действиях.

Действия SQA в ходе *интеграции и тестирования* включают:

- обеспечение гарантии готовности к тестированию всех поставляемых компонентов ПС;
- обеспечение гарантии, что все тесты выполняются согласно планам и процедурам тестирования, любые несоответствия фиксируются, и по ним принимается адекватное решение;
- обеспечение гарантии, что отчеты о тестировании сформированы в полном объеме и содержат достоверную информацию;
- подтверждение, что тестирование проведено полностью и программное

обеспечение и документация готовы к поставке;

- участие в проверке готовности к формальному тестированию (предварительным испытаниям) и обеспечение гарантии, что все работы стадии завершены.

Действия SQA в ходе *приемки и поставки ПС* включают, как минимум, обеспечение гарантии эффективного аудита итоговой конфигурации ПС с целью демонстрации готовности к поставке всех поставляемых компонентов.

Во время *эксплуатации и сопровождения ПС* происходят циклы мини-разработки с целью развития или исправления ПС. В ходе этих циклов разработки группа SQA проводит работу на мини-стадиях, описанную выше.

5.2. Профили процессов контроля качества

5.2.1. Элементы профиля процесса

Термин «профиль» стал широко использоваться в программной инженерии с появлением концепции открытых систем и в связи с необходимостью определения стандартов мобильности программ и данных (например, OSE, open system environment profile). Этот термин использовал также Дж. Д. Муса в концепции инженерии надежности применительно к описанию альтернативных вариантов процесса применения ПС (functional profile, operational profile) [7].

В. В. Липаев определяет профиль как «совокупность нескольких (или подмножество одного) базовых стандартов и/или других нормативных документов с четко определенными и гармонизированными подмножествами обязательных и факультативных возможностей, предназначенную для реализации заданной функции или группы функций. Функциональная характеристика (заданный набор функций, в частности, система качества) объекта стандартизации является исходной для формирования и применения профиля этого объекта или процесса. В профиле выделяются и устанавливаются допустимые факультативные возможности и значения параметров каждого стандарта и/или нормативного документа, входящего в профиль, выбранные из альтернативных вариантов. На базе одной и той же совокупности стандартов могут формироваться и утверждаться различные профили для разных проектов ПС и сфер применения» [8].

В.В. Липаев выделяет две категории профилей в ЖЦ ПС:

1) профили, регламентирующие архитектуру и структуру ПС и их компонентов – функций, интерфейсов, протоколов взаимодействия, форматов данных и др. и

2) профили, регламентирующие процессы ЖЦ.

Применительно к процессам обеспечения качества ПС предлагается конкретизировать понятие профилей второй категории - *профилей процессов*.

Любой процесс ЖЦ можно обобщенно рассматривать как агрегацию составляющих его элементов – действий, объектов, субъектов и ограничений, а под профилем процесса понимать совокупности стандартов, базовых требований, практических приемов и процедур, учетно-отчетных форм и др. для отдельных элементов в агрегации.

Схематически профиль любого процесса ЖЦ ПС представлен на рисунке 5.4.

Рис. 5.4. **Элементы профиля процесса ЖЦ ПС**

Характеристика элементов профиля процесса контроля качества[3] содержится в последующих разделах данной главы, за исключением элементов, которые были описаны ранее в этой главе, а также главе 1.

Требования к профессиональной зрелости исполнителей процесса характеризуют необходимый уровень компетентности специалистов, участвующих в его выполнении.

Стандарты рабочих продуктов процесса определяют требования к структуре и содержанию входных и выходных документов процесса, которые обычно представляют собой набор форм, разработанных и утвержденных для применения при выполнении процесса.

Метрики процесса включают множество методов и шкал для измерения стоимости, трудоемкости и продолжительности процесса, а также размера и сложности объектов деятельности процесса.

Профиль стандартов процесса составляют международные и другие стандарты, касающиеся планов процесса, а также методов и средств выполнения как процесса в целом, так и отдельных действий в процессе.

Если к разрабатываемой ПС предъявляются *повышенные требования* относительно безопасности функционирования, защиты информации, целостности системы (как, например, в критических системах управления ядерными реакторами), - это может накладывать отпечаток на спектр используемых методов, ресурсов и стандартов при выполнении процессов. Такого рода ограничения обычно регули-

---

[3] Исключительно для удобства изложения в данной главе используется обобщенное наименование «процесс контроля качества» для группы процессов, непосредственно связанных с качеством ПС - «обеспечения гарантии качества», «управления качеством», «измерения», «верификации» и др.

руются ведомственными стандартами (или стандартами предприятия), сужающими поле деятельности общепринятых стандартов.

Методы и средства выполнения процесса включают методическую и инструментально-технологическую поддержку выполнения процесса. Для процессов контроля качества этот элемент профиля включает методы проведения обзоров, инспекций, сквозного просмотра, сбора и обработки данных и др., а также набор стандартных инструментов статистического контроля процессов (гистограмм, диаграмм и др.), применяемых группой качества.

Далее перечисленные элементы профиля процесса контроля качества рассматриваются подробнее.

5.2.2. Требования к подготовке компетентных специалистов

В модели процессов ЖЦ ПС существует специальный организационный процесс «управление трудовыми ресурсами (кадрами)», цель которого состоит в подборе (подготовке) компетентных специалистов для выполнения работ по проектам ПС.

Компетентность специалистов образуется сочетанием знаний, мастерства (умения) и личных качеств (способствующих эффективному выполнению процессов), а они, в свою очередь, достигаются с помощью образования, специальной подготовки и личного опыта, которые должны подтверждаться перед назначением специалиста на ту или иную роль в области качества (рисунок 5.5).

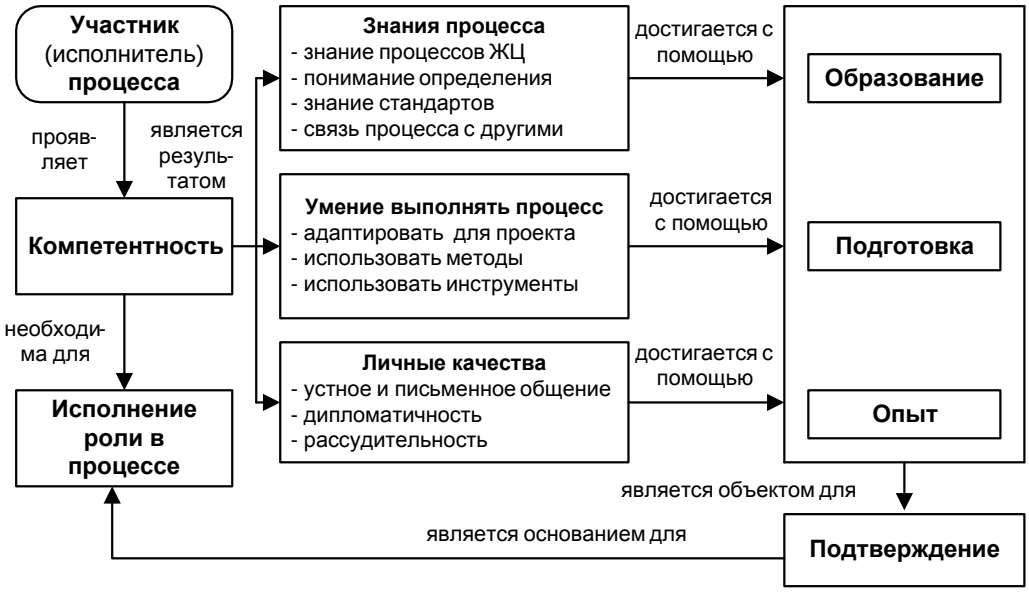


Рис. 5.5. **Составляющие компетентности для выполнения процесса**

В таблице 5.1 кратко охарактеризован круг знаний и навыков, необходимых инженеру по качеству для выполнения работ по процессам контроля качества, и указаны ссылки на доступные источники (преимущественно те, материал которых выходит за рамки данной книги).

Таблица 5.1. **Круг знаний специалистов по качеству программного обеспечения**[4]

Раздел знаний	Объем знаний	Ссылки на доступные источники
Основы программной инженерии	• Разделы SWEBOK • Структурный и объектно-ориентированный подходы к проектированию • Технологии разработки фирм Oracle, Rational Inc.	• Учебная и научная литература: [9], [10], [11], [12], [13], [14], [15] • Ядро знаний: [16] • Переводная литература: по продуктам фирм Oracle, Rational, языку UML
Стадии разработки ПС. Процессы ЖЦ	• Задачи и виды работ на стадиях • Виды и структура входных / выходных документов	• Стандарты: ГОСТ серии 34.XXX, ГОСТ серии 19.XXX, ДСТУ 3918-99 и др. • Приложение 3
Модели ЖЦ	• Каскадная, спиральная, итеративная и др. модели. Преимущества и недостатки	• Учебная и научная литература: [17] • Приложение 1
Парадигма качества	• Связь и отличие понятий и задач SQA, V&V, тестирования	• Учебная и научная литература: [8], [14], [17], [18] • Переводная литература: [19], [20] • Периодика: журнал «Проблемы программирования»
Модели качества	• Стандартизованная модель	• Учебная и научная литература: [14] • Стандарты: ДСТУ 2844-94 ДСТУ 2850-94
Основные метрики и показатели качества	• Размер, сложность, трудоемкость, стоимость разработки ПС • Меры дефектов • Надежность	• Учебная и научная литература: [15], [21], [22] • Переводная литература: [23], [24] • Периодика: журналы «Проблемы программирования», «Математические машины и системы», «Открытые системы»
Методы и инструменты анализа данных	• Реляционные базы данных • Методы статистической обработки информации • Приемы экспертного анализа	• Литература по прикладной статистике, метрологии, хранению и обработке данных, извлечению знаний • Периодика: ж. «Системні дослідження та інформаційні технології»
Методы улучшения процесса разработки	• Статистическое управление процессами • Модели достижения совершенства (зрелости) • Сертификация качества	• Литература по управлению качеством промышленной продукции • Стандарты ДСТУ серии 9000

[4] Краткую концентрированную информацию по многим вопросам инженерии качества можно почерпнуть из доступной в Интернет электронной энциклопедии «**Википедия**» (ru.wikipedia.org/wiki/), а также en.wikipedia.org/wiki/ (англ.), fr.wikipedia.org/wiki/ (франц.)

Совершенствование любого процесса ЖЦ ПС основано, прежде всего, на улучшении работы *каждого* специалиста, исполняющего определенную роль в процессе.

В 1995 году У.С. Хамфри (SEI) предложил концепцию «*персонального* (*личного) процесса участника разработки ПО*» (PSP, “Personal Software Process”) [25].

PSP – это схема и совокупность методов индивидуальной профессиональной деятельности исполнителей процессов ЖЦ, основанной на принципах планирования, учета, самоконтроля и личной ответственности за качество принимаемых решений и выполняемых действий. В основе PSP лежат принципы У. Е. Деминга и Дж. М. Джурана [26]. PSP помогает формулировать измеримые цели собственного процесса, фиксировать потраченное время, анализировать допускаемые и устраняемые ошибки, контролировать объем выполненной работы и повышать производительность труда. По словам У. Хамфри «рекомендуемая цель процесса состоит в создании бездефектных продуктов в рамках графика и запланированных затрат».

Повышение качества разрабатываемого продукта достигается по трем ключевым аспектам [27]:

- трассируя все допущенные дефекты, исполнитель определяет причины собственных ошибок и старается работать внимательнее;
- анализируя данные о дефектах, исполнитель начинает осознавать стоимость их устранения и необходимость определения наиболее эффективных способов обнаружения и локализации дефектов;
- исполнитель использует эффективные практические приемы PSP для предупреждения появления ошибок в дальнейшем.

Процесс PSP определяется на семи уровнях (рисунок 5.6а).

Каждый уровень процесса надстраивается над предыдущими, основываясь на приобретенных исполнителями навыках и накопленных исторических данных. На каждом уровне (начиная с PSP 0) добавляются новые элементы процесса и усложняются приемы работы:

- ***PSP 0.*** Исполнители используют привычные для них приемы работы и изучают основы PSP. Охватываемый материал – приемы учета времени разработки и регистрации каждого дефекта. Данные измерений используются для анализа процесса и планирования, а также в качестве эталонов при оценивании улучшений процесса;
- ***PSP 0.1.*** На этом уровне применения процесса добавляются приемы измерения объема работы (размера продукта) и формирования предложений по усовершенствованию процесса. Используется форма для регистрации обнаруженных проблем и предложений по их решению, которая помогает не упустить множество мелких деталей;
- ***PSP 1.*** Исполнители знакомятся с методом PROBE (PROxy Based Estimating, *оценивание по объектам-представителям*). Это специально разработанный для PSP метод, основанный на применении регрессионного анализа для оценки размера продукта по историческим данным и определения точности оценки;
- ***PSP 1.1.*** На этом уровне применения PSP добавляются приемы оценивания ресурсов и построения графика (расписания) работы, а также отслеживания затрат труда и сроков завершения каждой задачи. Это позволяет соотносить важность задач с трудоемкостью их решения и переупорядочивать задачи.

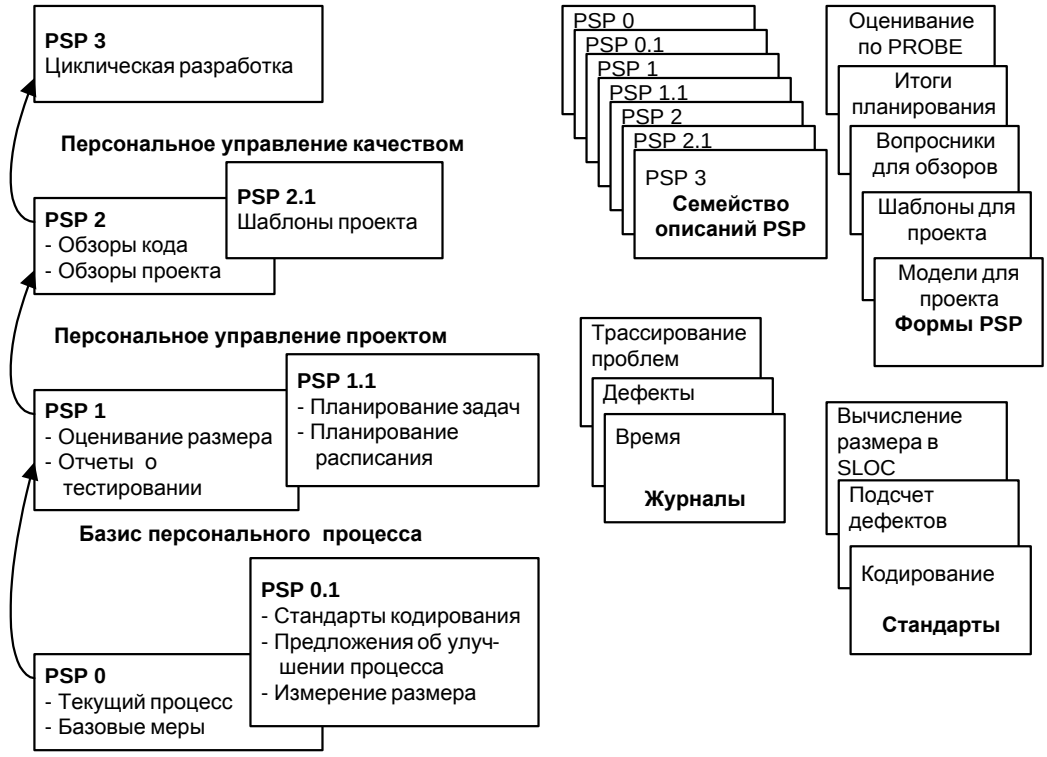


Рис. 5.6. **Эволюция и элементы процесса PSP**

- ***PSP 2.*** На этом уровне вводятся приемы обзора проекта и кода, а также оценивания качества. По данным о ранее допущенных дефектах разрабатываются персональные опросные листы (вопросники) для обзора проекта и кода.
- ***PSP 3.*** На этом уровне процесса исполнители осваивают приемы верификации проекта и методы адаптации PSP к условиям реальной среды разработки крупных проектов ПС.

На каждом уровне PSP используется множество однотипных журналов, форм, описаний (скриптов) и стандартов (рисунок 5.6 б). Описания определяют шаги процесса на каждом уровне, журналы и формы предоставляют шаблоны для учета и хранения данных, а стандарты обеспечивают нормативную поддержку работы исполнителей.

Для успешного применения PSP нужна специальная подготовка (обычно 120 – 150 часов интенсивной практической работы) и желательно последующее закрепление навыков в определенной производственной среде, что предотвратит «скатывание» к работе «по старинке».

Чтобы помочь каждому исполнителю, коллективу исполнителей (команде) и менеджерам проектов успешно применять методы PSP, У. Хамфри в 1996 году предложил концепцию «*коллективного процесса разработки ПО*» (TSP, Team Software Process) [28].

TSP предлагает схему итеративной эволюционной разработки программных продуктов с регулярным повторным планированием (настройкой) процесса (рисунок 5.7).

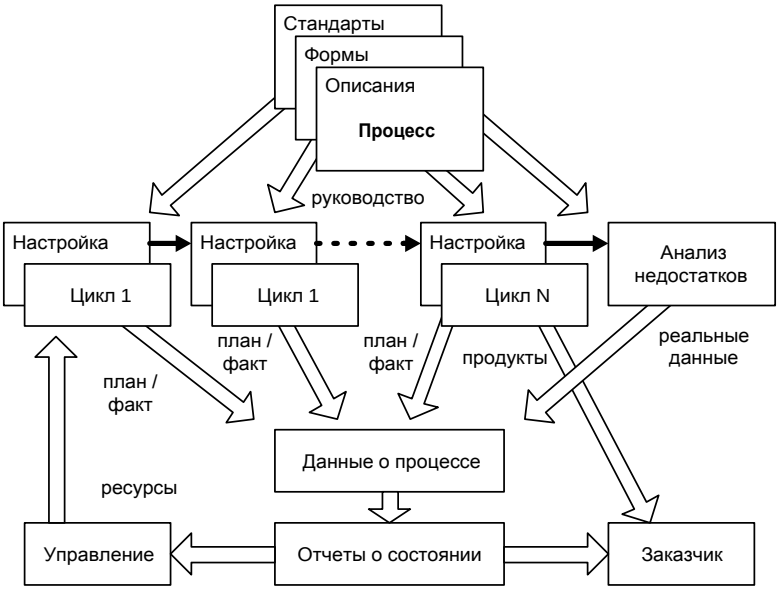


Рис.5.7. **Схема процесса TSP**

Во время 4-дневного периода настройки все члены команды определяют стратегию работы по проекту на ближайшие 3-4 месяца, выстраивают процесс и составляют коллективный и индивидуальные планы работ, которыми руководствуются в следующем цикле разработки.

Форма работы команды в период настройки – совещания (9 совещаний за 4 дня), которые готовятся командиром и проводятся по определенной схеме (рисунок 5.8).

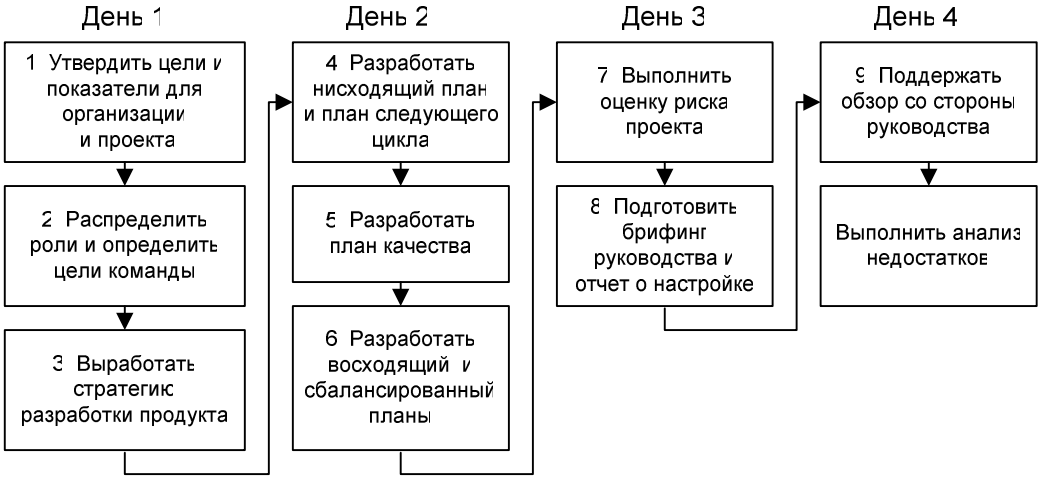


Рис. 5.8. **Схема процесса настройки**

На заключительном этапе настройки команда анализирует результаты этого процесса и готовит предложения по его улучшению. Соблюдение намеченного плана поддерживается и контролируется [28].

Практика применения процессов PSP и TSP в зарубежных фирмах-разработчиках ПС (в частности, Advanced Information Services, Boeing, Motorola Inc. и др.) свидетельствует о существенном повышении компетентности специалистов, производительности их труда, эффективности процессов ЖЦ, применяемых в проектах разработки ПС, и качества конечных программных продуктов.

Обобщив передовой опыт в области организационного строительства, управления трудовыми ресурсами и знаниями, в 90-е годы SEI предложил также модель *People CMM* (People Capability Maturity Model, модель зрелости процесса управления кадрами) [29].

Модель ***People CMM*** помогает организациям оценить зрелость кадровой политики, создать программу непрерывного совершенствования процесса подготовки и управления кадрами, а также повышения корпоративной культуры.

Модель описывает 5 уровней зрелости (стадий эволюции), на каждом из которых институциализируются практические приемы, открывающие новые возможности по организации труда коллектива (рисунок 5.9).

Рис. 5.9. **Уровни зрелости в модели People CMM**

Она может применяться как стандарт для оценивания существующей практики управления кадрами, а также приоритезации, планирования и реализации действий по ее усовершенствованию.

Структура People CMM представлена в форме матрицы, элементы которой - ключевые направления деятельности по управлению трудовыми ресурсами - взаимосвязаны и отнесены к четырем аспектам управления:

1. Развитие индивидуальных способностей
2. Формирование рабочих групп и корпоративной культуры
3. Стимулирование и управление производительностью
4. Управление кадрами.

Каждый элемент матрицы ассоциирован с аспектами управления и уровнями зрелости возможностей организации по управлению трудовыми ресурсами (рисунок 5.10).

Уровни Зрелости	**Аспекты управления трудовыми ресурсами**			
	Развитие индивидуальных способностей	**Формирование рабочих групп и корпоративной культуры**	**Стимулирование и управление производительностью**	**Управление кадрами**
5 Оптимизируемый	Непрерывное совершенствование возможностей		Организационное выравнивание производительности	Непрерывные инновации
4 Предсказуемый	Создание активов специализации (специализированных приемов работы) Наставничество	Интеграция различных видов специализации Формирование рабочих групп на доверии	Управление производительностью на основе измерений	Организационное управление Трудовыми ресурсами
3 Определяемый	Определение профилей специализации персонала Определение уровней квалификации	Определение рабочих групп Определение корпоративной культуры	Определение приемов работы с учетом специализации и квалификации Определение возможностей профессионального роста	Планирование трудовых Ресурсов
2 Управляемый	Организация обучения и повышения квалификации	Обеспечение взаимодействия и координации	Поощрение Управление производительностью Создание рабочей атмосферы	Укомплектование кадрами

Рис. 5.10. **Взаимосвязь ключевых направлений процесса в People CMM**

Чем выше уровень зрелости организации, тем больше возможностей для привлечения, удерживания, профессионального роста и эффективного использования специалистов.

Наряду с моделью SW-CMM, People CMM широко применяется в зарубежных фирмах. В Северной Америке, Европе и Австралии эту модель используют Lockheed Martin, Boeing, BAE Systems, Ericsson, IBM Global Services, Novo Nordisk IT A/S (NNIT), Citibank, U.S. Army, Advanced Information Services Inc. (AIS) и др.[29]. Более 40% организаций, достигших 4 и 5 уровня зрелости по модели CMM, используют People CMM для дальнейшего совершенствования процессов разработки программных продуктов. По результатам применения People CMM в Индии в 2001 году модель названа «оружием борьбы с утечкой мозгов».

5.2.3. Формы документов процесса контроля качества

Описание процессов в стандарте ISO/IEC 12207 выполнено по единой схеме:

- *Назначение* (*Purposes*) (цели) процесса;
- *Результаты* (*Outcomes*) выполнения процесса (продукты, артефакты, существенное изменение состояния, достижение определенных целей и требований);
- Перечень *действий* (*Activities*), составляющих процесс;
- Описание каждого действия и выполняемых *заданий* (задач) (*Tasks*).

Описания не содержат каких-либо требований к составу входных и выходных рабочих продуктов (РП) процессов. Однако, для выполнения задач SQA по проверке рабочих продуктов процессов, а также для оценивания эффективности процессов и их совершенствования, состав рабочих продуктов должен быть определен.

В части 5 стандарта ДСТУ ISO/IEC 15504-5 [30], содержащей пример модели оценивания, совместимой с эталонной, все процессы ассоциированы с входными и выходными рабочими продуктами (приложение А стандарта) и дано краткое описание этих рабочих продуктов (приложение С стандарта). Рабочие продукты отнесены к следующим категориям: РП уровня организации, РП ведения проекта и вспомогательные РП (таблица 5.2).

Описание каждого РП касается его сути (содержания), но не формы представления. Форма, в которой могут существовать однотипные рабочие продукты в разных проектах, обычно определяется применяемыми методологиями и CASE-инструментами разработки (например, SSADM, CDM Oracle), а также требованиями и рекомендациями специализированных отраслевых стандартов и руководств (например, стандарта Министерства Обороны США MIL Std. 498 «Software Development and Documentation» (Разработка и документирование ПО) [3] или стандарта Министерства Обороны Великобритании DEF STAN 0055 «Requirements for Safety Related Software in Defence Equipment» (Требования к ПО обеспечения безопасности военного оборудования) [31]).

Входными рабочими продуктами для процессов контроля качества являются практически все выходные продукты основных и поддерживающих процессов ЖЦ (документация проекта, планы и отчеты о выполнении процессов, данные проверок в контрольных точках проекта и др.). В ходе SQA проверяется их соответствие стандартам, собирается информация о «качестве» процесса программной инженерии (допущенных ошибках и причинах их появления) и контролируются темпы продвижения разработки (выполненные объемы работ, потраченное время, трудоемкость, затраты).

Таблица 5.2. **Классификация рабочих продуктов по ISO/IEC 15504-5: 2006**

Категория РП	Класс РП	Примеры рабочих продуктов
1. Организационные	1.1. Политика	Цель процесса, кадровая политика
	1.2. Процедура	Методология разработки, описание процесса, процедура поддержки потребителя, механизм связи
	1.3. Стандарт	Модель ЖЦ, стандарт кодирования
	1.4. Стратегия	Стратегия повторного использования, стратегия поставки
2. Проектные	2.1. План	План-график, план качества, план испытаний, план повторного использования
	2.2. Требования	Спецификация требований к продуктам, к услугам, к документам (внутренних или внешних)
	2.3. Проект	Технический проект, сценарий испытаний, контрольный пример, проект базы данных
	2.4. Реализация	Перечень компоновки (особенности реализации)
	2.5. Продукт	Репозиторий повторного использования, комплект поставки, система, руководство по установке, элемент конфигурации, документация пользователя
	2.6. Промежуточная поставка	Интегрированное ПО (компоненты в соответствии с перечнем компоновки)
3. Вспомогательные	3.1. Отчет	Отчет о состоянии, отчет о проблеме
	3.2. Протокол	Протокол совещания, протокол анализа риска, программа обеспечения качества, протокол аудита, результаты испытаний, протокол приемки
	3.3. Измерения	Оценка размера, критерий качества, измерения проекта
	3.4. Данные	Результаты анализа, эталонные данные, данные об эффективности процесса

По результатам всестороннего анализа полученной информации готовятся выходные рабочие продукты процессов контроля качества – планы проверок, отчеты о состоянии разработки, данные об эффективности процессов ЖЦ, рекомендации для руководителей проекта или организации. Некоторые образцы этих рабочих продуктов уже встречались ранее в этой книге (например, план измерения, план качества), остальные – будут представлены в последующих главах.

5.2.4. Ключевые метрики для контроля разработки

Ключевыми метриками, применяемыми при решении задач SQA, являются:

- метрики трудоемкости и стоимости разработки;
- метрики размера и сложности разрабатываемого программного продукта;
- метрики ошибок.

Трудоемкость и стоимость разработки. Существует немало методов оценивания трудоемкости и стоимости ПС, имеющих как достоинства, так и недостатки. Основные из них (например, SLIM, COCOMO, FPA и др.) подробно рассматриваются в главах 8 и 9.

Размер. Основная проблема и препятствие для применения ряда методов определения стоимости состоит в том, что для предсказания усилий на разработку нужно сначала *предсказать* размер конечной системы в единицах SLOC (число строк исходных инструкций кода). Существуют хорошие руководства по определению *длины* кода «готового» программного продукта. Однако, во-первых, они непригодны для прогнозирования, а, во-вторых, длина кода не всегда отражает *размер* современных программных продуктов (например, продуктов, предназначенных для интенсивной работы с базами данных средствами современных СУБД).

Достойную альтернативу измерению SLOC составляет определение размера ПО в *условных единицах функциональности* (FP, от Functional Points), выполняемое на ранних стадиях ЖЦ исходя из анализа функциональных требований к программному обеспечению. Методологию *анализа показателей функционального размера* (FPA, Function Point Analysis) предложил А.Альбрехт в 1979 году [32]. Обзор методов, разработанных на основе FPA и учитывающих *пользовательский взгляд* на разрабатываемый программный продукт, представлен в главе 8.

Сложность. Оценка таких характеристик качества ПС, как надежность или сопровождаемость, не может быть выполнена до тех пор, пока не получена хотя бы первая версия кода ПС. Однако еще до завершения разработки системы полезно знать, какие ее компоненты могут оказаться менее надежными, сложнее тестируемыми или хуже сопровождаемыми, и принимать это во внимание при распределении ресурсов разработки и тестирования. Многочисленные исследования показали, что хорошим «предсказателем» для этих целей служат метрики сложности [33]. Наиболее известными метриками сложности являются метрики М.Холстеда (1977 год) [34] и метрика «цикломатическое число» Т.МакКейба (1976 года) [35].

Как показано на рисунке 5.11, М.Холстед определил метрики сложности: размер программы (длина), размер словаря, предсказанный размер (с учетом развития) и объем программы, основываясь на синтаксических элементах программы (операторах и операндах). Он предложил также уравнения для оценки трудоемкости, времени программирования и количества ошибок.

Обозначения, введенные Холстедом:

$\mathbf{n_1}$ - количество разных операторов
$\mathbf{n_2}$ - количество разных операндов
$\mathbf{N_1}$ - общее количество операторов
$\mathbf{N_2}$ - общее количество операндов.

Основные метрики:

Размер (длина) программы**:** $N=N_1+N_2$

Размер словаря**:** $n=n_1+n_2$

Предсказанный размер: $\widehat{N} = n_1 \cdot \log_2 n_1 + n_2 \cdot \log_2 n_2$

Объем программы: $V = N \cdot \log_2 n$

Дополнительные метрики:

Трудоемкость разработки: $E = \dfrac{n_1 \cdot N_2 \cdot N \cdot \log_2 n}{2 \cdot n_2}$

Время, требуемое на разработку: $T=E/18$ секунд.

Количество ошибок: $B = V / 3000$

Рис. 5.11. **Метрики Холстеда**

Используя положения теории графов, Т.МакКейб предложил метрику, названную цикломатическая сложность, в основе которой - анализ графа потока управления программы (рисунок 5.12).

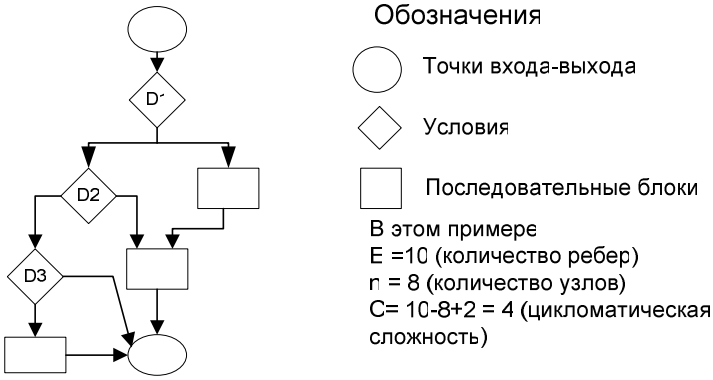


Рис. 5.12. **Пример вычисления цикломатической сложности**

Метрика сложности определяется числом линейно независимых путей в графе потока управления программы, от точки входа до точки выхода (называемых базовыми путями). Чем меньше число независимых путей, тем меньше тестов требуется для проверки всех возможных последовательностей управления элементом ПС. Вычисляется по формуле:

$$C = e - n + 2,$$

где C – цикломатическая сложность, e – количество ребер на графе, n – количество узлов. Рекомендуемое значение цикломатической сложности для обеспечения сопровождаемости и тестируемости кода - 10.

Фирмой McCabe and Associates (www.mccabe.com) создан набор инструментов структурного анализа и тестового покрытия кода, основанный на этой метрике.

Метрики Холстеда и МакКейба широко использовались в 80-е годы, хотя и подвергались критике за излишнее упрощение и очевидный «недоучет» сложности потоков данных или неструктурных программ, что и побудило их критиков к созданию множества новых метрик, учитывающих самые разные аспекты сложности программ (например, глубина вложенности, число узлов и другие). Однако эти новые метрики определялись на отдельных программах и не были чувствительны к декомпозиции системы на процедуры и функции. Позже был предложен новый класс метрик, учитывающих сложность информационных потоков между модулями [36]. Преимущество этих метрик состояло в том, что они могли применяться до этапа кодирования ПС, уже во время детального модульного проектирования ПС.

Метрики, используемые при объектно-ориентированном подходе (ООП). Хотя уже предложено немало метрик, учитывающих особенности объектно-ориентированного подхода к разработке ПС, они, как правило, не имеют под собой теоретического базиса и не утверждены формально [37]. Эти метрики предназначены для оценивания сложности таких конструкций и механизмов, как метод, класс, сцепление, связность, наследование. Наиболее распространенные метрики ООП представлены в таблице 5.3.

Таблица 5.3. **Метрики сложности, применяемые в ООП**

Наименование	Определение метрики
Цикломатическая Сложность (Cyclomatic Complexity)	Используется для оценки сложности алгоритма *метода*. Определяет число тестов, необходимых для тщательного тестирования метода. Чем ниже цикломатическая сложность, тем лучше метод. Метрика не может непосредственно использоваться для оценки сложности *класса* (ввиду действия механизмов наследования), однако цикломатическую сложность отдельных методов можно комбинировать, вычисляя сложность класса.
Взвешенное число Методов в Классе, или Взвешенная насыщенность класса (Weighted Methods per Class)	Определяется: - количеством методов, реализованных в классе, или - суммой оценок сложности методов (по цикломатической сложности каждого). Вторую меру определить сложно, так как не все методы могут быть доступны в иерархии классов (из-за механизмов наследования). Метрика используется для определения времени и усилий, необходимых на разработку и сопровождение класса. Чем больше число методов в классе, тем выше потенциальное влияние на классы-наследники и тем меньше возможности повторного использования.
Количество вызываемых удаленных методов Number Of Remote Methods	Удаленный метод - метод, который не определен в классе и его предках. При формировании значения метрики просматриваются все конструкторы и методы класса, и подсчитывается количество вызываемых удаленных методов.
Отклик на класс (Response for a Class)	Общее количество методов, которые могут вызываться экземплярами класса. Вычисляется как сумма числа локальных методов и числа удаленных методов. Чем больше число методов, которые могут быть вызваны из класса с помощью сообщений, тем сложнее класс, больше коммуникаций с другими классами.
Недостаток связности методов (Lack of Cohesion of Methods)	Вычисляется как число пар методов, не имеющих общих атрибутов, минус число пар методов, имеющих хотя бы один общий атрибут. Отрицательный результат отождествляется с нулем. Низкая связность – признак большой сложности и необходимости разделения класса. Высокая - свидетельство небольшой сложности класса и потенциальной возможности повторного использования.
Сцепление между классами (Coupling Between Classes)	Мера относительной независимости класса от других классов. Вычисляется как количество отдельных иерархий классов (*не связанных* между собой иерархическим наследованием), для которых данный класс является сцепляющим (то есть, зависящих от данного класса). Чем слабее сцепление классов, тем лучше.
Глубина Дерева Наследования (Depth of Inheritance Tree)	Максимальное число шагов по иерархии классов от данного класса до вершины (корня) иерархии (количество классов-предков в иерархии). Чем глубже находится класс в иерархии, тем больше методов наследуется, тем класс сложнее, однако выше потенциальные возможности повторного использования методов.
Количество наследников (Number of children)	Число непосредственных подклассов класса в иерархии наследования. Чем больше, тем больше вероятность неудачной абстракции класса, тем выше сложность, с другой стороны, тем больше возможность повторного использования.

Метрики ошибок. Обнаружение и своевременное устранение ошибок – основная задача процессов контроля качества ПС. Между тем, ни за рубежом, ни в Украине, не достигнуто согласия по определению основных понятий в этой области – дефект, ошибка, отказ. Эти понятия по-разному определяются не только в научной литературе по качеству и надежности программного обеспечения, но и в стандартах. Проще всего поступили разработчики стандарта IEEE Std.1044 «Standard Classification for Software Anomalies» (Стандартная классификация аномалий программного обеспечения), которые предпочли использовать единый термин «аномалия» (anomaly) вместо других – error, fault, failure, incident, flaw, problem, gripe, glitch, defect или bug[5] - что для целей этого стандарта оправданно. В Украине положение усугубляется внесением «погрешностей перевода» специальных терминов в переводную литературу. Поэтому в данном изложении мы даем свое толкование используемых терминов и указываем их приемлемые английские эквиваленты.

Существуют глубокие причинно-следственные связи между отказами ПС в эксплуатации, дефектами в поставляемом программном обеспечении, ошибками разработчика и изъянами в процессах создания ПС (рисунок 5.13).

Отказ *(failure)* - событие перехода ПС из работоспособного состояния в неработоспособное или получения результатов, которые находятся вне области допустимых значений.

Отказы ПС могут быть обусловлены как внешними причинами (ошибками элементов среды функционирования, в том числе человека-оператора), так и внутренними причинами - дефектами в ПС.

Дефект *(defect)* в ПС – запись элемента программы (кода) или текста документа (рабочего продукта), использование которой может привести к событию, заключающемуся в неправильной интерпретации этого элемента компьютером (ошибке (fault) в программе - ошибочному состоянию программы) или человеком (ошибке (error) исполнителя – заблуждению исполнителя).

Дефект всегда является следствием ошибки исполнителя процесса на любом из этапов разработки. Дефектом могут обладать спецификации требований, проектные документы, тексты кода, эксплуатационная документация и т.д. Выходные рабочие документы одних процессов, содержащие не устраненные при проверке дефекты, служат входными документами для других процессов, а дефекты в них – источником ошибок исполнителей этих процессов. Кроме того, ошибки исполнителей могут являться следствием *изъянов* в определении процессов (неправильная последовательность действий, неправильно выбранный инструмент и др.), способствующих неправильной интерпретации исходной информации человеком и принятию неверных решений, или просто недостаточной профессиональной зрелостью. Ошибки исполнителей, в свою очередь, приводят к дефекту в рабочем продукте, и цикл «*ошибка (error) – дефект (defect) – ошибка (error)*» повторяется.

Дефекты в коде, не обнаруженные в результате проверок текста кода, служат источником потенциальных ошибок и отказов ПС. «Сработает» дефект или нет, зависит от того, по какому сценарию будет работать с программой пользователь, и «столкнется» ли обработчик кода с неправильным элементом.

---

[5] Правильно перевести эти слова на русский язык можно лишь в том случае, если точно знать контекст их применения в тексте.

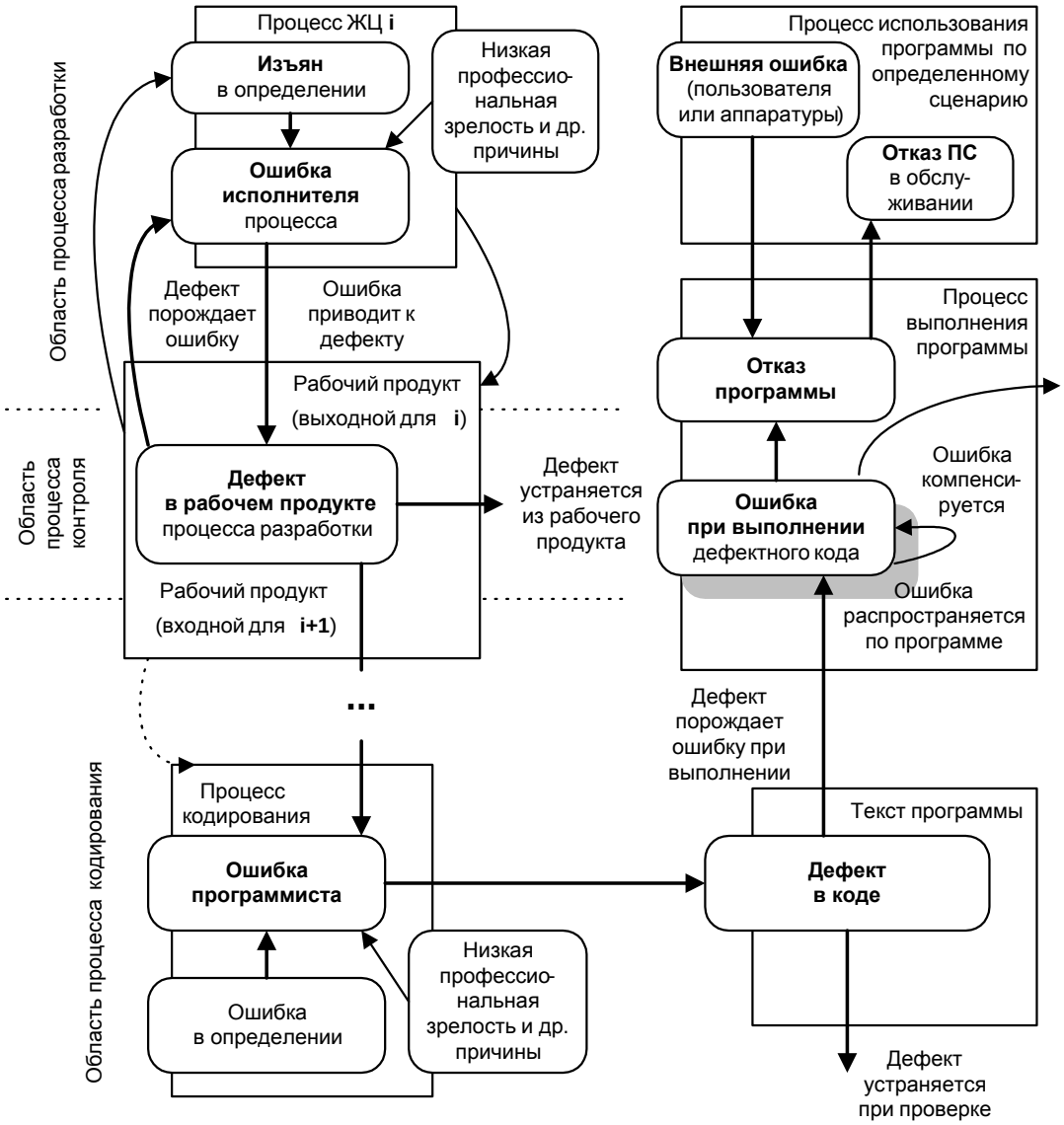


Рис. 5.13. **Связь ошибок, дефектов и отказов**

Если дефект «срабатывает» - возникает цепь ошибок, передаваемых от модуля к модулю. Если ошибка не компенсируется в программе (благодаря встроенным в программу средствам отказоустойчивости или в результате случайного «срабатывания» другого дефекта), - она может привести к аномалии в функционировании ПС. Считать ли аномалию отказом – зависит от определения понятия отказ в спецификации требований к разрабатываемой ПС. Обычно говорят, что в результате выполнения программы произошел *инцидент* (incident), который проявился в виде определенной аномалии. Последующий анализ инцидента и аномалии может показать наличие *проблемы* (problem) или ее отсутствие. О проблеме составляется отчет, который в виде входного документа направляется процессу «Управление ре-

шением проблем». Таким образом, схему отказа программы можно представить цепочкой

дефект в коде *(defect)* *[-* ***ошибка*** *(fault)]* *-* ***аномалия*** *(anomaly)* *=* ***отказ*** *(failure).*

Основными процессами, обычно исследуемыми в проблематике инженерии качества, являются процессы внесения ошибок, устранения дефектов и процесс отказа, а ключевыми метриками – «количество дефектов», «плотность дефектов» (количество обнаруженных дефектов, деленное на размер программы) и «интенсивность отказов системы» (число отказов за определенный промежуток времени).

Количество или плотность дефектов, обнаруженных в программе в ходе тестирования, нельзя считать хорошими индикаторами качества ПС, в большей степени они служат индикаторами серьезности процесса тестирования [38]. Тем не менее, регистрация дефектов определена как обязательное требование базового стандарта по качеству - ISO 9000-3:1997[6]. Регистрация дефектов служит основой для построения банков исторических *эталонных данных* по программным проектам и продуктам, которые могут использоваться для сравнения вновь разрабатываемых ПС с аналогами и улучшения процессов программной инженерии. Некоторые из них представлены в таблице 5.4 [39].

Таблица 5.4. **Эталонные данные по программным проектам**

Характеристика	**Показатели высокого потенциального качества ПС**	**В среднем**
Плотность дефектов в программном продукте, который реализует совсем новые функции		0,97 дефектов/УЕФ 14 дефектов/KSLOC
Плотность дефектов в программном продукте, который реализует не новые функции		0,14 дефектов/УЕФ 2 дефекта/KSLOC
Плотность выявленных дефектов в ходе разработки (для ПС в области организационного управления)		1,2 дефектов/УЕФ 17,4 дефектов/KSLOC
Эффективность устранения дефектов	> 95% всех дефектов устраняются во время разработки	> 56% всех дефектов устраняются во время разработки
Стабильность требований	< 2.5% изменений в базовых требованиях	
Ясность требований	> 97.5 % проверенных требований	
Плотность дефектов после тестирования	0.06 дефектов/УЕФ	0.44 дефектов/УЕФ 6,4 дефектов/KSLOC
Производительность разработки	39 УЕФ или 4250 SLOC в чел.-мес.	23 УЕФ или 2500 SLOC в чел.-мес.
Наименьшая (по отношению к среднему) плотность *потенциальных* дефектов в артефакте проекта (в расчете на УЕФ)	В требованиях - 0.20 В проектных решениях - 0.25 В коде - 0.25 В документации - 0.20 Не точно устраненных - 0.1 Всего 1.0 дефектов в УЕФ	В требованиях - 1.0 В проекте - 1.25 В коде - 1.25 В документации - 1.0 Не точно устраненных - 0.5 Всего 5.0 дефектов в УЕФ

[6] Серия стандартов ISO 9000 продолжает реформироваться. См. главу 12

Характеристика	Показатели высокого потенциального качества ПС	В среднем
Наименьшая плотность *фактических* дефектов (в расчете на УЕФ)	В требованиях - 0.02 В проекте - 0.0125 В коде - 0.003 В документации - 0.01 Не точно устраненных - 0.01 Всего 0.056 фактических дефектов/УЕФ	В требованиях - 0.16 В проекте - 0.10 В коде - 0.024 В документации - 0.08 Не точно устраненных - 0.08 Всего 0.444 фактических дефектов/УЕФ
Эффективность устранения дефектов (способность устранять дефекты без внесения новых)	CMM уровень 5 - 95%, CMM уровень 4 - 93% CMM уровень 3 - 91%, CMM уровень 2 - 89% CMM уровень 1 - 85%	

Средний показатель плотности дефектов программных продуктах – 6 дефектов в KSLOC для США и Европы и 2 дефекта в KSLOC для Японии.

В таблице 5.5 представлены данные о процессах выявления и устранения дефектов, опубликованные С. Кэном [40].

Таблица 5.5. **Структура дефектов по стадиям ЖЦ ПС (дефекты/KSLOC)**

Стадия ЖЦ	Унаследовано	Внесено	Всего присутствуют	Эффективность устранения	Устранено	Остаток
Анализ требований	Нет	1.2	1.2	Нет	Нет	1.2
Спецификация треб.	1.2	8.6	9.8	74%	7.3	2.5
Проектирование	2.5	9.4	11.9	61%	7.3	4.6
Кодирование	4.6	15.4	20.0	55%	11.0	9.0
Автоном. тестирование	9.0	Нет	9.0	36%	3.2	5.8
Интеграц. тестирование	5.8	Нет	5.8	67%	3.9	1.9
Испытания	1.9	Нет	1.9	58%	1.1	0.8
Эксплуатация	0.8	Нет	Нет	Нет	Нет	Нет

Количество (или плотность) дефектов в программе перед началом тестирования – важная метрика для определения, с одной стороны, эффективности процессов разработки (чем меньше допускается ошибок, тем лучше), и, с другой стороны, - эффективности процесса инспекции (чем больше устраняется дефектов, тем эффективнее процесс). Количество дефектов, содержащихся в программе на момент начала тестирования, полезная метрика для раннего прогнозирования надежности ПС и параметр ряда моделей роста надежности.

5.2.5. Стандартизация в области инженерии качества

Разработку стандартов осуществляют международные и национальные органы стандартизации, основные из которых представлены в таблице 5.6 наряду с указанием их адресов в Интернет (по данным, полученным на сайте www.memst.kz/int_links.html).

Таблица 5.6. **Международные и национальные органы стандартизации**

Обозначение	Наименование (англ.)	Наименование (рус.)	Адрес в Интернет
Международные организации по стандартизации			
ISO	International Organization for Standardization	Международная организация по стандартизации	www.iso.org
IEC/CEI	International Electrotechnical Commission	Международная электротехническая комиссия	www.iec.ch
CEN	European Committee for Standardization	Европейский комитет по стандартизации	www.cenorm.be
CENELEC	European Committee for Electrotechnical Standardization	Европейский комитет по стандартизации в области электротехники и электроники	www.cenelec.be
ETSI	European Telecommunications Standards Institute	Европейский институт по стандартизации в области телекоммуникаций	www.etsi.org
EOQ	European Organization for Quality	Европейская организация по качеству	www.eoq.org
IFAN	International Federation of Standards Users	Международная федерация пользователей стандартов	www.ifan-online.org
EASC	EuroAsia Council on Standartization, Metrology, and Certification	МГС СНГ – Межгосударств. совет по станд-ции, метрологии и сертификации СНГ	www.easc.org.by
WSSN	World Standards Services Network	Всемирная сеть служб стандартов	www.wssn.net
Национальные организации по стандартизации			
ANSI	American National Standards Institute	Американский национальный институт по стандартизации	www.ansi.org
BSI	British Standards Institution	Британская организация по стандартизации	www.bsi-global.com
DIN	Deutsches Institut fur Normund	Институт стандартизации Германии	www2.din.de
IEEE	Institute of Electrical and Electronics Engineers, Inc. IEEE Standards Association	Ассоциация по разработке стандартов. Институт инженеров по электротехнике и электронике (США)	www.ieee.org standards.ieee.org
NASA	National Aeronautics Space Administration	Нац. агентство по исследованию космич. пространства	www.nasa.gov
NIST	National Institute of Standards and Technology	Нац. институт по стандартзации и технологии (США)	www.nist.gov
SAA	Standards Australia	Стандарты Австралии	www.standards.com.au
SCC	Standards Council of Canada	Совет по стандартам Канады	www.scc.ca
ГОСТ	State Committee of the Russian Federation for Standardization and Metrology	Государственный комитет Российской Федерации по стандартизации и метрологии	www.gost.ru
ДСТУ	State Committee of Ukraine for Standardization, Metrology and Certification	Государственный комитет по стандартизации, метрологии и сертификации Украины	www.dssu.gov.ua

Базовыми стандартами по контролю и управлению качеством продукции являются международные стандарты серии ISO 9000, касающиеся *Системы управления качеством*[7], и соответствующие гармонизированные стандарты Украины (приложение 3). Аннотацию на стандарты ISO серии 9000 можно получить по адресу ISO в Интернет.

Международные стандарты (МС) в области информационной технологии разрабатываются *объединенным техническим комитетом* JTC1 (Joint Technical Committee) «Information technology», образованным ISO и IEC.

JTC1 включает множество *подкомитетов* (SC, Subcommittee)[8]. Ответственным за разработку международных стандартов по программной инженерии является подкомитет ISO/IEC SC7 «Software and system engineering»[9].

Для работы по проектам стандартов в подкомитете SC7 созданы рабочие группы (WG, Working Group), основные из которых перечислены в таблице 5.7.

Таблица 5.7. **Рабочие группы подкомитета SC 7 по программной инженерии**

Группа	**Наименование (англ.)**	**Наименование (рус.)**
WG 2	System software documentation	Документация ПО
WG 4	Tools and Computer Aided Software/System Engineering (CASE) environments	Инструменты и CASE-среды
WG 6	Software product evaluation and metrics	Оценивание программного продукта и метрики
WG 7	Life cycle management	Управление жизненным циклом
WG 9	System and Software assurance (safety, security and dependability)	Обеспечение гарантий системы и ПО (безопасности функционирования, защиты информации и надежности)
WG 10	Process assessment	Оценивание процесса
WG 12	Functional size measurement	Измерение объема функциональных возможностей (размера)
WG 19	Modeling languages, metadata, Open distributed processing (ODP) framework and components	Языки моделирования, метаданные, схема и компоненты открытой распределенной обработки
WG 20	Software Engineering Body of Knowledge	Ядро знаний в области программной инженерии
WG 21	Software asset management process	Процесс управления наработками в области программного обеспечения
WG 23	Systems Quality management	Управление качеством систем
WG 24	Software Life Cycles for Very Small Enterprises	Жизненные циклы ПО для очень мелких предприятий

[7] Поскольку в украинских стандартах термин «quality management» переводится как «управління якістю», в данном изложении используется термин «управление качеством» (хотя в соответствующих российских стандартах - «менеджмент качества»)

[8] По последним данным JTC1 насчитывает 17 подкомитетов с номерами 2-37. Нумерация не непрерывная. После роспуска отдельных подкомитетов список не перенумеровывается.

[9] Адрес JTC1/SC7 в Интернет: www.sqi.gu.edu.au/sc7/mirror/ (на дату: 01.06.2006)

Предложения по разработке международных стандартов или по приданию национальным стандартам статуса международных стандартов поступают от национальных органов стандартизации. Материал стандартизации проходит несколько стадий рассмотрения и уточнения и утверждается в качестве стандарта только в том случае, если его одобрили не менее 75% стран-членов ISO, принимающих участие в голосовании по стандарту.

Основные действующие стандарты, непосредственно касающиеся инженерии качества программных систем, представлены в приложении 3.

Обозначение рабочих документов ISO отражает текущее состояние соответствующих проектов стандартов (таблица 5.8).

Таблица 5.8. **Стадии разработки стандартов и обозначение документов**

Стадия	**Тип документа**		**Описание**
	Аббревиатура	**Наименование**	
1	AWI	Approved Work Item	Утвержденный рабочий материал для стандартизации
2	WD	Working Draft	Предварительный проект для обсуждения рабочей группой
3	CD	Committee Draft	Завершенный проект для голосования и подготовки технических замечаний национальными органами стандартизации
	CD TR или TS	Committee Draft Technical Report/ Specification	
4	CDV	Committee Draft for Vote (IEC)	Окончательный проект для голосования и редакционных правок национальными органами стандартизации
	DIS	Draft International Standard	
	FCD	Final Committee draft (JTC1)	
	DTR или DTS	Draft Technical Report/ Specification	
5	FDIS	Final Draft International Standard	Подготовленный к публикации текст, предназначенный для окончательного утверждения
6	ISO	International Standard	Опубликованный документ
	ISO TR или TS	Technical Report or Technical Specification	

При необходимости ISO пересматривает принятые международные стандарты и готовит их новые редакции. Так, со времени выхода в 1991 году стандарта ISO/IEC 9126 по качеству программного обеспечения, было подготовлено ряд его новых редакций (например, ISO/IEC TR 9126, Software Engineering – Product quality – Part 1 – Part 4, стандарт в четырех частях в редакции 2001-2004 года).

Наряду с подготовкой новых редакций отдельных стандартов, производится совместная реорганизация проектов стандартов. Так, принято решение о реорганизации проектов ISO 9126 и ISO 14598 в единую серию стандартов SQuaRE (Software Product Quality Requirements and Evaluation, Требования к качеству программного продукта и оценивание).

5.2.6. Архитектура стандартов SQuaRE

Одна из основных задач серии стандартов SQuaRE (и основное отличие от текущей версии стандартов ISO/IEC TR 9126) состоит в гармонизации их положений со стандартом ISO/IEC 15939, регламентирующим процесс измерения ПС [41], а также используемой терминологии с общепринятой терминологией в области метрологии, зафиксированной в международном словаре VIM [42].

Структура новой серии стандартов SQuaRE, предложенная рабочей группой WG6, включает 14 документов, сгруппированных в 5 разделов (таблица 5.9) [43].

Таблица 5.9. **Стандарты серии SQuaRE**

Стандарты	Описание
Раздел «Управление качеством» (ISO 2500n)	
Руководство по SQuaRE	Структура SQuaRE, используемая терминология, обзор документа, круг пользователей серии, эталонные модели и др.
Планирование и управление	Требования и руководства по планированию и поддержке управления оцениванием программного продукта
Раздел «Модель качества» (ISO 2501n)	
Модель качества и руководство	Модель внутреннего, внешнего и эксплуатационного качества продукта, характеристики и подхарактеристики внутреннего, внешнего и эксплуатационного качества
Раздел «Измерение качества» (ISO 2502n)	
Эталонная модель измерения и руководство по применению	Математические определения и практические руководства по выбору и применению метрик для измерения качества продукта на соответствующем уровне (внутреннем, внешнем, эксплуатационном)
Примитивы измерения	Множество базовых и производных метрик, используемых для конструирования метрик внутреннего, внешнего и эксплуатационного качества
Измерение внутреннего качества	Множество внутренних метрик продукта, используемых для количественного измерения внутреннего качества в терминах характеристик и подхарактеристик качества
Измерение внешнего качества	Множество внешних метрик продукта, используемых для количественного измерения внешнего качества в терминах характеристик и подхарактеристик качества
Измерение качества при использовании	Множество метрик продукта, используемых для количественного измерения эксплуатационного качества. Руководства по использованию мер эксплуатационного качества
Раздел «Требования к качеству» (ISO 2503n)	
Требования к качеству и руководство по определению	Руководство по выявлению требований к качеству продукта и спецификации требований к качеству. Применяется на стадии определения требований к продукту или при формировании входных данных для процесса оценивания продукта

Стандарты	Описание
Раздел «Оценивание качества» (ISO 2504n)	
Введение в оценивание качества и общее руководство	Основные требования к спецификации и оцениванию качества программного продукта, основные понятия и концепции оценивания, общая схема оценивания качества и требования к методам измерения и оценивания программного продукта
Процесс для разработчиков	Требования и практические рекомендации по оцениванию программного обеспечения параллельно с разработкой
Процесс для потребителей	Требования, практические рекомендации и руководства по систематическим измерениям и оцениванию качества готового программного продукта при его приобретении или при модификации существующего продукта
Процесс для оценщиков	Требования и практические рекомендации по оцениванию программного продукта в ситуации, когда результаты оценивания необходимы нескольким сторонам-участникам
Документация модуля оценивания	Структура и содержание документации, предназначенной для описания Модуля оценивания продукта (см. главу 12)

Группу стандартов ISO/IEC TR 9126 (части 1 - 4) в серии SQuaRE заменяют пять стандартов из раздела «измерение качества» (рисунок 5.14) [44].

Рис. 5.14. **Структура раздела серии ISO 2502n (предложение WG 6)**

Понятийная база этих стандартов расширена новым элементом - «примитив измерений» [45]. Рабочая группа WG6 предлагает набор примитивов измерений (базовых и производных метрик), которые могут использоваться в ходе жизненного цикла ПС для построения метрик внутреннего, внешнего и эксплуатационного качества. Это, например, такие базовые метрики как «время», «число функций», «число ошибок», «количество данных», «число операций», «число тестовых ситуаций». Однако, не очевидно, что новый элемент «приживется» в инженерии качества[10]. Серия стандартов SQuaRE продолжает развиваться.

О текущем состоянии этого перспективного проекта можно узнать из документов рабочей группы WG6 в Интернет[11].

---

[10] Анализ терминологии в VIM и ISO/IEC 15939 и концепций измерений, проделанный А.Абраном и его соавторами в [43], убеждает в обратном.

[11] Например, используя в качестве поисковой строки следующую: ”ISO/IEC/JTC1/SC7 WG6” “SQuaRE project”.

5.3. Инструменты анализа качества

5.3.1. Применение графических инструментов в инженерии качества

Для эффективного выполнения своих функций, группа SQA должна оценить собственные потребности в инструментах, поддерживающих процессы измерения, обеспечения гарантии качества и управления качеством, а также верификации и валидации.

В этом обзоре кратко рассматриваются графические инструменты, появившиеся несколько десятилетий назад и испытанные временем. Многие из них имеют японское происхождение, поэтому в литературе по качеству и управлению процессами (а также в Интернет) их можно найти под общим заголовком «японские инструменты» анализа качества [46]. Кроме того, отдавая дань создателям, некоторые инструменты называют их именами, например, карты У.Шухарта (контрольные карты) или диаграмма К.Исикавы (причинно-следственная диаграмма). Инструменты раннего периода уходят корнями в базовую математическую статистику и ориентированы на обработку числовых данных, а относительно новые – предназначены для визуализации логически взаимосвязанной нечисловой информации.

Достаточно полный обзор графических инструментов анализа данных содержится в электронном учебнике фирмы StatSoft (на русском языке) [47], а также в Интернет, например, на сайтах организаций SkyMark Corporation (www.skymark.com/resources/tools), Clemson University (deming.ces.clemson.edu/pub/tutorials/qctools/homepg.htm) и др. Далее перечислены лишь самые простые (базовые) инструменты.

Расслоение данных. Простейший из инструментов - *таблица расслоения данных*. Предназначена для классификации данных по нескольким критериям. Пример таблицы расслоения – форма сбора данных о дефектах (рисунок 5.15). Здесь все множество данных «расслаивается» исходя из двух критериев – «тип дефекта» и «дата обнаружения».

Тип дефекта	**Количество дефектов на дату**			**Сумма**
	Дата 1	**. . .**	**Дата n**	
Тип 1				
. . .				
Тип m				
Сумма				

Рис. 5.15. **Пример таблицы расслоения**

Для сопоставления данных по отдельным слоям удобно использовать диаграммы: круговые, точечные, столбиковые, ленточные и др.

Этот вид графических инструментов поддерживается средствами ведения электронных таблиц (например, Excel). Для построения диаграмм используется Мастер диаграмм.

Диаграмма выполнения. *Диаграмма выполнения (Run Chart)* или *временной график* - удобный инструмент контроля хода выполнения процесса в масштабе времени и выявления закономерностей в наступлении каких-либо контролируемых событий (например, изменения производительности труда в зависимости от дня недели или скорости передачи информации по каналам связи в зависимости от

времени суток). По оси ***x*** на графике – градация времени, а по оси ***y*** – градация события (рисунок 5.16).

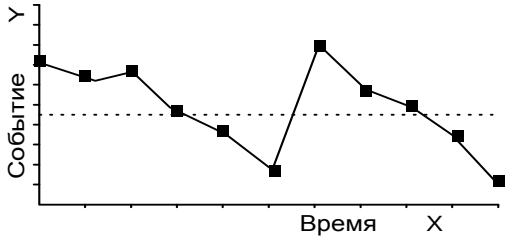


Рис. 5.16. **Вид диаграммы выполнения**

Диаграммы выполнения могут использоваться, также, для контроля устойчивости процесса и обнаружения вариабельности, вызванной особыми причинами. В этом случае для построения графика должно использоваться не менее 25 данных результатов наблюдений. Если 8 и более точек на графике оказывается по одну сторону центральной линии графика, параллельной оси ***x***, это свидетельствует о влиянии особой причины на процесс. Если график делает 6 последовательных «скачков» в одном направлении, это указывает на тренд процесса под влиянием особой причины. А если определенный фрагмент графика появляется 8 и более раз – нужно попытаться избавиться от особой причины. Нужно отметить, однако, что более мощным инструментом для контроля устойчивости процесса являются контрольные карты.

Диаграмма рассеяния. *Диаграмма рассеяния (Scatter Diagram)* используется для проверки гипотез о наличии определенной взаимосвязи между двумя измеряемыми величинами (например, размером модуля и плотностью дефектов, размером ПС и затратами и др.). По оси ***x*** указывается шкала измерения для одной величины (независимой переменной), а по оси ***y*** – для другой, зависимой переменной (рисунок 5.17).

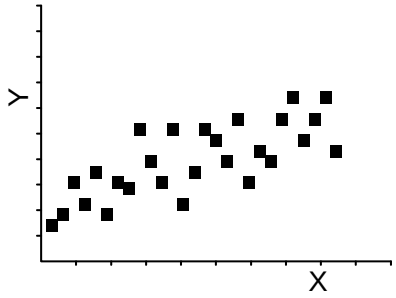


Рис. 5.*17.* **Вид диаграммы рассеяния**

Если при возрастании значений по оси ***x*** возрастают значения по оси ***y*** – существует положительная корреляция. Если при возрастании значения по оси ***x***, значения по оси ***y*** убывают – отрицательная корреляция.

Чем теснее группируются значения вокруг воображаемой кривой зависимости величин, – тем строже эта зависимость между величинами.

Если представить себе кривую зависимости сложно, - возможно зависимость отсутствует. Корреляция, однако, не всегда означает, что изменение одной величи-

ны влечет за собой изменение другой, поскольку может существовать и третья величина, влияющая на обе исследуемые. Все же построение диаграмм рассеяния полезно для выяснения, связаны ли величины каким-либо образом.

Столбиковая диаграмма. На с*толбиковой диаграмме (Bar chart)* последовательность значений представляется в виде столбцов (одному наблюдению соответствует один столбец). Если выбрано несколько переменных, - строится составная диаграмма, где все переменные отображаются одновременно в виде групп столбцов (одна группа для каждого наблюдения), что позволяет исследовать распределение данных по группам. Этот вид диаграмм удобен, например, для сравнения двух серий данных - измеренных и ожидаемых (рисунок 5.18).

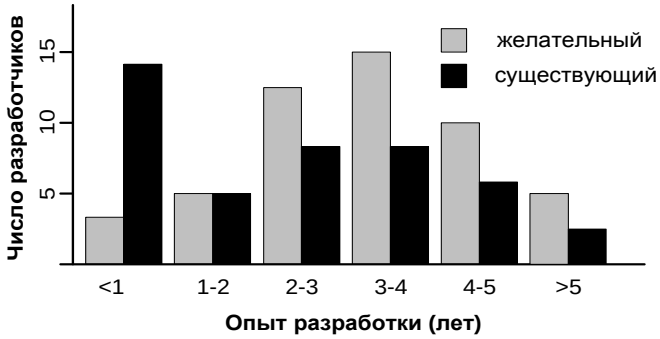


Рис.5.18. **Пример столбиковой диаграммы**

Два специальных типа столбиковых диаграмм - гистограммы и диаграммы Парето.

Гистограмма. *Гистограммы (Histogram)* создаются путем группирования результатов измерений по секциям и последующего подсчета количества «попаданий» измеренных значений в каждую секцию. Секции представляют собой непересекающиеся столбцы одинаковой ширины, в основании которых - равные отрезки действительной прямой, покрывающие область возможных значений измеренных величин (непрерывная шкала). Высота столбца пропорциональна числу попаданий значений в секцию.

Гистограммы могут использоваться для характеристики наблюдаемых значений практически любых атрибутов продукта или процесса (например, размера модулей, времени устранения дефекта, времени между отказами, количества дефектов, найденных при тестировании и др.).

Пусть, например, числа в таблице 5.10 – это данные 80 измерений количества времени (в часах), ежедневно затрачиваемого группой сопровождения ПС после ее сдачи в эксплуатацию, полученные менеджером за 16 недель наблюдений. Исследуется изменение количества часов потраченного времени по дням одной недели и по неделям.

Гистограмма для данных о процессе сопровождения ПС представлена на рисунке 5.19.

Высота столбца гистограммы отражает количество элементов наблюдений из таблицы 5.10, которые могут быть отнесены к одной секции.

Таблица 5.10. **Время, потраченное группой сопровождения ПС**

Недели	Пн	Вт	Ср	Чт	Пт	Среднее	Диапазон (размах)
1	50,5	43,5	45,5	39,8	42,9	44,44	10,7
2	44,3	44,9	42,9	39,8	39,3	42,24	5,6
3	48,8	51,0	44,3	43,0	51,3	47,68	8,3
4	46,3	45,2	48,1	45,7	44,1	45,88	4,0
5	40,6	45,7	51,9	47,3	46,4	46,38	11,3
6	44,4	49,0	47,9	45,5	44,8	46,32	4,6
7	46,0	41,1	44,1	41,8	47,9	44,18	6,8
8	44,9	43,4	49,0	49,5	47,4	46,84	6,1
9	50,0	49,0	42,6	41,7	38,5	44,36	11,5
10	44,5	46,5	41,7	42,6	41,7	43,40	4,8
11	43,8	41,8	45,5	44,5	38,6	42,84	6,9
12	37,2	43,8	44,8	43,5	40,9	42,04	7,6
13	50,0	43,4	48,3	46,4	43,4	46,30	6,6
14	52,3	45,2	42,2	44,8	42,8	45,46	10,1
15	50,0	46,2	47,4	42,2	47,0	46,56	7,8
16	47,3	49,7	48,0	42,0	41,0	45,60	8,7
В среднем						45,03	7,59

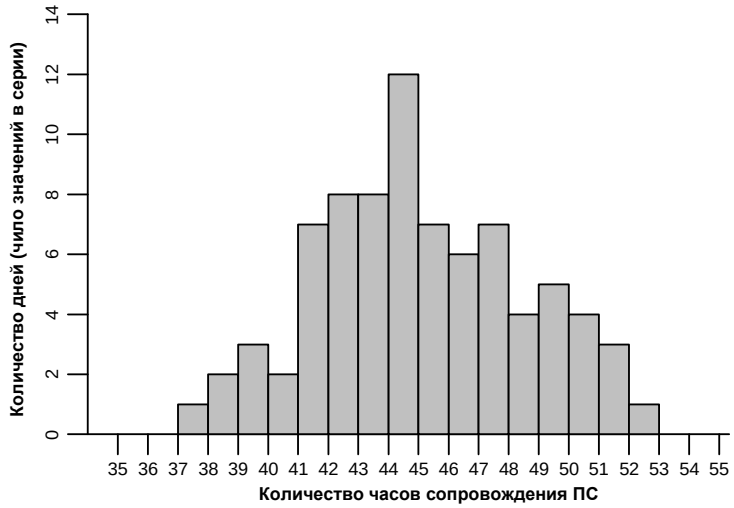


Рис. 5.19. **Пример гистограммы**

Например, 1 раз на сопровождение было потрачено от 37 до 38 часов работы персонала в день (значение 37,2), 2 раза – от 38 до 39 часов (значения 38,5 и 38,6) и т.д. Больше всего раз встречались данные в секции 44-45 (12 раз).

Диаграмма Парето. Полезность применения *диаграмм Парето (Pareto Chart)* в инженерии качества состоит в том, что они могут использоваться при анализе процессов для привлечения внимания разработчиков к тем факторам, которые оказывают *наибольшее* общее влияние на разрабатываемую систему, и отсеивания несущественных факторов, что позволяет вовремя выбрать правильное направление работы по улучшению процессов и продуктов. С помощью этих диаграмм можно, например, найти и графически проиллюстрировать ответы на такие вопросы, как «на предупреждении каких ошибок нужно сосредоточить усилия?» или «какие действия в процессе вносят наибольший вклад в проблему?» и др.

Диаграмма Парето представляет собой множество столбцов одинаковой ширины, упорядоченных по *убыванию* высоты и отображающих, в простейшем случае, счетчики частоты или количества наблюдаемых данных, сгруппированных по определенному критерию (рисунок 5.20).

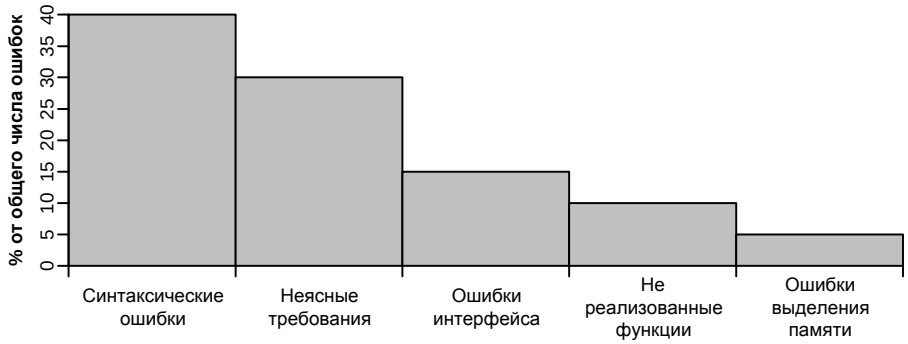


Рис. 5.20. **Пример диаграммы Парето**

При проведении более сложных видов анализа диаграммы Парето могут ранжировать факторы по убыванию оценок экономических последствий их влияния (в стоимостном выражении).

Диаграммы Парето удобны, также, для сравнения состояний процессов или продуктов до и после усовершенствований и определения эффективности выполненных действий по усовершенствованию. Однако, если процессы неустойчивы, сравнение диаграмм Парето может привести к неправильным выводам, поскольку демонстрируемые различия могут быть обусловлены неконтролируемыми вариациями процессов. Это значит, что диаграммы Парето лучше применять для исследования устойчивых процессов.

Контрольные карты. *Контрольные карты (Control Chart)* используются для анализа стабильности процессов. Все они имеют схожую структуру (рисунок 5.21).

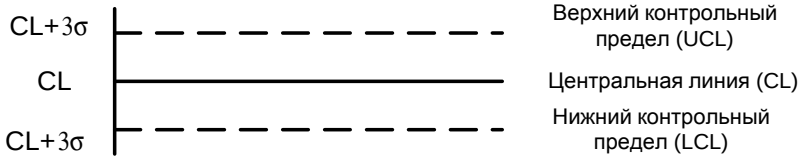


Рис. 5.21. **Структура контрольной карты**

Карта содержит *центральную линию* (CL, Central Line), которая представляет среднее значение исследуемой характеристики процесса, и две линии контрольных пределов – *верхний контрольный предел* (UCL, Upper Control Limit) и *нижний контрольный предел* (LCL, Lower Control Limit), которые вычисляются по одной из возможных мер вариабельности процесса, например, выборочным средним или диапазонам (размахам) значений.

Карты для выборочных средних часто называют X-картами и строят для получения ответа на вопрос, «изменяется ли основная тенденция процесса в период наблюдения?».

Карты для размахов выборки называют R-картами и строят для получения ответа на вопрос, «не влияют ли на дисперсию наблюдаемых величин какие-либо особые причины при выполнении процесса?».

X- и R-карты используются совместно для идентификации точек, в которых процесс выходит из-под контроля. Данные измерений характеристик продукта или процесса группируются в последовательные множества (подгруппы) и результаты группирования используются для вычисления пределов, по которым проверяется стабильность (контролируемость) процесса.

Традиционно (по Шухарту) контрольные пределы устанавливаются на уровне $\pm 3\sigma$ (сигма), где σ - это оцененное стандартное отклонение статистики, нанесенной на карту. У.Шухарт и его последователи обосновывали выбор пределов на уровне именно $\pm 3\sigma$ тем, что такой выбор позволяет не делать предположений о виде распределения подразумеваемой естественной вариации процесса и, кроме того, дает необходимую чувствительность к особым вариациям процесса, отсеивая несущественные общие. Поскольку значения, наносимые на карту, представляют статистические величины, они могут быть любой функцией измеримых свойств продукта или процесса. Эти значения определяются путем измерения таких атрибутов, как размер группы разработчиков, продолжительность работы между контрольными точками в ЖЦ или событиями, время, потраченное разработчиками на решение задачи, производительность, текучка кадров, число модулей, имеющих дефекты, число обнаруженных дефектов на 1000 модулей и т.д.

На рисунке 5.22 представлены примеры X- и R-карт, составленных по данным о процессе сопровождения ПС из таблицы 5.10.

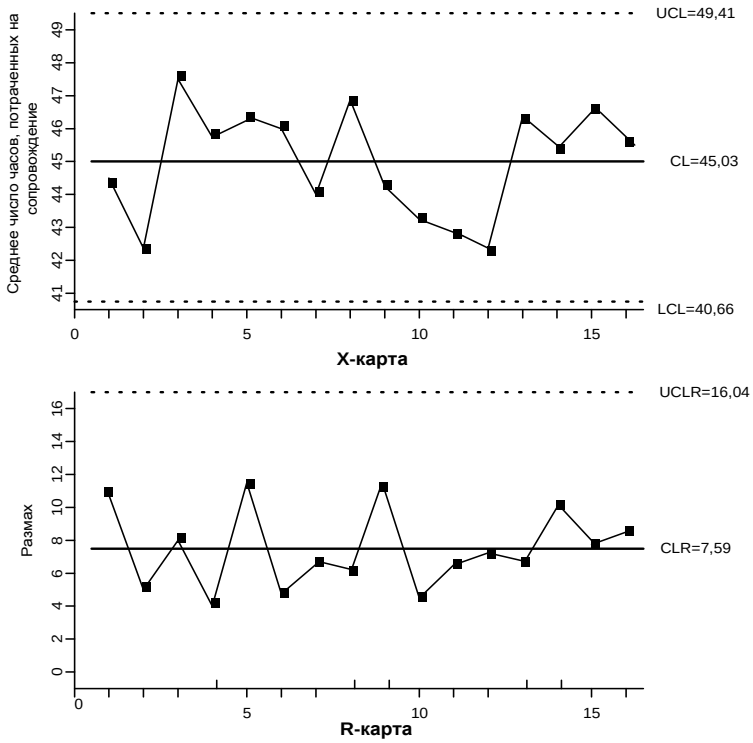


Рис. 5.22. **Примеры X-карты и R-карты**

Для расчета UCL и LCL используется специальная процедура, шаги которой перечислены в таблице 5.11.

Таблица 5.11. **Процедура построения контрольных карт X и R**

Шаг	Действия
1	Вычислить среднее и размах для каждой подгруппы данных (подвыборки)
2	Вычислить полное среднее, X, усредняя каждое из средних для k подгрупп
3	Вычислить средний размах, R, усредняя каждый из размахов для k подгрупп
4	Центральной линией для X-карты будет X. Центральной линией для R будет R
5	Найти по справочным таблицам значения A_2, D_3 и D_4 при размере подгруппы n
6	Вычислить A_2 R
7	Вычислить UCLx = X + A_2 R
8	Вычислить LCLx = X - A_2 R
9	Вычислить $UCL_R = D_4$ R
10	Вычислить $LCL_R = D_3$ R

В этой процедуре A_2, D_3 и D_4 – константы преобразования средних и размахов подвыборок в несмещенные оценки для пределов $\pm 3\sigma$. Их значения определяются по справочной таблице 5.12 или подобным ей таблицам, которые можно найти в литературе по статистическому управлению процессами.

Таблица 5.12. **Константы для вычисления контрольных пределов**

Объем подвыборки (n)	A_2	D_3	D_4
2	1,880	-	3,268
3	1,023	-	2,574
4	0,729	-	2,282
5	0,577	-	2,114
6	0,483	-	2,004
7	0,419	0,076	1,924
8	0,373	0,136	1,864
9	0,337	0,184	1,816
10	0,308	0,223	1,777

В рассматриваемом примере на рисунке 5.22 процесс устойчив. Если результирующая кривая на графике выходит за верхний или нижний контрольный предел, это указывает на существование проблем с процессом. Но, даже если кривая на картах X и R не выходит за контрольные пределы, однако ее конфигурация выражает определенную *тенденцию* поведения для следующих друг за другом выборок, это может указывать на то, что процесс все же вышел из-под контроля.

Считается, что процесс выходит из-под контроля, если:

- одна или более точек кривой вышли за контрольные пределы;
- обозначился сдвиг процесса – восемь точек подряд находятся по одну сторону CL;
- две из трех последовательных точек вышли за пределы уровня 2σ;
- четыре из пяти последовательных точек вышли за пределы уровня 1σ;
- повторяется один и тот же фрагмент кривой (определенная последовательность из семи точек);

- несколько точек находятся вблизи контрольных пределов.

Нужно отметить, что кроме X- и R-карт, существуют и другие полезные виды контрольных карт, используемые в инженерии качества ПС (например, XmR, U, Z, S и другие карты) [48].

Причинно-следственная диаграмма. *Причинно-следственная диаграмма (C&E, Cause and Effect diagram)* используется для того, чтобы установить множество возможных причин появления одной изучаемой проблемы. Другие названия этого графического инструмента - *диаграмма Исикавы*, *«рыбья кость»* (fish-bone), *диаграмма анализа первопричин* (root cause analysis). Наиболее эффективно применяется в условиях коллективной работы над проблемой в режиме мозгового штурма. Пример диаграммы показан на рисунке 5.23.

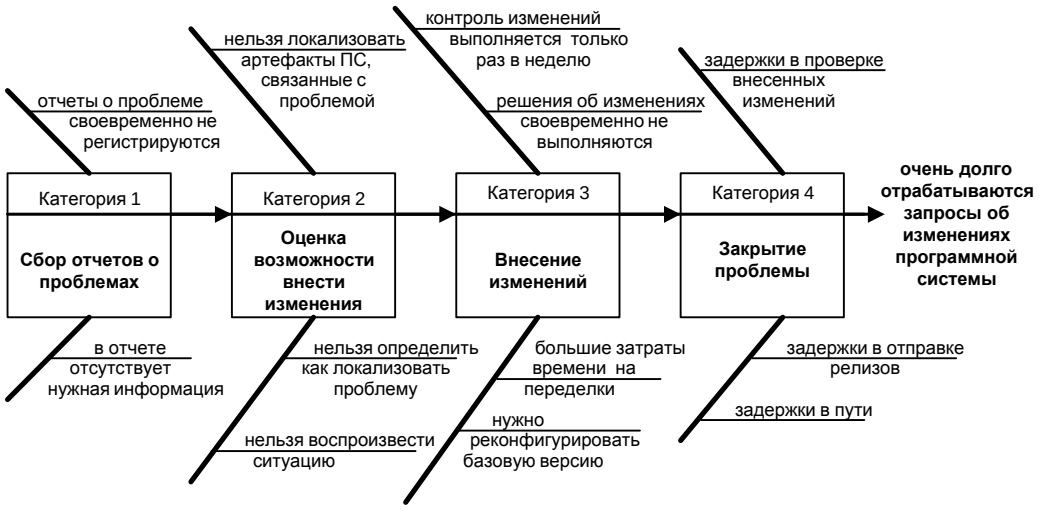


Рис. 5.23. **Пример C&E диаграммы для процесса сопровождения системы**

Основная линия диаграммы (хребет рыбы) отображает проблему (или характеристику, которую нужно улучшить или проконтролировать) и имеет обозначение на правом конце. Ответвления от основной линии (главные кости, отходящие от хребта) диаграммы размещаются под углом к ней и соответствуют главным причинам (или категориям причин), непосредственно связанным с проблемой (для наглядности категории указаны в блоках на основной линии диаграммы). Обычно выделяют 3 – 6 категорий причин. Далее каждое ответвление вновь интерпретируется как основная линия – представитель проблемы более низкого, второго уровня рассмотрения, и уже относительно этой линии производится ветвление причин. Практически полезная глубина ветвления – 4-5 уровней проблем. Таким образом, C&E диаграмма используется для представления всех обнаруживаемых потенциальных или реальных причин, детализируемых и упорядочиваемых по уровням важности. Она помогает установить *первопричины* существующей или возможной проблемы, идентифицировать области риска и ранжировать риски по степени относительной важности.

C&E диаграмма может также изображаться в виде лежащего дерева (рисунок 5.24).

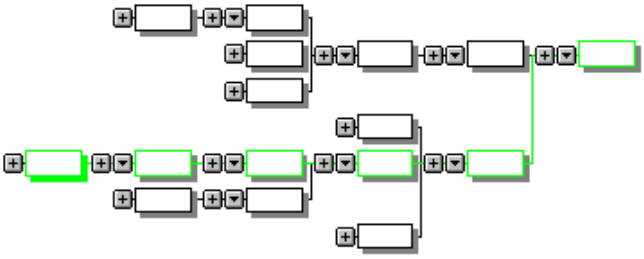


Рис. 5.24. **Пример древовидного изображения диаграммы C&E на компьютере**

Изображение в виде дерева имеет преимущества перед «скелетом рыбы», поскольку с ростом сложности диаграммы становится все сложнее находить и сравнивать разбросанные по диаграмме «равновеликие» причины, вносящие одинаковый вклад в исследуемую проблему. При древовидной структуре все элементы диаграммы, находящиеся на одном и том же каузальном уровне, выравниваются по вертикали.

Существует множество *других графических средств*, применяемых для представления и анализа информации о процессах и программных продуктах. Они хорошо известны и не нуждаются в описании. В их числе:

- диаграмма «сущность-связь» (entity-relations diagram),
- диаграмма потоков управления и данных (flow chart),
- контрольный листок (check sheet) (пример приведен в главе 4),
- матричная диаграмма (пример – совокупность матриц в методологии QFD),
- вопросник (checklist).

5.3.2. Применение методов интеллектуального анализа данных

Конец XX века отметился появлением и бурным развитием технологии *интеллектуального анализа данных* (ИАД) или *добычи данных* (DM, Data Mining), обусловленным необходимостью аналитической обработки сверхбольших объемов информации, накапливаемой в современных хранилищах данных [49], [50]. Элементы этой технологии находят применение и в инженерии качества ПС [51].

Процесс добычи данных в инженерии качества заключается в *извлечении ранее неизвестной* и потенциально полезной информации из множества разрозненных (сырых) исторических данных измерений с целью выявления скрытых закономерностей и принятия обоснованных решений относительно контроля качества программных продуктов и процесса разработки.

Особенность анализируемых данных состоит в том, что они могут представлять результаты измерений, выполненных в *различных* организациях с применением *разных* метрик (отвечающих целям этих организаций) и структурированных таким образом, что к ним сложно непосредственно применить автоматизированные или полу автоматизированные методы извлечения знаний без предварительной обработки. Суррогатные наборы данных могут содержать лишние или несущественные данные (шумы), массивы с пропущенными данными и др. Поэтому перед применением к ним алгоритмов DM данные из различных источников должны быть просеяны, интегрированы и согласованы.

Процесса добычи данных включает 4 шага:

Шаг 1. Выбор типов данных, для которых будет применяться алгоритм извлечения знаний, идентификация расположения данных, получение к ним доступа и транспортировка в одно централизованное хранилище – репозиторий - для последующего применения.

Шаг 2. Предварительная обработка данных, которая заключается в просеивании данных, форматировании, преобразовании типов, нормализации, выявлении неопределенных или отсутствующих значений, а также, «порождении» новых атрибутов (как, например, атрибут «серьезность дефекта», который может быть порожден для сущности «дефект» как функция от «времени обнаружения», «времени локализации» и «времени устранения» дефекта).

Шаг 3. Применение алгоритма добычи данных – извлечение интересной и потенциально полезной информации путем применения моделей и методов выявления знаний. Предварительно обработанные данные могут использоваться, например, для:

1. разработки точной классификационной модели предсказания количества дефектов в программных модулях (сама модель будет представлять собой новую информацию);
2. автоматического построения диаграмм, отражающих интересные ассоциации между характеристиками проекта и профилями ошибок (новая информация будет получена в результате последующего исследования диаграмм);
3. получения целостной картины проективных характеристик и профилей ошибок с помощью инструментов визуализации данных (новая информация будет извлечена в результате интерактивного исследования визуальных образов данных).

Эти примеры представляют основные направления добычи данных – построение моделей, автоматическое исследование образцов (pattern extraction) и визуальное исследование данных (visual data exploration).

Шаг 4. Интерпретация (ассимиляция) извлеченной информации – оценивание надежности и эффективности разработанных моделей или определение экспертом в данной проблемной области степени новизны и полезности новой информации.

В поиске интересной информации шаги процесса добычи данных могут повторяться неоднократно – аналитик может изменить критерий выбора данных, если посчитает информацию неинтересной; произвести дальнейшую фильтрацию данных, если они не адекватны алгоритму DM; уточнить алгоритм, если в результате его применения обнаруживается слишком мало интересных фактов.

Качество извлеченной информации зависит как от правдивости исходных «сырых» данных, так и от методов, используемых в процессе добычи данных, а также способностей аналитиков правильно выполнять действия по DM.

Выбор методов добычи данных определяется задачами, которые должны решаться с использованием анализа исторических данных в области программной инженерии. Основные задачи классифицируются по следующим категориям:

Задачи оценивания и прогнозирования. Исследование атрибутов множества сущностей (продуктов, процессов и ресурсов) и использование их значений для квантификации нового исследуемого атрибута. Например, прогнозирование трудоемкости разработки ПС по известным характеристикам проекта. Для решения этих задач могут использоваться такие методы DM, как *нейронные сети* (artificial neural network) [52] или *байесовские сети* (Bayesian belief network) [53], которые позво-

ляют строить достоверные прогнозы, не уточняя конкретный вид тех зависимостей, на которых прогноз основан. Подробнее о байесовских сетях – в п.5.3.3.

Задачи классификации. Исследование атрибутов заданной сущности и отнесение ее к определенному классу или категории, основываясь на значениях этих атрибутов. Например, отнесение модулей к одному из двух классов – «с большой вероятностью дефектов» или «с малой вероятностью дефектов». Для решения этих задач могут использоваться методы *деревьев классификации* (classification trees) [54] или *оптимального сокращения набора данных* (optimal set reduction) [55] (это метод иерархической классификации).

Задачи выявления ассоциаций. Поиск ассоциативных групп значений атрибутов, т.е. значений, почти всегда появляющихся вместе. Например, определение того, какие значения двух атрибутов - «опыт» и «подготовка» - для сущности «группа разработки» ассоциируются с характеристикой качества конечного продукта - «надежность». Для решения этих задач может использоваться метод *анализа взаимосвязанных событий* (анализа структуры транзакции) (в экономике этот метод называют методом *анализа структуры покупки* (market basket analysis)) [56] или метод *анализа значений атрибута* (attribute focusing) [57]. По первому методу анализируются «интересные» события и связанные с ними значения атрибутов, а по второму - «интересная» функция распределения значений или «интересные» корреляции между значениями атрибутов. Установленные факты отображаются с помощью столбиковых диаграмм, упорядочиваемых по степени «интереса», который они представляют для экспертов. Вопросы, которые возникают у экспертов в процессе анализа диаграмм, и побуждают к извлечению новых знаний.

Задачи кластеризации. Отличаются от задач классификации тем, что классы или категории, к которым должны быть отнесены сущности, заранее не заданы и должны быть сформированы в результате определения множества однородных подгрупп данных. *Разделение* популяции данных на подгруппы производится не в соответствии с какой-либо моделью классификации, а по измерениям *расстояния* между ними. Если, например, база данных содержит записи о модулях системы и нужно разделить все множество записей на группы, руководствуясь значениями атрибута «количество модификаций», расстояние будет измеряться разницей между количеством модификаций разных модулей. Это простейший вид кластеризации по одному критерию. Обзор методов и алгоритмов иерархической и неиерархической кластеризации данных можно найти, например, в работах [58] и [59].

Задачи визуализации данных. Визуализация данных заключается в отображении многомерных данных на двумерном экране компьютера. Достигается путем связывания записей данных с «визуальными атрибутами», каждый из которых далее ассоциируется с измерением реальных данных. Хороший пример использования приемов визуализации в программной инженерии - работы Data Visualization Research Group (AT&T Bell Labs) по визуализации исходного кода программ [60].

Задачи исследования визуализированных данных. Осмысление сложной информации с помощью интерактивного управления визуализацией многомерных неструктурированных наборов данных путем построения сценариев отображения данных в режиме «что если». Обзор современных инструментов визуальной добычи данных, позволяющих эксперту в проблемной области обнаруживать интересные образы данных без использования автоматизированных алгоритмов, представлен в [51].

5.3.3. Графические инструменты для построения байесовских сетей

Существует множество инструментальных средств для построения графических моделей, основанных на байесовских сетях [61, 62].

В таблице 5.13 представлены результаты анализа возможностей некоторых *доступных* в Интернет инструментов построения байесовских сетей, функционирующих под управлением Windows. В графе “Адреса” указаны ссылки на адреса сайтов в Интернет, на которых обеспечен доступ к инструментам.

Инструменты сравнивались по критериям:

- поддержка разных *типов* случайных величин в вершинах (*Н* – непрерывные (с Гауссовским распределением), *Д* – дискретные);
- наличие *API-интерфейса* для встраивания функций сети в программные приложения на Visual Basic и VBA;
- ограничение функциональных возможностей доступной версии.

Таблица 5.13. **Сравнение инструментов построения байесовских сетей**

Название	**Адрес**	**Тип СВ**	**API**	**Ограничение**
MSBNx	[63]	Д	Да	Полная версия. Без ограничений. Не коммерческий
Netica	[64]	Д/Н	Да	Нет API в demo-версии. Коммерческий
Serene	[65]	Д/Н	Нет	Serene 1.0 Demo (до 40 вершин). Не коммерческий
Hugin	[66]	Д/Н	Да	Версия Hugin Lite 6.5 (до 50 вершин, до 500 комбинаций значений СВ). Коммерческий

На рисунке 5.25 показан пример использования MSBNx для построения байесовской сети с дискретными случайными величинами в вершинах, представленной ранее в главе 3 (в п.3.2.6, рисунок 3.9).

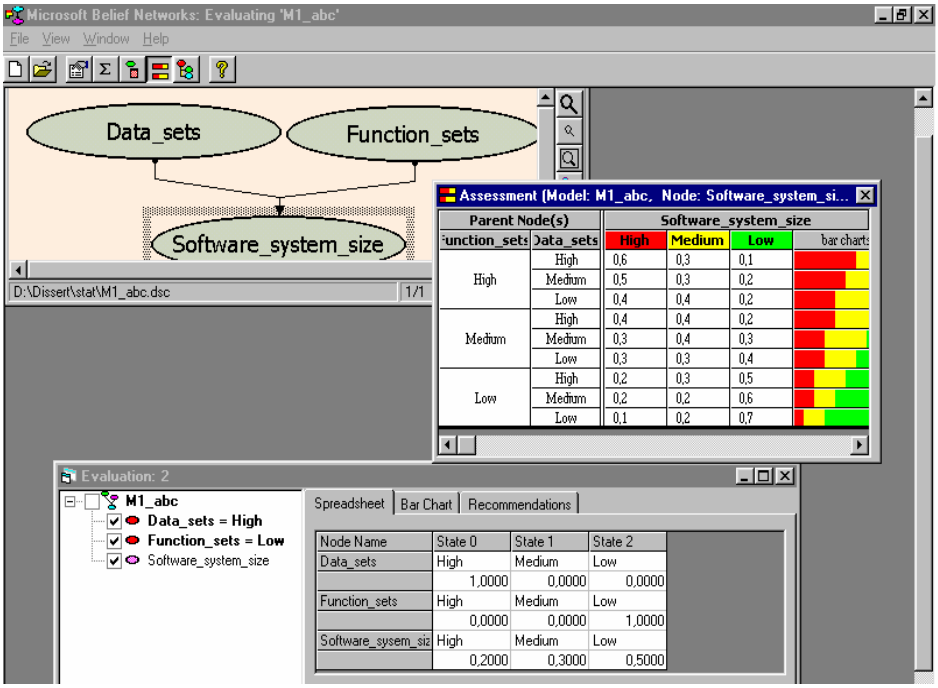


Рисунок 5.25. **Пример сети, построенной с помощью MSBNx**

На рисунке 5.26 показана инициализированная байесовская сеть для прогнозирования дефектов, представленная в главе 3 (п.3.2.6, рисунок 3.11). Сеть построена с помощью инструмента Hugin Lite 6.5.

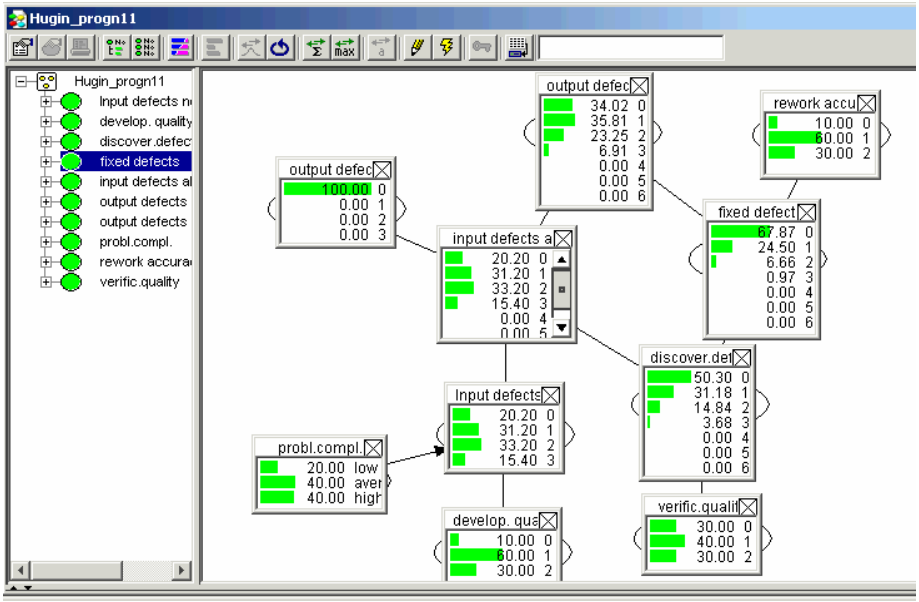


Рис. 5.26. **Априорные распределения вероятности в байесовской сети**

Правильность и чувствительность построенных байесовских моделей проверяется на оптимистическом и пессимистическом гипотетическом сценариях.

Для примера на рисунке 5.27 показан оптимистический сценарий событий при разработке программного приложения, моделируемый той же байесовской сетью для прогнозирования дефектов (см. рис. 3.11).

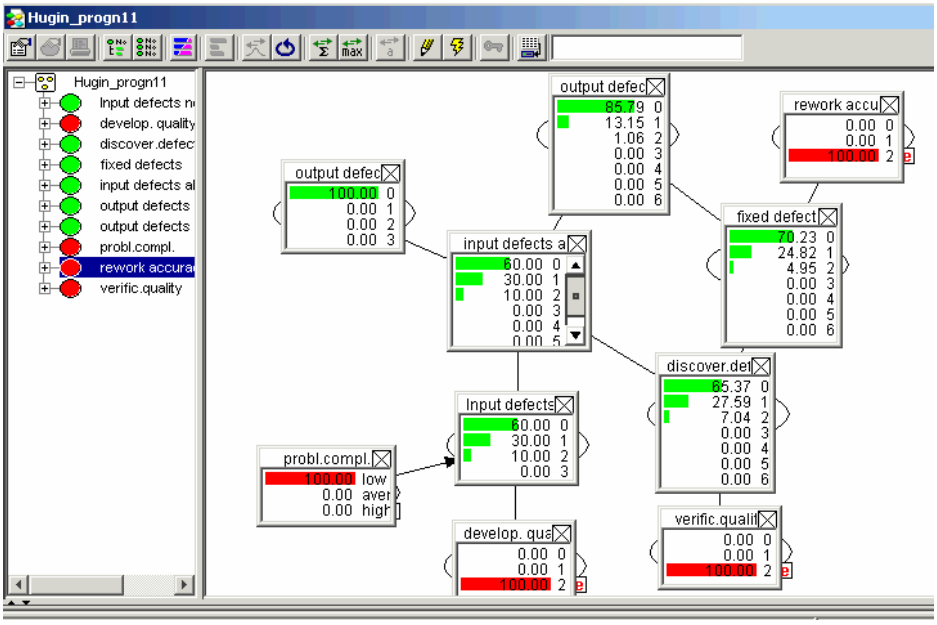


Рис. 5.27. **Оптимистичный сценарий событий в проекте**

Если при разработке приложения *сложность проблемы* квалифицируется как *низкая*, а *качество разработки* (вероятность предотвратить дефекты) и *проверки* (вероятность найти дефекты) *высокие*, а также *высокая точность* устранения дефектов (вероятность правильно откорректировать код), - плотность дефектов будет *очень низкой* (с вероятностью 86%).

5.3.4. Поддержка качества в CASE-инструментах

Современные CASE-инструменты автоматизируют большую часть периодически повторяющейся ручной работы разработчиков программных систем и при правильном использовании повышают производительность труда проектировщиков и программистов на всех стадиях ЖЦ ПС. В их состав входит множество компонентов (оконечных инструментов) и утилит, поддерживающих решение традиционных задач разработки ПС - описания деловых процессов в проблемной области, эскизного, технического, рабочего проектирования и макетирования программного и информационного обеспечения ПС.

Однако возможности CASE-инструментов далеко не ограничиваются поддержкой основных процессов ЖЦ. Они содержат средства, которые могут применяться при выполнении поддерживающих процессов ЖЦ и способствовать повышению качества ПС. Краткая характеристика некоторых интегрированных CASE-инструментов дана в приложении 4.

Литература к главе 5

1. *IEEE* Std. 730:1998. IEEE Standard for Software Quality Assurance Plans.
2. *NASA* Std. 2201:1993. Software assurance standard.
3. *MIL* Std. 498:1994. Software Development and Documentation.
4. *ISO/IEC* 12207:1995. Information Technologies – Software life cycle processes.
5. *NASA* GB A201:1995. Software assurance guidebook.
6. *IEEE* Std. 730-1:1995. IEEE Guide for Software Quality Assurance Planning.
7. *Мороз Г.Б.,* Коваль Г.И., Коротун Т.М. Концепция профилей в инженерии надежности программных систем // Математические машины и системы. – 2004. – С. 166 – 182.
8. *Липаев В.В.* Обеспечение качества программных средств. Методы и стандарты. Серия «Информационные технологии». М.: СИНТЕГ, 2001. – 380 с.
9. *Бабенко Л.П.,* Лавріщева К.М. Основи програмної інженерії: Навчальний посібник. – К.: Знання, 2001. – 269 с.
10. *Вендров А.М.* CASE – технологии. Современные методы и средства проектирования информационных систем. М.: Финансы и статистика, 1998.
11. *Липаев В.В.* Тестирование программ - М.: Радио и связь, 1986. – 296 с.
12. *Липаев В.В.* Отладка сложных программ – М.: Энергоатомиздат, 1993. - 296 с.
13. *Шураков В.В.* Надежность программного обеспечения систем обработки данных: Учебник. – М.: Финансы и статистика, 1987. – 272 с.
14. *Липаев В.В.* Выбор и оценивание характеристик качества программных средств. Методы и стандарты. Серия «Информационные технологии». М.: СИНТЕГ, 2001. – 228 с.
15. *Липаев В.В.* Качество программных средств. - М.: «Янус-К», 2002. – 400 с.
16. *Орлик С.,* Булуй Ю. Программная инженерия и управление жизненным циклом. - http://software-testing.ru/lib/se/

17. *Основы* инженерии качества программных систем / Ф.И. Андон, Г.И. Коваль, Т.М. Коротун, В.Ю. Суслов – К.: Академпериодика, 2002. – 504 с.
18. *Кулаков А.Ф.* Оценка качества программ ЭВМ. – К.: Техніка, 1984. – 167 с.
19. *Майерс Г.* Надежность программного обеспечения. – М.: Мир, 1980. – 360 с.
20. *Гласс Р.* Руководство по надежному программированию. – М.: Финансы и статистика, 1982. – 256 с.
21. *Липаев В.В.* Надежность программных средств. – М.: СИНТЕГ, 1998. – 231 с.
22. *Карповский Е.Я.*, Чижов С.А. Надежность программной продукции.–К.:Техніка., 1990
23. *Боэм Б.У.* Инженерное проектирование программного обеспечения. – М.: Радио и связь, 1985. – 511 с.
24. *Тейер Т.*, Липов М., Нельсон Э. Надежность программного обеспечения. – М.: Мир, 1981. – 325 с.
25. *Humphrey W.S.* The Personal Software Process // Technical Report CMU/SEI-2000-TR-022. – Software Eng. Inst., Pittsburgh, PA 15213-3890, 2000. - 55 p.
26. *Humphrey W.S.* The Personal Software Process: Status and Trends // IEEE Software, nov./dec. 2000. – P. 71 - 75.
27. *Ferguson P.* et al. Results of Applying the Personal Software process // IEEE Conputer, may 1997. – P. 24 – 31.
28. *Humphrey W.S.* The Team Software Process // Technical Report CMU/SEI-2000-TR-023. – Software Eng. Inst., Pittsburgh, PA 5213-3890, 2000. - 51 p.
29. *Curtis B.*, Hefley W.E., Miller S.A. People Capability Maturity Model. Version 2.0 Technical Report CMU/SEI-2001-MM-01. -Software Eng. Inst., Pittsburgh, PA 5213-3890, 2000.–735p.
30. *ISO/IEC* 15504-5:2006. Information Technologies - Process assessment - Part 5: An exemplar Process Assessment Model
31. *DEF* STAN 0055:1997. Requirements for Safety Related Software in Defence Equipment
32. *Albrecht A.*, Gaffney J. Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation // IEEE Trans. S. E.- 1983.- V.9.- N.11.-p.639-648.
33. *Fenton N.* A Critique of Software Defect Prediction Models // IEEE Trans. Soft. Eng.- 1999.- V.25.-N.5.-p.675-689
34. *Холстед М.Х* Начала науки о программах. – М.: Финансы и статистика, 1988. – 128 с.
35. *Watson A.H.*, McCabe T.J. Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric // NIST Spec. Publ. 500-235, NIST, Gaithersburg.- 1996. – 124 p.
36. *Fenton N.*, Neil M. Software metrics: successes, failures and new directions // J. Systems Software.- v.47, n.2-3.- 1999. - p. 149-157.
37. Applying and interpreting object oriented metrics - http://satc.gsfc.nasa.gov/support/ STC_APR98/apply_oo/apply_oo.html.
38. *Voas J.*, Morell L., Miller K. Predicting where faults can hide from testing // IEEE Software.- 1991.-March.- p.41-48.
39. *Коваль Г.И.* Подход к прогнозированию надежности ПО при управлении проектом // Проблемы программирования. –2002. - № 1 – 2. – С. 282 – 290.
40. *Kan S.H.* Metrics and models in software quality engineering // Addison-Wesley, 1995.
41. *ISO./IEC 15939* Software engineering – Software Measurement Process.
42. *International* Vocabulary of Basic and General Terms in Metrology (VIM). - ISO. -1993.
43. *W. Suryn*, A.Abran, A.April ISO/IEC SQuaRE. The second generation of standards for software product quality // www.lrgl.uqam.ca/publications/pdf-presentations/ 799.pdf

44. *http:* //www.info.fundp.ac.be/~nha/Monsite/PubsPdf/Rek(Square)2005.pdf
45. *ISO/*IEC *PDTR 25021* Software and System Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) – Measurement Primitives. - ISO/IEC JTC1/SC7 WG6, September 30, 2004, 6/N-521.
46. *Исикава К.* Японские методы управления качеством. – М.: Экономика, 1988. – 216 с.
47. *Электронный* учебник по промышленной статистике.- М., StatSoft, Inc. – 2001. (www.statsoft.ru/home/portal/textbook_ind/default.htm)
48. *Florac W.A.*, Park R.E., Carleton A.D. Practical Software Measurement: Measuring for Process Management and Improvement //CMU/SEI-97-HB-003.-Pittsburgh, 1997.–246p.
49. *Шапот М.* Интеллектуальный анализ данных в системах поддержки принятия решений // Открытые системы. - http://atlant.osp.ru/os/1998/01
50. *Буров К.* Обнаружение знаний в хранилищах данных // Открытые системы. - http://atlant.osp.ru/os/1999/05-06
51. *Mendonca M.,* Sunderhaft N.L. Mining Software Engineering Data: A Survey // www.dacs.dtic.mil/techs/datamining/datamining.pdf
52. *Khoshgoftaar T.M.,* Pandya A.S., Lanning D.L. Application of neural networks for predicting faults // Annals of Software Engineering. – n.1. – 1995. – P. 141 - 154.
53. *Fenton N., Neil M.* Making Decisions: Using Bayesian Nets and MCDA // Knowledge-Based Systems. – n.14. – 2001. – P. 307 - 325.
54. *Porter A.A.,* Selby R.W. Empirically Guided Software Development Using Metric-Based Classification Trees // IEEE Software. – v.7. – n.2. - March 1990. - P. 46 - 54.
55. *Briand L.C.,* Basili V.R., Hetmanski C.J. Developing Interpretable Models with Optimized Set Reduction for Identifying High-Risk Software Components //IEEE Trans.on Soft.Eng.. – v.19. – n.11. – 1993. - P.1028 - 1044.
56. *Hu Z.*, Chin W.N., Takeichi M. Calculating a New Data Mining Algorithm for Market Basket Analysis // Second Intern. Workshop on Practical Aspects of Declarative Languages (PADL'00), Boston, Massachusetts, Jan. 17-18, Springer Verlag.- 2000.- P. 169 -184.
57. *Colet E.*, Bhandari I. S. Statistical Issues in the Application of Data Mining to the NBA Using Attribute Focusing // Proc. of the Section on Statistics in Sports of the 1997 Joint Statistical Meetings, Anaheim, CA, American Statistical Association. - 1997. – P. 1-6.
58. *Jain K.*, Murty M. N., Flynn P. J. Data clustering: a review // ACM Computing Surveys. – 1999.- V.31. N. 3. - P. 264-323.
59. *Ng R.T.,* Han J. Efficient and Effective Clustering Methods for Spatial Data Mining // VLDB. – 1994. – P. 144-155.
60. *Information* Visualization Resources // www.cs.man.ac.uk/~ngg/InfoViz/People/ Publications/
61. *Heckerman D.,* Mamdani A., Wellman M. Real-world applications of Bayesian networks // Comm. ACM. – 1995. - № 3. – С. 25-26.
62. *Murphy K.P.* Brief Introduction to Graphical Models and Bayesian Networks. http:// HTTP.CS.Berkeley.EDU/~murphyk/Bayes/bayes.html.
63. *MSBNx*. Продукт Microsoft Research corp. // research.microsoft.com/ adapt/MSBNx/
64. *Netica.* Продукт Norsys Software corp.// www.norsys.com/download.html
65. *Serene*. ESPRIT Framework IV Collaborative Project 22187 //www.dcs.qmul.ac.uk/ ~norman/serene.htm
66. *Hugin Lite 6.5*. Продукт Hugin Expert //www.hugin.com/roducts_Services/ roducts/Demo/Download/

Глава 6. ПРОЦЕССЫ И МЕТОДЫ ПРОВЕРКИ

6.1. Процессы проверки в жизненном цикле

6.1.1. Назначение процессов проверки

Наряду с процессом обеспечения гарантии качества, в архитектуре процессов ЖЦ определены еще четыре *процесса поддержки*, всецело направленных на повышение качества ПС. Это процессы *Верификации*, *Валидации*, *Совместного просмотра* и *Аудита*[1], которые предназначены для проверки соответствия рабочих продуктов и процессов разработки своему назначению и подтверждения их способности обеспечить надлежащее качество конечного программного продукта.

Исходя из положений стандарта ISO/IEC 12207 [1] «назначение ***процесса верификации*** состоит в подтверждении того, что каждый *рабочий продукт ПС* (software work product) и/или услуга процесса или проекта должным образом отражают установленные требования. В результате успешного осуществления процесса:

- будет разработана и внедрена стратегия верификации;
- будут установлены критерии верификации всех необходимых рабочих продуктов ПС;
- будут выполняться надлежащие действия по верификации;
- будет выполняться поиск дефектов в рабочих продуктах и их устранение;
- будет обеспечиваться доступ заказчика и других заинтересованных сторон к результатам верификационной деятельности…

Назначение ***процесса валидации*** состоит в подтверждении того, что требования, касающиеся *конкретного применения рабочего продукта ПС* по назначению, удовлетворяются. В результате успешного осуществления процесса:

- будет разработана и внедрена стратегия валидации;
- будут идентифицированы критерии валидации для всех нужных рабочих продуктов;
- будет проводиться надлежащая валидационная деятельность;
- будут решаться все обнаруженные проблемы;
- будут предоставляться свидетельства того, что разработанные рабочие продукты ПС отвечают своему назначению;
- будет обеспечиваться доступность результатов валидационной деятельности для заказчика и других заинтересованных сторон».

Анализируя определения стандарта достаточно сложно установить грань между процессами верификации и валидации, поскольку оба процесса применимы к «рабочим продуктам ПС» (software work product). В действующем стандарте ISO/IEC 12207 (1995 года) эта грань проведена четче: понятие верификации связывается с программными продуктами (software products) деятельности по разработке ПС, а валидации – с «конечным программным продуктом» (final software product).

---

[1] Наименование группы и отдельных процессов ЖЦ соответствуют ДСТУ 3918-99. В аналогичном стандарте России ГОСТ Р ИСО/МЭК 12207 группа процессов поддержки названа группой *вспомогательных процессов*, процесс валидации - *процессом аттестации*, а процесс совместного просмотра - *процессом совместного анализа*, что не меняет сути этих процессов.

На практике процессы верификации и валидации обычно выполняются совместно, начиная с ранних стадий ЖЦ и применительно к установленному множеству *рабочих продуктов* проекта. Они определены в стандарте раздельно, скорее из соображений соблюдения «принципа модульности» архитектуры процессов [2, 3].

Подробнее задачи, решаемые при выполнении процессов верификации и валидации, рассматриваются в п.6.1.2.

Назначение процесса ***совместного просмотра***, в соответствии с ISO/IEC 12207, состоит в том, чтобы «поддерживать должный уровень *понимания заказчиком* того, как проект продвигается к целям договора, и того, какие именно дополнительные мероприятия должны быть проведены для обеспечения гарантии разработки продукта, который его удовлетворит. Совместные просмотры проводятся как на уровне управления проектом, так и на техническом уровне, и в течение всего жизненного цикла проекта. В результате успешного осуществления процесса:

- периодические просмотры будут производиться в заранее определенные сроки;
- состояние и продукты деятельности процесса будут оцениваться в ходе выполнения совместного просмотра при участии заказчиков, поставщиков и других ответственных лиц и пользователей;
- результаты просмотра будут доводиться до всех сторон, которых они касаются;
- мероприятия, решение о выполнении которых принято при просмотрах, будут отслеживаться до их завершения;
- проблемы будут идентифицироваться и решаться».

В ходе подготовки совместного просмотра должны быть определены сфера просмотра, темы, участники, график работы и требования к ресурсам, и установлены критерии просмотра. Критерии просмотра касаются способа выявления проблемы и ее решения, а также согласования результата просмотра его участниками.

Цель периодических совместных просмотров на уровне управления проектом – оценить *состояние проекта* по отношению к утвержденным планам, графикам, стандартам и руководствам, установить, нет ли необходимости в изменении планов, правильно ли распределены ресурсы, выполняется ли управление рисками.

Цель периодических совместных просмотров на техническом уровне – оценить *состояние рабочих продуктов ПС* по отношению к требованиям заказчика и критериям приемки, зафиксированным в договоре. В ходе технических просмотров нужно установить, ведется ли разработка рабочих продуктов по плану, соответствует ли степень их завершенности запланированной, соответствуют ли они стандартам и спецификациям, все ли изменения в них вносятся надлежащим образом и касаются только тех областей, которые определены процессом управления конфигурацией.

Цель внутренних периодических совместных просмотров *процесса* – оценить его состоятельность и пригодность для проекта.

Проблемы, обнаруженные при просмотре, упорядочиваются по степени серьезности для проекта и фиксируются в отчете наряду с предлагаемыми мероприятиями по их решению. Отчет направляется на согласование всем участникам просмотра и затем передается процессу решения проблем. Выполнение мероприятий контролируется.

Назначение ***процесса аудита***, согласно ISO/IEC 12207, состоит в «независимом установлении соответствия выбранных продуктов и процессов требованиям, планам и договору. В результате успешного осуществления процесса:

- будет разработана и внедрена стратегия аудита;
- аудиты будут проводиться;
- согласованность выбранных рабочих продуктов ПС и/или услуг или процессов с требованиями, планами и договором будет устанавливаться в соответствии со стратегией аудита;
- проведение аудитов пригодной для этого независимой стороной будет организовано;
- проблемы, обнаруженные во время аудита, будут идентифицированы, доведены до тех, кто несет ответственность за мероприятия по корректировке, и решены».

Стандарт определяет рабочие продукты и результаты деятельности по процессам ЖЦ, подлежащие аудиту:

– соответствие *кода* программного продукта проектной документации;

– адекватность *требований* к приемочным просмотрам и испытаниям, зафиксированных в документации, *приемке* программного продукта;

– соответствие *тестовых данных* спецификации;

– успешность испытаний программного продукта и его соответствие спецификации;

– корректность *отчетов об испытаниях;*

– *устранение расхождений* между фактическими и ожидаемыми результатами;

– соответствие *документации пользователя* установленным стандартам;

– соответствие выполненных *действий* применимым к ним требованиям, планам и договору;

– соответствие *затрат и сроков* плановым.

Аудит проводится в заранее установленные сроки. При подготовке плана аудита устанавливается сфера применения аудита, темы, участники, критерии начала и завершения аудита, график и нужные ресурсы.

Кроме аудитов разработки ПС могут также проводиться аудиты управления проектом и аудиты выполнения процессов, с целью определения их состоятельности и пригодности для проекта.

По результатам аудита составляется аудиторский отчет. В нем фиксируются проблемы, обнаруженные в процессе аудита, и предлагаемые мероприятия по их решению. Отчет распространяется среди участников аудита и направляется процессу решения проблем. Выполнение мероприятий контролируется.

Каждый из рассмотренных выше процессов должен *синхронизироваться* с основными процессами ЖЦ, для проверки рабочих продуктов которых он подключается (в контрольных точках проекта), и *координироваться с процессом SQA* относительно видов и способов проверок.

Уровень и масштаб планирования и выполнения проверок варьируется в зависимости от особенностей ПС и условий ее разработки – выбранной модели ЖЦ, исходных требований к ПС, используемых организационных подходов и процедур, размера, критичности и типа ПС, количества персонала проекта и участвующих в проекте сторон. Эти факторы определяют, какие из пяти процессов поддержки ка-

чества будут включены в модель процессов ЖЦ проекта в виде *отдельных* процессов, и какие задачи из других процессов будут дополнительно в них решаться. Так, в простейшем случае, для проекта может быть выбран, например, только процесс SQA, в который включены задачи процесса совместного просмотра, а процессы верификации и валидации, а также аудита – не использоваться. Однако для крупных проектов разработки критических ПС могут понадобиться и независимая верификация и валидация (IV&V), и периодические совместные просмотры, и аудиторские проверки третьей стороной.

Упомянутые выше факторы обусловливают и выбор *методов* для решения задач в рассматриваемых процессах – в зависимости от специфики проекта и требований к продукту проверки могут быть индивидуальными или коллективными, проводиться коллегами по проекту или независимыми экспертами и отличаться формой организации работ. Одни и те же методы могут использоваться в разных процессах. Стандарт ISO/IEC 12207 указывает лишь на то, ЧТО (*какие действия*) должны выполняться в рамках процессов, но не на то, КАК (*какими методами*) их выполнять. Часто возникает путаница из-за наличия одноименных процессов и методов, а также неточности перевода отдельных терминов. Так, процесс верификации иногда связывают исключительно с методом формальной верификации (доказательством корректности), а процесс совместного просмотра (joint review) путают с методом коллегиальной проверки (peer review) или технического обзора (technical review). Эти и другие статические методы выполнения проверок в различных процессах поддержки подробно рассматриваются в разделах 6.2 и 6.3, а динамические методы проверки – методы тестирования – в главе 7.

Рассматриваемые в этой главе методы проверки регламентируются следующими стандартами[2]:

- *IEEE Std. 1012* «IEEE Standard for Software Verification and Validation» (стандарт 1998 года). Это новая редакция стандарта 1986 года «IEEE Standard for Software Verification and Validation Plans»,
- *IEEE Std. 1059* «IEEE Guide for Software Verification and Validation Plans» (стандарт 1993 года),
- *IEEE Std. 1012a* «Supplement to IEEE Standard for Software Verification and Validation: Content Map to IEEE/EIA 12207.1-1997»,
- *IEEE Std. 1028* «IEEE Standard for Software Reviews» (стандарт 1997 года). Новая редакция стандарта 1988 г. «IEEE Standard for Software Reviews and Audits».

6.1.2. Цели и задачи верификации и валидации

Роль процессов верификации и валидации заключается в выполнении проверки и оценивании состояния рабочих продуктов ПС на протяжении процесса разработки[3] с целью своевременного выявления проблем проекта.

---

[2] Часто встречаются ссылки на разные редакции стандартов IEEE, а также на уже *отмененные* стандарты. Справку о статусе стандартов IEEE можно получить по адресу в Интернете: http://standards.ieee.org/db/status/.

[3] Вообще говоря, V&V может применяться на протяжении всего ЖЦ ПС и привлекаться для проверки рабочих продуктов процессов *приобретения*, *поставки*, *разработки*, *эксплуатации* и *сопровождения*. Однако далее рассматриваются в основном задачи V&V для процесса разработки.

В то время как SQA в большей степени акцентирует внимание на проверке соблюдения стандартов и действенности процессов, V&V сосредотачивается на технических аспектах разработки, отраженных в документах или компонентах ПС (договоре, исходных требованиях, проекте, коде, интегрированной ПС).

Цель V&V – проверить и подтвердить, что требования, предъявляемые к ПС, являются полными, точными, согласованными, корректно установленными и тестопригодными, а программные рабочие продукты - корректно их реализуют.

Процессы V&V оценивают элементы программного обеспечения в *контексте системы*, частью которой оно является, с учетом требований со стороны операционного окружения (среды эксплуатации), технических средств (hardware), интерфейсов, обслуживающего персонала системы и непосредственных пользователей программных продуктов.

Состав объектов проверки, критериев проверки, задач V&V, а также интенсивность и тщательность выполнения процессов V&V определяется *уровнем целостности* разрабатываемой системы (уровнями критичности ее компонентов).

Связь и отличие процессов верификации и валидации. Суть отличий процессов верификации и валидации удобнее сформулировать применительно к каскадной модели ЖЦ и традиционным стадиям разработки программного обеспечения систем (определение концепции, анализ требований, проектирование, реализация (построение), интеграция и тестирование, ввод в действие (инсталляция)).

Верификация означает проверку соответствия конкретных выходных рабочих продуктов каждой стадии требованиям к ним, установленным на предыдущей стадии. Оценивается корректность, полнота и точность реализации требований, а также приверженность действующим стандартам и нормам передовой практики. Верификация выполняется с целью *своевременного* обнаружения и устранения допущенных ошибок и обеспечения руководства проекта объективными сведениями, необходимыми для управления риском проекта. Результаты верификации служат базисом для оценки завершенности текущей стадии и принятия решения о переходе к следующей стадии проекта.

Валидация означает подтверждение того, что (те же) рабочие продукты ПО удовлетворяют *системным требованиям*, делегированным ПО (то есть, той части требований к системе, которая будет реализована программными продуктами), и отвечают потребностям системы (например, алгоритмы корректно моделируют физические законы системы; или, модели данных, функций, состояний и процессов адекватны деловым процессам банковской системы).

Если бы исходные требования к программному обеспечению системы полностью охватывались в начале проекта и были неизменны (что и предполагается при выборе каскадной модели разработки), процесс валидации можно было бы связывать только с исходными требованиями, тестами и конечным программным продуктом. В этом случае было бы оправданным бытующее мнение о том, что валидация отождествляется с тестированием при приемочных испытаниях. Однако, на практике, требования изменяются по ходу разработки. Да и современные CASE-среды позволяют сохранять и «тестировать» (трассировать к исходным требованиям) *промежуточные* продукты разработки.

Таким образом, и верификация, и валидация необходимы и возможны на всех стадиях создания ПО. Вопрос состоит только в правильном оценивании уровня целостности ПО и надлежащего объема V&V.

V&V поддерживает основные процессы ЖЦ и, как указывается в стандарте IEEE Std. 1012, «наиболее эффективна, когда применяется параллельно с процессами разработки ПО; в противном случае цели V&V могут не быть достигнуты. В этом стандарте процессы V&V рассматриваются совместно, поскольку их действия и задачи пересекаются и взаимно дополняют друг друга» [4]. Должен быть определен круг основных процессов в проекте, с которыми будет ассоциироваться V&V, и для них выбрано *минимально необходимое* множество задач проверки, определяемое уровнем целостности, установленным для программного обеспечения системы.

Уровни целостности программного обеспечения. Уровни целостности ПО системы связываются с *диапазоном* значений критических характеристик ПО (надежности, безопасности функционирования, защиты информации, производительности, сложности и др.), которые позволяют удерживать риски в приемлемых пределах. Критическое высокоцелостное ПО требует V&V, большей по объему и строгости, чем не критическое.

Уровни целостности могут назначаться для требований к программному обеспечению, функций, групп функций, программных компонентов или подсистем на самых ранних стадиях проекта, *согласовываться* всеми заинтересованными сторонами и *изменяться* по мере эволюции разработки в связи с выбором организацией-разработчиком или заказчиком определенных стратегий и методологий разработки и управления риском. Контроль и изменение уровня целостности – задача анализа критичности (V&V criticality analysis) в ходе процесса разработки.

Стандарт использует 4-уровневую схему целостности ПО как метод для определения минимума задач V&V в рамках уровня (таблица 6.1).

Таблица 6.1. **Уровни целостности ПО**

Критичность	**Описание**	**Уровень**
Высокая	Выбранная функция служит критическим фактором для эффективной работы системы	4
Большая	Выбранная функция важна для эффективной работы системы	3
Умеренная	Выбранная функция влияет на работу системы, но могут быть реализованы стратегии обхода для компенсации потери эффективности	2
Низкая	Выбранная функция вносит существенный вклад в работу системы. Если она не будет выполняться надлежащим образом, пользователь испытает неудобства	1

Если уровень целостности ПС или отдельных компонентов программного обеспечения системы ниже первого, применение V&V для них необязательно.

Ассоциация поуровневой схемы целостности с минимумом задач V&V описывается в *Плане верификации и валидации (SVVP)* (от Software Verification and Validation Plan), а обоснование назначения определенных уровней целостности проверяемым компонентам ПО - в *Отчете о задаче V&V* и в *Заключительном отчете о V&V*.

Нужно заметить, что предыдущая редакция стандарта IEEE Std. 1012 (1986 года) была «продукто-ориентированной»[4] и определяла структуру и содержание

[4] Это касается и большинства стандартов IEEE по программной инженерии, вышедших до 1996 года (то есть до появления ISO/IEC 12207) и теперь пересмотренных.

SVVP. Ныне действующий стандарт (1998 года) поддерживает процессо-ориентированный подход к разработке ПС и определяет процессы V&V в терминах действий и задач, а также устанавливает требования к управлению V&V, плану SVVP и отчетным документам.

Задачи V&V на стадиях разработки программной системы. В стандарте IEEE Std. 1012 приведен перечень и описание минимального набора задач V&V для *всех* основных процессов и уровней целостности программного обеспечения систем. На рисунке 6.1 показаны только *задачи процесса разработки* наряду с входящей информацией для их решения (входами V&V) и исходящей информацией – результатом их решения (выходами V&V), а в таблице 6.2 кратко описаны те же задачи с указанием номеров уровней целостности ПО, на которых они решаются.

Учитывая то, что задачи по V&V, которые должны решаться для ПС с низкой критичностью (первый уровень целостности), представляют наиболее широкий интерес, мы рассматриваем их далее подробнее.

Задачи верификации и валидации требований к ПС с низкой критичностью. Требования к ПС отражаются в документах *описания концепции ПС*, *спецификации требований к ПС*, а также *спецификации требований к интерфейсам ПС*[5]. В ходе рассмотрения этих документов решаются две задачи верификации и валидации: оценка требований к ПС и создание/верификация плана тестирования для V&V системы.

Оценка требований к ПС заключается в анализе всех видов требований с целью проверки и подтверждения их корректности, согласованности, полноты, точности, читабельности и тестопригодности.

Подтверждение *корректности* требований к ПС должно быть основано на:

– проверке, правильно ли распределены функции ПС по компонентам общесистемного и специального (разрабатываемого) программного обеспечения, и удовлетворяют ли они требования заказчика к ПС;

– проверке, согласуются ли требования к ПС с действующими в отрасли стандартами, нормами, положениями и правилами ведения бизнеса;

– анализе логики событий и потоков данных в деловом процессе и подтверждении, что последовательности смены состояний системы отвечают принципам, действующим в проблемной области, и другим нормам;

– проверке, удовлетворяют ли описания потоков данных и управления в ПС требованиям к ее функциям и производительности[6];

– проверке правильности использования (обработки) данных и их форматов.

---

[5] Структура и содержание документов описания требований к ПС могут регламентироваться, например, стандартами IEEE Std. 830, IEEE Std. 1233, ГОСТ 34.602.

[6] Сложившаяся практика, к сожалению такова, что описание деловых процессов, потоков управления и форматов данных выполняется лишь на стадии технического проектирования, а документ «техническое задание» больше напоминает протокол о намерениях.

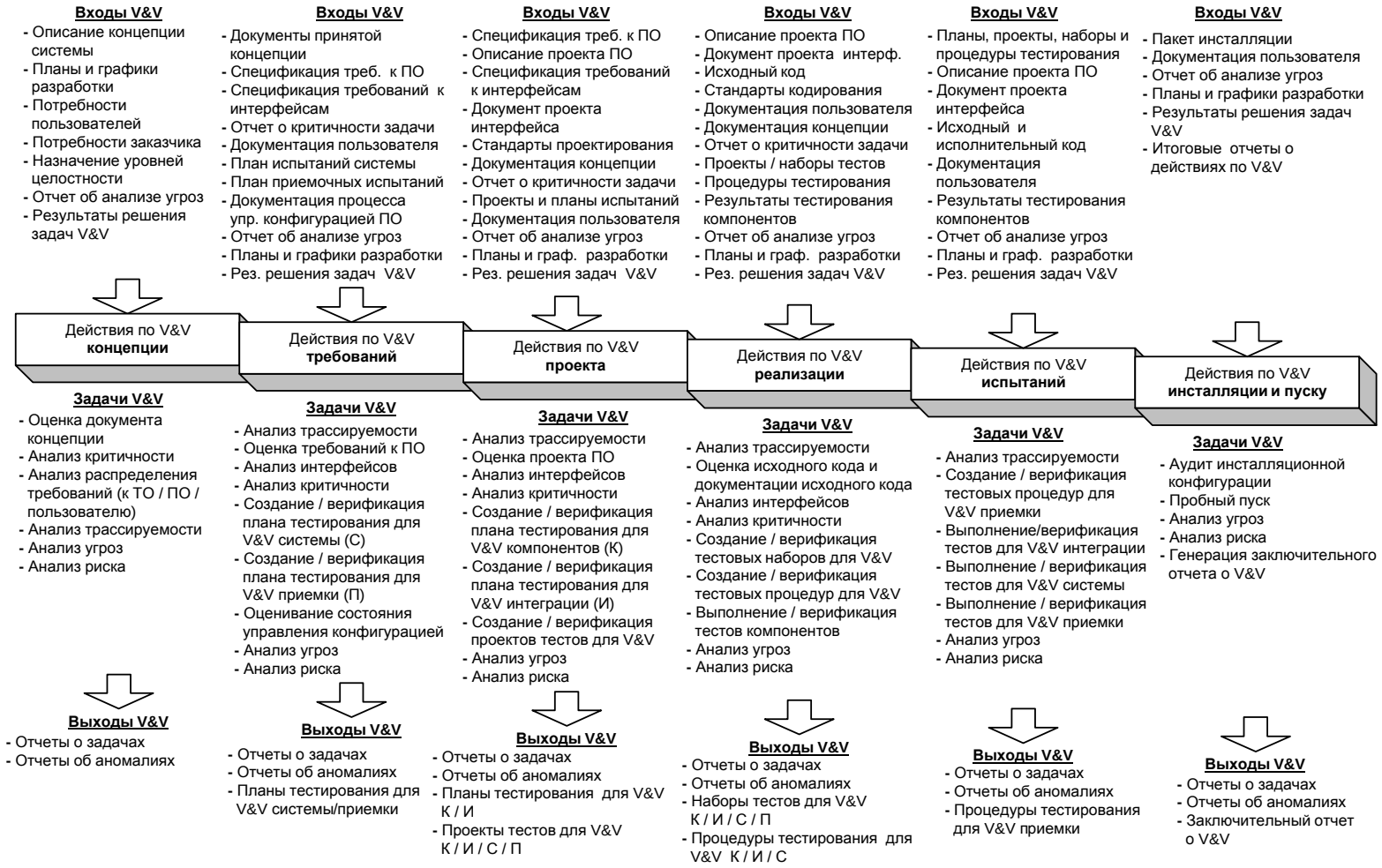


Рис. 6.1. **Задачи V&V для процессов разработки**

Таблица 6.2. **Характеристика задач V&V**

Задача	Описание задачи	Уровни
Анализ критичности	*Задача выполняется в составе работ по V&V концепции, требований, проекта* и *реализации ПО системы.* Проверяется корректность назначения уровней целостности требованиям, функциям, подсистемам, модулям ПО. Если частные уровни целостности не назначены – присваивается уровень целостности, назначенный системным требованиям. Если какому-либо компоненту назначен самый высокий уровень критичности, всем «влияющим» на него компонентам присваивается тот же уровень. Уровни целостности документируются в отчете о выполнении V&V. При каждой последующей проверке отчет пересматривается и обновляется по результатам анализа новых документов (описания требований, описания проекта и др.).	4, 3, 2
Анализ распределения системных требований	*Задача выполняется в составе работ по V&V концепции.* Проверяется корректность, точность и полнота распределения системных требований по компонентам технического и программного обеспечения, а также пользовательским интерфейсам в контексте требований пользователей. Проверяется, обеспечит ли схема распределения нужную производительность работы, удовлетворяют ли форматы данных, частота и протоколы передачи информации пользовательским требованиям к интерфейсам, предусмотрены ли механизмы обеспечения отказоустойчивости и восстанавливаемости приложений, отражены ли требования пользователей к сопровождению системы.	4
Анализ прослеживаемости	*Задача выполняется в составе работ по V&V концепции, требований, проекта, реализации и испытаний ПО.* Изначально в документе описания концепции идентифицируются все системные требования, которые полностью или частично будут реализованы программно. Проверяется, прослеживаются ли системные требования к требованиям заказчика. Критерий прослеживаемости – корректность, согласованность в деталях, полнота и точность отражения требований заказчика в требованиях к системе. На последующих стадиях разработки проверяется, соответственно, прослеживаются ли *требования к ПО* к *системным требованиям* (при V&V требований), *элементы проекта ПО* к *требованиям к ПО* (при V&V проекта), *компоненты исходного кода* к *соответствующим спецификациям в проекте* и наоборот (при V&V реализации), а все *тестовые процедуры* – к *планам тестирования* (при V&Vиспытаний).	4, 3, 2
Анализ угроз	*Задача выполняется в составе работ по V&V концепции, требований, проекта, реализации, испытаний и ввода в действие ПО системы.* Изначально по документу описания концепции определяется, существуют ли потенциальные угрозы системе или со стороны системы.	4, 3

Задача	Описание задачи	Уровни
	Оценивается вероятность появления каждой угрозы и ее серьезность, идентифицируются стратегии устранения угроз. Составляется отчет об анализе угроз. При последующих проверках уточняется, какие требования к ПО (проектные решения, особенности реализации) вносят вклад в каждую из ранее обнаруженных угроз.	
Анализ риска	*Задача выполняется в составе работ по V&V концепции, требований, проекта, реализации, испытаний и ввода в действие ПО системы.* Изначально по документу описания концепции, планам-графикам разработки и отчету об анализе угроз идентифицируются технические риски, а также риски управления проектом. Даются рекомендации по устранению, снижению или наблюдению рисков. Составляется отчет об их анализе. При последующих проверках анализ рисков возобновляется, риски пересматриваются и даются новые рекомендации по их устранению, снижению или наблюдению.	4, 3
Анализ интерфейсов	*Задача выполняется в составе работ по V&V требований, проекта* и *реализации ПО системы.* Изначально по документам описания концепции, описания требований к ПО и описания интерфейсов проверяется и подтверждается, что требования к интерфейсам ПО с ТО, пользователем, оператором и другими системами: - корректны. Правильно определены требования к внешним и внутренним интерфейсам, - согласованы. Непротиворечивы описания интерфейсов в документе спецификации требований и документе спецификаций интерфейсов, - полны. Описаны форматы данных и критерии производительности каждого интерфейса, - точны. Каждый интерфейс обеспечивает передачу информации с нужной точностью, и - тестопригодны. Существуют объективные критерии приемки для подтверждения требований. При V&V проекта по тем же критериям проверяются все проектные решения относительно интерфейсов с ТО, ПО, пользователем, оператором и другими системами. При V&V реализации по тем же критериям проверяется и подтверждается правильность программно реализованных интерфейсов.	4, 3, 2
Оценка документа описания концепции	*Задача выполняется в составе работ по V&V концепции ПО системы.* Проверяется, насколько документация концепции удовлетворяет требования пользователей и соответствует потребностям заказчика. Подтверждается, что функции системы, цикл ее использования, степень реализуемости и тестопригодности, функциональные требования, концептуальный проект архитектуры системы, требования к эксплуатации и сопровождению, а также требования к миграции наработок из старой системы в новую удовлетворяют пользователей.	4, 3, 2

Задача	Описание задачи	Уровни
Оценка требований к ПО	*Задача выполняется в составе работ по V&V требований к ПО.* Описана отдельно	4, 3, 2, 1
Оценка проекта ПО	*Задача выполняется в составе работ по V&V проекта ПО.* Описана отдельно	4, 3, 2, 1
Оценка исходного кода	*Задача выполняется в составе работ по V&V реализации ПО.* Описана отдельно	4, 3, 2, 1
Оценивание состояния управления конфигурацией	*Задача выполняется в составе работ по V&V требований к ПО.* Проверяется полнота и адекватность процесса управления конфигурацией (CM, от Configuration Management), описанного в документации по процессу. Поддерживающий процесс CM должен отвечать требованиям стандартов и соответствовать сложности, размеру и целостности ПС, а также планам проекта и чаяниям пользователей.	4, 3
Создание / верификация плана тестирования для V&V приемки	*Задача выполняется в составе работ по V&V требований к ПО.* Для уровня целостности 2 достаточно проверить, что План тестирования приемки, созданный группой разработки ПС при выполнении основного процесса «тестирование ПО», удовлетворяет потребностям проекта и стандартам. Для уровней целостности 3 и 4 должен быть разработан самостоятельный план тестирования приемки, *подтверждающего*, что ПО корректно реализует программные и системные требования в среде эксплуатации.	4, 3, 2
Создание / верификация плана тестирования для V&V системы	*Задача выполняется в составе работ по V&V требований к ПО.* Описана отдельно	4, 3, 2, 1
Создание / верификация плана тестирования для V&V интеграции	*Задача выполняется в составе работ по V&V проекта ПО.* Для уровня 2 достаточно проверить, что План тестирования интеграции, созданный группой разработки ПС при выполнении основного процесса «тестирование ПО», удовлетворяет потребностям проекта и стандартам (в частности, обеспечивает прослеживаемость интеграции к системным требованиям). Для уровней 3 и 4 должен быть разработан самостоятельный план интеграционного тестирования, *подтверждающего*, что ПО, создаваемое путем постепенной интеграции отдельных проверенных компонентов, корректно реализует требования.	4, 3, 2
Создание / верификация плана тестирования для V&V компонентов	*Задача выполняется в составе работ по V&V проекта ПО.* Для уровня 2 достаточно проверить, что План тестирования компонентов, созданный группой разработки ПС при выполнении основного процесса «тестирование ПО», удовлетворяет потребностям проекта и используемым стандартам (в частности, обеспечивает прослеживаемость компонентов к требованиям к ПО и проекту ПО).	4, 3, 2

Задача	Описание задачи	Уровни
	Для уровней 3 и 4 должен быть разработан самостоятельный план тестирования компонентов, *подтверждающего*, что каждый компонент корректно реализует предъявляемые к нему требования проекта.	
Создание / верификация проектов тестов для V&V	*Задача выполняется в составе работ по V&V проекта ПО.* Для уровней ниже 3 достаточно проверить, что Проекты тестов, построенные разработчиками для тестирования компонентов, интеграции, системы и приемки ПС, отвечают требованиям соответствующих планов тестирования и могут быть прослежены к системным требованиям. Для уровней 3 и 4 нужно разработать проекты дополнительных тестов для тестирования компонентов, интеграции, системы и приемки.	
компонента		4, 3, 2
интеграции		4, 3, 2, 1
системы		4, 3, 2, 1
приемки		4, 3, 2
Создание / верификация тестовых наборов для V&V	*Задача выполняется в составе работ по V&V реализации ПО.* Для уровней ниже 3 достаточно проверить, что Наборы тестов, построенные разработчиками для тестирования компонентов, интеграции, системы и приемки ПС, отвечают соответствующим проектам тестов и могут быть прослежены к планам тестирования и системным требованиям. Для уровней 3 и 4 нужно разработать наборы дополнительных тестов для тестирования компонентов, интеграции, системы и приемки.	
компонента		4, 3, 2
интеграции		4, 3, 2, 1
системы		4, 3, 2, 1
приемки		4, 3, 2
Создание / верификация тестовых процедур для V&V	*Задача выполняется в составе работ по V&V реализации ПО.* Для уровней ниже 3 достаточно проверить, что Тестовые процедуры, построенные разработчиками для тестирования компонентов, интеграции и системы, отвечают соответствующим наборам тестов и могут быть прослежены к планам тестирования и системным требованиям. Для уровней 3 и 4 нужно разработать дополнительные тестовые процедуры для тестирования компонентов, интеграции и системы. Подтвердить их соответствие планам.	
компонента		4, 3, 2
интеграции		4, 3, 2, 1
системы		4, 3, 2, 1
Создание / верификация тестовых процедур для V&V приемки	*Задача выполняется в составе работ по V&V испытаний ПО.* Для уровня 2 достаточно проверить, что Тестовые процедуры приемки, построенные разработчиками, отвечают соответствующим наборам тестов и могут быть прослежены к планам тестирования и системным требованиям. Для уровней 3 и 4 нужно разработать дополнительные тестовые процедуры для тестирования приемки. Подтвердить их соответствие планам.	4, 3, 2
Выполнение / верификация тестов для V&V компонентов	*Задача выполняется в составе работ по V&V реализации ПО.* Для уровня 2 достаточно используя результаты выполнения тестов компонентов разработчиками, а также планы и процедуры тестирования компонентов подтвердить, что ПО удовлетворяет критерию приемочного тестирования.	4, 3, 2

Задача	Описание задачи	Уровни
	Для уровней 3 и 4 нужно провести тестирование компонентов, проанализировать результаты и подтвердить, что ПО корректно реализует проект. Подтвердить, что результаты тестирования прослеживаются к ожидаемым, зафиксированным в планах. Документировать результаты тестирования в соответствии с требованиями, описанными в Плане тестирования, указывая отклонения результатов от ожидаемых. Использовать результаты тестирования для подтверждения того, что ПО удовлетворяет установленному критерию приемочного тестирования.	
Выполнение / верификация тестов для V&V интеграции	*Задача выполняется в составе работ по V&V испытаний ПО.* Описана отдельно.	4, 3, 2, 1
Выполнение / верификация тестов для V&V системы	*Задача выполняется в составе работ по V&V испытаний ПО.* Описана отдельно.	4, 3, 2, 1
Выполнение / верификация тестов для V&V приемки	*Задача выполняется в составе работ по V&V испытаний ПО.* Для уровня 2 достаточно используя результаты выполнения тестов приемки разработчиками, а также планы и процедуры тестирования приемки подтвердить, что ПО удовлетворяет критерию приемочного тестирования. Для уровней 3 и 4 нужно провести тестирование приемки, проанализировать результаты и подтвердить, что ПО удовлетворяет системным требованиям. Подтвердить, что результаты тестирования прослеживаются к ожидаемым, зафиксированным в планах. Документировать результаты тестирования.	4, 3, 2
Аудит инсталляционной конфигурации	*Задача выполняется в составе работ по V&V ввода в действие ПО.* Проверяется, все ли продукты ПО, необходимые для корректной инсталляции и применения ПО, присутствуют в инсталляционном пакете. Подтверждается, что все зависящие от среды эксплуатации параметры настройки и условия корректно учтены.	4, 3
Пробный пуск	*Задача выполняется в составе работ по V&V ввода в действие ПО.* Проводится анализ или выполняются тесты для проверки того, что инсталлированное ПО соответствует тому, которое должно быть подвергнуто V&V. Подтверждается, что программный код и база данных запускаются, работают и завершаются корректно. При переходе от одной версии ПО к другой подтверждается, что ПО может быть демонтировано без нарушения функционирования оставшихся компонентов системы.	4, 3

Задача	Описание задачи	Уровни
Генерация заключительного отчета о V&V	*Задача выполняется в составе работ по V&V ввода в действие ПО.* Нужно подытожить все работы по V&V, выполненные задачи и полученные результаты, включая местонахождение обнаруженных аномалий и их состояние (степень устранения). Провести оценивание качества всего ПО и выработать рекомендации.	4, 3, 2

Подтверждение *согласованности* требований к ПС должно быть основано на:

– проверке, все ли термины и определения тщательно документированы;

– проверке, согласованы ли взаимодействие функций и допуски относительно их взаимного влияния. Удовлетворяют ли они потребностям деловой сферы;

– проверке внутренней непротиворечивости требований к компонентам общесистемного и специального (разрабатываемого) ПО, а также согласованности с внешними требованиями к ПС.

Подтверждение *полноты* требований к ПС должно быть основано на:

– проверке, определены ли функциональные требования (относительно алгоритмов, режимов/состояний, входов/выходов, обработки исключительных ситуаций, составления отчетов и др.), а также требования к технологическому процессу и регламенту использования ПС;

– проверке, описаны ли интерфейсы с общесистемным ПО, техническими средствами и пользователями;

– проверке, установлены ли критерии эффективности ПС (например, скорость обработки сообщений, точность) и обеспечены ли средства контроля и отказоустойчивости ПС (контроль входных данных, откат транзакций и др.);

– проверке, удовлетворяют ли рассматриваемые документы требованиям, предъявляемым к ним процедурами управления конфигурацией.

Подтверждение *точности* требований к ПС должно быть основано на проверке, удовлетворяет ли точность вычислений (усечения, округления) и представления текстовой информации требованиям со стороны технических средств.

Подтверждение *читабельности* требований к ПС должно быть основано на:

– проверке, написана ли документация понятно, доходчиво, недвусмысленно, и отвечает ли профессиональному уровню круга ее читателей;

– проверке, определены ли все сокращения, аббревиатуры, термины и специальные символы.

Подтверждение *тестопригодности* требований к ПС должно быть основано на проверке наличия объективных критериев приемки для подтверждения каждого из положений документированных требований.

Задача *создания/верификации плана тестирования с целью V&V системы* выполняется по-разному, в зависимости от уровня целостности ПС.

Для ПС с уровнем целостности 1 или 2 выполнение этой задачи заключается в *верификации* соответствия Плана тестирования системы (System test plan), созданного группой разработки в ходе выполнения основного процесса «тестирование ПО», потребностям проекта и требованиям стандартов документирования испытаний (например, ДСТУ 2851-94 или IEEE Std. 829:1998). Проверяется, охватывает ли тестовое покрытие системные требования, удачно ли выбраны методы тестирования, можно ли будет отождествить полученные результаты с ожидаемыми (согла-

сована ли форма представления), осуществимо ли квалификационное тестирование и есть ли возможности выполнять и сопровождать план.

Для более высоких уровней целостности ПС проверки плана, созданного разработчиком, недостаточно. Должен быть построен самостоятельный план тестирования системы с целью *подтверждения* достижения требований, предъявляемых к ее программному обеспечению.

Задачи верификации и валидации проекта ПС с низкой критичностью. Деятельность по V&V проекта касается проверки проектных решений, принятых по каждому компоненту ПО, хранилищам данных и интерфейсам (внешним для ПО, между компонентами ПО и между отдельными модулями). Цели V&V – продемонстрировать, что требования к ПС корректно, точно и в полном объеме трансформированы в проект, и не запроектировано каких-либо дополнительных свойств ПС, не предусмотренных требованиями. В ходе рассмотрения документов проекта (*описания проекта ПС* и *описания проекта интерфейса*) решаются три задачи верификации и валидации:

– оценка проекта ПС,
– создание / верификация проектов тестов для V&V интеграции,
– создание / верификация проектов тестов для V&V системы.

Оценка проекта ПС заключается в анализе корректности, согласованности, полноты, точности, читабельности и тестопригодности элементов проекта, отраженных в документах. Для проверки и подтверждения правильности (по существу и по форме) информации, представленной в документах, используются те же критерии и принципы, что и при оценке требований к ПС.

Нужно заметить, что рекомендации по описанию проектов (SDD, Software Design Description) сформулированы в стандарте IEEE Std. 1016 [5] и отражают современный взгляд на формы представления декомпозиции функций и компонентов ПС, зависимостей, интерфейсов и алгоритмов модулей, а также на степень необходимой детализации проектных решений[7].

Стандарт IEEE Std. 1012 в части оценки проекта ПС также предполагает, что проверка проекта по указанным выше критериям осуществляется на всех уровнях декомпозиции проекта вплоть до описания модулей (например, на псевдокоде). Не касаясь глубины проверок можно сказать, что при верификации и валидации проекта нужно оценить:

- соответствие методов проектирования и стандартов требованиям проекта;
- соответствие потоков управления и данных функциональным требованиям и требованиям к производительности ПС (например, скорости передачи данных, точности);
- внутреннюю согласованность компонентов проекта;
- наличие в описании проекта проектных решений по всем функциям, процессу обработки информации, интерфейсам с техническими средствами, другими

[7] Этих рекомендаций нет в стандартах серии ГОСТ 34, устанавливающих требования к документам стадий эскизного и технического проектирования. Выделение и спецификация модулей ПО выполняется позже, на стадии подготовки рабочей конструкторской документации системы.

(не разрабатываемыми) компонентами ПО, а также по обеспечению контроля входных данных и отказоустойчивости ПС и др.;

- возможность проследить (трассировать) каждое проектное решение к требованиям, а также возможность использовать объективные критерии приемки для подтверждения каждого элемента проекта ПО и системы.

Задачи *создания / верификации проектов тестирования с целью V&V интеграции (системы)* выполняются по-разному, в зависимости от уровня целостности ПС.

Для ПС с уровнем целостности 1 или 2 выполнение этих задач заключается в *верификации* соответствия проектов тестов, созданных группой разработки в ходе выполнения основного процесса «тестирование ПО», соответствующим планам интеграционного (системного) тестирования и требованиям стандартов документирования испытаний (например, ДСТУ 2851 или IEEE Std. 829).

Для более высоких уровней целостности ПС проверка проектов тестов, созданных разработчиком, должна дополняться построением новых проектов тестов, соответствующих планам тестирования интеграции (системы) и прослеживаемых к системным требованиям, с целью *подтверждения* достижения требований, предъявляемых к интегрированному программному обеспечению системы.

Задачи верификации и валидации реализации ПС с низкой критичностью. Деятельность по V&V исходного кода касается проверки корректности, точности и полноты трансформации принятых проектных решений в исходный и выполнимый код и физическую структуру базы данных. В ходе рассмотрения *документов проекта* (описания проекта ПС и описания проекта интерфейса), *документации пользователя*, *документации исходного кода* и стандартов кодирования решаются следующие задачи верификации и валидации:

- оценка исходного кода и документации исходного кода,
- создание / верификация наборов тестов для V&V интеграции,
- создание / верификация наборов тестов для V&V системы,
- создание / верификация тестовых процедур для V&V интеграции,
- создание / верификация тестовых процедур для V&V системы.

Оценка исходного кода и документации исходного кода ПС заключается в анализе корректности, согласованности, полноты, точности, читабельности и тестопригодности компонентов исходного кода (документации исходного кода). Прежде всего, проверяется:

- реализует ли исходный код ПС проектные решения, описанные в документах проекта, функциональные и технические требования (производительность, точность и др.), описанные в документах требований к ПС, и учитывает ли он ограничения со стороны технических средств (разрешающая способность экрана, разрядность, правила усечения (округления) чисел и др.);
- удовлетворяет ли этот код требованиям пользователя (например, соответствуют ли экранные формы и отчеты (внешне и содержательно) ожиданиям пользователя);
- отвечает ли исходный код действующим стандартам (стандартам языка программирования, требованиям к комментариям, соглашениям именования, стандартам используемой операционной системы);
- существуют ли объективные критерии приемки каждого компонента исходного кода и можно ли протестировать код на соответствие этим критериям.

Рекомендации по форме представления и содержанию пользовательской документации дают стандарты IEEE Std. 1063 (пользовательская документация ПО) [6], ISO/IEC 9127 (пользовательская документация в пакетах ПО) [7], ISO/IEC 6592 (руководство по документированию компьютерных прикладных систем) [8].

Задачи *создания / верификации наборов тестов с целью V&V интеграции (системы)* выполняются по-разному, в зависимости от уровня целостности ПС.

Для ПС с уровнем целостности 1 или 2 выполнение этих задач заключается в *верификации* соответствия наборов тестов, созданных группой разработки в ходе выполнения основного процесса «тестирование ПО», соответствующим проектам тестов для интеграционного (системного) тестирования и требованиям стандартов документирования испытаний (например, ДСТУ 2851 или IEEE Std. 829).

Для более высоких уровней целостности ПС проверка наборов тестов, созданных разработчиком, должна дополняться построением новых наборов, соответствующих проектам тестирования интеграции (системы) и прослеживаемым к планам тестирования и системным требованиям, с целью *подтверждения* достижения требований, предъявляемых к интегрированному программному обеспечению системы.

Задачи *создания / верификации тестовых процедур с целью V&V интеграции (системы)* выполняются по-разному, в зависимости от уровня целостности ПС.

Для ПС с уровнем целостности 1 или 2 выполнение этих задач заключается в *верификации* соответствия тестовых процедур, созданных группой разработки в ходе выполнения основного процесса «тестирование ПО», соответствующим наборам тестов для интеграционного (системного) тестирования и требованиям стандартов документирования испытаний (например, ДСТУ 2851 или IEEE Std. 829).

Для более высоких уровней целостности ПС проверка тестовых процедур, созданных разработчиком, должна дополняться построением новых процедур, соответствующих наборам тестов для тестирования интеграции (системы) и прослеживаемых к планам тестирования и исходным системным требованиям, с целью *подтверждения* достижения требований, предъявляемых к интегрированному программному обеспечению системы.

Задачи верификации и валидации испытаний ПС с низкой критичностью. Деятельность по V&V испытаний касается проверки правильности интеграции компонентов ПС (между собой, а также с общесистемным ПО и техническими средствами) и подтверждении того, что требования к программному обеспечению и системе в целом, удовлетворены. Заключается в *подтверждении результатов* выполнения интеграционного, системного и приемочного тестирования (для уровней целостности 1 и 2) или *выполнении* дополнительного тестирования (для уровней целостности 3 и 4). Включает решение двух задач:

- выполнение / верификация тестов для V&V интеграции,
- выполнение / верификация тестов для V&V системы.

Исходными продуктами для выполнения V&V испытаний являются планы и процедуры для тестирования интеграции (системы), а также исходный и выполнимый код ПС.

Структура и содержание планов, проектов, наборов и процедур тестирования, а также стратегии и методы тестирования компонентов, интеграции, системы и приемки ПС подробно рассматриваются в главе 7.

6.1.3. Управление верификацией и валидацией

Управление действиями по V&V при выполнении основных процессов ЖЦ, для проверки результатов которых они вызываются, осуществляется со стороны организационного процесса ЖЦ – *процесса управления*.

В задачи процесса управления входит подготовка плана выполнения любого процесса (в том числе V&V), инициация реализации плана, мониторинг его выполнения, анализ проблем, обнаруженных при выполнении плана, определение степени завершенности задач и отчет о состоянии управляемого процесса перед руководством проекта.

Задачи управления верификацией и валидацией. В задачи управления V&V входит постоянный анализ деятельности по V&V, построение плана SVVP и его пересмотр в контексте изменений в планах проекта, а также координация результатов V&V с процессом разработки и другими поддерживающими процессами – SQA, управление конфигурацией, совместный просмотр и аудит.

В ходе управления V&V оценивается каждое предложенное изменение в системе и ПО, определяются требования к ПО, которых могут касаться изменения, и планируется выполнение задач V&V, связанных с проверкой внесенных изменений. По каждому предложенному изменению устанавливается, не приведет ли оно к появлению каких-либо новых угроз или рисков для ПО, и как оно отразится на уровнях целостности ПО. При изменении уровней целостности планы V&V пересматриваются – могут добавляться новые задачи V&V или расширяться сфера применения и интенсивность выполнения ранее запланированных задач V&V.

В контрольных точках проекта результаты V&V оцениваются, и принимается решение о переходе к следующей стадии (или множеству действий) в процессе разработки. Может быть инициировано повторное выполнение V&V.

Планы верификации и валидации разрабатываются для всех основных процессов ЖЦ ПС и могут обновляться по требованию *заказчика* (потребителя) (при выполнении им процессов приобретения ПС) или *разработчика* (поставщика) (при выполнении им процессов поставки), а также *группы сопровождения* (при выполнении процесса сопровождения).

Исходной информацией для построения (обновления) SVVP являются результаты выполнения работ (выходы) по проверяемому процессу, планы поставщика ПС, графики проекта ПС, а также выходы выполняемого процесса V&V. Изначально, деятельность по V&V планируется на основании положений контракта между заказчиком и разработчиком и концепции ПС (базовый план SVVP).

Организовать деятельность по V&V можно по-разному. Выбор формы организации работ в процессах V&V определяется масштабом, сложностью и критичностью системы. Действия по V&V могут выполняться только *разработчиками*, равными в правах, *группой разработки* (менеджерами и разработчиками), *группой качества*, совместно *представителями заказчика и разработчика*, только *заказчиками* или *независимыми экспертами*.

Процессы V&V, выполняемые организацией (подразделением), имеющей определенный уровень технической, административной или финансовой независимости от организации-разработчика (подразделения-разработчика), называют ***независимой верификацией и валидацией (IV&V, Independent V&V)***, а такую организацию (подразделение) – органом (группой) IV&V.

IV&V целесообразно применять к крупным дорогостоящим проектам или проектам критических по безопасности функционирования систем (safety critical system) для достижения объективности проверок и снижения риска разработки. Выбирая IV&V, как форму организации V&V проекта, заказчик полагается на орган IV&V как на объективную и незаинтересованную сторону для получения информации об истинном состоянии разработки, соблюдении сроков проекта и затратах на его выполнение.

Техническая независимость группы IV&V состоит в том, что ее члены не принимают участия в разработке ПС, однако обладают знаниями и опытом разработки систем данного класса и компетентны в принятии решений, касающихся технических аспектов ПС. Они не подвержены влиянию или давлению со стороны группы разработки при выполнении V&V. Техническая независимость обеспечивает возможность тщательно изучить тонкости принятых разработчиками решений и обнаружить ошибки, оставшиеся ими незамеченными. Для выполнения работ группа может использовать те же инструментальные средства поддержки разработки, что и разработчики, однако предварительно нужно провести испытания этих средств, чтобы гарантировать, что они не маскируют ошибок в анализируемом и тестируемом ПО. По возможности, группа IV&V должна применять или разрабатывать свой набор инструментов анализа и тестирования.

Административная независимость означает, что IV&V выполняется организацией, административно не связанной ни с разработчиком, ни с заказчиком. Орган IV&V вправе по своему усмотрению выбирать те части системы, для которых проводить V&V, определять аспекты и методы проверки и устанавливать график работ по V&V. Результаты проверки доводятся как до заказчика, так и до разработчика ПС. Координировать работы органа IV&V с заинтересованными организациями может третья сторона – организация-координатор.

Финансовая независимость означает, что финансирование работ по IV&V, а также контроль их выполнения осуществляется организацией, не зависящей от организации-разработчика ПС (например, заказчиком). Эта независимость защищает орган IV&V от неправильного распределения денежных средств или других форм финансовых ограничений, которые могут привести к задержке или прекращению выполнения задач IV&V и несвоевременному получению результатов проверки.

IV&V, обладающую полной технической, административной и финансовой независимостью, называют *классической* IV&V.

Структура и содержание плана верификации и валидации. В плане SVVP фиксируется информация, необходимая для управления и выполнения процессов, действий и задач по V&V в ходе ЖЦ ПС. Примерная структура плана показана на рисунке 6.2. Эта структура необязательна и может быть изменена при условии сохранения содержания плана.

Разъяснения по составлению плана SVVP содержатся в стандарте IEEE Std. 1012 [4] и вкратце приводятся ниже применительно к основным пунктам плана. Если по какому-то из пунктов плана информация не предоставляется, это должно указываться фразой «данная тема не охватывается планом» и обосновываться. В план могут быть добавлены и другие пункты.

План верификации и валидации ПС

1. Назначение
2. Ссылки
3. Определения
4. Организация работы по V&V
   - 4.1. Организация
   - 4.2. График работ
   - 4.3. Схема назначения уровней целостности ПО
   - 4.4. Распределение ресурсов
   - 4.5. Ответственности
   - 4.6. Инструменты, методологии и методы
5. Процессы и действия по V&V
   - 5.1. Процесс: Управление
     - 5.1.1. Действие: Управление V&V
   - 5.2. Процесс: Приобретение
     - 5.2.1. Действие: V&V поддержки приобретения
   - 5.3. Процесс: Поставка
     - 5.3.1. Действие: V&V планирования
   - 5.4. Процесс: Разработки
     - 5.4.1. Действие: V&V Концепции
     - 5.4.2. Действие: V&V Требований
     - 5.4.3. Действие: V&V Проекта
     - 5.4.4. Действие: V&V Реализации
     - 5.4.5. Действие: V&V Испытаний
     - 5.4.6. Действие: V&V Ввода в действие
   - 5.5. Процесс: Эксплуатация
     - 5.5.1. Действие: V&V Эксплуатации
   - 5.6. Процесс: Сопровождение
     - 5.6.1. Действие: V&V Сопровождения
6. Требования к отчетным документам V&V
7. Требования к административным процедурам
   - 7.1. Регистрация и принятие решений по аномалиям
   - 7.2. Политика итерации задач
   - 7.3. Регулирование отклонений от плана
   - 7.4. Процедуры контроля
   - 7.5. Стандарты, практические руководства и соглашения
8. Требования к документированию V&V

Рис. 6.2. **Структура плана SVVP**

Организация работы по V&V. В этом разделе должны быть описаны:

- вид V&V (IV&V), взаимосвязь процессов V&V с другими процессами ЖЦ. Должны быть четко определены связи (форма взаимодействия) между организацией (группой), выполняющей V&V, и разработчиком, полномочия для решения проблем, возникающих при выполнении задач V&V, и для приемки и утверждения результатов V&V;
- общий план выполнения действий, контрольные точки в V&V и механизм обратной связи с процессом разработки. Должна быть описана принятая модель ЖЦ разработки, этапы выполнения работ, включая даты завершения каждого этапа, предполагаемые сроки поставки программных продуктов и предполагаемые сроки

выполнения действий по V&V (согласованные со сроками завершения этапов). Сроки должны быть достаточными для эффективного выполнения задач V&V;

- согласованная схема уровней целостности ПО. Уровни целостности должны быть назначены отдельным компонентам (требованиям, функциям, модулям, подсистемам и др.), требующим применения различных подходов к V&V. Эти уровни должны пересматриваться перед началом выполнения действий по V&V для каждого основного процесса;
- ресурсы, необходимые для проведения V&V в соответствии с выбранным видом V&V и организационной структурой. Целесообразно указывать общую ориентировочную стоимость выполнения V&V, а также:
  - общую трудоемкость (в человеко-месяцах) и трудозатраты в разрезе элементов оргструктуры;
  - необходимую квалификацию персонала и потребности обучения;
  - прогнозируемую общую стоимость услуг по V&V (в рамках стоимости проекта и с учетом стоимости обучения);
  - стоимость приобретения или разработки применяемых инструментов проведения V&V (программных);
  - стоимость приобретения, эксплуатации и сопровождения аппаратного и общесистемного ПО в период выполнения V&V, необходимого для создания моделирующего комплекса;
  - затраты на удовлетворение специальных требований по обеспечению защиты, прав доступа или управления документацией;
- организационная структура коллектива, выполняющего V&V, с указанием сферы ответственности каждого его члена и решаемых задач, а также направлений взаимодействия между членами группы V&V;
- инструменты, методы и методологии, используемые при выполнении процесса V&V. Для критических систем они должны быть согласованы с заказчиком. Каждый инструмент, используемый при V&V, должен быть описан в стандартном формате с указанием названия, описания назначения, идентификационного номера, версии и т.п. Для разрабатываемых инструментов должны быть описаны их отличия от существующих инструментов, а также проведена оценка возможности их своевременной разработки.

Метод выполнения каждой задачи V&V должен быть тщательно идентифицирован и указаны ссылки на нормативные и методические документы, поддерживающие его применение. Все выбранные методы должны соответствовать выделенным ресурсам, план-графику и квалификации исполнителей.

Процессы и действия по V&V. В этом разделе должны быть идентифицированы и описаны все действия и задачи V&V для основных процессов ЖЦ. По каждому действию процесса V&V нужно описать:

- *задачи V&V* с указанием порядка их выполнения (критерия завершения), входов и выходов. Одна и та же задача может выполняться для компонентов ПО разных уровней целостности с разной степенью строгости применения методов и документирования выходов;
- *методы и процедуры* выполнения каждой задачи, а также критерии оценивания их результатов;
- *входы* для каждой задачи с указанием источника и формата;
- *выходы* для каждой задачи с указанием назначения, формата и адресатов;

- *график выполнения* задач с указанием даты начала и завершения выполнения, а также получения входной информации и отправки результатов;
- *ресурсы* для выполнения каждой задачи (по категориям – персонал, оборудование, инструменты, обучение);
- *риски и допущения*, связанные с каждой задачей, а также рекомендации по устранению, снижению или наблюдению рисков;
- *обязанности и ответственность* лиц, участвующих в решении задачи.

Требования к отчетным документам V&V. В этом разделе указываются виды, структура и содержание обязательных отчетов по V&V - Отчетов о задачах V&V, Итоговых отчетов о действиях V&V, Отчетов об аномалиях и заключительного отчета о V&V - и, возможно, других (необязательных) отчетов.

Требования к административным процедурам. В этом разделе устанавливаются требования по следующим аспектам административной поддержки V&V:

- *составление отчетов и принятие решений по аномалиям* – критерии определения и регистрации аномалий, список распределения (рассылки) отчетов для устранения аномалий, полномочия и сроки устранения аномалий, уровни критичности каждой аномалии. Подходы к классификации аномалий определяются стандартом IEEE Std. 1044 [9] и руководством [10];
- критерии определения, в какой степени необходимо осуществить *повторное выполнение задачи V&V* при изменениях входных данных или процедур, используемых при ее решении. Критерии могут основываться на оценке существенности изменений, уровня целостности ПО, отражения изменений на бюджете, графике и качестве;
- критерии и процедуры *оформления отклонений от плана*, касающиеся идентификации задачи, обоснования отклонений и степени влияния на качество ПО. Должны быть также указаны уровни ответственности за утверждение отклонений от плана;
- *процедуры контроля*, используемые при V&V. Они должны описывать, каким образом будут собираться, храниться и защищаться результаты V&V и рабочие продукты, если эти действия не поддерживаются процедурами SQA, управления конфигурацией и защиты информации. Как минимум должно быть описано, как данный план согласуется с действующими методами защиты, и как результаты V&V будут защищены от случайного или преднамеренного изменения;
- *стандарты, практические рекомендации и соглашения*, регулирующие выполнение задач V&V, включая внутренние стандарты (стандарты предприятия), методики и документы, разъясняющие политику организации.

Требования к документированию V&V. В этом разделе указываются назначение, формат и содержание документов всех видов и уровней тестирования ПС – планов тестирования, проектов тестов, наборов тестов, процедур тестирования и результатов тестирования (глава 7).

Полнота и правильность плана SVVP должны проверяться и подтверждаться руководством проекта. Для этого могут использоваться контрольные перечни вопросов, касающихся всех аспектов деятельности по V&V и учитывающих особенности прикладной области, для которой создается ПС.

Отчеты по верификации и валидации. Результаты выполнения действий и задач V&V документируются в сводном отчете о V&V, который может включать: Отчеты о задачах V&V, Итоговые отчеты о действиях V&V, Отчеты об аномалиях,

Тестовые документы V&V, Заключительный отчет о V&V и, возможно, другие отчеты, оговоренные в плане SVVP.

Отчет о задаче V&V кратко описывает ход ее выполнения и полученные основные результаты, а также включает характеристику обнаруженных аномалий, оценку степени изменения базовой версии ПО и состояния управления конфигурацией. Кроме того, он может включать описание вновь появившихся ограничений и обстоятельств, которые могут повлиять на дальнейшие действия по V&V (например, привести к изменениям в SVVP).

Итоговый отчет о действии по V&V любого из основных процессов ЖЦ обобщает результаты выполнения задач V&V и включает краткое описание задач, оценку качества ПО, обзор аномалий, идентификацию и оценку рисков и рекомендации относительно аномалий и рисков.

Отчет об аномалиях (проблемах, дефектах) описывает каждую обнаруженную аномалию и содержит оценку ее влияния на ПС и степень критичности. Подробнее этот вид отчетов описан в разделе 6.3 и главе 7.

Заключительный отчет о V&V составляется по завершении действий по вводу ПС в эксплуатацию или по завершении деятельности по V&V (если эти события не совпадают). Он включает обзор всех действий по V&V в ЖЦ ПС, всех результатов выполнения задач, обнаруженных аномалий и принятых относительно них мер, а также оценку качества ПО и описание извлеченных уроков и рекомендаций.

6.2. Виды и методы проверки программных систем

6.2.1. Классификация методов проверки

Методы, применяемые при выполнении процессов верификации, валидации, совместного просмотра и аудита, можно классифицировать по-разному, например:

- *динамические* (связанные с выполнением программ[8]) и *статические* (связанные с просмотром текста);
- *поисковые* (направленные на поиск ошибок) и *поддерживающие принятие решений* (применяемые для анализа рабочих продуктов с целью выявления каких-либо отличительных особенностей, выполнения оценок и др.);
- для *коллективной* работы и для *индивидуальной* (аналитической) работы;
- *формальные* (требующие четкого соблюдения метода) и *неформальные;*
- *коллегиальные* (выполняемые коллегами по работе, равными в правах, и не грозящие административными последствиями) и *ревизионные* и т.п.

Одни и те же методы могут применяться для выполнения разных процессов проверки, и, наоборот, при выполнении одного процесса проверки может использоваться несколько методов. Часто метод ассоциируется с задачей процесса, для решения которой он применяется, а сама задача имеет название, одноименное методу, например, анализ прослеживаемости, анализ критичности и др. Во избежание путаницы с одноименными наименованиями методов, процессов и задач далее при описании методов приводятся их эквивалентные названия на английском языке.

Для удобства изложения мы используем классификацию методов на коллективные и индивидуальные аналитические и попутно характеризуем их с точки зре-

[8] К динамическим методам мы относим и методы «прокрутки» программы вручную.

ния других классификационных признаков. Рассматриваются только методы статической проверки. Методы динамической проверки (методы тестирования) можно найти в главе 7.

6.2.2. Обзор аналитических методов

Аналитические методы предназначены для индивидуального исследования рабочих продуктов ПС с использованием (или без) инструментальных средств поддержки. Они могут быть отнесены к категории поисковых методов, методов поддержки принятия решений либо иметь двойное назначение. В таблице 6.3 дана краткая характеристика методов с указанием ссылок на доступные источники с их описанием. Нужно заметить, что ряд аналитических методов индивидуального применения, которые могут использоваться для целей V&V, имеют гораздо более широкое поле приложения (например, в процессах SQA, анализа требований, проектирования, тестирования или управления риском). Обзор многих методов анализа можно найти у Д. Веллес в работах [11], [12].

Таблица 6.3. **Методы анализа рабочих продуктов ПС**

Метод	Краткая характеристика и рекомендации по применению
Вопросники (Checklists)	Перечень вопросов, касающихся определенного аспекта проверки рабочего продукта. Создается для каждого вида рабочих продуктов (требований, проекта и др.) с учетом особенностей класса ПС (ПС реального времени, информационные и др.). Структурирован по категориям возможных дефектов. Оптимальный объем вопросника – 1 лист для каждой категории. Подробнее – в п.6.3.2. Может использоваться для поиска дефектов и обоснования принимаемых решений.
Анализ алгоритма (Algorithm analysis)	Цель - определить функциональную корректность алгоритма и его операционные характеристики. Предполагает повторный вывод уравнений или оценку пригодности численных методов.Алгоритмы анализируются с точки зрения их корректности, эффективности, простоты, оптимальности и точности. Исследует влияние усечений и округлений, оценивает обеспечиваемую точность хранения при использовании различных типов переменных (например, с простой или удвоенной точностью) и влияние преобразования типов. Обнаруживает неправильные, неэффективные (по памяти и времени) или неустойчивые алгоритмы; недостаточную точность вычислений; неработоспособность на полном диапазоне данных; неправильный анализ погрешности вычисления и др. Может использоваться для поиска дефектов.
Графический анализ (Graphical analysis)	*Анализ состояний* продуктов и процессов с использованием диаграмм: столбиковых, гистограмм, Парето, рассеяния, выполнения, причинно-следственной связи и др. Может использоваться для поиска дефектов и обоснования принимаемых решений.
Формальное доказательство правильности (Formal proof)	Применяется для математического доказательства соответствия *детальной формальной спецификации* (требований, проекта или кода) исходной спецификации. Предполагает, что для генерации спецификаций применялись формальные методы. Может использоваться для поиска дефектов.

Метод	Краткая характеристика и рекомендации по применению
Анализ интерфейса (Interface analysis)	Проверка интерфейсов различных типов - внешнего, внутреннего, аппаратного, программного, программно-аппаратного и программно-информационного (базы данных). Может включать следующее: - анализ ситуации, когда все переменные интерфейса имеют критические значения; - анализ ситуации, когда часть переменных интерфейса имеет критические значения, а остальные - имеют нормальные значения; - анализ ситуации, когда одна (поочередно каждая) переменная интерфейса принимает все значения из области значений, а остальные – только нормальные значения. Обнаруживает следующие типы ошибок: ошибки описания входов/выходов; несоответствие фактических и формальных параметров (точность, тип, количество); некорректное использование функций или вызов подпрограмм; несогласованность атрибутов глобальных переменных и др. Может использоваться для поиска дефектов.
Анализ потоков данных (Dataflow analysis) **Анализ потока информации** (Information flow analysis)	Применяется для обнаружения неопределенных входов/выходов или форматов данных, нарушения порядка обработки (чтения данных до их записи), отсутствия обработки (неоднократной записи в одни и те же переменные без использования (чтения) записанных данных). Использует информацию о потоках управления и о структурах хранения (переменных), в которые пишутся или из которых читаются данные, обрабатываемые приложением. Анализ потока информации – расширение анализа потоков данных, автоматизирован в современных CASE-инструментах. Примеры применения метода – в многочисленных публикациях на сайте: www.cc.gatech.edu/aristotle/Publications/dataflow.html Может использоваться для поиска дефектов и обоснования принимаемых решений.
Анализ влияния изменений (Change impact analysis)	Используется для определения объема и сложности внесения изменений в рабочие продукты на определенной стадии ЖЦ. Основные стадии – обзор запроса об изменениях, оценка степени распространения изменений, оценка затрат и ресурсов, анализ пользы/затрат. Использует анализ трассируемости, анализ взаимозависимостей элементов системы, анализ согласованности. Может использоваться для обоснования принимаемых решений.
Анализ базы данных (Database analysis)	Применяется для анализа программных приложений с большим объемом хранимых данных во внутренних структурах хранения или базе данных, для проверки целостности данных, возможности переполнения массивов и таблиц, непротиворечивости типов данных, способов их использования, привилегий доступа. Цель - гарантировать, что структуры данных, методы и права доступа не противоречат проекту приложения. Может использоваться для поиска дефектов.
Анализ хода работ (анализ освоенного объема) (Earned Value Analysis)	Графическое представление изменения соотношения планового и реального состояния проекта с течением времени. Используется *для контроля проекта*: соблюдения графика, финансовых затрат, трудовых ресурсов проекта. Применяется в пакетах управления проектами Microsoft Project, Open Plan, Suretrak, Primavera Project Planner, Spider Project (Россия). Сайт по управлению проектами - http://project.km.ru/ Может использоваться для обоснования принимаемых решений.

Метод	Краткая характеристика и рекомендации по применению
Анализ операционного профиля (Operational profile analysis)	Профиль - полное множество альтернатив (например, множество альтернативных сценариев работы пользователей), для каждой из которых существует определенная *вероятность* появления. Полнота множества альтернатив означает, что сумма вероятностей их появления равна единице. Анализ операционного профиля используется для рационального распределения усилий по V&V рабочих продуктов ПС, построения системы тестов и тестовых сценариев, максимально приближенных к реальным условиям среды функционирования ПС и др. Может использоваться для обоснования принимаемых решений.
Анализ трассируемости (Traceability analysis)	Существует несколько типов анализа трассируемости - трассировка требований, трассировка проекта, трассировка кода и трассировка теста. Проверяется, что каждое требование нашло отражение в проекте/коде, что все аспекты проекта/кода основаны на определенных требованиях, что тестирование дает результаты, совместимые с установленными требованиями. Основан на применении матриц трассировки требований: - матрица «*требования/метод оценивания*» - для проверки охвата всех требований определенной формой V&V; - матрица «требования/тесты» - для проверки охвата всех требований тестированием; - матрица «требования/модули» - для проверки покрытия каждого требования группой модулей (программных компонентов), реализующих это требование. Типы обнаруживаемых ошибок: - *при анализе трассируемости требований* - пропущенные функции, неправильное упорядочение требований по важности, спецификация ПО несовместима с другими спецификациями системы; - *при анализе трассируемости проекта* - нереализованные требования, неправильная интерпретация специфицированных требований, детальный проект не соответствует архитектурному; - *при анализе трассируемости кода* – несоответствие кода проекту, несоответствие кода стандартам, ПО в целом или отдельные компоненты не выполняют требуемые функции и др. Может использоваться для поиска дефектов и обоснования принимаемых решений.
Анализ деревьев событий (Event tree analysis) (ETA)	Идентифицирует компоненты, события отказов которых могут повлиять на безопасность функционирования системы. *Последствия* каждого события (вызывающего цепь других событий) трассируются до тех пор, пока не прояснятся опасности (самого верхнего уровня), связанные с этими событиями. По ходу идентификации хронологической цепочки событий определяется вероятность каждого события и комбинации событий. Дерево событий представляет полный спектр возможных *последствий* определенного события отказа компонента (являющегося корнем дерева). Не отражает *причин* появления события отказа. Подробнее – в главе 7, а также на сайте: www.cis.strath.ac.uk/teaching/ug/classes/52.422/risk.assessment.two.doc Может использоваться для поиска дефектов.

Метод	Краткая характеристика и рекомендации по применению
Анализ деревьев отказов (ошибок) (Fault tree analysis) (FTA)	Возможный отказ компонента системы рассматривается как событие верхнего уровня, *причины* которого скрыты внутри компонента. Анализ дерева отказов – это процесс *идентификации рисков*, связанных с компонентом системы. Дерево отказов – иерархическая структура, на вершине (в корне) которой – состояние отказа компонента, прилегающие ветви – события, приводящие к переходу в это состояние. Исток каждой ветви – состояние элемента компонента, приведшее к наступлению соответствующего события. Это *дедуктивный* метод поиска условий и факторов (причин) появления события отказа компонента. Трассирование событий может начинаться не с события отказа компонента системы, а с внешнего события, представляющего опасность. Подробнее – в главе 7, а также на сайте: www.cis.strath.ac.uk/teaching/ug/classes/52.422/risk.assessment.two.doc Может использоваться для поиска дефектов.
Анализ режимов (механизмов) и последствий отказа (Failure mode and effect analysis (FMEA)) **Анализ режимов и критичности отказов** (Failure mode and criticality analysis (FMECA))	Методы *предотвращения появления отказов*. Используются для идентификации типов потенциальных дефектов и механизмов их появления (режимов работы системы, в которых они могут появиться) с целью определения их возможного влияния на изучаемый объект (систему, компонент), а также классификации типов дефектов по критичности последствий (в критических системах). Широко применяются в инженерии надежности и безопасности технических систем. Представляют подход, обратный (индуктивный) по отношению к методу FTA (направление анализа – от события отказа отдельного компонента (с учетом его критичности или без) до опасного внешнего события для пользователя). Идентифицируют те компоненты, которые требуют повышенного внимания при разработке. Полезны при определении стратегии управления риском, тестирования ПО в условиях ограниченных ресурсов, обеспечения отказоустойчивости ПО, построения процедур сопровождения и др. Подробнее – в главе 7 или на сайте www.itemsoft.com/fmeca.shtml Могут использоваться для обоснования принимаемых решений.
Анализ опасностей (Hazard analysis) **Анализ угроз** (Threat analysis)	Опасность/угроза - состояние системы или среды, попадание в которое приводит к катастрофическим последствиям для общества. В программной инженерии термин «опасность» чаще связывается с процедурами обеспечения *безопасности функционирования* (safety), а «угроза» - с процедурами *защиты информации* (security). Методы применяются при разработке высокоцелостных систем. Используют метод ETA. Могут использоваться для поиска дефектов и обоснования принимаемых решений.
Анализ риска (Risk analysis)	Может использовать методы FMEA, FMECA, FTA, а также методы, представленные в главе 10 Может использоваться для обоснования принимаемых решений.
Анализ критичности (Criticality analysis)	Используется для определения стратегий V&V, применимых к компонентам (модулям) с высоким риском отказов. Ранжирует компоненты по степени вклада в безопасность. Устанавливает, в какой степени безопасность функционирования системы зависит от корректной работы отдельных компонентов ПО, и каким образом на нее могут повлиять отклонения в реализации ПО.

Метод	Краткая характеристика и рекомендации по применению
	Классификация компонентов по уровню критичности может выполняться с помощью указания их роли в обеспечении безопасности (не играет, играет косвенно, играет непосредственно, играет вспомогательную роль), а также с помощью индекса критичности. Использует результаты анализа опасности, FMEA, FMECA для идентификации требований (элементов проекта, кода), неправильная реализация которых может вызвать наиболее серьезные последствия. Может использоваться для обоснования принимаемых решений.
Проверка за столом (Desk checking) (Code reading)	Чтение исходного кода и комментариев программы независимым экспертом (не разработчиком) с целью обнаружения очевидных дефектов, проверки интерфейсов процедур, проверки соответствия стандартам соглашениям. Изучение содержательной стороны программы с целью ее сравнения с внешними спецификациями. Метод полезен для выявления следующих типов ошибок (дефектов): - *ошибки логики и управления:* недостижимый код, неправильное вложение циклов и ветвей, неправильная последовательность процессов, бесконечные циклы, метки без ссылок и др.; - *ошибки вычислений:* неправильный доступ к компонентам массива, несоответствие списков параметров; ошибки инициализации, устаревшие данные, неопределенные переменные, необъявленные переменные, неправильное употребление переменных (локальных и глобальных), неопределенные границы данных, неэффективная передача управления и др.; - *прочие:* вызов не существующих подпрограмм, ошибки входов/выходов, неэффективное программирование, плохой стиль. Может использоваться для поиска дефектов.
Ортогональная классификация дефектов (ODC) (Orthogonal defect classification)	Характеристика каждого обнаруженного дефекта по трем аспектам: - *тип дефекта*. Назначается исходя из характера действий, которые должны быть выполнены для устранения дефекта (исправление документации, интерфейсов, функций, алгоритмов, назначений значений и др.). Служит мерой совершенства продукта; - *триггер дефекта* (причина обнаружения). Направление проверки (в рамках одного из трех видов проверки – обзоры, автономное и функциональное тестирование, системное тестирование и испытания), при выполнении которой был обнаружен дефект (проверка полноты, проверка согласованности, путевое тестирование, нормальный режим и др.). Служит мерой эффективности V&V; - *влияние дефекта*. Критерии оценки воздействия, которое дефект может оказать (или уже оказал) на степень удовлетворенности конечного пользователя продуктом (функциональность, надежность, удобство применения и др.). Служит мерой качества продукта при использовании. ODC – парадигма измерения характеристик процессов и продуктов на основе причинно-следственного анализа дефектов. Подробности – на сайтах: www.research.ibm.com/softeng/ODC/ODC.HTM www.chillarege.com/odc/articles/odcrisk/odcissre.html Может использоваться для обоснования принимаемых решений.

Метод	Краткая характеристика и рекомендации по применению
Анализ сложности (Complexity analysis)	Используется для выявления компонентов проекта или кода, сложных для корректной реализации, тестирования и сопровождения. Может использоваться для обоснования принимаемых решений.
Прототипирование (Prototyping)	Применяется для согласования и оценки реализуемости требований, оценки проекта интерфейса пользователя, предсказания размера и сложности ПС, определения стратегий тестирования. Может использоваться для поиска дефектов.

6.2.3. Обзор методов коллективной проверки

Коллективные проверки предполагают участие в них нескольких (по крайней мере, двух) лиц и принятие *согласованного коллективного решения*[9]. Лица, принимающие участие в коллективных проверках, должны, с одной стороны, владеть методами индивидуальной аналитической проверки рабочих продуктов, а с другой - обладать навыками коллективной работы.

Коллективные проверки могут проходить либо в форме неформальных собраний, либо в форме формальных совещаний, заранее подготовленных и имеющих строго определенную процедуру проведения. Формальные проверки строже и считаются более эффективными, однако, их сложнее применять на практике. К тому же, в условиях нехватки знаний процедур их проведения и отсутствия опыта практического применения, формальные проверки могут не дать желаемой отдачи. Решение о типе проверки принимается руководством проекта на основе опыта разработчиков и критичности проекта. Допускаются компромиссные решения о проведении полуформальных проверок.

Объектами всех видов проверок являются любые *рабочие продукты* ПС (или их части), создаваемые на стадиях ЖЦ и в процессах ЖЦ. Действует принцип: «все, что может быть проверено, должно быть проверено тщательно и своевременно». Проверяться могут:

- документы ПС (описание требований, описание проекта, описание постановок задач, текст кода, эксплуатационная документация и др.), «представляющие» (характеризующие) ПС на определенной стадии ее существования;
- документы проекта разработки ПС (планы, протоколы и др.), характеризующие ход разработки ПС и ресурсы проекта;
- документы процесса ЖЦ ПС (планы, описание процесса, руководства по выполнению и др.).

Нужно заметить, что в стандартах, регламентирующих процессы проверки, нет «единодушия» в терминах, используемых для определения одних и тех же понятий. В частности, множество материалов, передаваемых для проверки, может называться и «рабочим продуктом» (в единственном и множественном числе), и «программным продуктом» (в единственном и множественном числе).

Далее мы будем считать, что независимо от вида проверки и стадии ЖЦ, на которой она проводится, объектом проверки является *один* рабочий продукт (или,

---

[9] Если проверка, осуществляемая группой лиц, не предполагает *коллективного* обсуждения и принятия *согласованного* решения, она не является коллективной (например, одновременное тестирование разных компонентов ПС разными тестировщиками).

для краткости, просто *продукт* проверки), который может представлять собой *совокупность* любых передаваемых для проверки материалов (документов, программ), например, совокупность тестов, совокупность описаний постановок задач и др.

Проведение формальных коллективных проверок регламентируется стандартом IEEE Std. 1028 [13]. В этом стандарте по единой схеме описаны 5 видов проверки:

технический обзор,
управленческий обзор,
инспекция,
сквозной контроль,
аудиторская проверка.

Для каждого из этих видов определены цели, роли и обязанности участников, требования к входам и выходам, критерии начала и завершения, а также методические указания по процедурам проверки.

Краткая характеристика перечисленных методов дана в таблице 6.4. Более подробное описание одного из самых распространенных и мощных методов проверки – метода формальной инспекции - содержится в разделе 6.3.

Таблица 6.4. **Обзор методов коллективной проверки**

Цель и объект проверки	**Участники и роли**
Технический обзор (Technical review)	
Систематическая оценка пригодности продукта *для использования по назначению* и идентификация расхождений со спецификациями и отклонений от стандартов, руководств и планов. Возможно рассмотрение альтернатив и выработка рекомендаций. *Инициируется* лицом, которое несет ответственность за разработку такой системы, функциональные и технические характеристики которой будут удовлетворять требованиям потребителя. Планируется (в планах управления проектом). Возможны дополнительные технические обзоры по запросу руководства проекта, группы качества, группы системной инженерии, группы программной инженерии. Поддерживает принятие решений ЛПР (лицом, принимающим решение) о возможности *перехода к следующей стадии ЖЦ*, о необходимых корректирующих действиях, касающихся технических аспектов разработки системы. *Объекты обзора* – спецификация требований, описание проекта, тестовые документы и др. Участники обзора (в количестве не менее 3 человек) подбираются лидером (с помощью руководства) из числа компетентных специалистов в области разработки ПС. Имеют достаточно времени для изучения продукта, способны обнаружить технические проблемы и предложить способы их решения (альтернативы). По запросу лидера могут прослушать вводный обзор процедур проверки или продукта.	Участники – группа проекта (руководство, заказчики и пользователи – необязательно) Роли: - *ЛПР* (принимает решение) - *лидер обзора* (обучен, готовит и ведет совещание), - *секретарь* (фиксирует проблемы), - *персонал проекта* (активно участвует в обсуждении, предлагает альтернативы) - *руководящий состав* (идентифицирует проблемы, требующие управленческих решений)

Цель и объект проверки	**Участники и роли**
Подготовленность участников к техническому обзору проверяется лидером. Если участники недостаточно подготовлены, лидер может перенести совещание. Продукт изучается участниками *до начала* совещания (этап подготовки). Описание проблем передается лидеру. Лидер упорядочивает проблемы по степени важности. Знакомит с ними авторов продукта. На совещании проблемы обсуждаются, оцениваются связанные с ними риски, *предлагаются технические решения*, даются рекомендации руководству. Если зафиксированы многочисленные проблемы или есть проблемы, классифицированные как критические, может быть запланирован новый обзор для *проверки переделок* продукта. *Процессы ЖЦ*: совместный просмотр, V&V, SQA	
Управленческий обзор (management review)	
Систематическая оценка состояния процессов ЖЦ с целью *контроля продвижения проекта*, определения состояния выполнения планов и графиков, распределения ресурсов, соблюдения стандартов и регулирующих норм, а также оценивание эффективности используемых приемов руководства. *Инициируется* лицом, несущим непосредственную ответственность за разработку системы. Планируется (в планах проекта). Поддерживает принятие решений ЛПР (лицом, принимающим решение) *о корректирующих действиях*, изменениях в распределении ресурсов проекта и др. *Объекты обзора* – планы проекта, планы любых процессов, отчеты о выполнении процессов, процедуры управления. Продукт проверяется участниками (в количестве не менее 2 человек) *до начала* совещания (этап подготовки). Описание проблем передается лидеру. Лидер упорядочивает проблемы по степени важности. Знакомит с ними авторов продукта. На совещании проблемы обсуждаются, оцениваются связанные с ними риски, даются рекомендации ЛПР, предлагаются *мероприятия по улучшению процессов*. *Процессы ЖЦ*: совместный просмотр, V&V, SQA	Участники - руководство проекта и персонал (заказчики и пользователи - необязательно) Роли: - *ЛПР* (принимает решение) - *лидер обзора* (обучен, готовит и ведет совещание), - *секретарь* (фиксирует проблемы), - *руководящий состав* (активно участвует в обсуждении), - *персонал проекта* (предоставляет необходимую информацию)
Коллегиальная проверка (Peer review)	
Общее название методов проверки, имеющих следующие *характерные особенности*: • участники проверки – коллеги автора продукта (равные в правах); • участие лиц, занимающих руководящие посты по отношению к любому из членов группы проверки, не допускается; • лидеры группы имеют специальную подготовку (по применению методов); • цель проверок – поиск дефектов, повышение качества продукта, совершенствование процессов; • рекомендуется сбор статистических данных (по продукту и процессу проверки). Коллегиальная проверка *институциализируется* как ключевое направление процесса программной инженерии на 3-м уровне зрелости по модели CMM.	

	Цель и объект проверки	Участники и роли
Формальная инспекция (Formal inspection)	Систематическая проверка продукта с целью *выявления и идентификации проблем*, включая дефекты, и отклонений от стандартов и спецификаций. Планируется (в плане проекта, плане V&V). Дополнительные инспекции могут проводиться по инициативе руководства проекта, группы SQA или автора. Проводится группой *обученных лиц*, владеющих методом инспекции (в количестве 3-6 человек). *Каждый* участник инспекции играет роль инспектора, независимо от других его ролей. Роли могут совмещаться. Автор не может быть координатором, референтом или секретарем. Продукт исследуется с *разных* позиций (например, код оценивается в контексте соответствия спецификации требований и проекта, тестопригодности, безопасности и др.). Координатор определяет, в каком *контексте* будет исследоваться продукт каждым инспектором. В помощь инспекторам разрабатываются вопросники. Готовность инспекторов к совещанию проверяется координатором. Обязательный элемент инспекции - *анализ* проблем и *исправление* выявленных дефектов. Рекомендуется *сбор статистических данных* о проблемах, дефектах, трудоемкости инспекции, трудоемкости устранения дефектов, потраченных ресурсах и др. Эти данные используются для совершенствования процессов разработки и повышения эффективности инспекции. *Объекты инспекции* – спецификация требований, описание проекта, исходный код, тестовые документы, пользовательская документация, руководство по сопровождению, процедуры инсталляции и др. Объект инспекции должен представлять собой *завершенный* продукт, отвечающий требованиям стандартов к его форме и содержанию. Продукт изучается инспекторами на этапе подготовки к совещанию. Перечень проблем передается координатору. На совещании референт делает *подробный обзор* продукта. По ходу чтений инспекторы поднимают проблемы. Согласованные дефекты регистрируются, передаются автору. Устанавливается срок переделки продукта. Исполнение проверяется. *Процессы ЖЦ:* основные процессы, V&V, SQA	Роли: - *координатор инспекции* (*обучен*, готовит совещание, отвечает за результаты), - *секретарь* (фиксирует проблемы), - *референт* (представляет продукт на совещании), - *автор* (на совещании дает разъяснения, после совещания – устраняет дефекты), - *инспектор* (при подготовке к совещанию ищет дефекты, активно участвует в чтениях и обсуждении проблем, может предлагать альтернативы)
Сквозной контроль	Это планируемый систематический контроль хода разработки продукта и оценка его состояния. Дополнительные обзоры продукта могут проводиться по запросу руководства проекта, группы качества или автора. Цели – *поиск дефектов*, улучшение продукта, обсуждение альтернативных решений, оценка соответствия стандартам и спецификациям.	Альтернативное название метода: *Сквозной просмотр*

Цель и объект проверки		**Участники и роли**
Сквозной контроль (Walkthrough)	Дополнительные цели – обмен применяемыми технологиями, приемами и стилями, обучение участников (в количестве 2 – 7 человек). *Объекты контроля* – те же, что и для инспекции. Продукт изучается группой контроля на этапе подготовки к совещанию. Проблемы и вопросы, выносимые на обсуждение, классифицируются как *общие* или *частные* и передаются лидеру. На совещании автор делает *общий обзор* продукта. По его окончании проводится обсуждение продукта, в ходе которого контролеры поднимают общие вопросы. Далее автор начинает *детальный (сквозной) просмотр* продукта (порядок просмотра выбирает автор или лидер), а контролеры задают свои *частные* вопросы только тогда, когда автор представляет соответствующий фрагмент продукта. По завершении обсуждения каждого фрагмента фиксируются решения и предложения по каждому из рассмотренных вопросов. *Процессы ЖЦ:* основные процессы, V&V, SQA	Роли: - *лидер* (определяет цели обзора, ведет совещание) - *секретарь* (фиксирует все решения, а также комментарии по поводу дефектов, стилей, улучшений и др.) - *автор* (представляет продукт (в целом и детально) и отвечает на вопросы) - *контролеры* (готовят общие вопросы к продукту в целом и частные вопросы к его частям)
Аудиторская проверка (Audit) (альтернативные названия - ревизия, инспекция)		
Независимая проверка продукта, процесса или множества процессов с целью оценивания соответствия стандартам, руководствам, планам, процедурам, условиям договора и другим критериям. Планируется (в планах проекта). Проводится группой аудиторов (1-5 человек). Ответственность за принятие решений относительно *необходимости* аудита, организации-аудитора, целей, критериев, сферы, объектов аудита, несет инициатор. Он может быть руководителем проверяемой организации, представителем заказчиков, пользователей или третьей стороны. Ответственность за достижение целей аудита и его *беспристрастное* проведение в установленном порядке несет ведущий аудитор. *Объекты проверки* – все виды планов, тексты договоров, рекламации заказчиков и пользователей, руководства пользователя, отчеты и статистические данные процессов проверки, программный продукт на всех стадиях ЖЦ, применяемые стандарты, нормы, руководства, процедуры и др. Перед началом аудиторской проверки инициатор (как правило) уведомляет о ней организацию-разработчика программного продукта, для того чтобы она могла обеспечить готовность к аудиту людей и проверяемых материалов. Далее ведущий аудитор готовит *план аудита* и получает его одобрение инициатором. В ходе подготовки к аудиторской проверке аудиторы изучают план аудита, информацию о проверяемой организации и продуктах, применимые при проверке и оценке стандарты, нормативы и руководства, критерии оценивания. Этап проверки открывается общим собранием группы аудиторов и проверяемой организации с целью обзора целей, сферы, процедур и графика аудиторской проверки.	Роли: - *ведущий аудитор* (готовит план аудита, подбирает группу и руководит ее работой, готовит отчет), - *секретарь* (фиксирует дефекты, решения, рекомендации группы аудита), - *аудиторы* (работают по плану аудита, не имеют личной заинтересованности, фиксируют замечания и рекомендации), - *инициатор* (инициирует процесс аудиторской проверки, утверждает ее план и результаты, распространяет отчет), - *проверяемая организация* (способствует аудиту - предоставляет документы и интервьируемых сотрудников, устраняет замечания аудиторов)

Цель и объект проверки	Участники и роли
Проверка состоит в сборе (путем опроса сотрудников, изучения документов) и анализе *объективных данных* о состоянии дел в проверяемой организации. Наблюдения обсуждаются и подтверждаются сотрудниками организации до проведения заключительного общего собрания. По завершении проверки готовится отчет. Ответственность за его объективность несет ведущий аудитор. Отчет утверждается инициатором проверки и направляется всем заинтересованным лицам проверяемой организации для устранения замечаний аудиторов. *Процессы ЖЦ:* аудит	

Методы неформальной проверки представляют собой вариации формальных проверок, адаптированные к целям и условиям проектов, в рамках которых разрабатываются программные продукты, и с учетом критичности этих продуктов.

Порядок решения проблем, обнаруженных при проверке, регулируется процедурами регистрации проблем (разногласий) и процедурами внесения изменений в рабочие продукты. Если рабочий продукт находится в сфере управления конфигурацией, нужно руководствоваться стандартом ISO/IEC 12207 (процессы управления конфигурацией и решения проблем), в противном случае - можно использовать рекомендации применяемого метода проверки. Типовая форма отчета о проблеме, обнаруженной при проверке, представлена на рисунке 6.3.

Сообщение о проблеме в рабочем продукте
Наименование проекта:_____ **Наименование рабочего продукта (части)** _____
Идентификатор проблемы:_____ **Подробное описание проблемы:**_____ _____
Серьезность проблемы: *очень важная* (возможен существенный дефект, повторить процедуру проверки), *умеренно важная* (возможен умеренный дефект, устранение проконтролировать), *не важная* (несущественный технический дефект, контроль необязателен) **Критичность проблемы для проекта:** *критическая* (дефект на верхнем уровне, глобальные ресурсоемкие переделки), *устранимая* (локальные ресурсоемкие переделки (время)), *легко устранимая* (не требует дополнительных затрат ресурсов) **Потенциальный ущерб:** оценка ущерба на каждой из последующих стадий ЖЦ в случае не решения проблемы **Приоритет устранения:** *немедленно, к определенному сроку, в следующей версии*
Состояние проблемы: *предполагаемая, открытая, закрытая, закрытая с замечаниями, повторно возникшая* **Хронология разрешения проблемы:** *4 важные даты, связанные со сменами состояний проблемы*
Дата обнаружения:_____ **Кто обнаружил проблему:**_____
Кому направляется: _____ **Ответственный за устранение:**_____
Составитель (подпись):_____ **Дата составления:**_____

Рис. 6.3. **Типовая форма отчета о проблеме**

Рекомендации по классификации проблем содержатся в стандарте IEEE Std. 1044 [9] и руководстве по его применению – IEEE Std.1044.1 [10].

6.3. Формальные инспекции

6.3.1. Элементы процесса инспекции

В программной инженерии термин «инспекция» определяет два понятия [14]:

– инспекция – это методология статического анализа, основанная на неформальной визуальной проверке выборочных продуктов разработки с целью обнаружения отклонений от стандартов разработки и других проблем. Инициируется извне проекта (например, руководством, заказчиком и др.) и выполняется группой, не включающей членов группы проекта. Может применяться в процессе аудита;

– инспекция – это процесс *формальной* проверки рабочего продукта *владельцем* этого рабочего продукта и группой его коллег по проекту с целью обнаружения ошибок, упущений, несогласованностей и проблематичных областей в рабочем продукте.

Именно это, второе определение понятия инспекции и используется далее.

Формальная инспекция – это инспекция, обладающая следующими отличительными особенностями:

- автор рабочего продукта – обязательный участник процесса инспекции;
- инспекция выполняется в установленном порядке в соответствии с утвержденными процедурами и графиками и с целью обнаружить все существенные дефекты и проверить их устранение автором;
- информация, извлекаемая при инспекции, собирается и используется для управления проектом, оценивания качества и улучшения процесса разработки;
- при проведении инспекции всегда используются контрольные вопросники, что облегчает поиск дефектов и помогает правильно их классифицировать;
- члены группы инспекции прошли *обучение* процессу инспекции;
- каждая инспекция соответствует определенному *типу инспекций* и проводится в соответствии с требованиями, предъявляемыми к инспекциям соответствующего типа.

Тип формальной инспекции определяется видом (типом) анализируемого рабочего продукта, стадией ЖЦ, на которой она проводится, содержательной направленностью контрольного вопросника, критерием начала проведения инспекции и критерием завершения инспекции.

Первое опубликованное описание процесса инспекции появилось в 1976 г в статье автора этого метода М.Фагана [15]. С тех пор, как отмечает А.Аккермен, инспекции неоднократно доказывали свое преимущество перед другими видами проверок, в частности, техническими обзорами и сквозным контролем, и, безусловно, способствовали улучшению качества ПС, повышению производительности труда разработчиков и совершенствованию управляемости процесса разработки [16].

Основные элементы процесса формальных инспекций таковы:

- семь четко определенных *этапов (шагов) процесса инспекции*, выполняемых в установленной последовательности (рисунок 6.4);
- пять строго установленных *ролей участников инспекции*: координатор (председатель, арбитр), референт (комментатор, чтец), секретарь (регистратор), инспектор, автор продукта (разработчик);

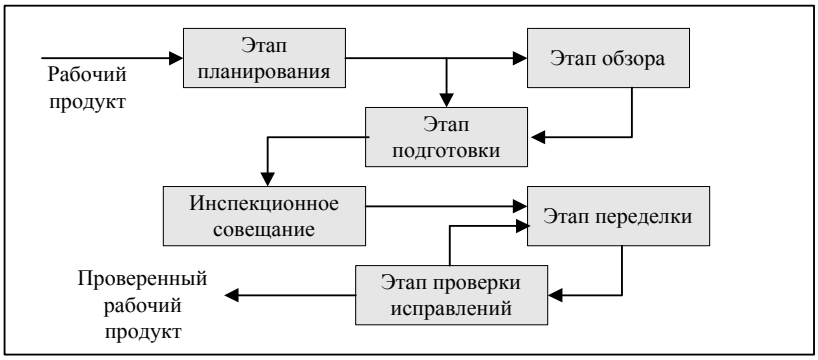


Рис. 6.4. **Этапы процесса формальной инспекции**

- четко определенная процедура сбора и классификации данных о рабочем продукте и процессе;
- инспектируемый рабочий продукт;
- инфраструктура для поддержки процесса инспекции.

Ключевые факторы успеха процесса инспекций - *обучение* участников и строгое соблюдение всех шагов и применяемых процедур. Очевидно, что эффективность формальной инспекции может быть достигнута в организациях-разработчиках, занимающих 2-й и выше уровень зрелости по модели CMM, поскольку на этих уровнях институциализированы такие направления деятельности, как обучение, планирование, контроль исполнения и др. [17].

Инспекции проводятся небольшой группой специалистов - от 4 до 7 человек[10], представляющих разные области знаний в программной инженерии. Усиленный состав может созываться при рассмотрении рабочих продуктов самого высокого уровня (концепции или требований). Дополнительные участники могут привлекаться в той степени, в которой необходимо знать их точку зрения о рабочем продукте. Поскольку цель инспекции – обнаружить и зафиксировать дефекты, роль инспекторов исполняют фактически все члены группы инспекции независимо от других исполняемых ими ролей.

Для определения готовности рабочих продуктов к рассмотрению, используется *критерий начала* инспекции, а для подведения результатов инспекции – *критерий завершения* инспекции.

Критерий начала инспекции устанавливается с учетом типа рабочего продукта, но в целом, основывается на предположении о том, что рабочий продукт будет в достаточной степени готов к использованию по назначению после устранения в нем дефектов, обнаруженных при инспекции. То есть, инспекции подлежат законченные («почти готовые») рабочие продукты. Для инспекции кода программ, например, критерий ее начала – логическая завершенность и отсутствие по крайней мере синтаксических ошибок, а для инспекции документа – соответствие общим требованиям к документам подобного типа.

Критерий завершения инспекции – наличие отметки о завершении действий,

---

[10] Нами описана классическая схема инспекции. В настоящее время проводятся исследования эффективности процесса инспекции, выполняемого только двумя инспекторами (например, [18]).

внесенных в перечень обязательно выполняемых для каждого этапа процесса инспекции [14]. Критерий завершения используется главным образом для того, чтобы гарантировать, что дефекты, обнаруженные при инспекции и классифицированные инспекторами как *существенные*, устранены. Устранение *умеренных* или *не существенных* дефектов, которые не оказывают значительного влияния на применение рабочих продуктов по назначению, может не включаться в критерий завершения.

Все работы по представляемому на рассмотрение рабочему продукту должны быть «заморожены» до завершения инспекции, в противном случае она теряет всякий смысл. В связи с этим, инспекция должна заранее *планироваться* в рамках сроков разработки рабочих продуктов.

По завершении каждой инспекции о ней составляется подробный отчет. Кроме того, результаты всех инспекций рабочего продукта обобщаются в Итоговом отчете об инспекциях.

6.3.2. Этап планирования

Планирование инспекции выполняется *координатором*, который устанавливает тип инспекции, проверяет готовность рабочих продуктов к инспекции соответствующего типа, подбирает квалифицированную инспекционную группу и распределяет роли, устанавливает место и время встречи группы для совместной работы, обеспечивает участников необходимыми инспекционными материалами.

Используя критерий начала инспекции, координатор, прежде всего, отсеивает рабочие материалы, которые заведомо не готовы для инспекции, а для остальных определяет, имеют ли они приемлемый объем, позволяющий завершить обсуждение в рамках одного заседания группы. Если объем проверяемого материала слишком большой, координатор делит его на части.

Затем координатор подбирает членов инспекционной группы, обладающих знаниями, навыками и опытом проведения инспекций и способных выполнить проверку рабочих продуктов соответствующего типа (не по форме, а по содержанию) и сформировать о них компетентное мнение.

Каждому участнику инспекции поручается определенная роль.

Координатор (председатель) руководит группой инспекторов и несет ответственность за результативность ее работы. Это самая активная роль на всех этапах инспекции (кроме устранения дефектов). Основная функция координатора – обеспечить условия, при которых группа не будет «выплескивать свои эмоции», инспекционное совещание не превратится в поиск *причин* ошибок, а обсуждение не увязнет в мелких проблемах, упуская существенные.

Координатор отвечает за сбор объективных данных в ходе инспекции, составление документов инспекции и осуществление обратной связи с менеджером проекта, а также с группой программной инженерии, которая может использовать результаты инспекции для принятия решений о совершенствовании процессов.

Ввиду важности данной роли для инспекции, координатор должен иметь специальную подготовку (пройти обучение).

Референт (комментатор). Несет ответственность за эффективность работы группы при рассмотрении рабочего продукта (далее просто *продукта*) в ходе ин-

спекционного совещания, зачитывая текст документа и комментируя его[11]. Лучшим кандидатом на эту роль может быть разработчик, непосредственно участвующий в работах на *следующей* стадии ЖЦ и заинтересованный в качестве продукта, являющегося для него исходным.

Секретарь (регистратор). Несет ответственность за регистрацию дефектов, признанных таковыми сообща в ходе инспекционного совещания, и точность изложения информации о каждом обнаруженном дефекте. Секретарь должен также регистрировать информацию о каждой поднятой, но не решенной проблеме, а также обо всех дефектах, обнаруженных в документах вышестоящего уровня.

Автор. Это разработчик инспектируемого продукта. Его основная обязанность – предоставить всю необходимую информацию о продукте, а также отвечать на вопросы инспекторов, чтобы гарантировать, что отдельные неясности (неверные толкования) не приняты ими за дефекты. Во время совещания он вместе со всеми ищет недостатки в своей работе.

Автор несет ответственность за устранение как *существенных*, так и *не существенных* дефектов, однако устраняет последние не к очередной встрече с инспекторами, а когда для этого появятся необходимые ресурсы.

Инспекторы. Независимо от исполняемой специальной роли в инспекции (координатор, референт, автор или секретарь) *все члены группы* считаются инспекторами и несут ответственность за обнаружение дефектов на этапе подготовки и во время инспекционного совещания.

Группа инспекторов формируется из разработчиков программного обеспечения. Исключение из этого правила составляют программисты, выполняющие кодирование отдельных фрагментов программного продукта, которые не должны привлекаться для разработки тестовых процедур для своего кода.

В качестве инспекторов могут также приглашаться лица из любых других групп проекта системы, организации-разработчика или организации-заказчика, както группа системной инженерии, группа тестирования, группа качества, администраторы системы и пользователи. «Внешние» инспекторы могут использоваться в качестве экспертов для повышения эффективности процесса инспекции.

Сформировав группу инспекции, координатор раздает ее участникам ***пакет материалов инспекции***, в состав которого входит:

1. Заметки для инспекционного совещания. Этот вид документа предназначен для того, чтобы информировать членов группы инспекции о предмете будущей инспекции.

Заметки должны иметь определенную структуру и содержать следующую информацию, подготовленную координатором:

- *титульная часть*. Включает административную информацию – название проекта, дату рассмотрения продукта, стадию проекта, идентификацию компонента проекта, частью которого является рабочий продукт, название и версию инспектируемого рабочего продукта, сведения о координаторе;
- *тип совещания*. Тип совещания отвечает его цели – новая инспекция (первое чтение) или повторная инспекция (проверка и обсуждение исправлений);
- *вид инспекции*. Указывается вид инспектируемого рабочего продукта (тре-

[11] При инспекции кода референт описывает структуру программы и комментирует программные конструкции, давая им собственные оценки.

бования, проект и др.);

- *расписание инспекции*. Указывается дата, время, продолжительность совещания, время, отведенное на подготовку к совещанию, и размер продукта;
- *состав группы инспекции*. Указывается ФИО каждого участника, сфера компетенции применительно к категории дефектов, отведенная роль в инспекции;
- *комментарии*. Ссылки на документы предыдущего уровня разработки проекта, которые могут содержать информацию, полезную для понимания существа продукта и проблем с ним.

2. *Рабочий продукт* на носителе. Это, скорее всего, твердая копия продукта (документа), хотя нельзя исключить возможность визуального анализа экранных объектов (форм, отчетов, диаграмм, фрагментов кода и др.) или использования всевозможных статических анализаторов, при условии, что для проведения инспекции создана соответствующая среда;

3. *Контрольные вопросники*. Это списки вопросов, каждый из которых касается определенной технической проблемы, которая может быть выявлена в результате поиска ответа на вопрос (приложение 5). Вопросники составляются для всех возможных типов рабочих продуктов (например, для описания требований к системе, описания проекта, описания набора тестов и др.). Вопросы классифицируются по категориям возможных дефектов в рабочем продукте соответствующего типа и, необязательно, по подкатегориям, что позволяет обратить внимание инспекторов на определенные области возможных проблем. Каждому вопросу присваивается составной порядковый номер, отражающий номер категории, номер подкатегории и номер самого вопроса (рисунок 6.5);

Обозначение вида продукта	**Наименование вида (типа) рабочих продуктов**		
	Номер категории (подкатегории)	**Наименование категории (подкатегории) дефектов**	
		Номер вопроса	**Формулировка вопроса для выявления дефекта определенной категории**

Рис.6.5. **Структура контрольного вопросника**

4. *Бланки форм (или журналы) регистрации проблем*. В журнале инспекторы будут описывать суть проблемы и связанную с ней информацию. Форма журнала может быть выбрана координатором произвольно, однако в нем должно найти отражение следующее:

- *идентификация инспектора* (код инспектора[12], его ФИО и координаты, сфера компетенции (применительно к виду продуктов и категориям дефектов));
- *дата* выполнения подготовки;
- *время*, отведенное на подготовку, и *время*, реально потраченное инспекто-

[12] Всевозможные «коды» и «номера» в документах инспекции могут сослужить свою службу, если координатор (или группа SEPG) попытается автоматизировать процесс и создаст базу документов инспекции. Это было бы очень полезно для статистической обработки данных инспекций, изучения эффективности процесса инспекции и др.

ром на подготовку к совещанию;

- *объем продукта*, охваченный проверкой;
- *описание каждой обнаруженной проблемы*, а именно:
  - - присваиваемый ей порядковый *номер*,
  - - *местонахождение* в продукте (номер страницы, раздела, параграфа, абзаца, предложения),
  - - *описание сути проблемы*,
  - - *категория предполагаемого дефекта* и *номер вопроса*, поиск ответа на который послужил толчком к обнаружению проблемы;
- *количество обнаруженных проблем* за время подготовки.

Нужно заметить, что в пакет материалов инспекции входят и другие документы, однако они не раздаются инспекторам на этапе планирования и поэтому рассматриваются дальше.

Формируя группу инспекции, координатор должен установить степень знакомства каждого инспектора с материалом и определить необходимость проведения *краткого обзора*. Если члены группы инспекторов выбраны из состава разработчиков проекта, тестировщиков, или инженеров по качеству, хорошо знакомых с проектом, этап обзора не требуется. Если нет – координатор планирует краткий обзор продукта его автором.

Для того чтобы инспекции были максимально эффективны, Д. Франкович предлагает руководствоваться следующими общими рекомендациями [19]:

- продолжительность планируемого совещания – не более 2 часов;
- периоды и продолжительность проведения инспекции должны быть согласованы с менеджером проекта и предусмотрены в плане проекта;
- обязательно должен быть подготовлен контрольный вопросник, отвечающий потребностям инспекции *определенного вида рабочих продуктов*;
- нужно предусматривать при планировании инспекции не менее двух дней на подготовку (продолжительность может колебаться в зависимости от размера и сложности рабочего продукта);
- объем рассматриваемого рабочего продукта не должен превышать 20 страниц;
- автор не должен исполнять роль координатора, референта или секретаря;
- группа инспекции должна сосредоточиться на обнаружении дефектов, а не на проблемах их устранения.

Цели, задачи и роли при планировании инспекции подытожены в таблице 6.5.

Таблица 6.5. **Этап планирования**

Цель	Организовать инспекцию и спланировать ресурсы для ее проведения
Вход	• Завершенный вариант рабочего продукта • Сопровождающие материалы • Критерий начала инспекции (наличие продукта, отвечающего минимальным требованиям к оформлению и содержанию) • Исторические данные инспекции (при наличии)
Задачи	• Проверить критерий входа • Подобрать группу инспекции • Определить, нужен ли обзор • Составить план-график (расписание) инспекции(й) • Сформировать и распространить пакет материалов инспекции

Измерения (меры)	• Трудоемкость планирования • Размер рабочего продукта
Выход	• Комплект материалов для инспекции сформирован и вручен • Запланирован обзор (при необходимости)
Роли	*Координатор:* • Проверяет, отвечает ли рабочий продукт критерию входа • Подбирает членов группы инспекции и автора (разработчика) • Принимает решение о том, нужен ли обзор • Планирует совещание(я) и комплектует материалы совещания *Автор:* • Рассматривает рабочий продукт совместно с координатором • Предлагает кандидатов в группу инспекции • Дает рекомендации о том, нужен ли обзор продукта

6.3.3. Этап обзора

Краткое обзорное совещание планируется в том случае, если инспекционная группа не знакома с материалом. Группа должна иметь представление об истоках рабочего продукта (знать содержание документов, на основе которых он был разработан). Для исходного кода, например, это должны быть документы описания требований и проекта (в части, касающейся этого кода).

В ходе обзора автор дает характеристику рабочего продукта и разъясняет его связь с другими продуктами, место в системе, функциональное назначение и предполагаемое использование. Обзор планируется в следующих обстоятельствах:

– инспекционная группа не знакома с классом программных систем, который представляет данный рабочий продукт;
– рабочий продукт новый или рассматривается впервые;
– инспекция нового проекта;
– при разработке рабочего продукта использованы новые методы.

Цели, действия и роли в обзорном совещании подытожены в таблице 6.6.

Таблица 6.6. **Этап обзора**

Цель	Ознакомление
Вход	Инспектируемый рабочий продукт Обзорный материал, подготовленный автором
Задачи	Координатор проводит обзорное совещание Автор представляет рабочий продукт
Измерения	Время автора на подготовку Продолжительность обзорного совещания
Выход	Возросший уровень понимания рабочего продукта Распределение обязанностей (при необходимости) Комплект рабочих материалов инспекции Материалы обзорного совещания

6.3.4. Этап подготовки

Это ключевой этап процесса инспекции, в ходе которого инспекторы готовятся к исполнению своих ролей в инспекционном совещании. Каждый инспектор самостоятельно изучает продукт, сосредотачивая внимание на тех его областях и тех аспектах проверки, которые входят в его компетенцию. Он выявляет проблемы, используя для этого контрольные вопросники, ориентированные на обнаружение

дефектов, типичных для данного вида рабочих продуктов, а также проверяет соответствие рабочего продукта тем продуктам предыдущих этапов разработки, которые служили для него исходными. В своей работе инспектор может пользоваться любыми аналитическими методами, рассмотренными в разделе 6.2.

Установленные недостатки фиксируются в журнале регистрации проблем наряду со временем, потраченным на изучение рабочего продукта. Заполненные журналы представляются координатору до начала инспекционного совещания.

Известно, что именно на шаге подготовки инспекции выявляется до 90% дефектов, поэтому важно правильно выбрать время его продолжительности. Оно должно быть не меньше длительности инспекционного совещания, а для очень сложных и критических систем может увеличиваться в несколько раз [20].

Существенное влияние на качество подготовки в отведенное на нее время оказывает опыт членов группы в проведении инспекций. Поэтому вопросам обучения инспекторов должно уделяться особое внимание.

Качество подготовки инспекции зависит также от личного опыта членов группы в проведении инспекций, а также от полноты и глубины контрольных вопросников, используемых для анализа рабочего продукта.

Цели, действия и роли на этапе подготовки подытожены в таблице 6.7.

Таблица 6.7. **Этап подготовки**

Цель	Достичь понимания продукта и идентифицировать потенциальные проблемы
Вход	• Комплект рабочих материалов для инспекции (заметки, вопросники, журналы) • Представление о рабочем продукте • Материалы обзора
Задачи	• Изучить рабочий продукт • Идентифицировать проблемы, используя специально подготовленный контрольный вопросник (либо типовой вопросник в приложении 5) • Фиксировать время на подготовку • Сформировать перечень обнаруженных проблем
Измерения	• Продолжительность подготовки • Количество проблем
Выход	• Сформированный сводный перечень проблем в журналах регистрации проблем • Зафиксированное время подготовки
Роли	*Инспектор:* • Изучает рабочий продукт • Идентифицирует проблемы, используя вопросник • Фиксирует время подготовки *Координатор:* • Решает все задачи, связанные с подготовкой инспекции • Устраняет следующие проблемы самой инспекции: - инспектор не может участвовать в инспекции - существенное отклонение рабочего продукта от требований *Референт:* • Решает все задачи, связанные с подготовкой к обсуждению продукта • Решает, как лучше представить рабочий продукт

6.3.5. Инспекционное совещание

Это кульминационный этап инспекции, в ходе которого участники совместно рассматривают рабочий продукт, обсуждая выявленные *проблемы* и, по согласованию, присваивая им *статус дефектов*, подлежащих устранению разработчиком.

Созывает совещание и далее координирует его работу координатор. В начале совещания референт (комментатор) дает общую характеристику продукта, а автор уточняет неясности. Если группа новая (или включает новых членов), координатор может начать совещание с краткого введения, напоминая участникам их роли и повторно объясняя цель инспекции и назначение продукта.

Далее референт переходит к последовательному (параграф за параграфом) изложению своего видения продукта и его связи с другими продуктами системы. Совещание строится таким образом, что любой инспектор может прервать референта, если считает, что обсуждаемый фрагмент рабочего продукта имеет, с его точки зрения, дефект. Чтение материала приостанавливается для краткого обсуждения. Если инспекторам удалось достичь согласия, - дефект квалифицируется по определенной категории и заносится секретарем в *список дефектов*, после чего чтение возобновляется.

Список дефектов представляет собой упорядоченное (в порядке регистрации) множество описаний дефектов.

Каждое описание дефекта можно считать логическим продолжением описания соответствующей проблемы, обнаруженной инспектором (или несколькими инспекторами независимо), хотя и не исключено совместное обнаружение дефекта, не замеченного при подготовке.

Описание дефекта должно включать:

- *порядковый номер дефекта в списке* (а возможно, и составной номер со ссылкой на код инспектора, обнаружившего проблему, и номер проблемы, признанной дефектом),
- *категория (подкатегория) дефекта* и *номер вопроса* (по вопроснику), поиск ответа на который послужил толчком к обнаружению проблемы;
- *характер дефекта* (упущение автора, ошибочное решение, ошибка верхнего уровня);
- *уровень серьезности дефекта* (существенный, умеренный, несущественный)
- *местонахождение* в продукте (номер страницы, раздела, параграфа, абзаца, предложения) или ссылка на описание соответствующей проблемы, признанной дефектом;
- *ориентировочное время* на устранение дефекта;
- *идентификация лица* (группы), которому (которой) продукт направляется на переделку;
- *тип повторной проверки продукта* (C – совещание автора с координатором, R – совещание автора с координатором и инспекторами);
- *отметка об устранении дефекта*. Дата подтверждения устранения дефекта. Проставляется только после повторной проверки переделок продукта (на этапе 7).

Уровень серьезности дефекта устанавливается для определения приоритета устранения обнаруженных дефектов в продукте. Дефекты, которые могут стать

причиной того, что система не будет соответствовать требованиям, должны классифицироваться как *существенные*, а такие дефекты, как типографские и грамматические ошибки, небольшие отклонения от стандартов - как несущественные. С уровнем серьезности дефекта связан и тип повторной проверки на *этапе проверки.*

Координатор должен пытаться ограничить время обсуждения каждой проблемы. Если обсуждение не закончено в рамках отведенного времени, он объявляет проблему нерешенной и продолжает совещание. Секретарь отмечает в соответствующем журнале нерешенную проблему как *открытую* для переноса ее обсуждения на дополнительное время (так называемый *«третий час»*).

Группа обязательно должна прийти к согласию по вопросу, является ли каждый предполагаемый дефект действительно дефектом. Иногда то, что считается дефектом, на самом деле может быть ошибкой инспектора, вызванной, например, не пониманием того, что имел в виду автор. Вопрос может быть решен изучением других документов - истоков рабочего продукта, которые должны быть предоставлены инспекторам. Если инспекторы находят, что и исходные документы также содержат ошибку, - вопрос остается открытым и его решение откладывается до следующего этапа инспекции – «третьего часа».

Если лимит времени совещания (2 часа) исчерпан, а чтение (разбор продукта) *не закончено*, назначается дата *повторного совещания*.

Если чтение завершено, но остались открытые проблемы, - принимается решение о переносе их обсуждения на «третий час».

После совещания автор и координатор должны определить сроки и тип проверки внесенных исправлений в продукт (этап 7). Автор получает копию списка выявленных дефектов для их устранения. Список дефектов используется автором при переработке продукта, а также служит отправной точкой на последующих этапах инспекции для идентификации устраненных дефектов.

Группа инспекции использует список дефектов для присвоения рабочему продукту *категории готовности* в зависимости от его состояния. Р. Ибенау (Ebenau) предлагает следующую категоризацию состояний рабочего продукта [21]:

- *продукт категории A* (от accept). Рабочий продукт принимается в полном объеме без дальнейших проверок и переделок. Это не значит, что он полностью свободен от дефектов, однако в нем не найдено существенных дефектов, а также большого количества несущественных, которые могли бы послужить причиной отклонения от спецификаций;
- *продукт категории C* (от conditional). Рабочий продукт принят условно с последующей проверкой переделок на совещании координатора с автором. Это значит, что отмечены некоторые существенные дефекты, но их в рабочем продукте немного и их переделка предположительно не повлечет за собой значительных изменений логики (концепции) рабочего продукта;
- *продукт категории R* (от Reinspect). Рабочий продукт подлежит повторной инспекции после переделок автором. Состояние продукта таково, что переделки должны быть подвергнуты повторной итерационной инспекции на совещании координатора, автора и, по крайней мере, одного инспектора. Это означает, что в продукте обнаружено значительное количество существенных дефектов или переделки могут повлечь за собой изменение исходной концепции рабочего продукта.

В конце совещания подводятся его итоги, подсчитывается общее число рассмотренных проблем, установленных дефектов и проблем, оставшихся открытыми.

Результаты совещания фиксируются в о*тчете об инспекции*, который составляется по завершении обсуждения продукта.

Отчет об инспекции. Должен включать:

- *титульная часть*. Идентификация проекта и продукта, вид инспекции, тип проведенного совещания;
- *состояние продукта*. Один из возможных вариантов текста:
  - «Чтение завершено. Продукту присвоена категория (A/C/R)»,
  - «Чтение не завершено. Будет продолжено на совещании <указание даты> с <точное указание места в рабочем продукте, с которого должно быть продолжено чтение>. Категория не присвоена»;
- *дата проверки* исправлений и *тип повторной проверки* (если продукту присвоена категория C или R);
- *список дефектов (набор описаний дефектов);*
- *итоговая информация:*
  - число рассмотренных проблем,
  - число зарегистрированных дефектов,
  - число открытых проблем, перенесенных на «третий час»,
  - число закрытых проблем <число проблем, закрытых на «третьем часе»>, из них установлено дефектов <число> (заполняется по завершении «третьего часа»);
- *комментарии*. Информация о необходимости рассмотрения новых документов для решения открытых проблем (например, документов предыдущего уровня разработки проекта), необходимость привлечения новых лиц для участия в обсуждении (экспертов, разработчиков продуктов предыдущих уровней) и др.;
- *подписи членов группы инспекции*. Указывается ФИО каждого участника и подпись.

Если в «третьем часе» нет необходимости, отчет об инспекции передается руководителю проекта для того, чтобы он мог оценить текущее состояние продукта и выделить ресурсы для его переделки.

Если обзор продукта не был доведен до конца (чтение прервано), - инспекторам передаются обновленные записки для инспекционного совещания, в которых указана дата повторной инспекции и зафиксировано место в рабочем продукте, с которого будет продолжено обсуждение.

В таблице 6.8 подытожены цели, процедуры и роли на этапе совещания.

Таблица 6.8. **Этап инспекционного совещания**

Цели	Обсудить рабочий продукт. Достичь консенсуса и зафиксировать дефекты
Вход	• Полный список предполагаемых проблем (в журналах) • Зафиксированная продолжительность подготовки
Задачи	• Введение в материал • «Чтения» рабочего продукта • Идентификация и фиксация дефектов в основном списке дефектов • Подведение итогов по дефектам (обзор дефектов) • Определение категории состояния рабочего продукта (A/C/R) A: *Принят* – отмечено несколько несущественных дефектов; устранение дефектов - на совести автора. C: *Принят условно* – автор должен отчитаться за переделку перед координатором

	R: *Отложен на повторную инспекцию* – подлежит повторной инспекции со стороны координатора, автора и по крайней мере одного инспектора
Измерения	• Данные о производительности (количество рассмотренных проблем, закрытых, открытых) • Данные о дефектах (количество, серьезность, количество дефектов, допущенных на предыдущем уровне разработки)
Выход	• Категория состояния продукта • Сформированный полный список дефектов • Сформированные заметки инспекционного совещания о нерешенных проблемах • Частично сформированный отчет об инспекции
Роли	*Автор:* • Сохраняет объективность, внимательно слушает, делает заметки • Ничего не опасается *Инспекторы:* • Объективны в оценках продукта, не руководствуются личным отношением к автору • Хорошо подготовлены и принимают активное участие в обсуждении • Заботятся не о форме, а о содержательной стороне высказываемых замечаний *Координатор:* • Вовремя начинает и заканчивает совещание • Поддерживает энергетику (темп) совещания на надлежащем уровне • Добивается достижения консенсуса членов группы по вопросам о дефектах *Референт:* • Выбирает наиболее эффективную последовательность представления рабочего продукта • Отвечает на вопросы, возникающие у членов группы *Секретарь:* • Знаком с классификационной схемой дефектов • Фиксирует все данные в общем списке дефектов • Готовит сводную информацию о дефектах для Итогового отчета об инспекции

6.3.6. Этап дополнительного обсуждения - «третий час»

«Третий час» – это условное название дополнительного периода времени вне совещания, который используется для обсуждения или закрытия открытых проблем, а также для того, чтобы автор мог уточнить вопросы, связанные с устранением дефектов.

«Третий час» может проходить в форме дополнительного совещания или индивидуальной работы инспекторов. Он следует не обязательно сразу после основного совещания и не ограничен одним часом.

Состав участников совещания не ограничивается теми, кто присутствовал на основном инспекционном совещании. Это могут быть инспекторы (необязательно все), а также менеджеры, внешние технические эксперты и т.п. В ходе «третьего часа» автор получает информацию, которая будет способствовать более эффективному внесению изменений в продукт, например, справки о существе дефектов, об-

наруженных в документах более высокого уровня, что поможет ему исправить собственный продукт, не дожидаясь устранения дефектов в исходных продуктах. В ходе «третьего часа» автор также участвует в закрытии открытых проблем, каждая из которых может быть идентифицирована как дефект, подлежащий устранению.

Если «третий час» проводится в форме индивидуальной работы, он используется для изучения открытых проблем, поиска информации для разрешения разногласий или для подготовки отчета о разногласиях или запросов на изменения, связанных с существенными дефектами, обнаруженными в документах более высокого уровня, находящимися под управлением конфигурацией.

Итоговая информация о результатах «третьего часа» вносится в отчет об инспекции, после чего он направляется руководителю проекта.

6.3.7. Этап переделки рабочего продукта

Цель этого этапа инспекции – исправить дефекты, обнаруженные при инспекции. До начала этапа автор должен быть обеспечен всей необходимой информацией и *реальными* временными ресурсами. Нужно учитывать, что дефекты в рабочем продукте могут быть вызваны ошибками, допущенными *другими лицами* на предыдущих стадиях разработки системы, а устранение этих дефектов невозможно без исправления ошибок, с которыми они связаны. Автор не должен брать на себя ответственность за чужие ошибки и исправлять их (или «додумывать» исправление), даже в том случае, если виновники ошибок далеко (или вообще больше не работают по данному проекту).

Автор несет ответственность за исправление всех дефектов, содержащихся в списке дефектов инспекции, - и существенных, и несущественных.

Существенные дефекты он обязан устранить в установленные сроки и отчитаться за их устранение перед координатором или группой инспекции. При наличии времени, в эти же сроки он может устранить и несущественные дефекты, в противном случае, они должны быть устранены до выпуска рабочего продукта (передачи его другому процессу).

Решение о любых корректировках рабочего продукта из-за существенных дефектов должно приниматься менеджером проекта. Каждое исправление должно быть зафиксировано в системе контроля за изменениями в проекте и прослеживаться менеджером проекта до тех пор, пока станет возможным повторное утверждение (визирование) рабочего продукта. В таблице 6.9 подытожены цели, процедуры и роли на этапе переделки рабочего продукта.

Таблица 6.9. **Этап переделки**

Цель	Устранить дефекты в рабочем продукте
Задачи	Автор устраняет все дефекты Автор уведомляет координатора о завершении переделок Автор сообщает о не устраненных дефектах менеджеру проекта
Измерения	Количество дефектов Трудоемкость переделки каждого дефекта
Выход	Измененный рабочий продукт, не содержащий зафиксированных ранее существенных дефектов
Роли	*Автор:* • Устраняет дефекты • Сообщает о не устраненных дефектах менеджеру проекта • Уведомляет координатора о завершении переделок

6.3.8. Проверка внесенных изменений или повторная инспекция

На этом этапе проверяются все изменения, сделанные автором в рабочем продукте в связи с устранением существенных дефектов, поскольку при попытке переделать продукт он мог внести новые дефекты.

В зависимости от категории состояния рабочего продукта, присвоенной ему на инспекционном совещании, координатор может:

- если продукт имел категорию C, сам проверить и визировать изменения;
- если продукт имел категорию R, провести процесс инспекции в полном объеме, но сосредоточить внимание инспекторов только на внесенных изменениях и их взаимозависимостях.

Повторная инспекция может быть проведена и для продукта категории C при наличии большого числа дефектов или если устранение некоторых дефектов требовало больших и взаимосвязанных исправлений.

Если все существенные дефекты устранены, все открытые проблемы разрешены и рабочий продукт удовлетворяет инспекторов, координатор «сдает» продукт, отмечая завершение инспекции в *итоговом отчете об инспекции*.

В противном случае в ранее составленном *отчете об инспекции* проставляется отметка об устраненных на момент проверки дефектах, обновляются даты предполагаемого устранения не устраненных дефектов, дата текущей проверки, дата и вид следующей повторной проверки и согласующие подписи. Автор опять возвращается к устранению дефектов.

Итоговый отчет об инспекции. В этом отчете помимо информации, отражаемой в отчетах об инспекции, составляемых (обновляемых) после каждого инспекционного совещания (вид продукта, его объем и прочее), содержатся:

- *сводные данные обо всех этапах инспекции* рабочего продукта - количество циклов инспекции (повторных инспекционных совещаний, повторных инспекций разных видов), количество участников, потраченное время и др. Эти данные нужны для определения трудоемкости процесса инспекции;
- *сводные данные о проблемах* - общее количество рассмотренных проблем, количество проблем, «снятых» после согласования, количество установленных дефектов. Эти данные могут оказаться полезными при определении состава участников будущих инспекций (поскольку свидетельствуют о профессиональной пригодности инспекторов и эффективности их работы);
- *сводный список дефектов.* Этот список строится на основе списка дефектов, который изначально был составлен на первом инспекционном совещании, и уточнялся по мере выявления новых и устранения найденных дефектов. Поскольку сводный список нужен, прежде всего, руководству проекта для определения *состояния* рабочего продукта, а также группе качества для определения эффективности *процесса разработки* продукта и поиска причин его возможной неэффективности, - список составляется в разрезе категорий дефектов (существенные, умеренные, не существенные), а в рамках каждой категории – в разрезе характера дефектов (рисунок 6.6).

При желании, информационные разрезы можно выбрать и иначе (например, по категориям (подкатегориям) вопросов в вопроснике);

- *подписи.*

Номер дефекта в списке дефектов	Категория дефекта	Уровень серьезности дефектов								
		Существенных			Умеренных			Не существенных		
		упущений	ошибок	ошибок верхнего уровня	упущений	ошибок	ошибок верхнего уровня	упущений	ошибок	ошибок верхнего уровня
1										
2										
…										
Итого										

Рис. 6.6. **Пример сводного списка дефектов**

Цели, задачи и роли на этапе проверки исправлений подытожены в таблице 6.10.

Таблица 6.10. **Этап проверки исправлений**

Цель	Завершить инспекцию и утвердить ее результаты
Вход	Измененный рабочий продукт, если он имел категорию C или R Рабочий продукт, если он имел категорию A Все формы для инспекции
Задачи	Координатор проверяет рабочий продукт: Если продукт имеет категорию R, перейти на этап планирования Если продукт имеет категорию C, проверить переделки. Если не все зафиксированные дефекты устранены, вернуться на этап переделки Если продукт имеет категорию A, утвердить решение инспекции Сформировать Итоговый отчет об инспекции Предоставить материалы инспекции менеджеру проекта Направить копию Итогового отчета группе программной инженерии
Измерения	Трудоемкость проверки Трудоемкость составления Итогового отчета об инспекции
Выход	Проверенный рабочий продукт Утвержденные результаты инспекции Составленный Итоговый отчет об инспекции
Роли	*Автор:* • Готовится к повторной инспекции, если продукт имеет категорию R • Выполняет и переделывает то, на что указал координатор *Координатор:* • Планирует повторную инспекцию, если продукт имеет категорию R • Проверяет переделки с автором, если продукт имеет категорию C • Направляет материалы инспекции менеджеру проекта • Оформляет Итоговый отчет об инспекции • Направляет копию Итогового отчета группе программной инженерии

6.3.9. Повышение эффективности процесса инспекции

Процесс формальной инспекции доказал свою эффективность как механизм обнаружения и устранения дефектов в рабочих продуктах во многих зарубежных организациях-разработчиках ПС. Он широко используется и пристально изучается (подтверждения этому можно найти, например, на сайте: www2.ics.hawaii.edu/

~johnson/FTR/Bib/bib-master.html).

Однако его внедрение еще не гарантирует, что он сразу будет максимально эффективным. Процесс должен адаптироваться группой программной инженерии к условиям его применения и постоянно улучшаться.

Главными элементами, которые должны «настраиваться» от инспекции к инспекции, является *скорость инспекции* (количество рассмотренных проблем за одно совещание) и *вопросники* [20].

Если на инспекционное совещание выносится слишком большой рабочий продукт, то в условиях ограничения срока инспекция может быть либо поверхностной, либо совещания будут проходить с отставанием от графика. Если продукт невелик – затраты на инспекции могут оказаться неоправданными.

В больших рабочих продуктах - инспекции в первую очередь должны подвергаться наиболее критические его части, а менее критические – планироваться на следующую инспекцию. Скорость инспекции должна корректироваться с учетом сложности продукта и наличия квалифицированных инспекторов.

Если процесс инспекции внедряется впервые, для проведения инспекций могут быть использованы типовые вопросники. Далее они должны настраиваться так, чтобы инспекторы уделяли внимание именно тем типам ошибок, которые присущи не только инспектируемому виду рабочих продуктов, но и классу ПС (например, встроенному ПО реального времени в бортовых компьютерах, системам с базами данных и др.). Настройка вопросников сделает более эффективным этап подготовки к инспекции, а также поможет структурировать процесс обсуждения материала. Однако, перегруженные информацией вопросники могут снизить эффективность инспекций, отвлекая внимание от существенных вопросов. Оптимальный объем вопросов, выносимых на одну инспекцию, ограничивается одним листом [20].

Для оценки эффективности базового процесса инспекции и установления тенденций в адаптированных процессах инспекции нужно регулярно собирать и анализировать данные об объеме инспектируемых продуктов, времени на подготовку и проведение совещаний, общем количестве дефектов, обнаруженных при каждой инспекции, типах дефектов и стадиях ЖЦ, на которых они обнаружены.

Для того чтобы сбор и обработка данных стала нормой при управлении процессом инспекции, в организации-разработчике должна быть внедрена *программа измерений* этого процесса и разработан *план измерений*[13].

Программу измерений предлагается строить на основе целеориентированного подхода и включать в нее следующие положения:

- определение целей инспекции;
- определение и разработка множества метрик;
- сбор метрических данных;
- проверка и утверждение метрик;
- анализ метрик;
- обобщение данных и определение тенденций процесса;

---

[13] Программа измерений может внедряться не только для повышения эффективности инспекции, но и любого метода формальной коллегиальной проверки, например, сквозного контроля.

- использование метрик для выработки решений по улучшению процесса инспекции;
- подготовка (обновление содержания) документов и инструментов базового процесса инспекции (входных и выходных форм, вопросников, структуры базы данных и приложений, используемых для автоматизации процесса инспекции);
- переподготовка инспекторов.

Метрики процесса инспекции. Метрики для включения в *план измерений* процесса инспекции разрабатываются с использованием парадигмы «цель-вопрос-мера» (глава 4).

Проверяемые цели инспекции таковы:

G1*: планировать действия по инспекции рабочего продукта ПС в качестве заключительного этапа работ на той стадии ЖЦ, на которой он разработан.*

G2*: идентифицировать и устранять дефекты в рабочих продуктах, начиная с самых ранних стадий ЖЦ.*

Вопросы, связанные с целью G1:

Q1: *Сколько времени необходимо для подготовки и проведения инспекции?*

Q2: *Достаточны ли ресурсы (время и персонал), выделенные для проведения инспекции?*

Q3: *Завершена ли запланированная инспекция?*

Q4: *Эффективен ли процесс инспекции для устранения дефектов?*

Вопросы, связанные с целью G2:

Q5: *Какие данные нужно собирать в ходе инспекции?*

Q6: *Как трассируется распространение дефекта?*

Q7: *Чем измеряется эффективность инспекции?*

Q8: *Какова оптимальная интенсивность (скорость) инспекции?*

Q9: *Как сопоставляются разные методы устранения дефектов?*

Атрибуты и метрики для оценки достижения целей представлены в таблице 6.11.

Таблица 6.11. **Собираемые данные и метрики процесса инспекции**

Вопрос	Метрика	Код	Описание
Q1	Время подготовки	Тпо	Суммарное время подготовки всех инспекторов. Время подготовки каждого инспектора должно быть указано в отчетах об инспекции.
Q1, Q5	Время проведения	Тпр	Время, потраченное группой инспекции на проведение инспекции.
Q3	Число инспекций, запланированных в проекте	Ипл	Общее количество инспекций рабочих продуктов, запланированное в плане проекта. Эти данные могут использоваться для сравнения с реальным числом инспекций, выполненных на определенную дату, чтобы узнать, соблюдается ли график инспекции. Эти данные могут быть получены из плана проекта.
Q3	Число проведенных инспекций	Ипр	Общее количество инспекций, проведенных на определенную дату по текущему плану проекта. Эти данные могут быть получены из отчетов об инспекциях.
Q2	Размер группы	Ргр	Количество членов группы инспекции, включая координатора, инспекторов и секретаря. Эти данные могут быть получены из отчетов об инспекциях.

Вопрос	Метрика	Код	Описание
Q3, Q4	Коэффициент выполнения	Квып	*Нормализованная величина (нормализация по количеству):* отношение количества инспекций, проведенных на определенной стадии проекта (или всего в проекте), к общему количеству инспекций, запланированных для данной стадии (или проекта). **Квып = Ипл / Ипр** Может использоваться для того, чтобы понять, как лучше планировать инспекции. Если Квып слишком мал, это может означать, что для выполнения всех запланированных инспекций выделено недостаточно времени.
Q1	Затраты времени на инспекцию	Тсум	Суммарное количество часов, потраченных на подготовку и проведение инспекции **Тсум = Тпо + Тпр** Может использоваться для определения затрат времени на инспекции по каждой стадии ЖЦ и по проекту в целом. Позволяет определить, растет (убывает) ли продолжительность инспекций по ходу проекта. Имея исторические данные можно определить, каковы минимальные и максимальные затраты времени.
Q2	Интенсивность подготовки к инспекции	Спо	Отношение размера продукта к среднему времени, потраченному на подготовку к инспекции одним инспектором (Тпо_ср) Среднее время на подготовку инспектора к инспекции продукта **Тпо_ср = Тпо / Ргр**
			Интенсивность подготовки к инспекции **Спо = Рпрод / Тпо_ср** Эта величина показывает взаимосвязь между количеством часов, потраченных на подготовку к инспекции, и размером группы инспекторов и продукта. Чем число больше, тем меньше времени тратится группой на подготовку к инспекции продукта. Имея исторические данные, можно определить, достаточна ли интенсивность подготовки к инспекции продукта данного вида и объема группой определенного размера.
Q2, Q4, Q8	Интенсивность проведения инспекции	Спр	Отношение размера продукта к среднему времени, потраченному на проведение инспекции одним инспектором (Тпр_ср). Среднее время на проведение инспекции продукта инспектором **Тпр_ср = Тпр / Ргр** Интенсивность проведения инспекции **Спр = Рпрод / Тпр_ср** Эта величина показывает взаимосвязь между количеством часов, потраченных на проведение инспекции, и размером группы инспекторов и продукта. Чем число больше, тем меньше времени тратится группой на проведение инспекции продукта.

Вопрос	Метрика	Код	Описание
			Имея исторические данные, можно определить, достаточна ли интенсивность проведения инспекции продукта данного вида и объема группой определенного размера.
Q5, Q6	Число дефектов, обнаруженных инспекцией	Добн	Общее количество дефектов, обнаруженных во время инспекции. Показатель эффективности процесса инспекции. Чем выше это число, тем выше отдача от выделенных ресурсов. Информация может быть получена по отчетам об инспекциях и описаний (списка) дефектов, а сами дефекты – классифицированы по типам и степени серьезности.
Q5, Q6	Число дефектов, не обнаруженных инспекцией	Днеобн	Общее количество дефектов в рабочем продукте, подвергавшемся инспекции, обнаруженных на последующих стадиях в ЖЦ. Определяется после окончания инспекции продукта на определенной стадии ЖЦ. Чем меньше дефектов скрылось от инспекции, тем выше ее эффективность (тем ближе мы к цели 2). Большое количество скрытых дефектов свидетельствует о проблемах в самом процессе инспекции.
Q5, Q7	Размер продукта	Рпрод	В зависимости от вида рабочего продукта размер определяется количеством страниц или строк, или числом функциональных элементов (экранов, отчетов, функций и др.), числом условных единиц функциональности (FP) или числом KSLOC.
Q7	Продуктивность инспекции	Иэф	*Нормализованная величина (нормализация по размеру или времени):* количество дефектов, обнаруженных инспекцией в продукте определенного размера (плотность дефектов), или – за определенный период времени (подготовки или проведения инспекции) (частота обнаружения). Может вычисляться в любой момент процесса инспекции.
Q6, Q7	Коэффициент обнаружения	Кобн	Процент дефектов, обнаруженных при инспекции, к общему числу дефектов **Кобн = 100 * (Добн / (Добн + Днеобн))** Не может быть вычислен до тех пор, пока инспектируемый продукт не пройдет тщательное тестирование и использование по назначению. Однако, может быть показателем раскрываемости дефектов на стадии ЖЦ, если вычисляется как процент дефектов, обнаруженных на стадии, к общему числу дефектов, уже обнаруженных в проекте на определенный момент. Чем выше это число – тем лучше раскрываемость.
Q9	Относительная эффективность метода	Мэф	Отношение количества дефектов, обнаруженных в единицу времени инспекции, к количеству дефектов (Дт), обнаруженных в единицу времени тестирования **Мэм = (Добн / Тсум) / (Дт / время тестирования)** Дефекты должны быть однотипны. Полезно применять для сравнения методов обнаружения дефектов в коде.

Сбор и анализ данных Ответственность *за сбор* объективных данных несет группа инспекции. Необходимые для этого формы и инструменты (например, таблицы Excel) должны быть подготовлены заранее группой SEPG или SQA. Собранные данные направляются группе SQA *для анализа* состояния процесса.

Перед тем как данные будут проанализированы, должна быть проверена их точность и согласованность. Для этого могут применяться критерии, выработанные в результате анализа однотипных исторических данных, либо опросы опытных инспекторов, а также выборочный аудит данных.

Анализ данных проводится с целью определения направлений улучшения процессов ЖЦ ПС или самой инспекции. Эффективными инструментами анализа данных являются (см. главу 5):

- *диаграммы Парето*. Могут использоваться для *ранжирования* проблем (дефектов) по критичности, серьезности, для принятия решения о *приоритете устранения* дефектов, *отслеживания причин* возникновения дефектов для предупреждения их появления в будущем;

- *диаграммы выполнения*. Отражают поведение определенной переменной во времени. Могут использоваться для *выявления тенденций* в процессе инспекции;

- *диаграммы рассеяния*. Могут использоваться для выявления возможной *корелляции двух переменных*, например, интенсивности проведения инспекций (Спр) и их способности к обнаружению дефектов (Кобн);

- *столбиковые диаграммы*. Могут использоваться для представления соотношения определенных характеристик продуктов или процессов (вычисляемых метрик) и наблюдаемых (базовых) величин - размера, стоимости, времени. Например, соотношения данных об эффективности инспекций (Иэф) и размере групп, выполнявших эти инспекции (Ргр);

- *причинно-следственные диаграммы*. Могут использоваться при проведении сеанса «мозгового штурма» определенной проблемы, связанной с самим процессом инспекции. Наиболее эффективны, если в их построении участвуют *несколько* (не менее трех) экспертов, хорошо понимающих разные аспекты определения процесса (входы, ресурсы, методы и др.) и способных взглянуть на причины, вызывающие проблемы, с различных точек зрения. Пример причинно-следственной диаграммы представлен на рисунке 6.7.

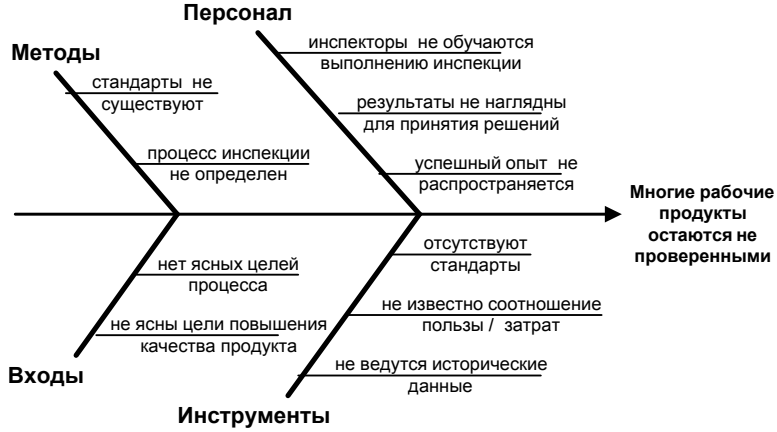


Рис. 6.7. **Пример причинно-следственной диаграммы**

Если в результате анализа собранных данных прослеживается тенденция к снижению эффективности процесса инспекции - должны быть подкорректированы процедуры инспекции. Анализ может показать необходимость изменения объема продукта, планируемого на каждую инспекцию, или объема вопросников, или потребность в дополнительном обучении инспекторов.

Если данные об инспекциях показывают, что меньше всего дефектов было найдено на поздних стадиях ЖЦ (например, в коде), это может свидетельствовать о том, что, либо инспекторы были плохо подготовлены, либо инспекции на ранних стадиях ЖЦ были очень хороши. В последнем случае, может быть принято решение об инспекции только критических участков кода.

Данные об инспекциях могут использоваться и для улучшения процесса разработки. Например, если данные инспекции кода свидетельствуют о том, что большой процент дефектов в коде обусловлен дефектами, внесенными при проектировании, - должны быть приняты меры по улучшению процесса проектирования и инспекций его рабочих продуктов. Способность определить, на какой стадии ЖЦ была допущена ошибка (и внесен дефект, послуживший первопричиной последующих дефектов), зависит от точности сбора данных. Любая попытка изменить применяемые в организации процессы ЖЦ должна предприниматься с большой осторожностью и с учетом разнообразных источников данных о процессе разработки. Процесс улучшения должен быть непрерывным, то есть данные о тенденциях должны быть актуальными и регулярно изучаться, быть доступными всем участникам процесса, группе качества и группе программной инженерии.

6.4. Требования к рабочим продуктам, предъявляемым для проверки

Для того чтобы проверки были эффективными, предъявляемые рабочие продукты должны отвечать определенным требованиям.

При проведении проверок проверяющие лица оперируют термином *проверяемый материал* (ПМ) применительно к рабочим продуктам любого вида.

Единицей функциональности ПМ может быть одно предложение, абзац или раздел, в котором содержится изложение законченной мысли. Например, для документа описания требований – это формулировка одного определенного функционального требования (функции) или нефункционального требования (нефункциональной характеристики) системы.

Проверяемый материал можно рассматривать как состоящий из *компонентов* – элементарных частей рабочего продукта, к которым можно применить *независимые операции* - модифицировать, добавить или удалить, не причиняя вреда другим компонентам.

К структуре ПМ предъявляются следующие требования (с которыми могут быть ассоциированы показатели качества ПМ) [22]:

Корректность. Каждое утверждение ПМ должно быть правильным.

Согласованность. Каждый ПМ должен содержать согласованные между собой утверждения.

Полнота. Каждый ПМ должен содержать информацию, необходимую для достижения его целей (целей, для которых ПМ разрабатывался). Каждый ПМ должен отвечать требованиям к содержанию, отраженным во всех применяемых к нему стандартах.

Ясность. Смысл и назначение каждого утверждения ПМ должны быть ясны. Предполагает:

определенность. Каждое утверждение ПМ должно быть четко идентифицированным. Если это пример – должно использоваться ключевое слово «например» и т.п.;

непротиворечивость. Каждое утверждение ПМ должно иметь только один смысл (значение).

Требования к определениям и обозначениям в тексте ПМ:

– *полнота*. ПМ должен включать определения для каждого технического термина и аббревиатуры;

– *первое использование*. Определение термина должно либо предварять его использование, либо следовать непосредственно за первым использованием. Использование термина во введении должно сопровождаться ссылкой на его определение;

– *идентифицируемость*. Первое использование термина, требующего своего определения, должно четко идентифицироваться (например, выделяться типом шрифта в тексте);

– *простота поиска.* Читатель ПМ должен иметь возможность легко найти определение каждого термина или аббревиатуры в единственном месте (глоссарии или списке сокращений);

– *согласованность*. ПМ должен использовать те же определения терминов и аббревиатур, что были введены в документах более высокого уровня;

– *обозначение.* Каждый компонент должен иметь уникальное обозначение. Для текстовых документов это может быть числовое обозначение (номера разделов, подразделов и т.п.) или текстовый мнемонический маркер (для облегчения контекстного поиска).

Модифицируемость. Предполагает соблюдение требований относительно:

– *уникальности размещения*. Каждое утверждение должно размещаться только в одном месте и только в одном компоненте. Повторения допустимы, если того требуют цели ПМ (например, повторения в учебных материалах, инструкциях и др.);

– *обозначения ссылок*. Ссылки на утверждение в ПМ должны включать обозначение компонента, в котором это утверждение появилось;

– *предметных указателей*. ПМ должен быть структурирован по тематическим разделам. Каждый компонент должен касаться одного аспекта проблемы. Компоненты, касающиеся одного и того же аспекта, должны группироваться.

Требования к оформлению ссылок на документы. Каждая ссылка на соответствующий документ должна быть четко идентифицирована. За исключением маленьких ПМ, в них необходимо включать раздел библиографии.

Требования к языку. Грамматические правила языка изложения должны соблюдаться.

Требования к идентификации. Эти требования касаются:

– *версии*. ПМ должен иметь уникальный идентификатор версии, отличающий данный ПМ от его предшественников;

– *автора*. ПМ должен иметь идентификацию авторов, которые несут ответственность за его разработку.

Требования к минимизации ПМ. Эти требования касаются:

– *неизбыточности*. ПМ не должен содержать утверждений, которые не являются необходимыми для достижения целей его использования;

– *простоты*. *ПМ не должен содержать утверждения, трудные для понима*ния и осложняющие поиск их смыслового назначения в ПМ.

Литература к главе 6

1. *ISO/IEC 12207:1995* / Amd.2:2004 Information technology – Software life cycle processes.
2. *Васютович В.*, Самотохин С., Никифоров Г. Регламентация жизненного цикла программных средств // Открытые системы, № 7-8. – 2000. – www.osp.ru/cio/2000/07-08/042.htm.
3. *ISO/IEC 15271:1998*. Information Technologies - Guide for ISO/IEC 12207 Software Life Cycle Processes.
4. *IEEE Std. 1012:1998*. IEEE Standard for Software Verification and Validation.
5. *IEEE Std. 1016:1998*. IEEE Recommended Practice for Software Design Descriptions.
6. *IEEE Std. 1063:1987*. IEEE Standard for Software User Documentation.
7. *ISO/IEC 9127: 1998*. User documentation and cover information for software packages.
8. *ISO/IEC 6592:2000*. Information Technology. Guidelines for the documentation of computer-based application systems.
9. *IEEE Std. 1044:1993*. IEEE Standard Classification for Software Anomalies.
10. *IEEE Std. 1044.1:1995*. IEEE Guide to Classification for Software Anomalies.
11. *Software* error analysis (на сайте hissa.nist.gov/SWERROR/).
12. *Reference* Information for the Software Verification and Validation Process (на сайте http://hissa.nist.gov/HHRFdata/Artifacts/ITLdoc/234/val-proc.html).
13. *IEEE Std. 1028:1997*. IEEE Standard for Software Reviews.
14. *Using* Formal Inspections in Software Quality Assurance http://www.cis.ksu.edu/~hankley/d841/Fa99/Chap3.html
15. *Fagan M. E.* Design and Code Inspections to Reduce Errors in Program Development // IBM Systems Journal.- v.15.- P. 182-211.
16. *Ackerman A.F.*, Buchwald L.S., Lewski F.H. Software inspections: An effective verification process // IEEE Software.- 1989.- V.6.-May.- P. 31-36.
17. *Модель* оценки технологической зрелости организаций-разработчиков ПО / Ф.И. Андон, В.Ю. Суслов, Т.М. Коротун, Г.И. Коваль, О.А. Слабоспицкая // Проблемы программирования. - 1998. - № 4. - С. 46 – 57.
18. *Emam K. El*, Laitenberger O. Evaluation capture-recapture models with two inspectors // IEEE Trans. On Softw. Eng. – 2001. – V.27. – N.9. – P. 851-864.
19. *Frankovich J.* The Software Inspection Process // Advanced Information Services.- http://sern.ucalgary.ca/courses/seng/621/W97/johnf/inspections.htm.
20. *Lee E.* Software Inspections: How to Diagnose Problems and Improve the Odds of Organizational Acceptance //CrossTalk. – aug. 1997.
21. *Strauss S.*, Ebenau R. Software Inspection Process // New York: McGraw-Hill, Inc. – 1993. – 363 p.
22. *Wheeler D. A.*, Brykczynski B., Meeson R. N. Software Inspection: An Industry Best Practice // IEEE Computer Society Press.- Los Alamitos, CA.-1996.- 325 p.

Глава 7. ТЕСТИРОВАНИЕ ПРОГРАММНЫХ СИСТЕМ

7.1. Основные понятия в области тестирования

Тестирование – неотъемлемая составляющая процесса программной инженерии, один из методов дальнейшего улучшения качества разработанного программного обеспечения системы посредством выявления оставшихся дефектов, не обнаруженных ранее всеми другими видами проверок.

Термин testing (тестирование) широко используется в литературе, но определяется по-разному.

Стандарт ANSI/IEEE Std. 610.12:1990 определяет термин *testing* в самом его широком смысле как *любое действие* по анализу ПО (включая методы как динамической, так и статической проверки) [1].

Г. Майерс определяет этот термин в самом узком смысле: «тестирование – процесс *выполнения* программы (или ее части) *с целью обнаружения* ошибок… Отладка (debugging) – диагностирование точной природы известных ошибок и их исправление» [2].

Слово testing может быть переведено не только как тестирование, но и как *испытания* (как это сделано в стандартах ДСТУ 2844-94 [3], ДСТУ 2853-94 [4]). Однако, понятие «испытания» нам привычнее связывать с завершающими стадиями ЖЦ, на которых выполняется не поиск ошибок, а подтверждение пригодности разработанного программного продукта.

В данной главе тестирование рассматривается как процесс *выполнения* программной системы (или элементов ПС) с целью *проверки ее соответствия установленным требованиям* и *обнаружения дефектов.*

По мере развития программной инженерии понимание целей и задач тестирования менялось. В работе [5] выделены следующие «исторические» периоды, отражающие прогресс в понимании целей тестирования и его места в жизненном цикле ПС:

1) период до 1956 года - ориентация на отладку;

2) период с 1957 по 1978 гг. - ориентация на установление соответствия ПС исходным требованиям;

3) период с 1979 по 1982 гг. – ориентация на обнаружение дефектов, оставшихся после реализации;

4) период с 1983 по 1987 гг. – ориентация на анализ, проверку и тестирование с целью оценки качества ПС на всех стадиях разработки;

5) период с 1988 по 1995 гг. – интеграция действий по проверке и тестированию в жизненный цикл ПС с целью предотвращения появления дефектов на *всех* стадиях разработки (с охватом всех действий по верификации, валидации и тестированию).

С появлением в 1995 году стандарта ISO/IEC 12207 [6], в котором все действия, связанные с созданием ПС, представлены в виде отдельных процессов ЖЦ, произошло разделение задач верификации, валидации и тестирования по отдельным *процессам*. Тестирование отнесено к *основным процессам*, но представлено не одним, а группой процессов. Кроме того, ряд задач тестирования решаются в рамках других процессов разработки, в частности, задачи планирования тестирования распределены по процессам: «Проектирование архитектуры системы», «Анализ требований к ПО», «Проектирование ПО». Автономное и интеграционное тестиро-

вание ПО выполняются в рамках процессов «Построение ПО» и «Интеграция ПО», поскольку неразрывно с ними связаны.

Таким образом, произошла *интеграция* тестирования с процессами разработки, и сегодня тестирование рассматривается как непрерывная деятельность, выполняемая на протяжении *всего процесса разработки*. Планирование тестирования должно начинаться на стадии анализа требований, а планы и процедуры тестирования должны систематически и постоянно уточняться по мере разработки системы.

В SWEBOK 2004 года [7] область знаний «Тестирование ПО» представлена пятью основными разделами (рисунок 7.1). Весь последующий материал главы структурирован таким образом, чтобы учесть рекомендации SWEBOK по их смысловому наполнению.

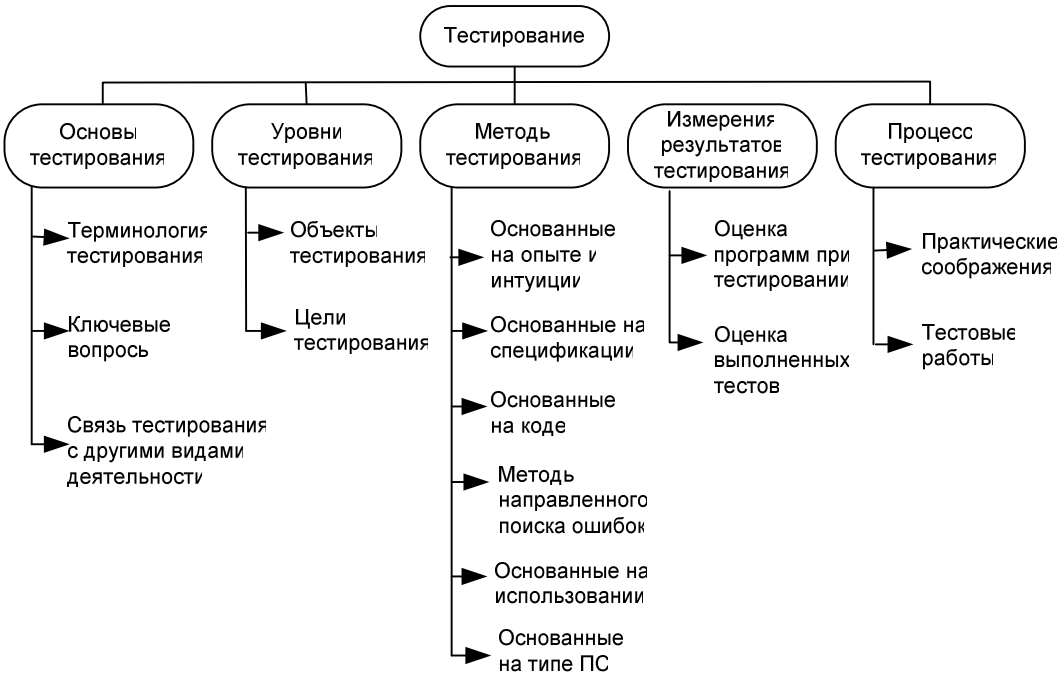


Рис. 7.1. **Область знаний «Тестирование ПО» в SWEBOK-2004**

Терминология тестирования. Тестирование заключается в *динамической* проверке поведения программы на *конечном* множестве тестовых данных, специальным образом *выбранных* из бесконечного входного пространства, на соответствие установленному *ожидаемому* поведению.

Выделенные курсивом слова ключевые для тестирования:

динамическое: тестирование всегда предполагает *выполнение* программы;

конечное: даже для небольшой программы теоретически возможно создать такое количество тестов, для выполнения которых могут понадобиться годы [2, 8]. Неполнота - одна из основных проблем тестирования, поскольку на практике полное множество тестов можно рассматривать как бесконечное. Количество же тестов, которые могут быть выполнены в ограниченные сроки, конечно. Таким образом, тестирование всегда предполагает некий «компромисс» между ограниченными сроками и потенциально неограниченным количеством тестов. Это приводит к известным проблемам тестирования, таким как принятие решений об адекватности

тестирования, и проблемам управления – оценке затрат (стоимости, времени, персонала) на тестирование;

выбранных: с проблемой адекватности тестирования связана проблема выбора ограниченного множества тестов. Методы тестирования, в основном, отличаются подходами к выбору множества тестовых данных из входного пространства;

ожидаемое поведение: нужно уметь определять, правильны ли полученные результаты выполнения программы, соответствует ли наблюдаемое выполнение ожиданиям пользователя или спецификациям. В литературе по тестированию это называется *проблемой оракула* [2] (эталона), для решения которой могут применяться разные подходы (оценка результатов, сравнение с существующим эталоном, согласование с пользователем трактовок понятий «дефект» и «отказ»).

Хотя основная цель тестирования – обнаружить дефекты в программной системе и установить ее функциональную пригодность, необходимо рассматривать и другие его цели - тестирование удобства применения, производительности и прочие. Отсюда следует два основных подхода к выполнению тестирования – деструктивный (отрицательное, разрушительное тестирование) и конструктивный (положительное или демонстрационное).

При деструктивном подходе тесты выбираются с целью обнаружения дефектов, и эффективным считается тест с высокой вероятностью их обнаружения.

При конструктивном подходе тесты выбираются для демонстрации соответствия характеристик ПС установленным требованиям пользователя или целям качества.

Тестирование программной системы на протяжении процесса разработки выполняется на нескольких уровнях. Для каждого уровня тестирования определяется категория объектов тестирования (вся система, программные компоненты, модули) и набор проверяемых целей (тестируемых характеристик).

Цели и объекты совместно определяют критерий выбора множества тестов, как относительно его полноты - «какой объем тестирования достаточен для достижения установленной цели?», так и его состава - «какие тесты должны быть выбраны для достижения установленной цели?». Первый вопрос касается выбора критерия покрытия, а второй – критерия выбора типов тестов.

На каждом уровне тестирование повторяется многократно, образуя циклы тестирование-исправление-повторное тестирование.

В современной программной инженерии все виды действий, связанные с тестированием, начиная с планирования до оценки результатов, должны интегрироваться в четко определенный, документируемый и контролируемый *процесс тестирования*. Документирование процесса тестирования облегчает взаимодействие между разработчиками, группой тестирования и руководством проекта, а также позволяет сделать процесс видимым, повторяемым и измеряемым.

К ***ключевым вопросам тестирования*** в SWEBOK отнесены следующие:

1. Критерии выбора тестов/Критерии адекватности тестов (или правила прекращения тестирования). Эти критерии могут применяться как для создания набора тестов, так и для проверки, насколько выбранные тесты адекватны решаемым задачам (тестирования), а также помогают определить, когда можно или необходимо прекратить тестирование.

2. Эффективность тестирования/Цели тестирования. Тестирование – это наблюдение за поведением программы, выполняемой в целях тестирования с за-

данными параметрами, по заданному сценарию или с другими заданными начальными условиями или целями тестирования. Эффективность теста может быть определена только в контексте заданных условий [9].

3. Тестирование для выявления дефектов [8]. В тестировании для выявления дефектов применяется деструктивный подход, а успешным считается тест, обнаруживающий дефект. Этот подход принципиально отличается от другого подхода, когда тесты запускаются для демонстрации того, что программа удовлетворяет предъявляемым к ней требованиями и, соответственно, тест считается успешным, если не найдено дефектов.

4. *Проблема оракула*. «Оракул» в тестировании – это любой агент (человек или программа), оценивающий поведение программы и формирующий заключение о результате теста (тест пройден или нет). Это заключение существенно зависит от трактовки понятия «отказ» и «дефект» в конкретном контексте.

5. Теоретические и практические ограничения тестирования. Ограничения обусловлены невозможность исчерпывающего тестирования и его экономической нецелесообразности для конкретных ПС. Тестирование должно проводиться с учетом риска отказа ПС и основываться на таких стратегиях тестирования, получивших распространение в современных методологиях разработки, как тестирование, основанное на риске (risk-based testing), и управляемое риском тестирование (risk-driven testing), которые кратко рассмотрены в этой главе.

6. Проблема неосуществимых путей. Неосуществимые пути – это пути потока управления программы, которые не могут быть выполнены ни при каких входных параметрах. Это сложнейшая проблема путевого тестирования и особенно его автоматизации [9]. Основные методы путевого тестирования рассмотрены в разделе 7.3.

7. Тестопригодность [10]. Этот термин имеет два разных значения. Первое – это степень легкости описания критериев тестового покрытия для ПС, второе – вероятность, что выполнение программы при тестировании приведет к ее отказу, при наличии дефекта.

Связь тестирования с другими видами деятельности. Тестирование имеет много общего с процессами верификации и валидации (V&V), рассмотренными в главе 6. Общность процесса тестирования с процессами V&V заключается в единстве состава и структуры планов, рекомендуемых IEEE Std. 829 [11], а также объектов и применяемых методов. Отличие этих процессов состоит в условиях их применения. Тестирование – основной процесс в ЖЦ, выполняемый *всегда*, для всех объектов ПО системы независимо от ее критичности. Процессы V&V, в современной трактовке стандартов IEEE Std. 1012 [12] и ISO/IEC 12207 [6] – поддерживающие процессы, которые могут применяться к *выбранным объектам* тестирования для проверки планов их тестирования и подтверждения того, что выполненное тестирование адекватно уровню критичности ПС. По отношению к процессу тестирования V&V выполняет контрольную функцию и подтверждает соответствие объектов установленным требованиям.

Тестирование ПС тесно связано с отладкой и собственно программированием, но охватывает гораздо более широкий круг проблем и участников – программистов, тестировщиков, аналитиков и инженеров по качеству.

7.2. Виды и уровни тестирования

Структурирование процесса тестирования ПС по уровням обычно связывается либо с *видами тестируемых объектов* (модуль, система или ее части), либо с проверяемыми *целями качества* ПС (функциональность, безопасность, надежность и др.) на разных стадиях ее создания и эксплуатации.

7.2.1. Уровни тестирования по видам объектов

Традиционно выделяется три уровня тестирования ПО: автономное[1] или модульное (unit testing), интеграционное[2] (integrating testing) и системное (system testing). В стандарте ISO/IEC 12207 прослеживаются *четыре* уровня тестирования:

- *модульное* (в процессе «Построение (Конструирование) ПО»),
- *интеграционное* (в процессе «Интеграция ПО»),
- *тестирование ПО* (как процесс) и
- *системное* (в процессе «Системная интеграция»).

Модульное тестирование предполагает проверку функционирования объектов в изоляции друг от друга. Оно обычно выполняется разработчиками с доступом к коду и чередуется с отладкой. Объектами тестирования являются отдельные процедуры (методы и классы при объектном подходе), программные модули и программные компоненты, состоящие из тесно связанных модулей.

Цели модульного тестирования – обнаружение дефектов в реализации функций объектов и подтверждение соответствия объекта спецификациям проекта (техническому проекту).

Наиболее полно вопросы систематического модульного тестирования освещены в стандарте IEEE 1008-87 “Standard for Software Unit Testing” [14].

Интеграционное тестирование предназначено для проверки правильности взаимодействия между программными объектами, протестированными автономно.

Современные систематические стратегии интеграции определяются архитектурой ПС и моделью разработки (обычно итерационной с приращениями).

Интеграционное тестирование выполняется при каждой сборке новой версии ПС с целью выявления дефектов в *интерфейсах* между интегрируемыми компонентами и подтверждения их соответствия проекту архитектуры ПС.

Тестирование программного обеспечения заключается в проверке функционирования интегрированной версии ПО системы в моделируемой среде. Цели тестирования на этом уровне – выявление дефектов в реализации *внешних функций* ПО и подтверждение его соответствия спецификации функциональных требований.

Выделение уровня ***системного тестирования*** обусловлено необходимостью обеспечения и оценки качества сопряжения ПО с другими, в том числе, не программными компонентами современных систем.

На этом уровне тестирования производится проверка соответствия ПС *целям качества*, установленным в требованиях к системе (с акцентом на нефункциональные требования), таким как надежность, устойчивость, производительность, переносимость и др., а также *внешним интерфейсам* с другими системами, средой, аппаратным обеспечением. Большая часть функциональных отказов и дефектов

---

[1] В ДСТУ 2941термин unit testing переведен как «блочное тестирование» [13].

[2] В ДСТУ 2941термин integration testing переведен как «комплексные испытания»

должна быть идентифицирована и устранена на предыдущих уровнях тестирования. Цели тестирования на данном уровне– выявление дефектов, связанных с неудовлетворительными техническими характеристиками функционирования системы, и подтверждение ее соответствия проекту архитектуры системы.

Взаимосвязь уровней и целей тестирования можно представить в виде модифицированной каскадной модели ЖЦ (так называемой ***V-образной модели***) [15], отражающей процессы декомпозиции и интеграции ПС и соответствующие уровни тестирования (рисунок 7.2).

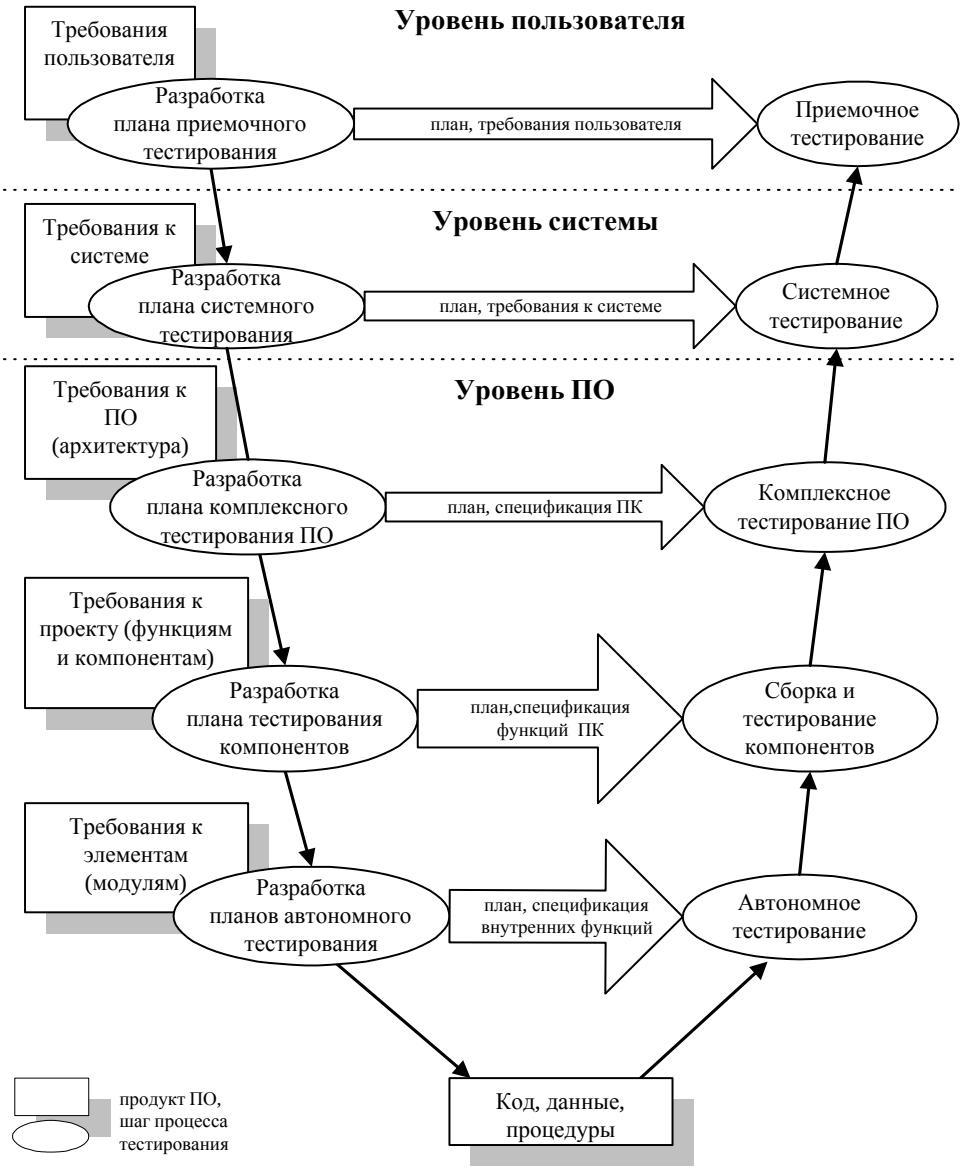


Рис. 7.2. **V-образная модель тестирования**

В этой модели тестирование рассматривается как непрерывный процесс, интегрированный в процесс разработки ПС и включающий два взаимосвязанных под-

процесса – *планирование* тестирования в рамках процессов разработки системы (левая ветвь на рисунке) и *проведение* тестирования соответствующих объектов (правая ветвь).

Хотя на рисунке для наглядности уровни тестирования отображены в привязке к традиционной каскадной модели разработки, - на практике в зависимости от архитектуры системы, а также принятой модели разработки, эти уровни могут объединяться, а разные задачи тестирования - выполняться параллельно.

На практике задачи *тестирования ПО* и *системного тестирования* часто объединяются в единый процесс, но для сложных ПС тестирование технических характеристик должно выполняться отдельно.

7.2.2. Виды испытаний программной системы

Помимо рассмотренных выше *уровней* тестирования объектов ПС, существуют *виды тестирования ПС в целом*, выполняемые на заключительных стадиях ее разработки и в ходе эксплуатации.

К ним, прежде всего, относятся *испытания ПС*. В ДСТУ 2853-94 определены следующие *виды испытаний* - *предварительные*, *приемочные*, *установочные* и *эксплуатационные*.

Предварительные испытания программной системы можно считать наивысшим уровнем ее «формального» *тестирования*, выполняе*мого в среде разработки* с целью обнаружения возможных оставшихся дефектов и оценки достигнутого уровня качества ПС. Здесь «формальный» означает, что тестирование выполняется строго в соответствии с установленными документированными процедурами и с участием представителей заказчика. Испытания осуществляются в два приема – тестирование ПО и тестирование системы. Следовательно, цель предварительных испытаний - обнаружение несоответствия ПС внешним спецификациям функциональных и технических характеристик.

Приемочные и все последующие испытания непосредственно не связываются с понятием тестирования, ибо их цель - *подтвердить соответствие* (или несоответствие) системы исходным требованиям пользователя. В соответствии со стандартом ISO/IEC 12207 эти испытания выполняются в рамках разных процессов ЖЦ.

Цель ***приемочных испытаний*** (приемочного[3], квалификационного тестирования) – определение степени соответствия разработанной ПС исходным требованиям заказчика (пользователя). Эти испытания проводятся исключительно в контексте требований заказчика (пользователя) и с его непосредственным участием и, как правило, в среде эксплуатации. В трактовке стандарта ISO/IEC 12207 приемочное тестирование, как вид тестирования, не включено в «Процесс тестирования», а выполняется в рамках процессов «Поставка» и «Приемка заказчиком» (потребителем) и связывается с процессом «Валидации».

Тестирование инсталляции (установочные испытания) выполняется в *среде эксплуатации* (в рамках процесса «Инсталляция ПС») и относится к уровню системного тестирования. Включает тестирование процедур инсталляции ПС с внешних носителей.

---

[3] Иногда под приемочным тестированием (acceptance test) понимают предварительное тестирование объекта тестирования при его передаче от разработчика группе тестирования, выполняемое с целью определения пригодности объекта для тестирования [8].

Перед окончательным выпуском системы проводится ***Альфа и Бета тестирование.*** Для этого система передается небольшой группе пользователей - внутренних (Альфа) или внешних (Бета), которые эксплуатируют систему и информируют разработчика о выявленных проблемах. Обнаруженные отказы и дефекты свидетельствуют о качестве выполненного ранее тестирования.

Альфа и Бета тестирование подобны *эксплуатационным испытаниям*, выполняемым на *этапе опытной эксплуатации системы*. Но, в отличие от них, обычно проводятся для систем, разрабатываемых для широкого круга пользователей, а не для конкретного заказчика.

Эти виды тестирования не регламентируются планом тестирования и непосредственно не входят в процесс тестирования.

Регрессионное (regression test)* и *повторное (re-test) тестирование. Эти понятия в литературе часто смешиваются, поскольку оба вида тестирования касаются повторного выполнения тестов. Однако они имеют некоторые отличия. Повторное тестирование выполняется на любом уровне тестирования для проверки внесенных изменений, и не регламентируется планом тестирования. Регрессионное тестирование связано с развитием ПС и особенно широко применяется в итерационных моделях разработки (для тестирования новых версий ПС). Оно заключается в *повторении* подмножества ранее выполненных тестов (для того, чтобы убедиться в том, что то, что ранее работало, еще работает), а также *разработке новых тестов* для проверки правильности внесенных изменений. В отличие от повторного тестирования, регрессионное тестирование планируется. При его планировании решается проблема отбора минимального набора регрессионных тестов [8].

7.2.3. Виды тестирования характеристик программной системы

По отношению к проверяемым *характеристикам* и целям качества любого объекта системы можно выделить следующие виды тестирования.

Функциональное тестирование (называемое также тестированием *на соответствие* или тестирование *корректности)* направлено на проверку соответствия наблюдаемого поведения системы (или отдельных элементов) спецификациям (внешним и внутренним). Выполняется, как правило, в среде разработки по планам, разработанным на основе спецификаций функциональных требований.

Цель функционального тестирования состоит в *обнаружении несоответствий* между реальным поведением реализованных функций и исходными требованиями, представленными в спецификации функций, и *определении полноты* функционального покрытия относительно спецификации функций. Выполняется на уровнях модульного тестирования и тестирования ПО в целом. Функциональные тесты могут быть получены по внешним спецификациям функций, проектной информации или прототипу ПО.

Тестирование безопасности (Security testing) заключается в проверке возможности несанкционированного доступа к ПС. Выполняется путем моделирования возможных атак внешних пользователей (в том числе других ПС) с целью «взломать» систему защиты.

Тестирование удобства применения (эргономичности) предназначено для оценивания простоты применения и изучения системы конечными пользователями, эффективности ее использования для решения задач пользователя и способности к восстановлению при ошибках пользователя. Часто включает также тестирование

документации (on-line, справочной службы, подсистемы обучения).

Этот вид тестирования требует знания современных стандартов по эргономике, например, ISO 13407:1999 [16], а также согласованных с заказчиком требований к пользовательскому интерфейсу. Применяется в рамках процесса анализа требований (тестирование прототипа), тестирования программных компонентов, реализующих интерфейс, тестирования ПО и приемочных испытаний системы.

Тестирование технических характеристик. Это процесс поиска несоответствий программной системы *целям качества* (целевым требованиям), зафиксированным во внешних спецификациях наряду с описанием ее функций.

Исходной посылкой для его выполнения является измеримость и принципиальная достижимость установленных целевых требований (к надежности, производительности, совместимости, конфигурации и др.). Эти виды тестирования получили названия, соответствующие целям тестирования, и обычно проводятся в рамках тестирования системы в специально моделируемой среде, максимально приближенной к среде эксплуатации, или в среде эксплуатации.

Тестирование на надежность. Этот вид тестирование рассматривают как основное средство улучшения надежности, поскольку с ростом количества выявленных и устраненных дефектов надежность ПС растет. Тестирование на надежность выполняется на наборах данных, выбранных случайным образом из входного пространства в соответствии с *операционным профилем* [17, 18]. Все отказы при выполнении тестов регистрируются применительно к интервалам времени между отказами (или моментам времени наступления отказов от начала тестирования). Процесс возникновения отказов рассматривается как случайный (стохастический) процесс (Марковский или Пуассоновский), а для количественной оценки достигнутой надежности применяются модели роста надежности.

Тестирование производительности *(Performance testing)*. Выполняется для проверки достижения установленных требований к производительности или для оптимизации производительности. Иногда подразделяется на следующие виды:

- *нагрузочное тестирование* (*load testing*) - проверка работоспособности ПС при ожидаемых нормальных загрузках;

- *тестирование на устойчивость (stress testing)* - проверка работоспособности в нестандартных условиях (при чрезмерных и отсутствующих загрузках);

- *тестирование объема (volume testing)* - проверка внутрипрограммных или системных ограничений, связанных, например, с большими массивами данных, предельными объемами БД и другими характеристиками объема.

Тестирование конфигурации (Configuration testing). Выполняется для проверки работоспособности ПС в разных конфигурациях (операционных системах).

Сравнительное тестирование (Back-to-back testing). Вид тестирования, при котором один и тот же набор тестов выполняется на двух реализованных версиях, а результаты сравниваются. Особенно актуален в итерационных моделях разработки.

Тестирование восстановления (Recovery testing). Проводится для проверки процедур восстановления системы после отказов.

Управляемая тестами разработка (Test-driven development) [19]. Подход к разработке, согласно которому тесты модулей создаются до начала их кодирования. Тесты исполняют роль некого прототипа модуля и заменителя спецификации требований. Подход применяется в экстремальном программировании (XP) (глава 11).

7.3. Методы тестирования

7.3.1. Классификация и краткий обзор методов тестирования

Методы тестирования различаются *подходами к проектированию тестов*.

Традиционно методы тестирования разделяют на две категории - «черный ящик» (без доступа к исходному коду) и «белый» ящик (с доступом к исходному коду) [20, 21].

Более подробная классификация методов тестирования, основанная на подходах к проектированию тестов, представлена на рисунке 7.3 и кратко описана ниже.

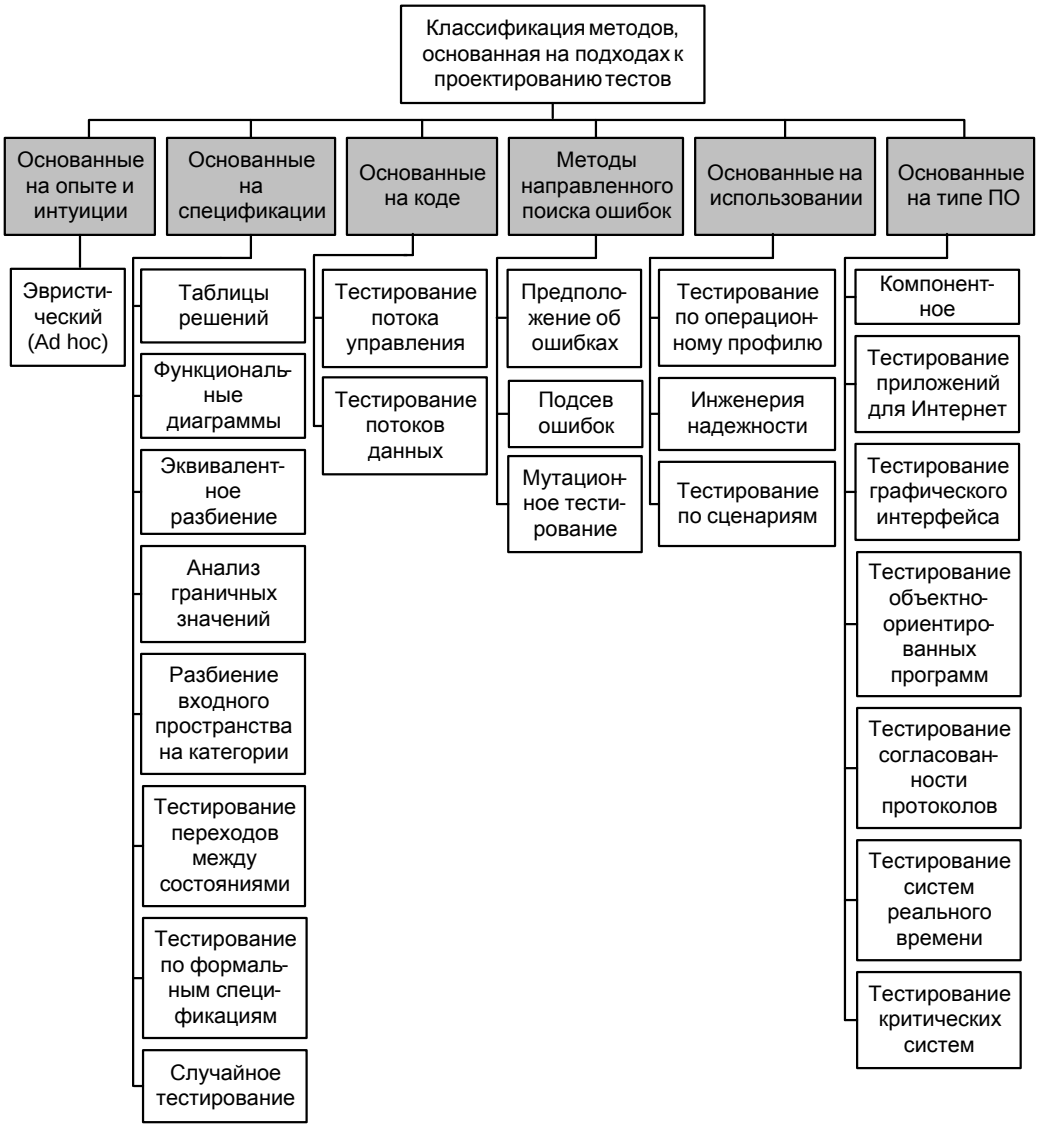


Рис. 7.3. **Классификация методов тестирования**

1. Методы тестирования, основанные на опыте и интуиции.

Специализированное тестирование (Ad hoc). Это наиболее распространенный (хотя и не считающийся систематическим) подход, при котором тесты разрабатываются исходя из интуиции тестировщика и его опыта в тестировании подобных систем. Эффективность подхода полностью определяется мастерством тестировщика. Подход не требует подробной спецификации функций ПО, но и не обеспечивает оценивания полноты тестового покрытия. Развитием подхода можно считать «исследовательское тестирование» (exploratory testing), основные принципы которого – совмещение изучения программного продукта с проектированием тестов и их выполнением. Такое тестирование обычно осуществляется независимыми тестировщиками применительно к завершенным программным продуктам. Пример применения подхода рассмотрен в 7.3.2.

2. Методы, основанные на спецификации (или методы функционального тестирования).

Эти методы широко известны и подробно описаны в литературе [2, 8, 9, 10, 22–29]. В традиционной классификации их относят к методам «черного ящика». Они применяются для тестирования внешних и внутренних функций программы и предполагают наличие спецификации (формальной или не формальной), используемой в качестве эталона. Отличаются между собой подходами к выбору тестовых данных из множества входов (входного пространства) функций.

К основным методам функционального тестирования относятся:

- таблицы решений;
- функциональные диаграммы;
- эквивалентное разбиение;
- анализ граничных значений;
- разбиение входного пространства на категории;
- тестирование переходов между состояниями;
- тестирование по формальным спецификациям.

Метод, использующий ***таблицы решений*** для проектирования тестов, был предложен Дж.Гуденафом и С.Герхартом [22]. Каждая колонка такой таблицы представляет комбинацию условий, которые могут существенно повлиять на выполнение программы. Эти условия идентифицируются на основе анализа спецификаций. Затем определяется множество их возможных значений, и устанавливаются ограничения относительно их совместимости. Таким образом, сокращается количество тестовых предикатов (например, ограничение может говорить о том, что, если условие 1 выполняется, то ни условие 2, ни условие 3 не могут выполняться).

Второй метод, предложенный Б.Эльмендорфом и описанный Г.Майерсом, заключается в преобразовании спецификации в ***функциональные диаграммы*** [21]. По этому методу вначале идентифицируется каждая отдельная функция, затем определяются все причины, существенно влияющие на ее поведение, и все ответные реакции (следствия). Следующий шаг состоит в построении графа (диаграммы), связывающего комбинации причин с ожидаемыми реакциями на них. Далее для каждого следствия, указанного на графе, определяются тестовые наборы путем перебора всех комбинаций причин, порождающих это следствие.

Хотя строгое соблюдение метода может обеспечить построение эффективных тестов, он сложен в практическом применении для функционально сложных про-

грамм, поскольку с ростом числа причин усложняется граф причинно-следственных связей, а установление ограничений сопряжено с добавлением новых промежуточных узлов.

В методе ***эквивалентного разбиения*** [8, 23] множество значений входных данных функции, образующих ее входное пространство, разбивается на набор подмножеств таким образом, что в каждое подмножество попадают значения, эквивалентные друг другу с точки зрения их использования в тестах для обнаружения ошибок. Говорят, что все тесты, которые могут быть построены на основе эквивалентных значений, представляют один «класс эквивалентности», и для тестирования достаточно выбрать только самые эффективные из них. Более подробно метод описан в п.7.3.3.

В методе ***анализа граничных значений***, дополняющем предыдущий метод, данные выбираются на границах входной области, поскольку многие отказы происходят из-за дефектов, связанных с обработкой предельных значений входов [23]. Подробнее метод описан в п.7.3.4.

Простое и ценное расширение этого метода – тестирование *устойчивости*, когда тестовые данные выбираются также и *вне области* для тестирования отказоустойчивости программы к недопустимым входам.

Методы эквивалентного разбиения и анализа граничных значений считаются *базовыми* методами функционального тестирования и должны применяться совместно при проектировании набора тестов для каждого уровня тестирования.

В развитие этих методов был создан метод, основанный на спецификациях функций, и использующий для построения функциональных тестов ***разбиение входного пространства*** функций на определенные категории (category partition method или CP-метод) [24]. Суть метода заключается в ряде последовательных декомпозиций функции, начиная с исходной функциональной спецификации, и кончая отдельными деталями каждой тестируемой процедуры, и создании спецификаций тестов на основе выделения категорий информации о параметрах функции и условий ее выполнения. Подробнее метод описан в п.7.3.5.

В методе тестирования ***переходов между состояниями*** программа (реализующая функцию) представляется в виде модели, отражающей все возможные состояния ее выполнения, переходы между этими состояниями, события, вызывающие переходы, и дальнейшие действия по обработке данных, определяемые этими переходами.

Описание спецификаций на формальном языке (имеющем точно определенный синтаксис и семантику) позволяет автоматически строить по ним функциональные тесты и в то же время обеспечивает эталон для проверки результатов. Существует целый ряд методов ***генерации тестов по формальным спецификациям*** [25-28].

Все упомянутые методы функционального тестирования относятся к «систематическим» методам в отличие от случайных (стохастических) и статистических методов.

При ***случайном*** тестировании входные данные для тестов выбираются случайно. В качестве инструмента для генерации входных данных могут применяться датчики случайных чисел. Для интерактивных программ тестировщик использует случайную комбинацию действий с программой, пытаясь выявить области ее неустойчивого функционирования.

3. Методы, основанные на анализе кода (или структурные).

В традиционной классификации структурные методы относят к методам «белого ящика». Согласно этим методам структура программы представляется в виде направленного графа, в котором идентифицируются потоки управления или потоки данных. Соответственно, методы делятся на две основные категории: тестирование *потока управления* (путей) и тестирование *потока данных*. Методы этой категории наиболее изучены и описаны в литературе [2, 20, 21, 29], поэтому рассматриваются очень кратко.

В методах ***тестирования потока управления*** данные из входного пространства выбираются таким образом, чтобы обеспечить максимальное покрытие кода. Основные методы приведены ниже:

Тестирование строк. Выбираются данные, обеспечивающие выполнения всех строк (операторов) программы. Этот метод дает самый слабый критерий покрытия, называемый «покрытие строк», и приемлем для программ, не содержащих логических условий и циклов.

Тестирование ветвлений. Выбираются данные, обеспечивающие выполнение путей, выделяемых в программе с помощью логических условий, принимающих значения *Истина* и *Ложь*. В приведенном ниже примере для проверки ветвления достаточно двух тестов (при A<B и при A ≥ B):

```
if (A<B) then
 …
else
…
endif
```

Нужно заметить, что для удовлетворения критерия покрытия строк достаточно было бы выполнить один (любой) из этих двух тестов.

Тестирование логических условий. Если ветвления в программе образуются в результате выполнения сложных логических условий, данные для их тестирования должны выбираться таким образом, чтобы проверить все значения логических условий. Например, для полной проверки представленного ниже логического условия:

```
if (A<B and C = 1) then
 …
else
…
endif
```

необходимо уже четыре теста (один - для случая, когда условие выполняется, и три - для остальных случаев):

1) A < B C = 1

2) A < B C ≠ 1

3) A ≥ B C=1

4) A ≥ B C≠1

Этот метод дает наиболее полный критерий покрытия кода программы, который называется критерием «*логические условия*». Опять таки, для удовлетворения критерия покрытия строк было бы достаточно одного теста, а для покрытия ветвлений – двух.

Тестирование циклов. Тесты разрабатываются для проверки каждого цикла при граничных значениях переменных цикла и внутри их границ (для тестирования устойчивости циклы проверяются при значениях вне границ). Этот метод дает критерий покрытия, называемый «*все циклы*».

В *методе* ***тестирования потока данных*** тестовые данные выбираются таким образом, чтобы проследить пути каждой переменной в программе от назначения значений до самого последнего использования (для всех переменных). Этот метод требует большого количества тестов, поэтому на практике трассируются только наиболее критичные значения переменных. Особый интерес, например, представляют значения переменных, участвующих в операциях деления (*необходимо предотвращать возможность обращения делителя в 0!*), условия и циклы, выполнение которых зависит от значений переменных. Метод тестирования потока данных может также применяться для поиска трудно обнаруживаемых ошибок времени выполнения, связанных с использованием памяти.

В заключение краткого обзора структурных методов нужно отметить, что они обычно применяются на уровне модульного тестирования программы ее разработчиком и чередуются с отладкой, хотя могут использоваться и в процессах V&V для проверки критических программ.

Методы функционального и структурного тестирования должны рассматриваться не как альтернативные, а как взаимодополняющие, поскольку используют разные источники информации для построения тестов и предназначены для выявления разных типов ошибок.

4. Методы направленного поиска ошибок.

Эти методы используются для проектирования тестов, специально предназначенных для обнаружения ошибок *определенных типов*.

К основным методам этой категории относятся:

- предположение об ошибках (error guessing);
- подсев ошибок (error seeding) [30];
- мутационное тестирование (mutation testing) [30].

Согласно методу выдвижения ***предположений об ошибках*** на основании «исторической» информации об ошибках, обнаруженных в подобных программах, опыта тестировщика, а также каталогов известных ошибок, составляется список возможных ошибок и ошибочных ситуаций. Примеры возможных типов ошибок представлены в разных публикациях, в частности в [8]. Одним из «классических» примеров применения метода является проверка деления на 0. В традиционной классификации метод относится к категории методов «черного ящика».

В работе [31] рассмотрен подход к тестированию и отладке программ, написанных на языке Java, основанный на анализе «шаблонов» (паттернов) типичных ошибок. Основная его идея – учиться на чужих ошибках, чтобы не допускать собственных.

Методы подсева ошибок и мутационного тестирования предназначены для проверки *тщательности* уже выполненного тестирования. В традиционной классификации они относятся к категории методов «белого ящика».

В методе ***подсева ошибок*** в код, протестированный на определенном наборе тестов, специально вносится небольшое количество ошибок, а затем программа повторно тестируется. Если при тестировании обнаруживаются не все внесенные

ошибки, набор тестов считается не достаточным. Отношение количества обнаруженных внесенных ошибок к общему количеству внесенных ошибок предполагается примерно равным отношению количества найденных реальных ошибок к общему числу ошибок, содержащихся в программе. Если выявлены все внесенные ошибки, то считается, что, либо набор тестов достаточен, либо, что эти ошибки было слишком легко найти.

При ***мутационном тестировании*** создается много копий основной программы, в каждую из которых вносится небольшое изменение, называемое «мутацией», для имитации ошибки в операторе или операнде, не отражающееся на синтаксической корректности программы. Каждая копия тестируется на одном и том же наборе тестов. Если в процессе тестирования все внесенные изменения обнаружены, программа считается протестированной адекватно и корректной (могут быть также обнаружены ранее не выявленные ошибки). Для того чтобы метод был эффективным в обнаружении ошибок, требуется большое количество мутантов, что возможно только при автоматической их генерации.

5. Методы, основанные на анализе ожидаемого использования.

К этой категории в [7] отнесено два основных метода:

- статистическое тестирование;
- тестирование по операционному профилю.

При ***статистическом тестировании*** входные данные для проектирования тестов выбираются из входного пространства в соответствии с частотой их появления в будущих сценариях использования программы. При выполнении тестов фиксируются моменты отказов и вычисляются интервалы между отказами MTBF (Mean Time Between Failure), обычно в единицах процессорного времени (CPU) или времени выполнения. Полученная информация используется для оценки надежности и предсказания момента завершения тестирования [18]. Метод применяется на уровне системного тестирования. Для определения частоты использования функций программы, а также моментов отказов, в код вставляются команды-счетчики.

Тестирование по ***операционному профилю*** применяется в рамках методологии инженерии надежности (SRE, от Software Reliability Engineering) и называется методом SRET (от Software Reliability Engineered Testing). Методология SRE охватывает полный цикл разработки программных систем с повышенными требованиями к надежности, а тестирование SRET (являющееся по сути статистическим) основано на приоритезации входов не только по частоте, но и с учетом критичности функций, режимов и последствий отказов систем при эксплуатации [17, 32].

К категории методов, основанных на анализе ожидаемого использования, можно отнести также метод тестирования по ***сценариям возможного использования***, суть которого заключается в идентификации возможных сценариев работы пользователя и разработке по ним сценариев тестирования (test-cases). Этот метод применяется во всех современных инструментах автоматизации функционального тестирования программ, имеющих интерфейс пользователя (инструменты Capture-Replay). Он также используется в популярной методологии RUP (Rational Unify Process) [33], согласно которой сценарии тестирования формируются на этапе анализа и проектирования по набору «бизнес-сценариев» (use-cases).

Обобщением данного метода можно считать ***тестирование на базе моделей*** (*model-based testing*, *model-driven testing*), применяемое в рамках соответствующего подхода к разработке (*model-driven development*) [34].

Такие известные подходы к тестированию как тестирование, основанное на риске (*risk-driven testing*, *risk-based testing*), представляют не методы разработки тестов, а скорее стратегии направленного тестирования и минимизации набора тестов. Эти стратегии подобны тестированию по операционному профилю, но проводятся методами систематического тестирования и учитывают не только частоту использования, но и величину риска отказа ПС.

6. Методы, учитывающие специфику программной системы.

Рассмотренные выше методы универсальны и применимы к любым типам ПС. Однако они не учитывают характерных особенностей построения систем разного типа, требующих применения специфических *подходов* к тестированию.

Руководствуясь рекомендациями SWEBOK 2004 года, мы далее кратко охарактеризуем следующие методы:

- тестирование объектно-ориентированных программ;
- компонентное тестирование;
- тестирование Web-приложений;
- тестирование графического интерфейса пользователя;
- тестирование протоколов на соответствие;
- тестирование систем реального времени;
- тестирование критических систем.

Особенности типов ПС обусловливают не столько применение специальных методов проектирования тестов, сколько выбор методов и видов тестирования, наиболее подходящих для определенных *объектов тестирования*.

Хотя каждый из перечисленных выше типов ПС заслуживает тщательного рассмотрения, мы ограничимся краткой характеристикой их основных особенностей, а в 7.3.8 подробнее остановимся на тестировании Web-приложений.

Тестирование объектно-ориентированных программ (ООП). Тестированию ООП посвящено достаточно литературы, например [31, 33, 35, 36].

Специфика тестирования ООП связана, в первую очередь, с:

- итерационным инкрементным подходом к разработке, что приводит к усложнению интеграционного и регрессионного тестирования;
- событийно-управляемой природой ОПП, что требует тестирования не только структуры программы, но и ее поведения.

На модульном уровне выполняется автономное тестирование классов: их структуры и методов (функций-членов) класса традиционными методами, основанными на коде.

На уровне интеграции классов выполняется тестирование иерархии наследования и тестирование взаимодействий между объектами. Должны быть проверены все возможные внешние вызовы методов класса - как непосредственные обращения, так и вызовы, инициированные получением сообщений. В качестве формальных моделей выступают модель классов ПО и модели каждого класса.

На уровне системы применяется тестирование по сценариям использования (use-cases) – начиная с анализа требований и проектирования ПС.

Основные методы (подходы) тестирования ООП перечислены в таблице 7.1.

Таблица 7.1. **Основные методы тестирования ООП**

Задача тестирования	Методы/подходы
Разработка тестовых наборов (test-case)	Тестирование, основанное на ошибках (шаблонах ошибок) Тестирование, основанное на сценариях использования
Тестирование внутренней структуры класса	Стохастическое тестирование классов Эквивалентное разбиение на уровне класса Таблицы решений
Тестирование взаимодействия классов	Стохастическое тестирование классов Эквивалентное разбиение на уровне классов Тестирование переходов между состояниями

Парадигма ***компонентного программирования*** (глава 2) и компонентная программная инженерия – относительно новые направления, основанные на концепциях повторного использования и распределенной разработки. Основные особенности построение системы из существующих компонентов таковы:

- процесс разработки ПС из компонентов (фактически сборки) отделен от процесса разработки самих компонентов;
- разработка ПС включает поиск (приобретение) нужных компонентов; их адаптацию и тестирование;
- основные усилия в ЖЦ сосредоточены на интеграции («встраивании») компонентов в ПС.

Компонентное тестирование проходит на нескольких уровнях:

- тестирование компонента в изоляции (автономное);
- тестирование в процессе сборки;
- системное тестирование.

Тестирование компонента в изоляции выполняется при создании нового компонента. Помимо традиционных методов тестирования корректности реализации, тестируются также возможности повторного использования компонента при разных конфигурациях сред.

В наибольшей мере специфика компонентного тестирования проявляется в процессе сборки компонентов. Например, возможны проблемы несовместимости разных компонентов (или их версий). Помимо этого, требуется тестирование связующего кода, который, собственно, и формирует ПС.

На стадии системного тестирования применяются обычные методы функционального тестирования. Специфика их применения для данного типа ПС проявляется в сложности поиска причин ошибок, особенно, если ошибка, обнаруженная в одном компоненте, связана с другим компонентом. В этой связи повышается важность контрактных интерфейсов, которые обеспечивают спецификацию входов и выходов компонента и делают возможной проверку корректности данных [37].

Особенности ***тестирования Web-приложений*** связаны в первую очередь с тремя основными факторами: моделью ЖЦ, спецификой объектов (архитектурой приложения) и важностью тестирования технических характеристик качества (безопасность, конфигурация, производительность и др.).

Тестирование графического интерфейса пользователя всегда включает тестирование удобства применения, проверку на соответствие стандартам представления (например, Windows), а также тестирование конфигурации (на разных типах мониторов). Процесс планирования и проектирования тестов систематизиру-

ется путем применения контрольных вопросов, которые отражают требования к интерфейсу, подлежащие проверке.

Тестирование протоколов. Бурное развитие сетевых технологий, усложнение их архитектуры, взаимодействие нескольких разнородных сетей многократно увеличивают число интерфейсов, подлежащих тестированию.

В работе [38] рассмотрены следующие задачи тестирования протоколов:

- тестирование соответствия (аттестационное);
- тестирование производительности;
- тестирование совместного функционирования;
- тестирования взаимодействия;
- тестирование функциональности;
- мониторинг.

Тестирование на соответствие заданной спецификации – это стандартизированный и широко распространенный метод проверки корректности реализации протокола. Инструментальным средством для такого тестирования служит специализированный язык написания тестов TTCN [38]. Стандартизация подходов к тестированию соответствия протоколов осуществляется международными организациями ETSI, ITU-T и ISO. Основной документ - стандарт ISO 9646:1994 [39].

Однако одно лишь тестирование соответствия не может гарантировать полной корректности реализации, так как не предполагает проведение тестирования «под нагрузкой» и проверку поведения системы во всем диапазоне возможных значений параметров спецификации. Поэтому наряду с ним необходимо проведение и остальных перечисленных видов тестирования.

Тестирование совместного функционирования производится с применением эталонной системы или с использованием системы тестирования, имитирующей эталонную. Для имитации эталонной системы, с которой должно стыковаться тестируемое оборудование, служат эмуляторы протоколов и генераторы вызовов. При использовании реальной системы в качестве эталонной применяются анализаторы протоколов, которые осуществляют мониторинг интерфейса, соединяющего тестируемую систему с эталонной. Подробнее эти вопросы рассмотрены в [38].

При тестировании ***систем реального времени*** система *всегда* представляется в виде формальной модели. Поэтому на модульном уровне применяются формальные методы, основанные на формальной спецификации. Одним из методов функционального тестирования, специально предназначенным для систем этого класса, является тестирование переходов между состояниями (*state transition testing*) [36]. Обязательно также применение методов статистического тестирования.

Тестирование ***критических систем*** включает не только динамические, но и статические методы верификации, рассмотренные в главе 6, такие как *анализ критичности*, *анализ деревьев событий* и *деревьев отказов* [40] (кратко рассмотренные далее в этой главе), а также *инспекции* документов и исходного кода. Для минимизации стоимости тестирования и риска отказа в ходе анализа ПС выделяются критические модули, требующие углубленного тестирования. Для них применяются наиболее строгие критерии покрытия кода.

7.3.2. Исследовательское тестирование

Методы исследовательского тестирования предназначены для изучения программы и обнаружения в ней ошибок одновременно. Эти методы применяются в стратегиях тестирования, основанных на риске.

Один из методов исследовательского тестирования и процедура его применения для тестирования функциональности и устойчивости программных продуктов в среде Windows, предложены в [41].

Процедура включает следующие шаги.

Шаг 1. Исследование. Изучение назначения и функций программного продукта, типов обрабатываемых данных и областей его возможной неустойчивости (факторов риска отказа).

Успех выполнения шага зависит от наличия информации о программном продукте и его потенциальных пользователях, а также времени для его изучения. Для получения информации о продукте может использоваться документация (в том числе Help), сведения об аналогичных продуктах, изучение продукта путем его кратковременного использования. Задачи шага:

1. Формирование списка функций (иерархии функций).
2. Разбиение функций на основные и второстепенные. В таблице 7.2 приведен пример критерия разбиения – по важности.
3. Выявление областей возможной неустойчивости продукта (подверженных отказам функций и данных).

Таблица 7.2. **Пример разбиения функций на основные и второстепенные**

Определение	Критерии отнесения
Основная функция – любая внешняя функция (видимая пользователю), отказы в которой означают несоответствие продукта своему назначению.	1. Функции, являющиеся определяющими для использования продукта по назначению. 2. Функции, часто используемые обычными пользователями в сеансе работы. 3. Функции, которые могут использоваться редко, но их отказы приводят к значительным отрицательным последствиям.
Второстепенная функция – функция, без которой продукт может использоваться.	Редко используемые функции, отказы при выполнении которых не могут иметь существенных последствий.

Примеры областей возможной неустойчивости функций:

- функции обработки внешних событий;
- функции, интенсивно использующие оперативную память;
- очень сложные функции;
- функции, использующие средства Windows и/или изменяющие ее параметры (настройки параметров);
- функции, манипулирующие конфигурацией Windows;
- функции анализа входных данных и обработки ошибок;
- функции, подменяющие базовые функции Windows (например, восстановление удаленных файлов);
- функции или группы функций, использующие много параллельных процессов;

- функции, работающие со многими файлами одновременно;
- функции, работающие с файлами, расположенными в сети.

Примеры областей возможной неустойчивости при обработке данных:

- *документы*: длинные, много одновременно открытых;
- *записи*: длинные, много записей, пустые, сложные;
- *списки*: длинные, пустые или многоколоночные;
- *поля*: ввод большого количества символов; очень большие значения числовых полей;
- *объекты*: много, крупные, сложные и др.

Шаг 2. Проектирование тестов. Определение стратегий выполнения тестов и критериев оценивания результатов (как, например, в таблице 7.3).

Таблица 7.3. **Пример критериев прохождения тестов**

Определение цели	**Критерий прохода**	**Критерий отказа**
Функциональность – способность продукта выполнять требуемые функции.	Функция выполняется в соответствии с ее назначением.	Хотя бы одна функция не выполняется в соответствии с ее назначением.
	Любое неправильное выполнение функции не приводит к серьезным последствиям при корректном использовании.	Неправильное выполнение приводит к серьезным последствиям даже при корректном использовании.
Устойчивость – способность продукта функционировать без отказов при повышенных нагрузках (по времени, памяти).	Работа продукта при повышенных нагрузках не приводит к разрушению Windows. Работа продукта не приводит к потере данных, зависанию, отказу. При тестировании не было обнаружено отказов или любых дефектов при выполнении основных функций.	Работа продукта разрушает Windows. Работа продукта может привести к потере данных, зависанию, отказу. При тестировании хоть одной из основных функций наблюдались отказы.

Шаг 3. Выполнение тестов. Выполнение тестов, наблюдение и фиксация результатов и использование этой информации для формирования мнения о работе продукта. Задачи шага:

- тестирование всех основных функций;
- тестирование идентифицированных областей потенциальной неустойчивости;
- выборочное тестирование второстепенных функций;
- регистрация отказов;
- фиксация любых обнаруженных проблем (для дальнейшего изучения).

Шаг 4. Построение эвристик. Эвристики представляют неформальные правила (основанные на опыте тестировщика и здравом смысле) принятия решения о том, считать ли работу продукта приемлемой или нет, и как дальше продолжать тестирование.

Шаг 5. Определение критерия покрытия. В данной процедуре используется следующий критерий покрытия:

- протестированы все основные функций;
- протестированы выбранные второстепенные функции;
- протестированы выбранные области возможной неустойчивости (следует выбрать от пяти до десяти функций и протестировать их на тестовых данных, которые могут привести к неустойчивой работе продукта).

Рекомендуемое распределение времени тестирования: 80% времени – на основные функции, 10% - на второстепенные и 10% - на области потенциальной неустойчивости.

Процедура тестирования по данному методу выполняется циклически.

7.3.3. Эквивалентное разбиение

Суть метода эквивалентного разбиения состоит в том, что все множество тестов, которые теоретически можно построить для тестирования программы, реализующей функцию, разбивается на подмножества тестов, образующих «классы эквивалентности», из которых для выполнения выбираются только наиболее представительные (с наибольшей вероятностью обнаружения ошибки).

Отнесение группы тестов к одному классу эквивалентности может основываться на следующих практических критериях [8]:

- тесты включают значения одних и тех же входных данных;
- при запуске тестов выполняются одни и те же операции;
- от выполнения тестов ожидаются одинаковые результаты;
- либо ни один из тестов не вызывает выполнения блока обработки ошибок, либо все тесты вызывают выполнение этого блока (в предположении, что программа вообще содержит блоки обработки ошибок).

Классы эквивалентности подразделяются на *допустимые классы эквивалентности*, представляющие правильные входные данные и входные события (события-входы), и *недопустимые классы эквивалентности*, представляющие все остальные данные или события. Для представления классов эквивалентности можно воспользоваться табличной формой (ее пример дан в таблице 7.4).

При выделении классов эквивалентности можно руководствоваться рядом общих правил:

- если событие-вход ассоциируется с *областью* значений (например, как в таблице 7.4), то определяется один допустимый класс эквивалентности и ряд недопустимых (в таблице их пять);
- если событие-вход ассоциируется со значением, представляющим количество чего-либо (например, от 1 до 20), то определяется один допустимый класс эквивалентности ($1 <= X <= 20$) и ряд недопустимых ($X < 1$ и $X > 20$);
- если для параметра программы допускается только определенный перечень значений (например, перечень конкретных наименований чего-либо, хранящихся в базе данных), то определяется один допустимый класс эквивалентности для значений из этого перечня и один недопустимый (например, для значений, отсутствующих в перечне). В этом случае граничные значения не определяются;
- любой элемент списка параметров программы или объектов (списки, меню, кнопки панели инструментов) представляет отдельный класс эквивалентности.

Таблица 7.4. **Пример описания классов эквивалентности**

Входные события	**Допустимые классы эквивалентности**	**Недопустимые классы эквивалентности**
Ввод числа	Числа от 0 до 9000	Числа меньше 0 и числа больше 9000 Пробел Нечисловые символы Вычисляемое выражение, результат вычисления которого находится вне допустимого интервала
Ввод первой буквы фамилии	Первый символ должен быть заглавной буквой	Первый символ строчная буква Первый символ – не буква

Тесты разрабатываются вначале для допустимых классов эквивалентности, а затем – для недопустимых:

- для *допустимых*: по одному тесту внутри класса и по одному на границах класса (например, если параметр программы содержит список, нужно из списка выбрать первый, последний и любой промежуточный элемент, а для числовых значений в диапазоне от 1 до 100, выбрать 1, 100 и любое число внутри диапазона);

- для *недопустимых*: по одному тесту для каждого класса (например, для числовых значений в приведенном выше примере - выбрать 0 и 101, пробел и нечисловой символ).

7.3.4. Анализ граничных значений

По этому методу тесты разрабатываются для направленного тестирования *границ* классов эквивалентности. Если область входных данных класса представляет непрерывный диапазон значений, то выбираются допустимые значения, расположенные на нижней и верхней границах диапазона. Если диапазон дискретный (например, элементы списка значений) – выбираются первый и последний допустимые элементы.

При этом анализируются также все выходные классы эквивалентности - ожидаемые результаты тестов. Если ожидаемые результаты представляют значения из непрерывного диапазона значений, то входные данные для их получения выбираются так, чтобы получить значения, находящиеся на верхней и нижней границах допустимого диапазона. В том случае, если результаты представляют собой наборы значений результатов, входные данные выбираются для получения первого и последнего элементов (например, проверить вывод на печать первой и последней строки отчета).

Отличие данного метода от метода эквивалентного разбиения состоит в следующем:

- при анализе граничных значений выбираются только такие элементы в классе эквивалентности, которые позволяют проверить тестом каждую границу этого класса;

- при разработке тестов рассматриваются не только входные значения, но и ожидаемые результаты.

7.3.5. Разбиение входного пространства на категории

Процедура применения этого метода включает следующую последовательность шагов.

Шаг 1. Декомпозиция функции на функциональные элементы, которые могут тестироваться независимо. Для больших и сложных функций декомпозиция выполняется иерархически. С каждым функциональным элементом связывается один или несколько модулей, его реализующих. Побочным положительным эффектом выполнения этого шага является анализ качества декомпозиции проекта по функциональному признаку [42].

Шаг 2. Идентификация параметров и условий среды, влияющих на поведение функции. Параметры являются входами к функциональному элементу, а условия среды определяют способ реализации функционального элемента и характеристики состояния системы во время его выполнения. Например, для функций интерфейса пользователя параметрами могут быть поля ввода, поля вывода, список выбираемых элементов и др., а условиями среды – файлы, поля таблицы базы данных и др.

Реальные тесты для функционального элемента представляют собой композицию определенных значений параметров и условий среды, выбранных с целью максимизации шансов поиска дефектов в реализации элемента.

Шаг 3. Выделение категорий информации, характеризующей каждый параметр и условие среды. Категория является основным свойством (или характеристикой) параметра или условия среды. В приведенном выше примере параметров и условий среды для функций интерфейса пользователя можно выделить следующие категории:

- для параметра «поле ввода» - категории тип, размерность, состояние, содержание, обязательность и др.;
- для условия среды «окно» - категории координаты, цвет окна, цвет символов, рамка и др.

Выбор категорий производится путем поиска в спецификации функции ключевых фраз, которые описывают, как ведет себя функциональный элемент при определенных сочетаниях параметров и условий среды.

Шаг 4. Разбиение каждой категории на четкие альтернативы, которые включают различные виды значений, возможные для категории. Альтернативы являются классами разбиения, из которых будут выбираться представительные элементы для построения тестов. В таблице 7.5 приведены примеры категорий и возможных альтернатив применительно к упомянутым параметрам функций интерфейса пользователя.

Шаг 5. Формирование формальной спецификации теста для каждого функционального элемента. Спецификация теста состоит из списка категорий и списков альтернатив в каждой категории. Информация из спецификации теста используется далее для получения множества фреймов теста, являющихся основой для создания реальных тестовых наборов. Фрейм теста состоит из множества выделенных альтернатив, причем каждая категория представлена не более чем одной альтернативой (или нулем). Построенная спецификация теста практически не ограничена, поскольку в ней не учтены возможные отношения между альтернативами. Полное число фреймов, сгенерированных по такой спецификации, равно произведению числа альтернатив в каждой категории. С целью сокращения числа возможных тестовых фреймов устанавливаются ограничения для взаимосвязанных альтернатив.

Таблица 7.5. **Пример разбиения на категории и альтернативы**

Параметр	Категория	Альтернативы
Поле выбора	Содержание	Текст Пиктограмма Значок с текстом
	Состояние	Поле не указано/выбрано Поле не указано/не выбрано Поле указано/выбрано Поле указано/не выбрано
	Тип поля	Однозначный выбор Многозначный выбор Расширенный выбор
Меню	Способ выбора	Выбор курсором Выделенная буква/(комбинация букв) Функциональная клавиша Выбор с помощью клавиатуры

Типичное ограничение указывает на то, что альтернатива из одной категории не может появиться вместе в тестовом фрейме с определенными альтернативами из других категорий.

Шаг 6. Генерация тестовых фреймов. Поскольку процесс формирования тестовых фреймов связан с перебором большого числа комбинаций альтернатив с учетом ограничений, он должен быть по возможности, автоматизирован. Генерируемый тестовый фрейм будет содержать по одной альтернативе для каждой категории.

Шаг 7. Преобразование тестовых фреймов в тестовые наборы и разработка тестовых сценариев. Реальные тестовые наборы для функционального элемента являются композицией определенных значений параметров и условий среды, представленных альтернативами соответствующих категорий и подобранных с целью максимизации шансов обнаружения дефектов в реализации функционального элемента.

Тестовый сценарий формируется из последовательности связанных тестов для одного или более функциональных элементов, требующих одинаковых условий среды. Например, один тестовый сценарий для тестирования интерфейса пользователя должен обеспечивать прохождение одного сценария работы пользователя в программе при фиксированных условиях среды на разных типах данных (допустимых, граничных, недопустимых).

Этот метод обладает рядом очевидных достоинств:

- позволяет охватить сразу оба основных аспекта тестирования - проверку полноты реализации функций и обнаружение различных классов дефектов в наиболее уязвимых частях программы;
- функциональный анализ является неотъемлемой частью метода и выполняется параллельно со спецификацией функциональных требований (либо сразу после), обеспечивая своевременное устранение дефектов спецификаций;
- хотя спецификация теста должна охватывать все возможные категории информации и варианты сочетания альтернатив, - применяя механизм ограничений и руководствуясь практическими соображениями, можно управлять объемом тестирования;

- процесс перебора альтернатив и «отсеивания» невозможных или нежелательных их сочетаний может быть автоматизирован, что избавит тестировщика от рутинной работы.

7.3.6. Тестирование переходов между состояниями

По этому методу тесты проектируются для проверки переходов между состояниями программы (например, изменения изображения на экране, смены состава и активности элементов меню и др.). Поскольку количество возможных состояний и переходов между ними может быть гораздо больше, чем можно протестировать, при проектировании тестов следует руководствоваться практическими соображениями [8]:

- набор тестов должен охватывать наиболее вероятные последовательности действий пользователей;
- если есть зависимые состояния и синхронизируемые события, для каждого из них должны разрабатываться отдельные тесты;
- отдельные тесты следует проектировать также для проверки недопустимых переходов между состояниями (для выявления возможных нежелательных побочных эффектов);
- после выполнения минимального набора тестов следует провести набор случайных тестов (выбирая произвольные режимы и сценарии выполнения).

В качестве инструментов при проектировании тестов используют:

- диаграммы переходов в состояния (State-transition diagram) [36];
- таблицы переходов в состояния (State-transition tables).

Диаграмма перехода в состояния отображает множество состояний в рассматриваемом контексте, события, которые вызывают переходы между состояниями, и возможные действия. На рисунке 7.4 представлен общий вид диаграммы.

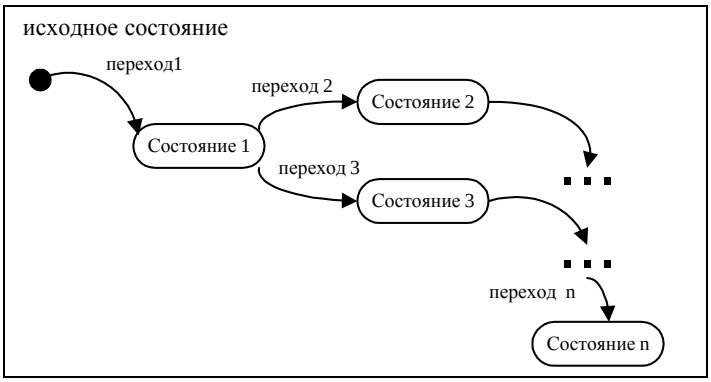


Рис.7.4. **Общий вид диаграммы переходов состояний**

Таблицы переходов в состояния предназначены для изображения той же информации, но в табличном виде (таблица 7.6).

Таблица 7.6. **Структура таблицы переходов состояний**

Состояние	Событие	Действие	Следующее состояние

Этот метод изначально предназначался для тестирования программ реального времени, но впоследствии стал применяться при проектировании тестов для интерактивных программ: тестирования состояний меню (активно, не активно, выбрано), навигации между окнами, синхронизации элементов интерфейса в диалоговых окнах.

7.3.7. Тестирование, основанное на моделях программной системы

Как самостоятельное направление тестирование на базе моделей (model-based testing (MBT)) оформилось к середине 90-х годов прошлого века в связи с автоматизацией разработки тестов [43]. Хотя, по сути, все методы тестирования применяют модели, для данного направления характерно следующее:

- модель представлена в формальном виде;
- модель применяется для генерации тестов или эталона.

В энциклопедии wikipedia (www.wikipedia) тестирование на базе моделей определяется как «…тестирование ПО, при котором тесты разрабатываются полностью или частично по модели, описывающей некоторые аспекты (как правило, функциональные) тестируемой системы».

Тесты в целом образуют набор, именуемый *абстрактным набором тестов* (abstract test suite), который непосредственно не может применяться для тестирования, а служит основой для создания *выполняемого набора тестов*, генерируемого (выводимого) по модели. Разработка же самой модели делается параллельно с разработкой ПС или непосредственно для существующей ПС.

Тестирование на базе моделей - это обобщение и расширение известных «классических» методов, применяемых для автоматического построения тестов, например:

- теория конечных автоматов;
- переходы между состояниями;
- доказательство теорем и др.

Более подробную информацию о данном подходе можно получить, например, на сайте http://www.model-based-testing.org/.

7.3.8. Тестирование Web-приложений

Web-приложения можно рассматривать как клиент/серверные приложения, в которых функциональность реализуется как на серверной, так и на клиентской стороне, а пользовательский интерфейс имеет стандартизированную архитектуру, в которой [44]:

1) для взаимодействия с пользователем используется *Web-браузер*;

2) взаимодействие с пользователем четко разделяется на этапы, в течение которых браузер работает с одним описанием интерфейса, а эти этапы, в свою очередь, разделяются однозначно локализуемыми обращениями от браузера к приложению;

3) для описания интерфейса применяется стандартное представление;

4) взаимодействие между браузером и приложением осуществляется по стандартному протоколу (HTTP) и четко формализовано;

5) функциональность Web-приложения распределена между удаленным сервером и клиентскими компьютерами пользователей.

Таким образом, тестирование Web-приложений проводится на трех уровнях:

- интерфейса пользователя;
- сервера (серверов);
- протоколов их взаимодействия.

В целом, тестирование основано на использовании сценариев, описывающих последовательность действий виртуальных пользователей.

Функциональное тестирование проводится как на уровне интерфейса приложения, так и на уровне его взаимодействия с сервером.

Тестирование интерфейса обычно сводится к проверке корректности вводимых данных, правильности навигации по сайту и других характеристик интерфейса и связано с удобством применения Web-приложения.

В таблице 7.7 приведен пример общего контрольного вопросника для проверки удобства применения интерфейса Web-сайта, который может быть полезен для планирования тестирования (опубликован компанией IT-Online) [45].

Таблица 7.7. **Контрольные вопросы для проверки удобства применения Web-приложений**

Архитектура и навигация сайта
Соответствует ли структура сайта целям, для достижения которых он предназначен?
Понятна ли схема навигации?
Можно ли определить, в каком месте сайта вы находитесь?
Легко ли найти на сайте нужную информацию?
Разумно ли количество элементов в панелях навигации?
Логично ли отсортированы элементы?
Соответствуют ли названия гиперссылок названиям страницы?
Отчетливо ли выделены гиперссылки ?
Отчетливо ли выделена ссылка на главную страницу?
Существует ли возможность поиска информации на сайте?
Существует ли карта сайта?
Каждая ли страница позволяет понять, на каком сайте вы находитесь?
Можно ли управлять навигацией по сайту?
Планировка и дизайн сайта
Не превышает ли размер страницы размер окна?
Повторяется ли схема планировки на всех страницах?
Существует ли отчетливый фокус на каждой странице?
Видна ли визуально планировка?
Эффективно ли используется выравнивание и группировка?
Достаточно ли хорош контраст?
Не громоздкая ли планировка?
Нравится ли вам сайт эстетически?
Содержание сайта
Понятны и лаконичны ли тексты на сайте?
Организован ли текст в виде небольших блоков?
Встречаются ли в тексте грамматические и орфографические ошибки и опечатки?
Содержат ли страницы вводный текст?
Поддерживают ли мультимедийные компоненты задачи пользователя?
Являются ли единицы измерения, используемые на сайте, понятными?
Представлены ли на сайте время и дата создания страниц?
Представлены ли на сайте номера контактных телефонов?
Представлены ли на сайте адреса с почтовыми индексами?

Формы и взаимодействие
Соответствуют ли формы задачам пользователя?
Поддерживают ли диалоги логичную последовательность шагов?
Очевидна ли кнопка или ссылка для перехода к следующему шагу диалога?
Все ли элементы форм используются по назначению?
Сгруппированы ли элементы формы по смысловому назначению?
Понятно ли выглядит кнопка отправки формы?
Графика
Приемлемо ли качество используемой графики ?
Все ли графические элементы имеют альтернативные текстовые надписи?
Содержат ли графические элементы информацию о размере файла?
Оптимизированы ли графические элементы для передачи по Интернету?
Реагируют ли графические элементы на движения мышки?
Используется ли анимация? Приемлем ли объем графических файлов?
Цвета
Подходящий ли выбор цветов для сайта?
Не слишком ли много цветов?
Используются ли цвета логично и последовательно?
Адекватно ли различаются цвета в черно-белом режиме?
Оформление текста
Понятны ли тексты?
Приемлем ли размер шрифта?
Имеет ли шрифт подходящий цвет и достаточно ли он контрастный?
Отформатирован ли текст так, чтобы в строке было от 10 до 12 слов?
Достаточна ли ширины поля вокруг текста?
Устойчивость к ошибкам
Должен ли пользователь что-нибудь запоминать, переходя между страницами?
Возникает ли предупреждение при попытке совершения необратимых действий?
Можно ли отменить рискованные действия?
Перехватываются ли возникающие ошибки локально, без обращения к серверу?
Содержат ли страницы с сообщением о возникших ошибках полезную информацию?
Содержат ли страницы с пустыми результатами поиска советы по расширению условий поиска?
Существует ли система контекстной помощи (справки)?
Структурирована ли помощь по задачам пользователя? Объясняет ли она пользователю, как совершить то или иное действие?
Платформа и особенности реализация
Достаточно ли быстро происходит загрузка страниц (от 3 до 15 секунд)?
Все ли гиперссылки работают правильно?
Существуют ли поврежденные графические элементы?
Написан ли текст на страницах так, чтобы его могли найти поисковые системы?
Работает ли сайт с разными браузерами пользователя?
Работает ли сайт на мониторах высокого и низкого разрешения?
Используются ли нестандартные plug-in? Являются ли они необходимыми и полезными?

Вопросы тестирования *серверной части Web-приложений* рассмотрены в разных источниках [44, 46-50], например, в [44] предложена оригинальная технология тестирования UniTesK.

Наряду с функциональным тестированием в Web-приложениях возрастает роль тестирования ***технических характеристик***, прежде всего, надежности и восстанавливаемости, безопасности, производительности, конфигурации (в разных браузерах и платформах).

Вопросы *тестирования безопасности* охватывают, помимо проверки корректности ввода, анализ таких распространенных аспектов «уязвимости» Web-приложений, как плохие механизмы аутентификации, логические ошибки, непреднамеренное раскрытие информации, а также традиционные ошибки в обычных приложениях. Так, в работе [49] предложена методология систематического тестирования, охватывающая следующие аспекты проверки:

1. Идентификация окружения web-приложения. Включает анализ информации об используемых языках скриптов, Web-сервере, операционной системе. Для извлечения такой информации рекомендуется:

– проанализировать отклик на HTTP-запросы HEAD и OPTIONS (из заголовка или любой страницы, содержащей строку SERVER);

– исследовать формат и текст информации о 404-й ошибке сервера (и других). Некоторые системы имеют легко узнаваемые сообщения об ошибках, а также часто позволяют узнать версии используемых языков. Можно преднамеренно запросить страницы, приводящие к подобным ошибкам, а также использовать альтернативные методы запроса (POST, PUT и т.п.) для извлечения информации из сервера;

– проверить распознаваемые типы файлов/расширения/каталоги. Многие Web-серверы по-разному реагируют на запросы файлов с поддерживаемыми и неизвестными расширениями. Следует попытаться запросить файлы со стандартными расширениями, такими как “.ASP”, “.HTM”, “.PHP”, “.EXE”, и следить за появлением какого-либо необычного результата или кодов ошибок;

– проверить исходные тексты доступных страниц. Исходный текст страницы, сгенерированной Web-приложением, может указывать на используемое программное окружение;

– попытаться исказить входные данные для получения ошибок в скриптах. Это также позволит получить информацию об окружении Web-приложения;

– использовать сканирование TCP/ICMP и сервисов. Для этого рекомендуется использовать анализаторы приложений (в частности, Amap и WebServerFP) или средства сканирования Nmap и Queso.

2. Тестирование скрытых элементов форм и раскрытие исходных текстов. Включает проверку всех исходных текстов страниц на наличие любой полезной информации, непреднамеренно оставленной разработчиком - это могут быть фрагменты скриптов, расположенных внутри HTML-кода, ссылки на подключаемые или связанные скрипты, неправильно розданные права доступа к критичным файлам с исходным текстом. Должно быть проверено наличие каждой программы и скрипта, ссылки на которые были найдены. В случае обнаружения они тоже должны быть протестированы.

3. Определение механизмов аутентификации. Необходимо проверить, как выбранный механизм применяется к каждому ресурсу, используемому Web-приложением. Для этого следует попытаться получить доступ ко всем ресурсам через каждую точку входа. Многие Web-системы предлагают средства поддержки сессий, основанные на сохранении в cookie или Session-ID псевдоуникальной стро-

ки, характеризующей их статус. В том случае, если данная строка является простым хэшем или строкой, составленной из известных элементов, система может оказаться уязвимой к таким видам атаки, как, например, прямой перебор, повторная отправка, или попытка восстановления.

Некоторые особенности тестирования Web-приложений связаны с процессом разработки. Как правило, для создания Web-приложений используются методы «ускоренной» разработки, в которых:

– тестирование встроено в ЖЦ и выполняется параллельно с разработкой;
– небольшие проектные группы (или один Web-дизайнер);
– активное участие заказчика (не всегда возможно);
– очень сжатые сроки разработки версий ограничивают время на тщательное планирование и выполнение тестирования;
– фаза интеграции Web-приложения обычно отсутствует (нет интеграционного тестирования) поэтому основное внимание уделяется функциональному тестированию приложений в моделируемой среде и системному – в реальной среде;
– стадия сопровождения и развития – неотъемлемая часть ЖЦ, следовательно, требуется постоянное регрессионное тестирование.

Сжатые сроки тестирования приводят к необходимости его автоматизации. Особенно это касается тестирования производительности и надежности сервера. Перечень наиболее распространенных инструментов тестирования различных фирм-производителей можно найти в приложении 4.

Среди множества публикаций по тестированию можно выделить переводную монографию [50], посвященную вопросам тестирования Web-приложений, а также перечень статей, посвященных разным аспектам тестирования в Интернет, например, на сайте http://www.rspa.com/reflib/TestingWebApps.html#testingMethods

7.3.9. Особенности выполнения тестирования в моделях ЖЦ

Процесс тестирования ПС и его интеграция с другими процессами разработки существенным образом зависит от принятой в проекте модели ЖЦ.

В таблице 7.8 представлены основные модели ЖЦ и подходы к тестированию, применяемые в этих моделях (по материалам сайта http://www.qalabs.com).

Таблица 7.8. **Подходы к тестированию, применяемые в моделях ЖЦ**

Модель ЖЦ	Применяемые подходы к тестированию
Каскадная	**Ключевая особенность модели:** Каскадная модель с циклами обратной связи, моделированием или прототипированием. Тестирование начинается после завершения кодирования. **Основные методы/подходы**: 1. Передача кода группе тестирования после его готовности. 2. Пользователи привлекаются к тестированию на этапе приемочных испытаний. 3. После выпуска продукта проводится дополнительное тестирование (опытная эксплуатация). **Примечание:** Улучшение тестирования возможно в конце проекта, что увеличивает время для возможности улучшения процесса и только в следующих проектах.

Модель ЖЦ	Применяемые подходы к тестированию
Спиральная	**Ключевая особенность модели**: использует прототипирование и перепланирование с постоянным оцениванием рисков и ограничений в рамках каждого прототипа. **Основные методы/подходы**: 1. Тестирование, основанное на оценке риска (risk-driven testing). Тестирование применяется для идентификации рисков. Анализ дефектов служит ключом для идентификации «узких» мест в ПО. 2. Тестирование начинается на ранних стадиях ЖЦ. 3. Критерий завершения тестирования, основанный на риске. Тестовое покрытие – не основная цель. Стратегия тестирования формируется с учетом рисков потенциально оставшихся дефектов и помогает принять решение о переходе к следующему витку спирали. **Примечания:** 1. В рамках стадии ЖЦ в одном витке спирали тестирование выполняется так же, как и в каскадной модели, но на меньшем объеме ПС и менее продолжительно. 2. Общее время тестирования в рамках ЖЦ больше, поскольку больший промежуток времени между его началом и выпуском ПС
Personal Software Process (PSP)	**Ключевая особенность модели**: Обучение разработчиков методам идентификации, анализа и улучшения собственных процессов. Цель – дать возможность каждому улучшить собственную производительность и достичь установленных целей качества. **Основные методы/подходы**: 1. Методы верификации (инспекции, персональные просмотры, коллективные проверки). 2. Раннее устранение дефектов. Разработчики поощряются за раннее устранение дефектов; фиксацию и управление дефектами, регулярный анализ дефектов. 3. Стремление к постоянному улучшению процесса. **Примечание:** Персональное управление дефектами (фиксация своих дефектов сразу после их внесения, персональные проверки проекта и кода по установленным руководствам и контрольным вопросникам).
Модели ускоренной разработки (Agile Programming)	**Ключевая особенность модели:** Итеративный подход, постоянное автоматизированное тестирование разработчиками, тесное взаимодействие, минимум документации (в том числе по тестированию). **Основные методы/подходы**: 1. Управляемая тестами разработка (test-driven development). Разработчики пишут тесты до кодирования и (при необходимости) обновляют их для повышения полезности 2. Автоматизация. Когда тесты «стабильны», они автоматизируются, предпочтительно группами так, чтобы их можно было ассоциировать с модулем или проектом системы. 3. Принцип «достаточно хорошее качество». Достаточность связывается со сложностью ПС, критичностью ее функций, терпимости заказчика к ошибкам и др. 4. Тестирование рассматривается как стратегия снижения риска отказа. 5. Приемочные тесты обычно разрабатываются заказчиком и должны быть готовы до начала проектирования и кодирования.

7.4. Анализ результатов тестирования

7.4.1. Система отслеживания проблем

На эффективность тестирования и устранения дефектов оказывает большое влияние степень четкости процесса подготовки и анализа отчетов о проблемах, а также наличие хорошего инструмента для его поддержки. При отсутствии промышленного инструмента, группа тестирования может создать свою систему отслеживания проблем, например, с помощью MS Access.

Цели системы отслеживания проблем:

– отслеживание состояния тестирования и устранения дефектов;

– организация взаимодействия между сотрудниками и решение спорных вопросов относительно классификации и приоритетов устранения дефектов;

– определение причин дефектов и выявление «узких мест» в процессах разработки и тестирования.

С отчетами о проблемах в процессе тестирования работают тестировщики, разработчики, руководитель проекта и группа качества. Поэтому для эффективного решения проблемы важно определить полномочия пользователей системы отслеживания проблем (как исполнителей процесса «Решение проблем»).

Процесс решения проблем, обнаруженных при тестировании, может быть представлен следующей последовательностью шагов.

Шаг 1. Открытие проблемы. При обнаружении проблемы тестировщик составляет отчет о проблеме («открывает проблему») и передает его программисту для анализа. При автоматизированном ведении отчетов тестировщик помещает отчет в базу данных.

Шаг 2. Изучение. Программист анализирует отчет и принимает решение (согласен или нет). Если проблема вызвана случайным отказом, связанным, например, со сбоем оборудования, средой тестирования, нарушениями технологии тестирования или непониманием тестировщика - она отклоняется. Если установлено, что проблема связана с дефектом в ПО, программист устанавливает приоритет устранения дефекта. В спорных случаях решение о приоритете устранения принимает руководитель проекта.

Шаг 3. Действие. Программист выполняет устранение дефекта и повторное автономное тестирование и возвращает отчет с отметкой «Исправлена» тестировщику. При автоматизированном ведении отчетов тестировщик находит отчеты с этой отметкой в базе данных.

Шаг 4. Закрытие. Тестировщик выполняет проверку исправлений и *закрывает проблему*. Закрыть отчет о проблеме может только тестировщик. При автоматизированном ведении отчетов о проблемах они хранятся в базе данных для последующего анализа, формирования отчетов, а также для накопления исторических данных. Отчеты об отклоненных дефектах могут удаляться из базы данных (но только тестировщиком!).

Шаги 1 - 4 касаются отслеживания каждого отдельного дефекта, но для анализа результатов и управления процессом тестирования используются обобщенные и классифицированные данные о дефектах. Этот анализ и обобщение выполняется менеджером группы тестирования и менеджером проекта. Обобщенные данные используются также группой качества для анализа текущего состояния проекта.

7.4.2. Классификация дефектов, обнаруженных при тестировании

Существуют разные подходы к классификации дефектов, которые вносятся в ПС и выявляются на разных стадиях ее разработки. Вопросы классификации дефектов, выявляемых при выполнении процессов верификации и валидации, уже рассматривались в главе 6. Наиболее полная классификация представлена в стандарте IEEE Std. 1044:1993 [51]. В нем вместо термина *дефект* используется термин *аномалия* для обозначения любых отклонений в ПС, ее спецификациях, документации, а также планах или процедурах, связанных с разработкой или использованием ПС. В этом разделе рассматриваются вопросы классификации дефектов, которые обнаруживаются при тестировании.

Наиболее важными аспектами классификации дефекта являются: *симптом; серьезность; приоритет устранения; стадия разработки и источник; тип.*

Симптом дефекта касается его видимого проявления, наблюдаемого тестировщиком при выполнении тестов. Описание симптома необходимо для анализа дефекта разработчиком и определения истинной причины дефекта.

Рекомендуемая IEEE Std.1044:1993 классификация симптомов такова:

- Аварийное завершение программы
- Не ожидаемое поведение программы
- "Зависание" программы
- Проблема ввода
  - Корректные данные не вводятся
  - Неправильные данные вводятся
  - Описание данных отсутствует или неправильно
  - Параметры не полные или отсутствуют
- Проблема вывода
  - Неправильный формат
  - Неправильные результаты/данные
  - Вывод не полный или отсутствует
  - Грамматика/синтаксис
- Неудовлетворительная производительность
- Ощущение общего отказа продукта
- Сообщение об ошибке системы
- Другое

Для классификации дефектов ***по серьезности*** стандарт IEEE Std.1044:1993 рекомендует такую порядковую шкалу:

- Критический
- Серьезный
- Значительный
- Незначительный
- Не дефект

В каждом конкретном случае «серьезность» дефекта зависит от типа системы и должна определяться в контексте последствий дефекта для системы (или для пользователя). Пример отнесения дефекта к категории серьезности представлен в таблице 7.9 (в ней каждому уровню серьезности присвоен код).

Таблица 7.9. **Описание серьезности дефекта**

Код	Серьезность	Описание
1	Критический	Дефект приводит к отказу всей системы, подсистемы, или компонента. Дальнейшее тестирование и/или использование системы невозможно.
2	Серьезный	Дефект приводит к отказу компонента ПО, потере данных.
3	Значительный	Дефект приводит к получению некорректных, неполных или противоречивых результатов, либо дефект касается удобства использования системы.
4	Незначительный	Дефект не приводит к отказу, не ухудшает удобство использования системы и может быть легко обойден.
5	Не дефект	Ошибки в тестах, сбои аппаратной или программной среды.

С каждым дефектом связывается ***приоритет устранения***. Для классификации дефектов также может использоваться порядковая шкала (таблица 7.10).

Таблица 7.10. **Описание приоритетов устранения дефекта**

Код	Приоритет	Описание
1	В первую очередь	Дальнейшая разработка и/или тестирование не может выполняться до устранения дефекта
2	Обратить особое внимание	Дефект должен быть устранен как можно быстрее, поскольку его наличие отрицательно влияет на разработку и/или тестирование
3	В порядке очереди	Дефект должен устраняться в порядке плановых действий по разработке. Его устранение может быть отложено до выпуска новой версии
4	Отложить	Устранение дефекта может быть отложено на неопределенный срок. Он может быть устранен в следующей версии или вообще не устраняться
5	Отклонить	Не дефект. Возможно случайный сбой или ошибка тестировщика

Классификация дефектов по ***стадиям разработки*** соотносит дефекты со стадиями разработки (и процессами), на которых они были внесены.

Классификация дефектов ***по источникам*** соотносит дефекты с рабочими продуктами стадий разработки, использование которых привело к появлению дефектов в коде ПО, например:

- Спецификация (требований, функций, проекта, интерфейса, данных)
- Код (дефекты кодирования)
- Документация на систему
- Планы и процедуры тестирования

Ниже перечислены основные ***типы дефектов***, рекомендуемые IEEE Std.1044:1993.

- Логические
- Вычислений
- Интерфейса
- Обработки данных
- Ввода данных
- Обработки ошибок и др.

Собранные и классифицированные данные о дефектах составляют основу для вычисления метрик тестирования.

7.4.3. Измерение результатов тестирования

Количественное измерение результатов тестирования основывается на применении метрик оценивания текущего состояния *объектов* тестирования (метрики продукта) и *процесса* тестирования (оценка выполненных тестов).

К основным *метрикам оценивания продукта* можно отнести метрики трех категорий, представленных ниже.

Метрики подсчета дефектов, указанные в таблице 7.11.

Таблица 7.11. **Метрики подсчета дефектов**

Метрика	Обозначение	Описание
Количество дефектов	Дфакт	Сумма всех дефектов, обнаруженных при тестировании за выбранный промежуток времени. Вычисляется по открытым и закрытым отчетам о проблемах.
Плотность дефектов	Пл_Дфакт	Вычисляется в виде отношения количества обнаруженных дефектов к размеру ПО (KSLOC или FP). **Пл_Дфакт = (Дфакт/Размер)**

Метрики тенденций дефектов *(профили дефектов)*, представленные в таблице 7.12. Эти метрики предназначены для определения тенденций в появлении дефектов в ПС, а также динамики устранения дефектов. Профили дефектов могут вычисляться по любой классификационной категории: приоритетам устранения, типам, серьезности.

Таблица 7.12. **Метрики профилей дефектов**

Метрика	Обозначение	Описание
Профиль открытых дефектов	ПрДоткр	Отношение не устраненных (открытых) дефектов к общему количеству обнаруженных дефектов **ПрДоткр = Доткр/Дфакт**
Профиль закрытых дефектов	ПрДзакр	Отношение устраненных (закрытых) дефектов к общему количеству обнаруженных дефектов **ПрДзакр = Дзакр/Дфакт**
Профиль серьезности	ПрСер	Отношение количества серьезных дефектов к общему количеству обнаруженных дефектов **ПрСер = Дсер./Дфакт**
Средний возраст открытых дефектов	ВОЗРоткр	Отношение временных интервалов (в днях) открытых дефектов к их количеству **ВОЗРоткр = Дней откр/Доткр**, где Дней откр - общее количество дней между датой обнаружения дефектов и датой вычисления метрики.
Средний возраст закрытых дефектов	ВОЗРзакр	Отношение временных интервалов закрытых дефектов к их количеству **ВОЗРзакр = Дней закр/Дзакр** где Дней закр – общее количество дней от открытия до закрытия дефектов, Дзакр – количество закрытых дефектов.

Метрики надежности, представленные в таблице 7.13. Эти метрики вычисляются по данным об отказах и требуют помимо подсчета отказов (дефектов) измерения интервалов времени между отказами.

Таблица 7.13. **Метрики надежности**

Метрика	Обозначение	Описание
Интенсивность отказов (Failure Rate)	FR	Количество отказов в единицу времени (например, в час) **FR= Дфакт/T** где Дфакт – количество обнаруженных отказов (дефектов), T –период наблюдения, выраженный в выбранных единицах времени (например, в часах).
Среднее время между отказами (Mean Time Beetween Failure)	MTBF	Отношение суммы интервалов времени между последовательными отказами к количеству обнаруженных отказов **MTBF = Tc/ Дфакт** где Tc – сумма интервалов времени между обнаруженными отказами.

Метрики *процесса* тестирования предназначены для определения *состояния* выполнения тестирования (метрики покрытия) и динамики обнаружения дефектов.

Метрики покрытия - служат индикаторами полноты выполненного тестирования относительно выбранных критериев покрытия и вычисляются в виде отношения количества выполненных тестов к требуемому количеству. Эти метрики подразделяются на структурные и функциональные.

Метрики *структурного покрытия* вычисляются для выбранного структурного критерия как отношение количества *выполненных* строк кода (ветвлений или логических условий) к общему их количеству в модуле.

Метрики *функционального покрытия* вычисляются как отношение количества выполненных функциональных тестов к количеству тестов, требуемых для удовлетворения критерия, соответствующего выбранному методу функционального тестирования.

Метрики динамики обнаружения дефектов служат для измерения процесса тестирования во времени (таблица 7.14). Они также связаны с метриками покрытия.

Таблица 7.14. **Метрики динамики процесса тестирования**

Метрика	Обозначение	Описание
Динамика выполнения тестов	Твып	Вычисляется как отношение количества выполненных тестов к запланированному количеству **Твып = (Тфакт/Тплан)** где Тфакт – количество выполненных тестов, Тплан – количество запланированных тестов
Динамика обнаружения дефектов	Тдин	Вычисляется как отношение количества тестов, которые выявили дефекты, и выполненных тестов **Тдин = Тдеф/Тфакт**
Общее состояние выполнения тестирования	Тпроц	Вычисляется как отношение количества успешно пройденных тестов к запланированному количеству **Тпроц = Тпр/Тплан** Где Тпр – количество пройденных тестов

7.4.4. Критерии завершения тестирования

Как известно, наиболее распространенный критерий завершения тестирования – исчерпано время, выделенное на его проведение. Этот критерий никак не учитывает выполненного объема тестирования и риска отказа ПС из-за оставшихся дефектов. Более строгие критерии основываются на количественных измерениях.

В подходе, *основанном на метриках покрытия*, критерии формируются путем вычисления метрик функционального и структурного покрытия и отражают объем выполненного тестирования. Структурные критерии применяются при автономном тестировании и основаны на методах структурного тестирования, а функциональные - применяются на всех уровнях и основаны на методах функционального тестирования.

Согласно подходу, *основанному на профиле дефектов*, тестирование прекращается, если нет новых и открытых дефектов серьезности 1, 2, 3. Этот критерий применяется при функциональном и системном тестировании.

Согласно подходу, *основанному на оценках интенсивности отказов*, тестирование продолжается до тех пор, пока не будут достигнуты установленные в требованиях значения метрик надежности (интенсивность отказов и/или среднее время работы без отказа). Критерий применяется на уровне системного тестирования и предполагает статистическое тестирование (по операционному профилю [17]).

Поскольку ни один из критериев не гарантирует полноты тестирования, при принятии решения о завершении тестирования необходимо использовать комплексные критерии. Например, в работе [52] сформулирован комплексный критерий завершения тестирования для информационных систем:

- все запланированные функциональные тесты прошли (Тплан = 100%);
- структурное тестирование было выполнено набором тестов, который обеспечил 100% покрытие строк, 80% покрытие логических условий и 100% покрытие вызовов процедур;
- нет открытых дефектов серьезности 1, 2 и 3 и плотность дефектов ниже, чем 0.5 дефектов на KSLOC;
- интенсивность обнаружения отказов не выше 40 новых отказов на 1000 часов тестирования;
- продолжительность непрерывного функционирования ПС без отказа достигает 100 часов.

В условиях ограниченных ресурсов на тестирование критерий завершения может быть сформулирован исходя из оценок риска отказа ПС. Он учитывает, какие идентифицированные риски устранены путем тестирования, и какова серьезность оставшихся дефектов. Согласно данному критерию, тестирование может быть завершено, если все известные дефекты серьезности 1, 2 и 3 закрыты, новые - не обнаружены, а риск отказа из-за оставшихся дефектов настолько мал, что дальнейшее тестирование экономически не выгодно.

7.5. Описание процесса тестирования

7.5.1. Модель процесса тестирования

Как уже отмечалось, в соответствии с процессным подходом к разработке ПС все действия, связанные с тестированием, начиная с планирования до оценки результатов, должны быть объединены в четко определенный и документирован-

ный процесс тестирования.

Появление отдельного описания процесса тестирования вызвано, прежде всего, тем, что в стандарте ISO/IEC 12207:1995 задачи тестирования «размыты» по многим процессам ЖЦ, что не позволяет четко сформулировать обязанности группы тестирования на всех стадиях разработки ПС и определить трудоемкость подготовки и проведения тестирования на всех уровнях. Выделение задач тестирования, решаемых в рамках разных процессов ЖЦ, в единый процесс позволяет *заблаговременно* создать среду и определить ресурсы тестирования, установить объемы и сроки тестирования в рамках плана проекта ПС.

Общая модель процесса тестирования представлена на рисунке 7.5.

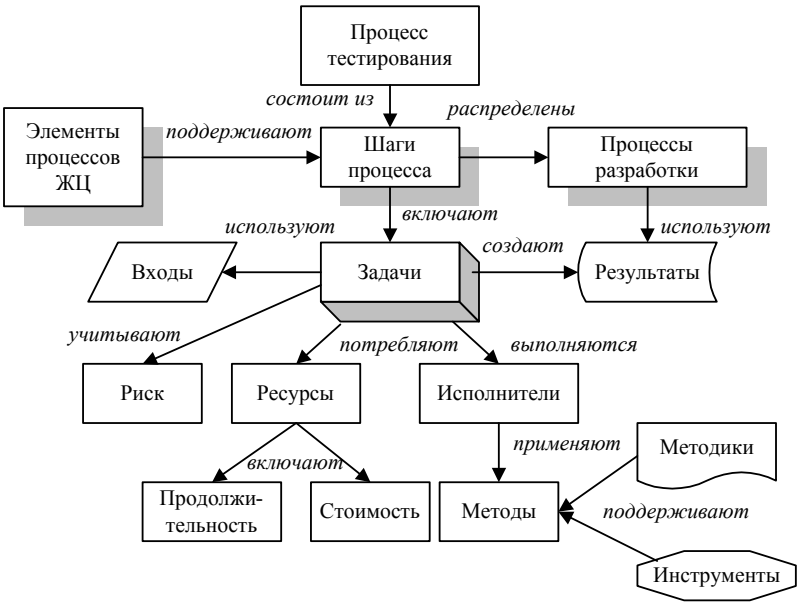


Рис. 7.5. **Модель процесса тестирования**

Цели тестирования выбираются и упорядочиваются на основе *анализа рисков* (в областях риска отказа ПС и срыва проекта ПС), что крайне важно при разработке ПС в условиях ограниченных ресурсов. Описание процесса представлено в виде последовательности 10 шагов процесса, но в зависимости от принятой модели ЖЦ эти шаги могут перекрываться, а отдельные задачи выполняться параллельно.

Шаги процесса охватывают три типичные стадии любого процесса: *планирование*, *выполнение* и *анализ результатов* (для *каждого уровня* тестирования). Выполняются циклически на всех уровнях тестирования и связаны с процессами разработки.

Каждый шаг процесса состоит из набора решаемых *задач*, имеет критерий начала и входные данные (входы). Результатом выполнения каждого шага (или задачи) является набор *рабочих продуктов процесса тестирования*. Описание структуры процесса, входы и результаты выполнения его задач представлены в таблице 7.15 и последующих пунктах раздела, а взаимосвязь рабочих продуктов тестирования – на рисунке 7.6.

Таблица 7.15. **Структура процесса тестирования**

Шаг процесса	Задача	Критерий начала и входные данные	Результат
1. Создание группы тестирования ***Цель:*** определить состав группы и распределить обязанности по выполнению задач тестирования	1.1. Определение сферы деятельности группы тестирования 1.2. Определение участников процесса тестирования 1.3. Распределение обязанностей и формирование плана работы группы тестирования	***Критерий начала***: наличие плана работ по проекту ***Входы:*** план и стратегия выпуска версий ПС, план работ по проекту, данные о квалификации и опыте членов группы тестирования	1. Утвержденные сфера деятельности и распределение ответственности за тестирование 2. План работы группы ***Критерий завершения:*** Сформированный и утвержденный план работы группы тестирования
2. Анализ риска ***Цель:*** своевременно выявлять проблемы, связанные с выполнением процесса тестирования, а также функции и компоненты ПС, которые должны быть протестированы более тщательно	2.1. Идентификация риска 2.2. Приоритезация риска 2.3. Распределение ресурсов 2.4. Создание базы данных по оценкам риска	***Критерий начала:*** наличие согласованных и упорядоченных по критичности требований к ПС, разработанный проект архитектуры ПС ***Входы:*** возможные факторы риска проекта, план/стратегия выпуска версий, требования к системе, возможные типы отказов	1. Список возможных рисков 2. Результаты анализа рисков, виды отказов, упорядоченные по критичности 3. Основанный на риске план распределения ресурсов на тестирование 4. База данных по оценкам риска
3. Определение целей тестирования ***Цель:*** выработать общую стратегию тестирования исходя из анализа риска.	3.1. Идентификация целей тестирования 3.2. Определение критериев прохода тестов 3.3. Упорядочение целей тестирования	***Критерий начала:*** наличие согласованных и упорядоченных по критичности требований к ПС. ***Входы:*** спецификация требований, результаты анализа рисков, прототип (при наличии) или первая версия ПС.	1. Перечень целей тестирования. 2. Перечень объективных критериев прохода тестов. 3. Перечень целей тестирования, упорядоченный по приоритетам.

Шаг процесса	Задача	Критерий начала и входные данные	Результат
4. Разработка плана тестирования ***Цель:*** установить объемы, ресурсы, сроки и исполнителей для тестирования объектов на каждом уровне тестирования и отразить их в плане тестирования.	4.1. Разработка плана приемочных испытаний системы 4.2. Разработка плана тестирования системы 4.3. Разработка плана тестирования ПО 4.4. Разработка плана интеграционного тестирования 4.5. Разработка плана автономного тестирования 4.6. Разработка плана регрессионного тестирования	***Критерий начала:*** наличие плана работ по проекту, готовность соответствующих рабочих продуктов процессов разработки, являющихся исходными для построения конкретного плана тестирования. ***Входы***. Определяются для каждого уровня	Планы для каждого уровня тестирования
5. Разработка тестов ***Цель:*** разработать детальные планы тестирования и подготовить тестовые данные.	5.1. Определение подходов к разработке тестов 5.2. Проектирование и разработка тестов 5.3. Подготовка тестовых данных 5.4. Подготовка журналов по тестированию 5.5. Проверка тестовых документов	***Критерий начала:*** разработанные планы тестирования. ***Входы:*** планы тестирования	1. Выбранные методы разработки тестов по уровням тестирования 2. Проекты тестов, описания наборов тестов, процедуры, описание сценариев тестирования 3. Реальные наборы данных в файлах, в БД или вводимые вручную 4. Шаблоны журналов для каждого уровня тестирования 5. Список исправлений, изменений и дополнений, необходимых для уточнения тестов

Шаг процесса	Задача	Критерий начала и входные данные	Результат
6. Автономное и интеграционное тестирование *Цель:* выполнить тестирование, зафиксировать результаты	6.1. Автономное тестирование 6.2. Повторное тестирование после устранения дефектов 6.3. Анализ результатов автономного тестирования 6.4. Интеграционное тестирование 6.5. Повторное тестирование после устранения дефектов 6.6. Анализ результатов интеграционного тестирования	***Критерий начала:*** наличие объекта тестирования ***Входы:*** планы, тестовые документы и тестовые данные.	1. Заполненные журналы 2. Отчет о выполнении тестирования
7. Тестирование ПО ***Цель:*** выполнить функциональное тестирование выпускаемой версии ПО согласно планам и зафиксировать результаты	7.1. Утверждение среды тестирования 7.2. Утверждение ресурсов тестирования 7.3. Тестирование ПО 7.4. Повторное тестирование после устранения дефектов 7.5. Анализ результатов тестирования 7.6. Разработка комплекта регрессионных тестов 7.7. Автоматизация повторного тестирования	***Критерий начала:*** передача на тестирование базовой версии ПО. ***Входы:*** план тестирования ПО, проекты тестов, описания наборов тестов, процедуры и сценарии тестирования, журналы тестирования, базовая версия ПО, документация на ПО.	1. Заполненный журнал, отчеты о проблемах, промежуточные отчеты 2. Отчет о выполнении цикла тестирования 3. Комплект регрессионных тестов 4. Тестовые процедуры функционального тестирования ***Критерий завершения шага:*** Достижение установленного критерия завершения тестирования
8. Системное тестирование ***Цель:*** выполнить тестирование *нефункциональных* характеристик	8.1. Утверждение среды тестирования 8.2. Утверждение ресурсов тестирования 8.3. Системное тестирование	***Критерий начала:*** передача на тестирование базовой версии ПО. ***Входы:*** план системного тестирования, проекты и описания	1. Утвержденный протокол среды тестирования 2. Утвержденный протокол ресурсов тестирования 3. Заполненный журнал, отчеты о

Шаг процесса	Задача	Критерий начала и входные данные	Результат
выпускаемой версии системы и зафиксировать результаты	8.4. Повторное тестирование после устранения дефектов 8.5. Анализ результатов тестирования 8.6. Тестирование инсталляции 8.7. Автоматизация тестирования	наборов тестов, сценарии и процедуры тестирования, архитектура системы, внешние интерфейсы системы, базовая версия ПО, эксплуатационная документация.	проблемах, промежуточные отчеты 4. Отчеты о проблемах, промежуточные отчеты 5. Промежуточные отчеты, отчет о выполнении цикла тестирования. 6. Отчеты о проблемах 7. Тестовые скрипты для системного тестирования
9. Анализ результатов тестирования ***Цель***: определить степень готовности протестированной версии ПС для предъявления на «формальное тестирование» (испытания) или выпуска	9.1. Анализ данных о результатах тестирования 9.2. Подготовка решений и рекомендаций по результатам тестирования 9.3. Подготовка итогового отчета о результатах тестирования 9.4. Проверка решений и отчета	***Критерий начала:*** Завершение всех видов тестирования ***Входы*:** журналы функционального и системного тестирования, отчеты о проблемах, недельные отчеты о результатах функционального и системного тестирования, спецификация функциональных и нефункциональных требований к ПС.	1. Собранные и классифицированные данные 2. Решения и рекомендации 3. Итоговый отчет о результатах тестирования 4. Результаты проверки
10. Регрессионное тестирование ***Цель***: проверить, что все расширения, изменения и настройки не затрагивают функциональные возможности ПС, и что изменения сделаны правильно.	10.1. Выполнение регрессионного тестирования 10.2. Повторное тестирование после исправления дефектов 10.3. Анализ результатов регрессионного тестирования 10.4. Подготовка итогового отчета о повторном тестировании	***Критерий начала***: Регрессионное тестирование выполняется после внесения любых изменений и расширений в следующей версии ПС .	1. Заполненный журнал регрессионного тестирования 2. Отчеты о проблемах 3. Промежуточные отчеты 4. Итоговый отчет

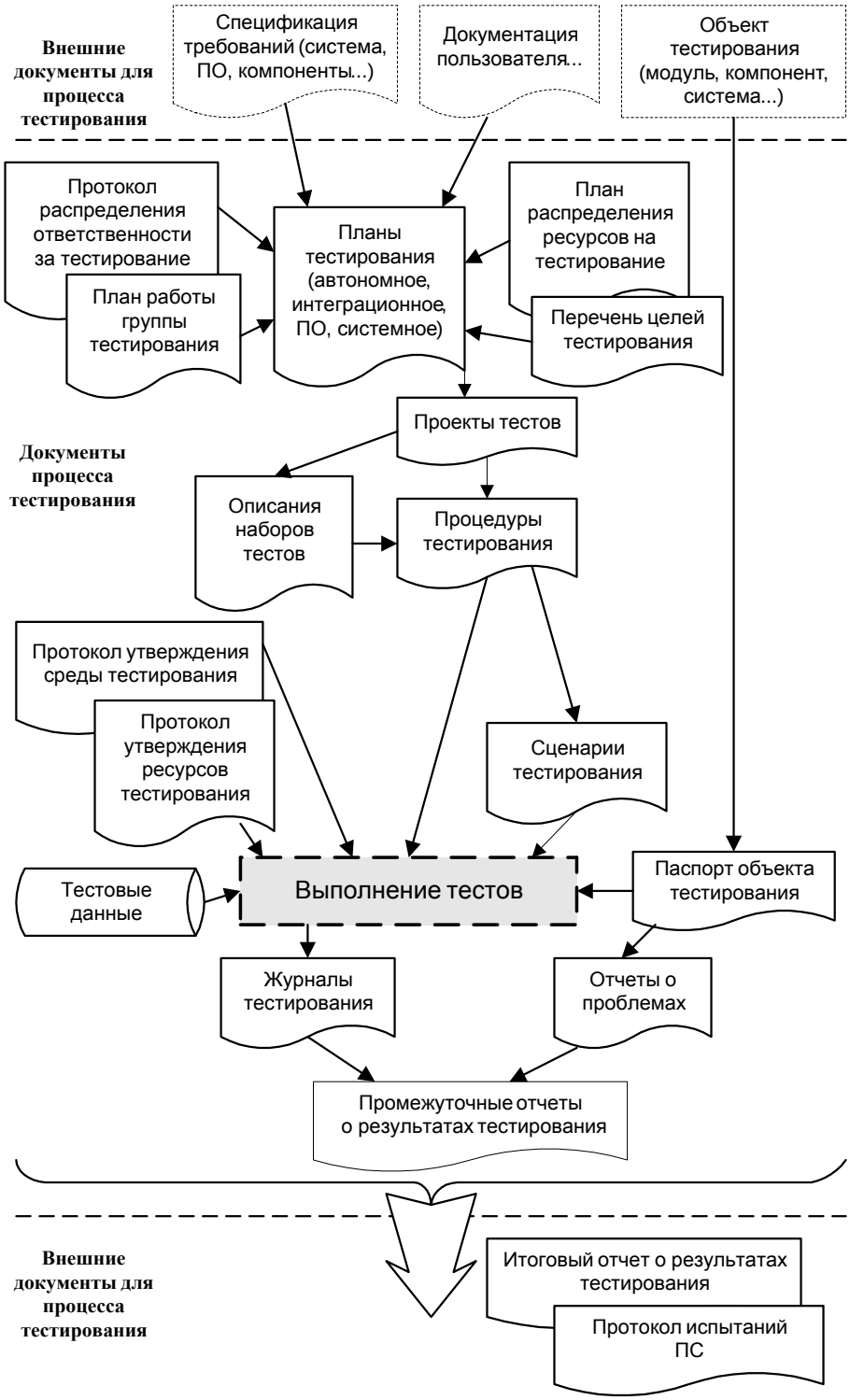


Рис.7.6. **Взаимосвязь рабочих продуктов процесса тестирования**

В таблице 7.16 представлено примерное распределение обязанностей в рамках шагов процесса.

Таблица 7.16. **Распределение обязанностей по выполнению процесса тестирования**

Шаг процесса тестирования	Исполнитель
Создание группы тестирования	Руководитель проекта (менеджер)
Анализ риска	Руководитель проекта (менеджер), руководитель группы тестирования, аналитики проекта, группа качества
Определение целей тестирования	Руководитель группы тестирования, тестировщики, группа качества
Разработка планов тестирования	Тестировщики
Разработка тестов	Разработчики, тестировщики
Автономное и интеграционное тестирование	Разработчики, тестировщики
Тестирование ПО	Тестировщики
Системное тестирование	Тестировщики
Анализ результатов тестирования	Руководитель группы тестирования, тестировщики, группа качества
Регрессионное тестирование	Тестировщики

Ниже кратко рассмотрены основные задачи, решаемые в рамках процесса тестирования

7.5.2. Создание группы тестирования

Создание группы тестирования - первый, но очень важный шаг в управлении процессом тестирования, на котором определяется ее уровень и подчиненность, порядок взаимодействия с разработчиками и заказчиками. Решаемые в этом шаге задачи и вопросы кратко описаны ниже петитом.

Задача 1.1. Определение сферы деятельности группы тестирования.

Уровень группы:

- *организация*. Участвует в тестировании всех проектов организации и подчиняется руководителю организации;
- *отдел (подразделение).* Участвует в тестировании проектов, разрабатываемых подразделением, и подчиняется руководителю отдела;
- *проект.* Участвует в тестировании определенного проекта - внутренняя группа, входящая в группу разработки и подчиняющаяся руководителю (менеджеру) проекта.

Уровни тестирования: автономное, интеграционное, тестирование ПО, системное тестирование. Ответственности на всех уровнях.

Потребность в инструментах тестирования. Их выбор, приобретение и освоение, или разработка собственных инструментов.

Задача 1.2. Определение участников процесса тестирования.

Определить круг лиц, которые будут принимать участие в процессе тестирования. Состав и количество участников процесса тестирования определяется имеющимися ресурсами (трудовыми и материальными), объемом, сложностью и критичностью проекта.

В процессе тестирования прямо или косвенно участвуют:

- представители заказчика и пользователи;

- руководитель проекта;
- группа качества;
- группа тестирования;
- разработчики (аналитики и программисты);
- администратор базы данных.

Пользователи привлекаются к процессу тестирования для:

- согласования серьезности отказов и дефектов;
- регистрации отказов и дефектов при опытной эксплуатации;
- участия в подготовке планов приемочного тестирования;
- оценки удобства применения и адекватности поддержки решения задач в ПрО.

Задача 1.3. Распределение обязанностей и формирование плана работы группы тестирования.

Примерное распределение обязанностей в рамках шагов процесса уже было представлено в таблице 7.16. Распределение задач тестирования между членами группы определяется численным составом группы. Например, в крупных проектах разработку планов, сценариев, проектов тестов осуществляет аналитик («архитектор тестов»), а выполнение тестов – рядовые тестировщики. В небольших проектах всю работу может выполнять один человек.

После создания группы тестирования и распределения обязанностей, разрабатывается план работы группы тестирования, в котором отражаются перечень задач тестирования, которые должны быть выполнены, ответственные исполнители за каждую задачу и ориентировочные сроки начала и завершения (согласованные с планом работ по проекту).

Требования к квалификации тестировщиков. Во многих организациях на роль тестировщиков приглашаются специалисты с более низкой квалификацией, чем программисты. Однако тестирование требует от человека как широкого кругозора, так и определенных деловых качеств. Вот, например, перечень качеств, которыми должен обладать идеальный тестировщик (по материалам сайта www.it-online.ru/library):

- умение разрушать программные продукты, не чувствуя при этом никаких угрызений совести;
- умение разрабатывать и выполнять сценарии и процедуры тестирования;
- умение описывать последовательность событий и конфигурацию системы, которые приводят к возникновению проблемы (например, четко документировать процедуры и результаты, умение устно передавать информацию разработчикам, другим тестировщикам и руководству);
- способность критиковать и корректно воспринимать критику;
- умение противостоять давлению (тестирование всегда является завершающей стадией любого процесса разработки, и, как правило, проходит в стрессовых условиях);
- способность быстро осваивать новые технологии;
- терпение. Нужно быть готовым выполнять прогоны тестов столько раз, сколько нужно для того, чтобы выявить проблему, после чего повторно выполнить тесты, чтобы убедиться в ее корректном устранении.
- гибкость мышления – способность быстро переключиться на тестирование нового программного продукта или даже отказаться от тестирования одного продукта в пользу другого, обладающего более высоким приоритетом.
- широта мышления - способность одновременно видеть общую картину и умение сосредоточиться на деталях.

- быть экспертом в нескольких областях – группе тестирования могут потребоваться специалисты по базам данных, по коммуникациям, по сетевым технологиям, по тестированию пользовательских интерфейсов, а также специалисты из других областей.

7.5.3. Анализ риска

Теме анализа и управления рисками проекта разработки ПС посвящена отдельная глава. Процесса тестирования, главным образом, касаются риски, связанные с техническими аспектами разработки, средой и технологией разработки, а также вопросы управления проектом. Результаты анализа риска используются для выработки общей стратегии тестирования, определения состояния тестирования путем отслеживания отказов и дефектов и определения критериев завершения задач тестирования. Решаемые в этом шаге процесса тестирования задачи и вопросы кратко описаны ниже петитом.

Задача 2.1. Идентификация риска.

Решение задачи заключается в рассмотрении и анализе функциональных и нефункциональных требований к ПС, характеристик ее размера и сложности, а также среды и условий использования, и выявлении факторов возможного риска для процесса тестирования и будущего функционирования ПС.

Помимо использования общих вопросов, предлагаемых в таксономии рисков (приложение 6), при идентификации риска группа тестирования должна учитывать **риски, связанные с качеством ПС**, в частности, по следующим атрибутам:

Функциональность.

- Какие функции наиболее критические (важные)[4]?
- Какие наиболее вероятные отказы могут быть связаны с выполнением этих функций?

Надежность.

- Определены ли типы и серьезность возможных отказов в ПС?
- Какие компоненты ПО могут вносить вклад в отказы системы? Могут ли отказы привести к потере и разрушению данных?
- Какие процедуры восстановления предусмотрены в проекте?
- Предусматривается ли обработка ошибок при вводе данных, ошибок пользователя, интерфейсов со средой и др?

Удобство применения.

- Установлены ли требования или критерии оценивания удобства интерфейса пользователя и процедур обслуживания?
- Согласованы ли требования к интерфейсу между компонентами ПО?
- Будет ли группа тестирования принимать участие в проверке документации?

Производительность.

- Как распределены требования к производительности в системе по уровням? На уровне администратора БД? На уровне сети? На уровне приложений?
- Какие функции ПО требуют высокой производительности?
- Какова относительная частота использования функций в приложениях?

Безопасность.

- Какие требования установлены к разграничению доступа к ПС?

---

[4] Классификация критичности функций приведена в главе 6.

Для выявления потенциальных **рисков, связанных с объемом и сложностью** тестирования, рассматриваются вопросы, касающиеся характеристик ПС и условий ее эксплуатации, например:

Сложность ПС.

- Какой предполагаемый размер и сложность ПС?
- Количество функций, входных (входных документов), их размер и др.?
- Какой объем выходных документов (результатов), их сложность?

Условия эксплуатации.

- Какое планируемое количество пользователей ПС? Будут ли они работать одновременно?
- Можно ли прогнозировать скорость роста БД?
- Известны ли любые «пиковые нагрузки» системы?

Характеристики среды.

- Типы и разрешающая способность мониторов, на которых должна работать ПС?
- Типы ОС и их версий, типы и версии общесистемного ПО?
- Какие системные драйверы будут использоваться?
- Какие системные возможности будут использоваться?
- Будут ли они работать в новой версии?

Кроме того, при анализе риска рассматриваются вопросы, касающиеся **эффективности процесса тестирования**, например:

Требования и спецификации системы и ПО.

- Существуют ли требования и подробные спецификации?
- Описаны ли все функции ПО, форматы экранов, интерфейсы системы, все входы и выходы системы? Доступна ли эта информация для группы тестирования?
- Участвует ли группа тестирования в анализе требований?

Модель разработки.

- Какая выбрана модель разработки?
- Если каскадная, то достаточны ли спецификации для планирования и разработки тестов (по V-образной модели)?
- Если модель итерационная, то есть ли прототип или первая версия ПС, чтобы начать планирование тестирования?

Проверки.

- Планируются ли внутренние действия по V&V: коллегиальные проверки, инспекции и др. для своевременного выявления дефектов проекта?

Сроки.

- Возможны ли сдвиги сроков передачи объектов на тестирование группе тестирования? Если да, то будут ли сдвигаться и сроки завершения тестирования?

Среда тестирования.

- Будет ли создана требуемая конфигурация для выполнения тестирования?

Процесс тестирования.

- Каким будет сам процесс тестирования?
- Кто будет принимать участие в процессе тестирования, их опыт и квалификация?

В результате рассмотрения вопросов, определяемых таксономией рисков, формируется *список рисков*, которые могут перерасти в проблемы и привести к срыву процесса тестирования и/или выпуску программного продукта с серьезными дефектами.

Следующий шаг в идентификации рисков – определить вероятность и серьезность каждой потенциальной проблемы для процесса тестирования. На рисунке 7.7 предложены две шкалы оценки.

Вероятность возникновения (1-99%)	Описание серьезности проблемы
Низкая (1%-25%)	**Критическая** – проблема требует немедленного решения, может привести к срыву процесса тестирования, серьезные дефекты в критических функциях не будут обнаружены.
Средняя (26% - 49%)	**Серьезная** – может привести к тому, что важные тесты не будут выполнены и серьезные дефекты не будут обнаружены.
Значительная (50% - 74%)	**Значительная** – может привести к задержке выполнения задач тестирования. Некоторые тесты не будут выполнены.
Высокая (75% - 99%)	**Незначительная** – может быть решена в ближайшее время.

Рис. 7.7. **Назначение распределения вероятности и серьезности проблемы**

Поскольку в условиях ограниченных ресурсов всеобъемлющее тестирование невозможно, главный интерес для процесса тестирования представляет идентификация ***рисков отказов*** системы и определение вклада в риск отказа отдельных функций и компонентов ПО. Методы анализа риска, которые могут использоваться при идентификации отказов для сложных систем, описаны в разделе 7.6.

Для оценки риска отказов применяется тот же подход, что и для оценки риска проекта, однако по серьезности классифицируются не проблемы, а отказы. Отнесение той или иной аномалии в работе ПС к отказу (за исключением очевидных), а также определение серьезности отказов, обусловлено типом и критичностью ПС.

Величина риска отказа ПС в данном процессе тестирования определяется как комбинация вероятности возникновения отказа и серьезности его последствий. Для назначения вероятности отказа используется та же шкала, что и для проблем (см. рисунок 7.7).

Вероятность отказа может ассоциироваться с:

Частотой использования функций. Чем чаще будут использоваться функции, тем больше вероятность, что пользователь «столкнется» с необнаруженными дефектами.

Частотой взаимодействия функции со средой. Чем больше функция взаимодействует со средой, тем больше вероятность возникновения отказа (ввод данных, манипулирование данными в базе данных, доступ к удаленным источникам данных, расположенными на сервере и др.).

Сложностью функции. Чем сложнее функция, тем больше дефектов может содержать код в начале тестирования. Подходы к оценке сложности функций подробно рассматриваются в главах 8 и 9. Хотя сложность функции не повышает вероятности отказа, а связана только с количеством возможных дефектов в реализации, это представляет потенциальный риск отказа.

Определение *серьезности* отказа. Для *каждой* функции рассматриваются возможные последствия ее отказа и оценивается их серьезность. Нужно учитывать, что выполнение даже редко используемых функций с серьезными последствиями отказа также сопряжено с риском. Любые отказы в критических функциях рассматриваются как серьезные. Величина риска отказа функции определяется по традиционной матрице рисков (глава 10).

Подобная оценка риска является не точной и должна уточняться при выполнении тестирования, но она все же позволяет выявить области потенциального риска отказа и более рационально использовать ресурсы тестирования.

Задача 2.2. Упорядочение рисков по приоритетам.

Решение задачи заключается в назначении приоритета каждому риску исходя из его величины. Как риски проекта, так и риски отказа упорядочиваются в порядке убывания величины риска.

Для учета рисков проекта (возможных проблем процесса тестирования) может использоваться форма, структура которой подобна представленной на рисунке 7.8.

Проект:		**Версия:**	**Дата анализа:**	
Идентификатор риска	**Описание потенциальной проблемы**	**Приоритет устранения**	**Ответственный за решение**	**Дата решения**

Рис. 7.8. **Форма описания риска проекта**

Для фиксации *рисков отказа* функций предлагается также использовать табличную форму, в которой указывать все внешние функции ПС, в порядке убывания величины их риска, наряду с описанием факторов риска, например, как на рисунке 7.9.

Проект: X		**Версия**: X.01	**Дата анализа**: 20.09.2006
Функция	**Наименование**	**Риск отказа**	**Описание риска**
Ф1	Первичный ввод информации в БД для формирования отчетов	высокий	Функция часто используется. Отказы могут привести к потере данных, невозможности выполнения других функций.
Ф2	Формирование недельного отчета	высокий	Отказы приведут к потере времени персонала.
Ф3	Ведение канцелярии	средний	Отказы не затрагивают других функций ПС.

Рис. 7.9. **Пример таблицы риска отказа внешних функций ПС**

Задача 2.3. Распределение ресурсов.

Результаты оценки риска применяются для распределения ресурсов при планировании тестирования. Наиболее распространенный подход к распределению – правило Парето (80/20), согласно которому 80% ресурсов должно выделяться на тестирование областей высокого и среднего риска, а 20% - низкого. Однако план выделения ресурсов должен регулярно пересматриваться, поскольку возможна недооценка величины вероятных рисков на ранних стадиях разработки.

Задача 2.4. Создание базы данных по оценке риска.

Результаты оценки рисков могут храниться в базе данных (или в таблице MS Excel) и использоваться, как для улучшения самого процесса оценки риска, так и для управления эффективностью процесса тестирования.

7.5.4. Определение целей тестирования

Цели тестирования - упорядоченное множество видов объектов тестирования, с каждым из которых ассоциируется критерий завершения тестирования. Задачи, решаемые для определения целей, представлены ниже петитом:

Задача 3.1. Идентификация целей тестирования.

При определении целей тестирования могут использоваться следующие подходы:

1. *Идентификация функций.* Составляется список всех внешних (пользовательских) функций ПС. Этот подход может использоваться для небольших приложений, а также в тех случаях, когда нет явно выраженных сценариев работы.
2. *Идентификация сценариев использования ПС.* Составляется список сценариев работы пользователей, в котором указывается последовательность и частота выполнения функций. Подход применяется для ПС с четко выявленными сценариями работы (системы

организационного управления, информационные системы автоматизации деловых процессов).

3. *Идентификация программных компонентов ПС,* которые будут тестироваться.

Полученный список дополняется целями тестирования нефункциональных характеристик, соотнесенных с компонентами ПС.

Идентификация целей производится на основе анализа соответствующих документов системы. Если разработка выполняется с помощью CASE-средств – из описания проекта в их среде.

Задача 3.2. Определение критериев прохождения тестов.

После построения списка целей нужно определить критерий принятия решения о достижении каждой из них (как узнать, что цель достигнута?). Критерии могут быть качественными и количественными. Качественные критерии формируются по ожидаемому результату выполнения тестов (проход/отказ). Количественные критерии основываются на метриках покрытия (например, все запланированные тесты прошли), количестве оставшихся дефектов и интенсивности отказов.

Задача 3.3. Приоритезация целей тестирования.

Решение задачи состоит в определении приоритетов выполнения тестов (*высокий, средний, низкий*), которые устанавливаются на основе анализа риска проекта и риска отказа.

7.5.5. Разработка плана тестирования

План тестирования определяется в стандарте IEEE Std. 829:1998 [11] как «документ, в котором определены объем, подходы, ресурсы и продолжительность тестирования. В нем указываются тестируемые объекты (элементы), характеристики, выполняемые задачи тестирования, ответственные исполнители по каждой задаче, а также риски, связанные с планом». Этот стандарт не регламентирует ни уровни, ни объекты тестирования на этих уровнях, используя общие термины «тестируемые элементы» применительно к любым объектам (модулям, компонентам, подсистемам или системе) и «характеристики» применительно к любым тестируемым характеристикам, и может использоваться для разработки планов на каждом уровне тестирования.

Планы могут уточняться по мере разработки системы (например, при уточнении требований или сроков), но должны быть готовы и проверены перед началом выполнения тестирования на соответствующем уровне.

Структура плана тестирования, рекомендуемая стандартом IEEE Std. 829, приведена ниже. Следует отметить, что общие задачи планирования испытаний перечислены и в ДСТУ 2853-94 [4], однако структура плана не представлена.

Идентификатор плана

Уникальный идентификатор, присваиваемый плану.

Введение

Указывают краткое описание объектов тестирования и тестируемые характеристики, а также уровень плана (автономное, интеграционное и пр.). Здесь же указывают ссылки на документы по проекту и используемые стандарты.

Тестируемые элементы

Перечисляют объекты тестирования с указанием их версии и уровня тестирования, характеристики носителей, содержащих объекты тестирования, требования к аппаратным средствам или требования к инсталляции объекта перед началом его тестирования. Указывают ссылки на необходимые документы (например, спецификации требований и проекта, руководство пользователя, руководство по инсталляции, руководство по обслуживанию), а также на любые отчеты о проблемах, касающиеся тестируемых объектов.

Тестируемые характеристики

Указывают все тестируемые характеристики и их комбинации. На следующем шаге процесса тестирования этот раздел дополняется перечнем проектов тестов, связанных с каждой характеристикой и каждой комбинацией характеристик.

Не тестируемые характеристики

Указывают, что *не будет* тестироваться и почему (например, переносимость системы).

Подход

Описывают общий подход к тестированию. Для каждой группы характеристик или их комбинации указывают подход, который должен гарантировать адекватность тестирования. Указывают основные методы, стратегии и инструменты, которые планируется применять для выполнения тестирования на соответствующем уровне.

Указывают минимально необходимую степень охвата элементов процессом тестирования, критерии, которые будут использованы для принятия решения об адекватности тестирования (функциональное покрытие, допустимое количество дефектов, частота отказов или время безотказного функционирования). Кроме того, указывают методы, применяемые для трассировки требований. Идентифицируют существенные ограничения, такие как степень готовности элемента к тестированию, ресурсы и сроки.

Критерии прохождения теста

Указывают критерии, которые будут использоваться для установления факта - прошел или не прошел тестируемый объект каждый тест.

Критерии приостановки и возобновления тестирования

Перечисляют причины, по которым могут быть приостановлены работы по тестированию, а также задачи, которые необходимо повторить при возобновлении тестирования.

Документирование результатов тестирования

Перечисляют документы, применяемые для документирования результатов тестирования. Рекомендуемый набор документов включает: план тестирования; проекты тестов; описания набора тестов; описания тестовых процедур; паспорта объектов, передаваемых на тестирование; журналы тестирования; отчеты о проблемах.

Задачи тестирования

Перечисляют задачи процесса тестирования, решаемые при подготовке и выполнении тестирования, зависимости между задачами, требования к квалификации исполнителей.

Среда тестирования

Указывают нужную аппаратно-программную конфигурацию и инструменты.

Обязанности

Ответственные исполнители по каждой задаче тестирования.

Вопросы кадров и их подготовки

Указывают требования к количественному составу участников процесса тестирования и уровню квалификации специалистов, а также необходимость в их обучении.

Календарный план

Указывают ориентировочные даты начала и завершения задач процесса тестирования, а также то, какие ресурсы и когда должны быть выделены.

Риск и непредвиденные обстоятельства

Указываются возможные риски, связанные с выполнением задач процесса тестирования, и предпринимаемые меры по их уменьшению.

Утверждения

Указывают фамилии и должности лиц, обязанных утвердить план, и резервируют место для их подписей и дат утверждения.

Задачи данного шага выполняются на *каждом* уровне тестирования. Они описаны ниже петитом.

Задача 4.1. Разработка плана приемочных испытаний системы.

Согласно стандарту ISO/IEC 12207 планирование приемочных испытаний не связано с процессом тестирования, но группа тестирования может привлекаться для их подготовки и проведения.

Этот план должен отражать интересы разных пользователей системы: конечных пользователей, администраторов системы (сети, БД), персонала сопровождения (развития).

Задача 4.2. Разработка плана тестирования системы.

Этот план должен охватывать все задачи тестирования и разрабатываться на основе требований к системе. В нем перечисляются все уровни и виды тестов, и указываются ссылки на соответствующие планы.

В плане должны быть указаны все объекты тестирования:

- все внешние функции и компоненты ПО, упорядоченные по приоритетам тестирования;
- требуемые конфигурации аппаратных и программных платформ;
- требуемые интерфейсы с другими системами;
- документация потребителя;
- учебные материалы.

Все планируемые виды тестирования, в порядке приоритета важности:

- функциональное;
- надежности;
- производительности;
- конфигурации;
- безопасности;
- восстановления;
- регрессионное.

Для определения продолжительности и трудоемкости тестирования могут использоваться методы оценки размера и сложности, описанные в главах 8 и 9.

Задача 4.3. Разработка плана тестирования ПО.

Этот план определяет стратегию проверки функциональных характеристик ПО.

В плане должны быть отражены цели функционального тестирования ПО:

- все внешние функции, упорядоченные по приоритетам тестирования;
- описание интерфейса пользователя (форматы экранов, меню и т.д.);
- характеристики производительности, распределенные по компонентам ПО;
- форматы всех типов данных и результатов (например, все отчеты);
- документация пользователя для проверки ее соответствия ПО;
- все сообщения об ошибках, формируемые ПО;
- возможные области неустойчивого функционирования (граничные значения);
- все таблицы БД.

Задача 4.4. Разработка плана интеграционного тестирования.

Этот план разрабатывается на основе плана интеграции ПО и должен отражать перечень компонентов ПО, порядок интеграции и сроки выполнения тестирования.

Планирование тестирования включает следующие действия:

- идентификация интерфейсов между компонентами ПО и порядка интеграции;
- идентификация требуемых интерфейсов компонентов ПО с другими компонентами системы (повторно-используемыми компонентами, средой, оборудованием);
- идентификация интерфейсов компонентов ПО с пользователем. Если компоненты имеют эти интерфейсы (графические интерфейсы), то в план должно быть включено тестирование их согласованности, а также представления на разных типах мониторов и при разной разрешающей способности экрана.

Задача 4.5. Разработка плана автономного тестирования.

Планирование автономного тестирования включает следующие действия:

- идентификация функций ПО, реализуемых в компоненте ПО;
- идентификация модулей, связанных с реализацией критических функций;
- идентификация сложных и подверженных отказам модулей;
- определение областей возможного риска отказа модулей;
- определение источников и форматов входных и выходных данных (тестовые данные, генераторы данных);
- определение граничных значений;
- определение требований к полноте тестирования (по метрикам структурного и функционального покрытия);
- определение критерия завершения.

Этот план может быть разработан в виде проекта тестов.

Задача 4.6. Разработка плана регрессионного тестирования.

План разрабатывается на основе плана интеграции ПО и должен включать:

- перечень внесенных изменений;
- перечень тестов, которые должны быть выполнены повторно;
- тесты для проверки совместимости внесенных изменений с существующими требованиями (новые тесты);
- компоненты системы, которые должны тестироваться повторно (характеристики, функции, интерфейсы, компоненты, в которых были обнаружены и устранены ошибки).

В набор регрессионных тестов нужно отбирать небольшое количество ранее выполненных тестов: тесты проверки последних изменений в ПО, тесты, при выполнении которых были выявлены серьезные дефекты, тесты сложных функций и граничных условий.

Планы должны подвергаться проверке с целью выяснения их полноты и адекватности общему плану разработки системы, плану качества, исходным требованиям на разработку ПС. Утвержденные планы передаются группе управления конфигурацией (кроме плана автономного тестирования) для контроля внесения изменений.

7.5.6. Разработка тестов

Шаг выполняется для *каждого* уровня тестирования и в соответствии с планом тестирования на определенном уровне. Решаемые задачи перечислены ниже петитом.

Задача 5.1. Определение подходов к разработке тестов.

Решение задачи заключается в выборе подходов/методов разработки тестов для каждого уровня тестирования и выборе стратегии их выполнения.

Определение общего подхода. Идентифицировать области риска отказа, которые будут отслеживаться при тестировании (таблица риска отказов внешних функций). В таблице 7.17 перечислены основные подходы/методы разработки тестов с указанием уровней, на которых их целесообразно применять.

Таблица 7.17. **Применение подходов/методов на уровнях тестирования**

Метод/подход	Автономное	Интеграционное	ПО	Системное
Методы структурного тестирования	+	+		
Эвристические			+	+
Таблицы решений	+	+		
Функциональные диаграммы	+	+	+	
Эквивалентное разбиение	+	+	+	
Анализ граничных значений	+	+	+	
Разбиение на категории	+	+	+	

Метод/подход	Автономное	Интеграционное	ПО	Системное
Тестирование переходов состояний	+	+	+	+
Тестирование по спецификациям	+	+	+	
Предположение об ошибке	+	+	+	
Мутационное тестирование	+			
Случайное тестирование			+	+
Операционный профиль				+
Тестирование по сценариям			+	+

Определение требований к полноте охвата функций. Для каждой функции установить требования к полноте охвата набором тестов в зависимости от ее вклада в риск отказа.

Задача 5.2. Проектирование и разработка тестов.

Заключается в разработке проектов тестов, описаний наборов тестов, процедур и сценариев тестирования, в соответствии с выбранными стратегиями.

В ***проекте тестов*** уточняется подход/метод тестирования, определяется перечень функций, тестируемых этим методом, и указываются *наборы тестов* и *процедуры*, по которым будет выполняться тестирование функций и их комбинаций. Проект тестов может быть разработан для одного независимого компонента ПО (например, генератора отчетов, интерфейса пользователя) или для группы связанных тестов.

Структура этого документа, рекомендуемая IEEE Std. 829, приведена ниже.

Идентификатор проекта тестов

Уникальный идентификатор проекта тестов

Тестируемые характеристики

Указывают элементы и описывают характеристики и комбинации характеристик, охватываемые данным проектом. Для каждой характеристики (или комбинации) указывают ссылки на связанные с ней требования в спецификации требований или описании проекта.

Уточненный подход

Перечисляют используемые методы тестирования и анализа результатов выполнения тестов (например, применение утилит сравнения файлов, визуальная проверка), критерии подтверждения выполнения тестов. Указывают общие характеристики любых наборов тестов. Например, ограничения на допустимые входы для каждого набора тестов, условия среды, дополнительные процедурные требования и любые зависимости тестов.

Идентификация тестов

Перечисляют идентификаторы и краткое описание каждого набора тестов, связанного с данным проектом. Отдельные наборы тестов могут использоваться более чем в одном описании проекта. Перечисляют идентификаторы и краткое описание каждой процедуры, связанной с данным проектом.

Критерии прохождения тестов

Указывают критерии, используемые для определения факта, прошла ли конкретная характеристика или их комбинация тесты.

Описание ***набора тестов*** предназначено для определения *конкретного* набора тестов, связанных с проектом тестов. Структура этого документа, рекомендуемая стандартом IEEE Std. 829, приведена ниже.

Идентификатор теста

Уникальный идентификатор набора тестов

Тестируемые элементы

Перечисляют и кратко описывают проверяемые элементы и характеристики. Для каждого элемента указывают ссылки на соответствующую документацию.

Описание входов

Указывают все входы/условия, требуемые для выполнения набора тестов. Входы могут указываться значениями входных данных или именами (для таблиц, файлов транзакций). Указывают все используемые базы данных, файлы, сообщения, области памяти. Указывают любые требуемые зависимости между входами (например, по времени).

Описание результатов

Указывают все результаты и характеристики (например, время ответа), требуемые для тестируемого элемента. Для каждого результата или характеристики указывают точные значение (с допустимой погрешностью).

Условия среды

Указывают требования к характеристикам аппаратной конфигурации.

Указывают требования к системному и прикладному ПО, необходимому для выполнения наборов тестов (например, ОС, компиляторы, инструменты тестирования, СУБД и др.).

Прочее

Любые другие специальные требования или требования к квалификации персонала.

Особые процедурные требования

Указывают любые ограничения, касающиеся процедур выполнения наборов тестов (например, настройки, условия вмешательства оператора, правила останова тестов).

Зависимости между тестами

Перечисляют идентификаторы наборов тестов, которые должны выполняться перед данным набором тестов, и указывают причины.

Более простая структура описания набора тестов может быть представлена в виде таблицы MS Excel (рисунок 7.10).

Дата: _____ Автор_____
Идентификатор теста _____Тип _____
Описание _____
ПС/Версия/ Модуль/Объект _____
Цель тестирования: _____
Приоритет: _____
Предусловие выполнения:_____
Описание тестовых условий: _____
а. Правильные условия_____
б. Не правильные условия_____
Тестовые данные:
а. для правильных условий:_____
б. для не правильных условий:_____
Ожидаемое поведение (результаты):
а. для правильных условий:_____
б. для не правильных условий:_____
Полученные результаты:
Отклонения:
Идентификатор проблемы: _____
Идентификатор тестовой процедуры: _____

Рис. 7.10. **Упрощенная структура описания набора тестов**

Стандартная структура больше подходит для формального тестирования (при проведении испытаний), для тестирования критических компонентов или тестирования в рамках процессов V&V. В остальных случаях – можно использовать для описания наборов тестов структуру, представленную на рисунке 7.10.

Процедура тестирования – это документ, содержащий подробные инструкции для выполнения наборов тестов. Структура этого документа, рекомендуемая стандартом IEEE Std. 829, приведена ниже, а на рисунке 7.11 представлена более простая структура описания процедуры тестирования.

Идентификатор процедуры тестирования

Уникальный идентификатор процедуры.

Назначение

Указывают назначение процедуры и ссылки на связанные с ней наборы тестов, а также ссылки на соответствующие разделы документации (например, на процедуры использования тестируемого элемента).

Специальные требования

Указывают перечень всех процедур, предшествующих данной, требования к квалификации исполнителей, требования к аппаратно-программной среде и инструментам.

Шаги выполнения процедуры

Перечисляют действия, необходимые для выполнения процедуры:

- *регистрация*. Указывают методы и форматы протоколирования результатов выполнения тестов, фиксации проблем и любых событий, касающихся выполнения процедуры;
- *установка/настройка*. Указывают действия, необходимые для подготовки к выполнению процедуры.
- *запуск*. Указывают действия, необходимые для начала выполнения процедуры.
- *выполнение*. Указывают действия, необходимые во время выполнения процедуры.
- *измерения*. Указывают методы выполнения измерений результатов работы теста (например, указывают, как измеряется время выполнения).
- *приостановка*. Указывают, как прервать выполнение теста.
- *возобновление*. Указывают любые предусмотренные точки возобновления работы процедуры и действия, требуемые для возобновления процедуры в каждой точке.
- *останов теста*. Указывают действия, нужные для завершения работы процедуры.
- *восстановление***.** Описывают действия, необходимые для восстановления среды.

непредвиденные события. Описывают действия, предпринимаемые при возникновении аномальных событий при выполнении процедуры.

Дата: _____Автор:_____
Идентификатор процедуры:_____ Тип _____
Описание:_____
Цель тестирования (тестовое требование):_____ Приоритет:_____
Файл спецификации _____
Идентификаторы тестов:
_____
_____
_____

Рис. 7.11. **Упрощенная структура процедуры тестирования**

Помимо документов, рекомендуемых стандартами, для объединения логически связанных процедур могут разрабатываться сценарии тестирования.

Предлагаемая структура описания ***сценария тестирования*** приведена ниже.

Идентификатор сценария:_____
ПС/Версия: _____
Общее описание:
Указывают, как использовать сценарий
Предусловия выполнения:
Указывают действия по настройке среды и подготовке к выполнению тестов (например, должны быть заполнены файлы или БД).
Постусловия выполнения:
Что необходимо делать после выполнения сценария (удалить тестовые записи и др.).
Выполнение:
Указывают пошаговые инструкции для выполнения каждого набора тестов с отметками о результатах.
Время:
Указывают ожидаемое время выполнения сценария.
Примечания:
Указывают любые расхождения между полученными и ожидаемыми результатами, а также описывают любые проблемы, возникшие при выполнении тестов.

Задача 5.3. Подготовка тестовых данных.

Заключается в подготовке наборов данных (например, в файлах, БД), необходимых для выполнения тестов. Тестовые данные должны быть максимально приближены к реальным и включать допустимые, граничные и недопустимые значения.

Задача 5.4. Подготовка журналов по тестированию.

Журналы предназначены для регистрации выполняемых тестов. Заполнение журналов осуществляется по ходу тестирования.

Структура **журнала тестирования**, рекомендуемая стандартом IEEE Std. 829, представлена ниже. При желании, можно использовать структуру журнала, предлагаемую стандартом ДСТУ 2851 [53].

Идентификатор журнала
Уникальный идентификатор журнала
Общие сведения
Перечисляют тестируемые элементы с указанием их версии и ссылки на паспорт объекта
Указывают условия аппаратно-программной среды тестирования.
Регистрация работ и результатов
Указывают дату и время начала и завершения работы (запуска и завершения теста), а также фамилию оператора, регистрирующего события.
Описание выполнения
Указывают идентификаторы выполняемых процедур тестирования и ссылки на их описания. Перечисляют участников: тестировщиков, операторов, наблюдателей и их функции.
Результаты выполнения процедуры
Для каждого теста фиксируют видимые результаты (ожидаемые / не ожидаемые), а также место их размещения (экран, файл.). Отмечают успешно или не успешно выполнен тест.
Среда тестирования
Указывают любые условия среды, касающиеся выполнения теста.
Аномальные события
Для каждого события, появление которого не ожидалось, фиксируют предыдущее и последующее событие/условие. Фиксируют любые обстоятельства, касающиеся невозможности начать выполнение процедуры или ее завершить.
Идентификаторы отчетов о проблемах
Перечисляют идентификаторы отчетов о проблемах (инцидентах), при их возникновении.

Задача 5.5 Проверка тестовых документов.

Цель проверки - гарантировать, что все цели тестирования (тестируемые функции и области риска отказа) охвачены наборами тестов. Эта задача может также выполняться в рамках процесса верификации.

Для проверки может использоваться таблица покрытия (рисунок 7.12)

Функция		**Оценка риска**	**Идентификатор набора тестов**
Идентификатор	**Описание**		

Рис. 7.12. **Таблица покрытия функций наборами тестов**

При проверке можно использовать контрольный вопросник, включающий, например, такие вопросы:

- Охватывают ли наборы тестов входные данные с максимальными, минимальными и нормальными значениями?
- Включены ли в набор граничные и недопустимые данные/условия? Можно ли придумать другие неправильные входные условия, не охваченные тестами?
- Включены ли в набор тестов тесты проверки сообщений об ошибках?
- Включены ли тесты для обнаружения возможных ошибок?

В инструментах управления тестированием с каждой функцией, указанной в иерархии целей, можно связать набор тестов и процедуры.

7.5.7. Автономное и интеграционное тестирование

Задачи, решаемые на этом шаге тестирования, описаны ниже петитом.

Задача 6.1. Автономное тестирование.

Выполняется программистом параллельно с разработкой и отладкой программных компонентов, с использованием основных методов тестирования (функционального и структурного). Методами функционального тестирования программист должен гарантировать, что каждая функция компонента протестирована на допустимых, граничных и недопустимых данных. Методами структурного тестирования, он должен убедиться в том, что все операторы кода и условия протестированы не менее 1 раза, все вызовы и обращения к процедурам, модулям и функциям компонента протестированы не менее 1 раза. Методом предположения о дефектах могут быть разработаны дополнительные тесты для выявления ожидаемых ошибок.

Подробные рекомендации по выполнению автономного тестирования приведены в стандарте ANSI/IEEE Std. 1008 [14].

Задача 6.2. Повторное тестирование после устранения дефектов.

Выполняется для проверки того, что исправления сделаны правильно, и что новые ошибки не внесены.

На уровне автономного тестирования отказы и дефекты не фиксируются в форме отчетов о проблемах, а результаты исправления отмечаются в журнале (в таблице Excel).

Задача 6.3. Анализ результатов автономного тестирования.

Для определения подверженных дефектам модулей и уточнения оценок риска отказа могут использоваться метрики подсчета дефектов и распределения дефектов по модулям (профили дефектов). Для таких модулей разрабатываются дополнительные тесты.

По результатам автономного тестирования разработчик готовит *отчет*, в котором указываются:

- достигнутая полнота по метрикам структурного и функционального покрытия;
- состояние всех протестированных модулей с указанием оставшихся дефектов;
- недостаточно протестированные модули и области (данные и условия среды) и причины, по которым они не были протестированы (нехватка времени, низкий риск отказа);

- возможные риски отказа, связанные с оставшимися дефектами;
- реальное время и трудоемкость тестирования.

В случае преждевременного завершения процесса из-за недостатка времени или обнаружения дефектов, требующих изменений в проекте, необходимо отразить эту информацию в отчете с указанием всех не устраненных дефектов.

*З*адачи 6.1- 6.3 выполняются циклически (кроме подготовки отчета) до достижения установленного критерия завершения тестирования. Минимальные цели, которые должны быть достигнуты при автономном тестировании:

- 100% покрытие операторов и не менее 85% условий;
- все запланированные функциональные тесты прошли (все области данных, подверженные отказам охвачены);
- нет не устраненных серьезных дефектов;
- общая устойчивая работа компонента в среде разработки.

Задача 6.4. Интеграционное тестирование.

Интеграция и тестирование может выполняться программистами или группой тестирования, в зависимости от условий интеграции и «распространенности» проекта, но в любом случае интегрируемые компоненты должны находиться под контролем конфигурации.

Это означает, что все изменения в совместно используемых библиотеках и схемах БД, а также интерфейсах между компонентами должны выполняться согласованно, а обнаруженные отказы фиксироваться в журнале тестирования.

Задача 6.5. Повторное тестирование после устранения дефектов.

Задача выполняется циклически до достижения установленного критерия завершения тестирования (версия ПО собрана). Повторяются только те тесты интеграции, при выполнении которых были обнаружены проблемы.

Задача 6.6. Анализ результатов интеграционного тестирования.

Решение задачи заключается в том, чтобы определить степень готовности интегрированной версии ПО к функциональному тестированию.

По результатам интеграционного тестирования группа тестирования готовит отчет и паспорт объекта тестирования (версии ПО).

Структура ***паспорта протестированного объекта***, рекомендуемая IEEE Std. 829, представлена ниже. Паспорт объекта подтверждает его готовность к тестированию и служит сопроводительным документом этого объекта.

Идентификатор паспорта объекта

Уникальный идентификатор объекта

Передаваемые элементы

Перечисляют передаваемые на тестирование элементы с указанием их версии и ссылок на соответствующие документы по этим элементам и план тестирования. Указывают ответственных за передаваемые элементы.

Местонахождение элементов

Указывают носители, на которых размещены элементы, имена передаваемых файлов и библиотек, пути доступа.

Состояние элементов

Указывают состояние передаваемых элементов, любые отклонения от проектной документации и плана тестирования.

Подписи

Указывают подпись ответственного за передачу.

7.5.8. Тестирование программного обеспечения системы

С момента определения версии ПО как базовой, она отторгается от разработчиков и «замораживается». Внесение изменений в базовую версию должно выполняться только с целью исправления дефектов и на основании отчетов о проблемах, обнаруженных тестировщиками. При продолжительном тестировании наряду с отчетами о каждой проблеме должны составляться промежуточные отчеты о ходе тестирования. По завершении функционального тестирования готовится сводный отчет о выполненном цикле функционального тестирования.

Задачи, решаемые в этом шаге тестирования, описаны ниже петитом.

Задача 7.1. Утверждение среды тестирования.

Перед началом тестирования должна быть создана соответствующая аппаратная и программная среда, включающая, например, рабочие станции, сервер, нужные версии компонентов общесистемного ПО и т.д., а также установлены и конфигурированы инструменты тестирования. Необходимо проверить, что среда установлена и соответствует требуемой. Любые отклонения или задержки в установке должны фиксироваться и рассматриваться как возможный риск срыва процесса (например, некоторые тесты не будут выполнены). Возможная структура протокола приведена на рисунке 7.13.

Протокол утверждения среды тестирования

Дата: _____

Шифр (идентификатор) проекта: _____

Уровень и вид тестирования	Требуемая конфигурация среды (по плану)	Установленная конфигурация	Возможные последствия

Утверждено:

_____ Дата: _____

Рис. 7.13. **Структура протокола утверждения среды тестирования**

Задача 7.2. Утверждение ресурсов тестирования.

Необходимо проверить, что все требуемые ресурсы (трудовые, финансовые, временные) выделены в соответствии с планами и достаточны для проведения тестирования. Любые отклонения от плана, например, сдвиг сроков передачи версии ПО группе тестирования, нехватка тестировщиков, должны фиксироваться и рассматриваться как возможный риск срыва процесса. Возможная структура протокола приведена на рисунке 7.14.

Протокол утверждения ресурсов тестирования

Дата: _____

Шифр (идентификатор) проекта: _____

Уровень и вид тестирования	Требуемые ресурсы (по плану)	Фактические ресурсы	Возможные последствия

Утверждено:

_____ Дата: _____

Рис. 7.14. **Структура протокола утверждения ресурсов тестирования**

Задача 7.3. Тестирование ПО.

Группа тестирования использует проекты и описания наборов тестов, а также процедуры, основанные на упорядоченных по *критичности* сценариях функционального тестирования ПО.

Помимо функционального тестирования на этом уровне выполняется проверка документации на ПО (руководства пользователя) и справочной системы (Help) по критериям:

- простота использования и понятность. Определяется, насколько легко найти нужные сведения;
- достоверность. Выполняются все описанные в документации примеры и процедуры пользователя и сверяются ПО, документация и Help;
- полнота и точность. Определяется, все ли функции и действия описаны? Все ли гиперссылки указаны правильно?

Тестирование по документации помогает выявить дефекты в ПО.

Все результаты выполнения тестов протоколируются в журнале тестирования, а любые отклонения или отказы фиксируются в ***отчетах о проблемах*** (рисунок 7.15).

Отчет о проблеме

1. Идентификатор отчета:_____2. Идентификатор паспорта объекта_____
3. Идентификатор теста _____
4. Дата: _____5. Время:_____
6. Проект: _____
7. Программа/Модуль: _____8. Версия: _____
9. Среда тестирования: _____ 10. Аппаратная среда: _____
11. Уровень тестирования: _____ 12. Цикл:_____
13. Симптом: _____
14. Серьезность (1-4): _____
15. Частота появления: _____
16. Статус (открыто/закрыто): _____
17. Описание: _____
18. Составитель отчета _____19. Передан: _____
20. Комментарий:
21. Решение:_____
22. Приоритет устранения(1-5): _____
23. Рассмотрено:_____ Дата_____
24. Проверено_____ Дата:_____
25. Считать отложенным (да/нет):_____

Рис. 7.15. **Структура отчета о проблеме**

*Отчет о проблеме***.** Это основной документ, предназначенный для взаимодействия между разработчиками и группой тестирования. В нем фиксируются любые непредвиденные или некорректные результаты выполнения тестов, обнаруженные группой тестирования. Основной целью составления отчета является исправление обнаруженного дефекта, поэтому отчет должен составляться сразу после обнаружения проблемы, содержать четкое описание проблемы и способа ее воспроизведения.

Первые три поля предназначены для увязывания отчета с тестовыми документами, что нужно для анализа проблемы программистом.

Поля 4 и 5 (дата и время) служат для последующего отслеживания устранения проблемы (при анализе результатов эти данные используются для определения возраста дефектов, анализа тенденций и динамики обнаружения дефектов).

Следующие поля (6-10) используются для идентификации объекта и среды тестирования.

Поля 11-12 нужны для анализа процесса тестирования.

В поле 13 указывается краткое описание *проявления* проблемы (симптома), определяемого тестировщиком, а в поле 14 - серьезность проблемы (критическая, серьезная и др.). Классификация проблем по данным категориям представлена в разделе 6.5. В поле 15 указывается порядковый номер появления данной проблемы (при повторном ее обнаружении). В поле 16 указывается состояние проблемы (открыта или закрыта). В поле 17 указывается описание проблемы и способ ее воспроизведения.

В поле 18 указывается фамилия тестировщика (или другого составителя отчета, например, пользователя), а в 19 – фамилия программиста. В поле 20 программист может записать, как устранена проблема, почему не устранена или отложена, и с чем она связана.

Поле 21 используется для описания состояния проблемы, например, «Отклонить», «Рассматривается», «Не может быть исправлена», «Исправлена» и заполняется программистом (в отличие от поля 16, которое заполняется тестировщиком). В поле 22 указывается код приоритета устранения (классификация приведена в разделе 6.5). В поле 23 указывается подпись программиста, ответственного за устранение проблемы, а в поле 24 – тестировщика, проверившего устранение дефекта. Поле 25 заполняется тестировщиком, если он не согласен с приоритетом устранения «Отложить», или «Отклонить», установленным программистом или руководителем проекта.

Для отслеживания динамики выполнения тестирования используются сводные данные об обнаруженных дефектах в виде промежуточных отчетов группы тестирования (например, еженедельно). В результате анализа этих отчетов могут быть уточнены первоначальные оценки рисков отказа функций, определены недостаточно протестированные области ПО и для них разработаны и выполнены дополнительные тесты.

На рисунках 7.16 и 7.17 приведено два типа недельных отчетов (промежуточных) о результатах тестирования, которые составляются на основании отчетов о проблемах.

Отчет о динамике обнаружения дефектов

Дата:_____ **Тип отчета:** недельный

Идентификатор отчета о проблеме	**Дата обнаружения**	**Описание**	**Серьезность**

Всего обнаружено дефектов:_____
Из них:
Критических:_____
Серьезных:_____
Незначительных:_____
Продолжительность тестирования:_____

Рис.7.16. **Отчет о количестве обнаруженных дефектов**

Отчет о динамике устранения дефектов

Дата:_____ **Тип отчета:** недельный

Идентификатор отчета о проблеме	**Дата открытия**	**Описание**	**Приоритет устранения**	**Дата закрытия**

Всего обнаружено: _____
Всего устранено:_____
Открыто: _____

Рис. 7.17. **Отчет о динамике устранения дефектов**

Первый из отчетов - основной отчет тестировщика о результатах недельной работы. В нем указываются все обнаруженные за неделю проблемы, упорядоченные по серьезности.

Второй вид отчета отражает динамику устранения дефектов и составляется тестировщиком по данным о «закрытых» отчетах о проблемах.

Задача 7.4. Повторное тестирование после устранения дефектов.

Задача выполняется циклически до достижения заданного критерия завершения.

Задача 7.5. Анализ результатов тестирования.

После выполнения запланированных тестов производится анализ покрытия функций набором тестов и анализ оставшихся не устраненных дефектов. Определяется риск отказов из-за оставшихся дефектов и принимается решение о продолжении или завершении тестирования ПО. По завершении тестирования группа тестирования готовит отчет о выполнении цикла тестирования (рисунок 7.18).

Отчет о выполнении тестирования ПО

Дата составления отчета:_____

Проект/Программа:_____Версия:_____ Цикл:_____

Дата начала тестирования: _____ Уровень/Задача тестирования:_____

Серьезность дефектов	**Устранено**	**Отложено**	**Отклонено**
Критические			
Серьезные			
Средние			
Прочие			
Всего			

Продолжительность тестирования (дней):_____

Трудоемкость тестирования:_____

Отчет составил:_____

Рис. 7.18. **Отчет о выполнении цикла тестирования**

Задача 7.6. Проектирование комплекта регрессионных тестов для регрессионного тестирования.

Цель: определить наборы тестов, сценарии и процедуры, которые должны быть сохранены для регрессионного тестирования в следующей версии ПО.

Решение должно быть основано на функциональных требованиях к системе, результатах анализа риска и анализе дефектов, которые имели место при функциональном тестировании.

***Задача 7.7. Автоматизация наборов тестов для регрессионного тестирования** (необязательная).*

При автоматизированном тестировании процедуры создаются в виде тестовых скриптов (автоматизированных процедур). Эти скрипты могут быть сгенерированы автоматически путем «проигрывания» сценария работы или написаны вручную на языке скриптов инструмента тестирования. Тестовый скрипт должен храниться и обновляться при каждом новом выполнении повторного тестирования.

Связанные логикой работы отдельные процедуры объединяются в тестовые сценарии посредством создания процедур-оболочек (shell procedure).

7.5.9. Системное тестирование

На этом шаге тестирования решаются следующие задачи, представленные ниже петитом.

Задача 8.1. Утверждение среды тестирования. Аналогична задаче 7.1.

Создание среды и имитация реальных и предельных нагрузок на систему требует значительных ресурсов, поэтому для моделирования условий среды требуются инструменты тестирования.

Задача 8.2. Утверждение ресурсов тестирования. Аналогична задаче 7.2.

Задача 8.3. Системное тестирование.

Выполняется в соответствии с планом, проектами и описаниями наборов тестов, сценариями и процедурами тестирования, основанными на сценариях работы пользователей. Все результаты выполнения тестов фиксируются в журналах тестирования, а обнаруженные отказы - в отчетах о проблемах, и передаются разработчику для анализа и устранения.

Системные тесты должны быть разрушительными по характеру и нацелены на то, чтобы найти области, где система не соответствуют техническим характеристикам.

При тестировании на надежность и производительность, тестовые данные выбираются из входного пространства по частоте использования функций в сценариях (по методам, основанным на анализе использования). Эти виды тестирования требуют измерения интервалов времени выполнения и моментов возникновения отказов. Системные тесты должны охватывать наиболее критичные по производительности и надежности функции системы, особенно те, отказы в которых могут привести к серьезным последствиям.

Помимо системных тестов, на этом уровне должна проверяться эксплуатационная документации на систему - на соответствие функциям и целям, реализованным в системе.

Для отслеживания динамики выполнения тестирования, как и при тестировании ПО, формируются сводные данные об отказах в виде промежуточных отчетов (см. рисунки 7.16 и 7.17). В результате анализа этих отчетов могут быть уточнены первоначальные оценки рисков отказа, и для них разработаны и выполнены дополнительные тесты. Эти данные составляют основу для вычисления метрик надежности и прогнозирования надежности ПС при эксплуатации.

Задача 8.4. Повторное тестирование после устранения дефектов.

Эта задача выполняется циклически до достижения установленного критерия завершения, инициируется по результатам анализа отчетов о проблемах и еженедельных отчетов.

Задача 8.5. Анализ результатов тестирования. Аналогична задаче 7.5.

Для системного тестирования критерий завершения формируется по каждому виду целевого тестирования.

Задача 8.6. Тестирование инсталляции.

Цель - выполнить тестирование процедур инсталляции в моделируемой и реальной среде. Это тестирование должно выполняться на «чистой» машине (без установленных компонентов системы).

Задача 8.7. Автоматизация тестирования.

Для выполнения этой задачи *должны* применяться инструменты сбора статистики о временных показателях выполнения и моментах возникновения отказов (путем встраивания счетчиков выполнения). Кроме того, могут использоваться специальные промышленные инструменты тестирования (приложение 4).

7.5.10. Анализ результатов тестирования

Задачи, решаемые в этом шаге тестирования, представлены ниже петитом.

Задача 9.1. Анализ данных о результатах тестирования.

Решение задачи состоит в том, чтобы собрать и обобщить все данные о результатах функционального и системного тестирования, выполнить классификацию и анализ отказов и дефектов. Вычислить метрики тестирования. По результатам анализа определить степень достижения установленного критерия завершения тестирования.

Задача 9.2. Подготовка решений и рекомендаций по результатам тестирования.

Сформулировать рекомендации, касающиеся продукта и процесса тестирования, и пояснить преимущества, получаемые в случае принятия рекомендаций, оценить стоимость и ресурсы, требуемые для их осуществления.

Задача 9.3. Подготовка итогового отчета о результатах тестирования.

По результатам функционального и системного тестирования группа тестирования готовит итоговый отчет, отражающий степень достижения целей тестирования и включающий рекомендации, основанные на анализе результатов тестирования. Стандарты ДСТУ требуют также подготовки протокола испытаний [53].

Этот отчет должен давать характеристику всех выполненных действий по тестированию, и направляться руководителям проекта и представителям заказчика (пользователя).

Структура **итогового отчета**, рекомендуемая в [11], приведена ниже.

Идентификатор отчета

Введение

Приводят наименование выпускаемой версии системы и общую оценку результатов ее функционального и системного тестирования, ссылки на план (планы) тестирования, проекты тестов, описания наборов тестов и процедур, сценарии тестирования, журналы тестирования, отчеты о проблемах. Перечисляют протестированные объекты (компоненты системы, процедуры). Описывают программно-аппаратную среду, в которой проводилось тестирование.

Отклонения от планов тестирования

Указывают любые отмеченные отклонения от планов и их причины, не выполненные тесты, отклонения от проектных спецификаций, среды тестирования и др.

Оценка адекватности тестирования

Указывают степень соответствия процесса тестирования критериям, установленным в планах тестирования. Отмечают любые характеристики и их комбинации, которые были протестированы недостаточно, с указанием причины.

Выводы по результатам тестирования

Подводят общий итог тестирования. Указывают все устраненные дефекты. Перечисляют все не устраненные дефекты.

Оценивание

Дают общую оценку каждого протестированного объекта. Эта оценка должна основываться на результатах тестирования и критериях прохождения (не прохождения) тестов каждым объектом.

Заключение

Обобщают данные об использовании ресурсов, например, количество тестировщиков, стоимость и продолжительность тестирования (включая время на планирование и разработку тестов), использованное машинное время.

Указывают рекомендации, сделанные по результатам тестирования, и ожидаемые преимущества, а также стоимость и ресурсы, требуемые для их осуществления.

Утверждения

Указывают фамилии и должности лиц, обязанных утвердить или завизировать отчет, и резервируют место для их подписей и дат утверждения.

Задача 9.4. Проверка решений и отчета.

Выполняется в форме совместного просмотра отчета группой тестирования, разработки и представителями заказчика (при необходимости).

Примечание. Задачи проведения приемочных испытаний выполняются аналогично задачам шагов 8-9 (согласно планам), но в реальной среде эксплуатации и с участием заказчика (пользователя).

7.6. Методы анализа риска отказа программной системы

Для идентификации и оценки рисков отказов ПС и ее компонентов применяются формальные и неформальные методы анализа. Выбор того или иного метода определяется критичностью ПС относительно серьезности последствий отказов.

Наиболее известные формальные методы анализа риска отказа – *анализ дерева событий* (ETA - Event Tree analysis), *анализ дерева отказов* (FTA - Fault Tree analysis), *анализ режимов и последствий отказов* (FMEA - failure modes and effects analysis) [54].

Все три метода широко применяются в мировой практике в качестве инструментов анализа сложных технических и программно-технических систем (особенно, критических по безопасности) на стадиях анализа требований, проектирования и реализации, с целью определения возможных типов отказов, выработки проектных и технических решений для минимизации последствий отказов.

В последнее время появились аналоги этих методов, специально предназначенные для использования в программной инженерии. Так, аналог FTA - метод SFTA (software fault tree analysis), а аналог FMEA - SFMEA (software failure modes and effects analysis).

Основные методы и их аналоги для ПО могут применяться совместно для анализа отказов на уровне системы [40], поскольку многие отказы бывают вызваны *комбинацией* аппаратных отказов, программных отказов или ошибок пользователя. В этом случае дерево программных отказов представляет поддерево отказов системы, а построение структурно-логической схемы системы выполняется в два этапа:

- построение деревьев событий и отказов на уровне системы, элементами которой являются функциональные компоненты системы;
- построение деревьев отказов для каждого функционального компонента (а также для программных компонентов и отдельных модулей).

7.6.1. Анализ дерева событий

Метод анализа дерева событий предназначен для идентификации возможных сценариев использования системы, которые могут привести к ее отказам. В нем используется индуктивный подход для моделирования возможных событий в системе и их последствий.

Дерево событий представляет логическую схему системы, которая определяет при заданном исходном событии (корень дерева) множество возможных элементарных конечных событий (последствий), каждое из которых является реализацией определенных комбинаций промежуточных альтернативных событий, включая:

- успешное или неуспешное выполнение действий;
- работоспособные состояния или отказы компонентов системы;
- правильные или ошибочные действия пользователей.

В зависимости от вида промежуточных событий различают функциональные и системные деревья событий.

В функциональных деревьях промежуточные события - это успешное или не успешное *выполнение функций* системы. Построение этих деревьев делается для выявления возможных сценариев использования системы и выделения из них сценариев, при которых возможны отказы в выполнении функций.

В системных деревьях промежуточные события – это успешные *действия* или отказы системы и ее компонентов (а также ошибки пользователей, которые могут привести к отказам). Разработка этих деревьев проводится с целью детализации событий, полученных в результате построения функциональных деревьев событий, для определения сценариев возникновения отказов на уровне системы.

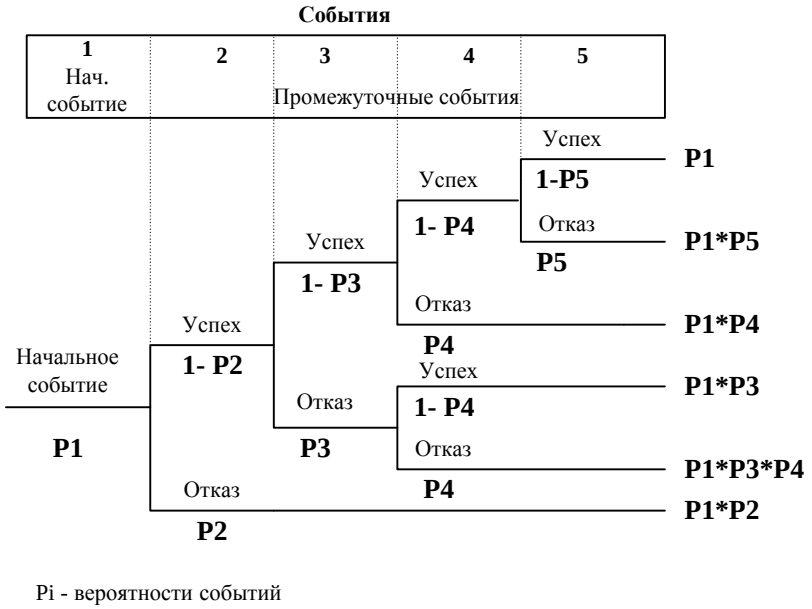
Затем, для каждого сценария возникновения отказа могут разрабатываться деревья отказов и ошибок пользователей (методами FTA и SFTA).

Порядок применения метода можно представить в виде последовательности следующих шагов:

Шаг 1. Определить границы анализируемой системы и границы анализа. Рассматриваются условия эксплуатации системы, факторы среды, важность, порядок и частота использования функций системы, ее необходимые интерфейсы с другими системами, требования к надежности и производительности.

Шаг 2. Идентифицировать возможные начальные события. Выполнить предварительный анализ системы для идентификации начальных событий или категорий событий. Начальное событие может представлять отказ компонента системы или некоторое внешнее событие (например, ошибочные действия пользователя, неправильные входные данные).

Шаг 3. Идентифицировать возможные последствия. Для каждого начального события идентифицировать его возможные последствия. Отобразить совокупность событий и последствий в виде дерева событий. Над деревом перечисляются события (начальное и промежуточные). Дерево событий представляет собой бинарный граф, вершина которого (корень дерева) – начальное событие. Две дуги, исходящие из корня дерева – представляют два возможных исхода события (успех или неуспех). Процедура повторяется до тех пор, пока не будет «достигнуто» элементарное событие, связанное с угрозой со стороны системы, которое далее не анализируется. Общая структура дерева представлена на рисунке 7.19.



Pi - вероятности событий

Рис. 7.19. **Общий вид дерева событий**

Шаг 4. Определить сценарии возникновения отказов (отрицательных событий). Для каждого начального события определить сценарии (цепочки распространения негативных последствий), которые могут привести к отказам (пути на дереве: от начального до конечного элементарного события).

Шаг 5. Проанализировать сценарии отказов. Для каждого конечного элементарного события на дереве событий определить частоту и последствия, которые с ним связаны. Назначить вероятности последствий событий (Pi) исходя из частоты. Для этого, возвращаясь от листьев дерева к его корню, вычислить вероятности каждого последствия события.

Шаг 6. Оценить потери из-за отказов.

Серьезность последствий отказов рассматривается в контексте ожидаемых потерь из-за отказов системы при ее эксплуатации.

Шаг 7. Принять решение о том, как снизить возможные потери в результате цепи отказов. Оценить ожидаемый эффект (пользу) направленного или дополнительного тестирования системы, характеризующийся, например, *экономическим* показателем - величиной снижения финансовых потерь заказчика.

7.6.2. Анализ дерева отказов

Метод анализа дерева отказов был разработан в начале 60-х годов, на протяжении многих лет применялся в различных инженерных дисциплинах и считается одним из основных методов анализа надежности и безопасности сложных и критических систем.

В отличие от дерева событий в данном методе используется дедуктивный анализ: от события отказа до событий, возможно послуживших его причиной.

Дерево отказов строится для каждого возможного отказа (или наиболее серьезных отказов). Начиная с исходного анализируемого события отказа (вершины дерева), то есть отказа системы или ее компонента (по существу, конечного события), для каждого рассматриваемого события выявляются события-предшественники, служащие непосредственными причинами его возникновения. Построение дерева заканчивается выявлением элементарных событий, которые далее не детализируются. Эти события не могут быть описаны математически, ввиду отсутствия информации о статистическом распределении событий в аналогичных системах или компонентах, и не могут быть промоделированы экспериментально. Графически дерево отказов изображается в форме графа (рисунок 7.20).

Примерами исходных событий для ПО могут быть: потеря данных и нарушение их целостности, отказы при выполнении компонентов (функциональные отказы), несанкционированный доступ к системе и данным и другое.

Примерами элементарных событий могут быть: аппаратные сбои, ошибки во входных данных, ошибки в программных модулях.

Порядок применения метода можно представить в виде последовательности следующих шагов.

Шаг 1. Определение границ анализируемой системы и уровня вложенности, до которого отказы будут рассматриваться. Это делается для упрощения структуры дерева и для построения иерархии деревьев (на уровне системы и на уровне компонентов).

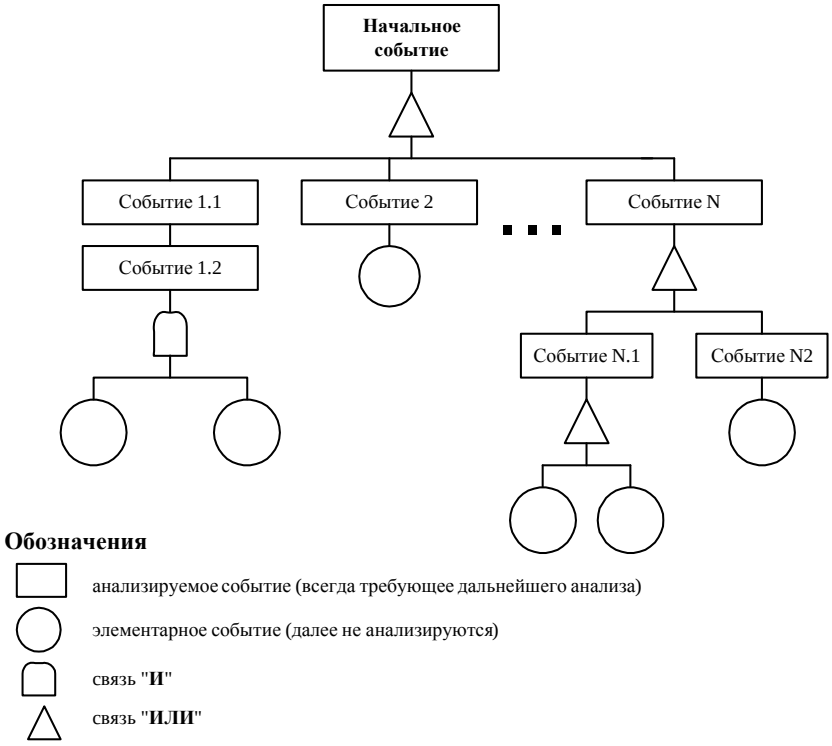


Рис. 7.20. **Общий вид дерева отказов**

Шаг 2. Идентификация начальных анализируемых событий системы. Эти события представляют возможные отказы, идентифицированные с помощью деревьев событий. Из всех возможных отказов выбираются наиболее важные, для каждого из которых будет строиться дерево отказов.

Шаг 3. Построение дерева отказов. Изначально предполагается, что идентифицированный отказ произошел, и затем применяется нисходящий анализ для выявления возможных причин его возникновения. Это делается с помощью образования ветвей дерева, корень которого – отказ. Дерево строится вплоть до элементарных событий, которые не будут дальше декомпозироваться. Для изображения событий и связей между ними применяются стандартные графические обозначения, основные из которых изображены на рисунке 7.20. Для более сложных деревьев применяются дополнительные графические символы [40].

Шаг 4. Выявление непосредственных причин возможных событий в системе. При определении этих причин важно не пропустить промежуточные события. При таком подходе дерево формируется систематически, начиная с начального события (корня), через промежуточные уровни, вплоть до элементарных событий (листьев), представляющих возможные причины отказа (и компоненты, отказ которых может произойти). Процесс определения причин продолжается до тех пор, пока все причины начальных событий не будут декомпозированы до элементарных событий.

Шаг 5. Для каждого события идентифицировать все непересекающиеся причины и определить минимальное их количество. Выполняется путем качественного (визуального) анализа дерева отказов.

Шаг 6. Назначить вероятностное распределение элементарных событий (причин). Количественный анализ основан на вычислении условной вероятности

возникновения отказа (начального события) при известных вероятностях элементарных событий. В качестве вероятностного показателя отказа может использоваться, например, вероятность невыполнения заданных функций или потери данных, частота или интенсивность возникновения элементарных событий.

Шаг 7. Вычисление условной вероятности отказа. Выполняется по правилам Булевской алгебры (сложения и умножения элементарных событий).

7.6.3. Анализ причин и последствий отказов

Метод анализа режимов и последствий отказов (FMEA – failure mode and effect analysis) и его расширение для ПО (SFMEA – software failure mode and effect analysis) - табличные методы, которые применяются в качестве вспомогательных для анализа и прогнозирования надежности технических и программно-технических систем и могут использоваться как вместе с описанными выше методами анализа деревьев событий и отказов, так и самостоятельно. Эти методы применяются для идентификации *типов возможных отказов* и механизмов их появления (режимов) в отдельных компонентах системы. Используют индуктивный анализ с целью определения условия возникновения отказов в компоненте (модуле) и последствий этих отказов для всей системы [54].

Для каждого компонента разрабатывается отдельная таблица, в которой описывается характер и степень влияния определенного режима отказа на систему. Реквизиты полей таблицы включают режим отказа и его причину, последствия отказа для системы, серьезность отказа, требуемые действия по изменению компонента или другие способы предотвращения отказа и контроля условий его появления (рисунок 7.21).

Компонент	**Режим отказа и причина**	**Последствия**	**Серьезность**	**Меры предотвращения**

Рис. 7.21. **Структура таблицы отказов**

7.7. Инструменты тестирования

При современных темпах роста сложности ПС, их тестирование должно поддерживаться автоматизированными инструментами. В последнее десятилетие рынок инструментов тестирования бурно развивается. На смену анализаторам кода и генераторам тестовых данных для структурного и функционального тестирования, работающим на определенных программных и аппаратных платформах, а также инструментам, поддерживающим отдельные методы тестирования, приходят промышленные интегрированные инструменты поддержки разных методов тестирования, работающие практически на всех аппаратных и программных платформах.

7.7.1. Классификация инструментов тестирования

Единых подходов к классификации инструментов тестирования нет, поскольку сейчас преобладают интегрированные инструменты. Достаточно общая схема классификации инструментов, применяемых при тестировании, приведена в работе [55], представлена на рисунке 7.22 и кратко описана ниже.

Инструменты управления тестированием
- Менеджеры конфигурации
- Менеджеры проекта

Инструменты, поддерживающие анализ требований и проекта
- Анализаторы планов, требований и проектов
- Построители прототипов/эмуляторы системы
- Инструменты трассировки требований
- Планировщики тестов

Инструменты поддержки тестирования на этапе реализации и сопровождения

Статические анализаторы исходного кода:
- Аудиторы
- Измерители сложности
- Инструменты генерации перекрестных ссылок
- Измерители размера

Инструменты подготовки тестов:
- Экстракторы данных
- Генераторы тестов по спецификации требований
- Генераторы тестовых данных (по разным методам)
- Планировщики тестов

Инструменты выполнения тестов:
- Динамические анализаторы покрытия
- Инструменты Захвата/Проигрывания
- Отладчики
- Эмуляторы
- Анализаторы сети
- Анализаторы производительности
- Анализаторы ошибок периода выполнения
- Инструменты управления выполнением тестов

Тестовые оценщики
- Компараторы (утилиты сравнения файлов)
- Конверторы и анализаторы данных
- Трассировщики дефектов/изменений

Рис.7.22. **Виды инструментов тестирования**

Инструменты управления тестированием

Менеджеры конфигурации – отслеживают и контролируют внесение изменений в ПС на протяжении ее разработки и сопровождения и поддерживают целостность разработанных и поставляемых версий ПС. Основные функциональные характеристики этого типа инструментов: отслеживание дефектов; управление запросами на внесение изменений, отслеживание изменений в коде.

Менеджеры проекта - поддерживают функции планирования и отслеживания на этапах разработки и сопровождения системы (оценивание сроков, ресурсов, построение сетевых графиков и др.). Инструменты данного типа можно применять и для поддержки планирования и отслеживания процесса тестирования, например, для распределения списка задач по исполнителям и отслеживания их выполнения.

Инструменты, поддерживающие анализ требований и проекта

Анализаторы планов, требований и проектов - оценивают спецификации на полноту, непротиворечивость и соответствие установленным стандартам для спецификаций.

Построители прототипов/эмуляторы системы – объединяют действия по тестированию с действиями по анализу и проектированию и позволяют быстро промоделировать требования и проектные решения.

Инструменты трассировки требований – позволяют установить связи требований с проектом, кодом и тестами.

Планировщики тестов – поддерживают процесс планирования тестирования на всех его уровнях.

Инструменты поддержки тестирования на этапе реализации и сопровождения

Статические анализаторы исходного кода. Исследуют исходный код без его реального выполнения. Могут применяться в качестве вспомогательных инструментов при планировании и тестировании, но в большей степени – это инструменты измерения и оценивания качества кода.

Аудиторы – анализируют код на соответствие стандартам кодирования и установленным правилам.

Измерители сложности – вычисляют метрики кода для определения атрибутов сложности путем оценивания таких характеристик кода, как поток управления, операнды/операторы, данные, структура системы.

Инструменты генерации перекрестных ссылок – обеспечивают ссылки между разными сущностями (переменными).

Измерители размера - вычисляют число строк исходного кода (SLOC).

Инструменты подготовки тестов. Включают группу инструментов, поддерживающих подготовку тестовых данных или разработку тестов.

Экстракторы данных – строят тесты, извлекая информацию из существующих баз данных или тестовых наборов.

Генераторы тестов по спецификации требований – строят тесты по требованиям, написанным на формальном языке спецификации.

Генераторы тестовых данных (по разным методам) – генерируют входные данные для тестирования в соответствии с методом, реализованным в инструменте. Некоторые генераторы создают данные с использованием датчиков случайных чисел (для статистического тестирования).

Планировщики тестов – поддерживают разработку и ведение планов тестирования.

Инструменты выполнения тестов. Наиболее представительная группа инструментов.

Динамические анализаторы покрытия – оценивают покрытие кода тестами по отношению к операторам, путям или модулям. Основаны на встраивании в код специальных операторов для отслеживания путей выполнения.

Инструменты Захвата/Проигрывания - автоматически записывают действия тестировщика при выполнении программы в виде автоматической тестовой процедуры (скрипта захвата) и затем воспроизводят (проигрывают) эти скрипты.

Отладчики – непосредственно поддерживают автономное тестирование, хотя их назначение – поиск и устранение ошибок (отладка). Некоторые отладчики имеют возможности анализа покрытия.

Эмуляторы – могут применяться вместо отсутствующих компонентов системы (обычно аппаратных, например эмуляторы терминалов).

Анализаторы сети – специальный класс инструментов тестирования, предназначенных для анализа трафика сети. Позволяют моделировать работу многих терминалов.

Анализаторы производительности – отслеживают временные характеристики системы или ее компонентов.

Анализаторы ошибок периода выполнения – отслеживают работу программы на наличие ошибок, связанных с заказом-освобождением памяти, нулевыми указателями и т.п. Автоматически отслеживают использование стеков и очередей.

Инструменты управления выполнением тестов – общее наименование категории инструментов, которые автоматизируют разные функции настройки и выполнения тестов, очистки системы после выполнения тестов. К этой категории относят также тестовые драйверы и заглушки.

Тестовые оценщики. К этому классу относят инструменты, выполняющие функции выявления подверженных ошибкам компонентов.

Компараторы (утилиты сравнения файлов) – сравнивают результаты тестирования и выявляют отклонения (например, сравнивают битовые образы экранов, файлы). Инструменты Захвата/Проигрывания обычно имеют в своем составе такие компоненты.

Конверторы и анализаторы данных – конвертируют данные в удобный для интерпретации формат и выполняют разные виды статистического анализа этих данных.

Трассировщики дефектов/изменений – хранят информацию о дефектах и генерируют отчеты. Обычно входят в состав систем управления конфигурацией и инструментов управления тестированием.

7.7.2. Выбор инструментов тестирования

Выбор набора инструментов тестирования для отечественных разработчиков зависит в первую очередь от их доступности, а также типа ПС и методологии разработки. Минимальный набор инструментов для подготовки и выполнения тестов представлен на рисунке 7.23.

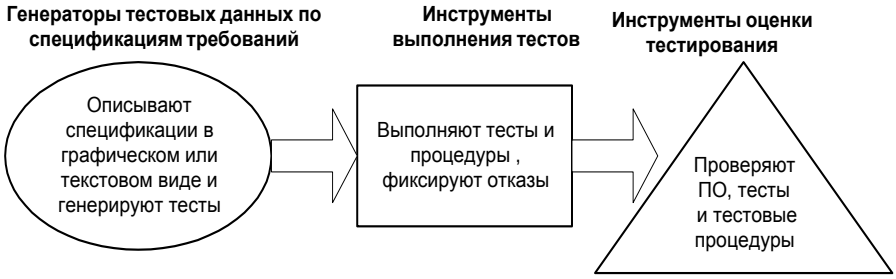


Рис.7.23. **Набор инструментов для поддержки выполнения тестирования**

В работе [56] приведены общие требования к набору инструментов, необходимых для тестирования клиент-серверных систем. Инструменты должны:

- поддерживать запись и проигрывание на уровне объектов (для графических интерфейсов пользователя);
- обеспечивать управление тестированием;

- иметь простой и удобный язык записи сценариев;
- поддерживать разные методы сравнения результатов тестирования;
- работать на всех требуемых платформах;
- поддерживать внешние языки (например, Visual Basic);
- поддерживать несколько методов тестирования;
- иметь систему контекстной помощи, удобную и полную документацию.

Литература к главе 7

1. *IEEE Std. 610.12:1990.* IEEE Standard Glossary of Software Engineering Terminology.
2. *Майерс Г.* Искусство тестирования программ.-М: Финансы и статистика, 1982. –196 с.
3. *ДСТУ 2844-94.* Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення.
4. *ДСТУ 2853-94.* Програмні засоби ЕОМ. Підготовлення і проведення випробувань.
5. *Gelperin D.*, Hetzel B. The Growth of Software Testing // Commun. ACM, 1988.-V. 31.- № 6.- P. 687-695.
6. *ISO/IEC 12207:1995*. Information technologies. Software life cycle processes. – 61p.
7. *Software* Engineering Body of Knowledge (SWEBOK). // ISO/IEC JTC1/SC7 N2517, Software & System Engineering Secretariat, Canada, 2004. – 202 p.
8. *Тестирование* программного обеспечения: Пер с англ. / Канер С., Фолк Д., Нгуен Е. // - К. Издательство «DiaSoft», 2000. – 544 с.
9. *Zhu H.*, May J.H.R. Software Unit Test Coverage and Adequacy // ACM Computing Surveys.- 1997.- V. 29.- Dec.- P. 366-427.
10. *Perry W.* Effective Methods for Software Testing, John Wiley & Sons.- 1995.
11. *IEEE Std 829:1998*. Standard for Software Test Documentation. - 52 p.
12. *IEEE Std. 1012:1998.* IEEE Standard for Software Verification and Validation.
13. *ДСТУ 2941-94.* Розроблення систем. Терміни та визначення.
14. *ANSI/IEEE Std. 1008:1987.* IEEE Standard for Software Unit Testing. – 31p.
15. *Developing* a Testing Maturity Model: Part I. / Burnstein, I., T. Suwannasart and C. R. Carlson // Crosstalk, STSC.- 1996.- August.- P. 21-24.
16. *ISO/IEC* 13407:1999. Human-centred Design Processes for Interactive Systems.
17. *Концепция* профилей в инженерии надежности ПС / Г.Б. Мороз, Г.И.Коваль, Т.М.Коротун // Математичні машини і системи". – 2004.- №1.- С. 166-184.
18. *Мороз Г.Б.* Пуассоновские модели роста надежности программного обеспечения и их приложение. Аналитический обзор // УСиМ.- 1996.- №. 1-2 - С. 69-85.
19. *Страничка* в электронной энциклопедии «Википедия» http://en.wikipedia.org/wiki/Test_driven_development
20. *Бабенко Л.П.*, Лаврищева К.М. Основи програмної інженерії. Навчальний посібник. – К.: Знання, 2001. –269 с.
21. *Майерс Г.* Надежность программного обеспечения. - М.: Мир, 1980. - 360 с.
22. *Zave P.* An Operational Approach to Requirements Specification for Embedded Systems // IEEE Trans. Software Eng..- 1982. -V.8. - № 3.
23. *Weyuker* E.J., Ostrand T.J. Theories of program testing and the application of revealing subdomains // IEEE Trans.Soft.Eng.- 1980.-V.6. - №.3. - P. 236-246.
24. *Richardson D.*, Clarke L. A partition analysis method to increase program reliability. // in proc. of yet 5th international Conf. Soft. Eng., San Diego, CA, mar.9-12, 1981, -P. 244-253.

25. *Software* unit test coverage and adequacy. / H.Zhu., P.A.V.Hall, J.H.R.May // ACM Computing Surveys.-1997.- V.29.-N.4. -P. 336-427.
26. *Software* testing based on formal specifications: a theory and tool. / G.Bernot., M.C.Gaudel, B.Marre // Software Engineering Journal.-1991.- November. – P. 287-405.
27. *Dick J.*, Faivre A. Automating the generation and sequencing of test cases from model-based specification. // FME'93: Industrial-Strenght formal method, LNCS 670, Springer Vertag, 1993. -P. 268-284.
28. *Horcher H.,* Peleska J. Using formal specifications to support software testing. // Software Quality Journal.-1995.- № 4. – P. 309-327.
29. *Липаев В.В*. Тестирование программ. -М.: Радио и связь, 1986. -296 с.
30. *Wendy* W. Peng, Dolores R. Wallace. Software Error Analysis. // NIST Special Publication 500-209. - March, - 1993.
31. *Аллен Э.* Типичные ошибки проектирования. Библиотека программиста (Bug Patterns in Java) /СПб.: Питер, 2003. – 224 с.
32. . *Определение* целей и задач инженерии надежности программного обеспечения. / Г.Б.Мороз, Г.И.Коваль, Т.М.Коротун // Проблемы программирования. Вып.1, 1997.- С. 98-106.
33. *Jacobson I.,* Booch G., Rumbaugh J. The Unified Software Development Process. Addison-Wesley, 1999. – 463 p.
34. *Hartman A.* Ten Years of Model Based Testing /http://react.cs.uni-sb.de/mbt2006/talks/ ModelBasedTestingSoberEvaluation.PDF
35. *Макгрегор Дж.,* Сайкс Д. Тестирование объектно-ориентированного программного обеспечения. Практическое пособие (A Practical Guide to Testing Object-Oriented Software). ДИАСОФТ (ISBN 966-7992-12-8), 2002, 432 с.
36. *Шлеер С.*, Меллор С. Объектно-ориентированный анализ: моделирование мира в состояниях. – К.: Диалектика., 1993. – 240 с.
37. *Crnkovic I.*, Larsson S., Chaudron M. Component-based Development Process and Component Lifecycle./ http://www.mrtc.mdh.se/publications/0953.pdf
38. *Гольдштейн* Б.*С.*, Ехриель И.М., Рерле Р.Д. Тестирование телекоммуникационных протоколов: проблемы и подходы. / Сети и системы связи online - http://www.ccc.ru/magazine/depot/02_12/read.html?0303.htm 20023.pdfISO 9646
39. *ISO 9646:1994.* Information Theory - Open System Interconnection - Conformance Testing Methodology and Framework.
40. *Leveson N.* Safeware: system safety and computers. Addison-Wesley Publishing Company, 1995. - 680 p.
41. *Bach J.* General Functionality and Stability Test Procedure. http://www.satisfice.com/tools/procedure.pdf.
42. *Функциональное* тестирование программного обеспечения информационно-расчетного типа. / Бернатович О.В., Коваль Г.И., Коротун Т.М.. // Проблемы программирования, Вып. 3, 1998. -С. 51-58.
43. *Hartman A.* Ten Years of Model Based Testing. A Sober Evaluation - http://react.cs.uni-sb.de/mbt2006/talks/ModelBasedTestingSoberEvaluation.PDF
44. *Сортов А.А.*, Хорошилов А.В. Функциональное тестирование Web-приложений на основе технологии UniTesK. http://citforum.ru/SE/testing/func_testing/#1
45. *Общий* оценочный лист тестирования usability web-сайта http://software-testing.ru/lib/it-online/site-usability-checklist.htm

46. *Whittaker J.A.* How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional (ISBN 0-321-36944-0), 2006.
47. *Nguyen H.*, Johnson R., Hackett M. Testing Applications on the Web (2nd Edition): Test Planning for Mobile and Internet-Based Systems (ISBN 0-471-20100-6)
48. *Тестирование* производительности Web-серверов http://www.sibinfo.ru/archive/news/03_01_08/server_testing.html
49. *Мельбурн Д.*, Джорм Д. Тестирование web-приложений "на проникновение" (часть 1) (перевод Дмитрия Леонова) /http://www.bugtraq.ru/library
50. *Стотлемайер Д.* Тестирование Web-приложений //М.: «КУДИЦ-ОБРАЗ» (ISBN 5-93378-064-2), 2003. - 240 с.
51. *IEEE Std 1044:1993.* Standard Classification for Software Anomalies. -21 p.
52. *Dreger*, Brian J. Function Point Analysis. New Jersey; Prentice-Hall Advanced Reference Series, Computer Science; 1989.
53. *ДСТУ 2851-94.* Програмні засоби ЕОМ. Документування результатів випробувань.
54. *Lutz R.* et al. Applying Integrated Safety Analysis Techniques (Software FMEA and FTA) // JPL D161168, Quality Assurance Office, 1988. - 15 p.
55. *Daich G.T.* et al. Software Test Technology Report. / STSC, August 1994. - P. 70.
56. *Eckerson, W.W.* Client-Server Test Tools: Client-Server Computing Has Created a Need for Robust New Test Tools and Test Methodologies. //Open Information Systems.-1994.- V.9.-№11. - P. 3-21.

Глава 8. МЕТОДЫ ОЦЕНКИ РАЗМЕРА ПРОГРАММНОЙ СИСТЕМЫ

8.1. Методология анализа показателей функциональности

8.1.1. Истоки методологии FPA

Организации-разработчики программных систем в течение многих лет искали приемлемые количественные методы для измерения производительности труда, оценивания эффективности процессов и управления затратами на разработку ПС. Камнем преткновения было отсутствие надежной единицы измерения *размера* программного обеспечения. Такая, на первый взгляд очевидная метрика, как число строк исходного кода SLOC (Source Lines of Code), не могла давать достоверные результаты, поскольку не учитывала следующее:

- число строк исходного кода зависит от уровня мастерства программиста. Фактически, чем выше мастерство программиста, тем меньшим количеством строк кода ему удастся обойтись для реализации определенной функциональной возможности (или *функциональности*) ПС;
- высокоуровневые языки или языки визуального программирования требуют гораздо меньшего числа строк кода, чем, например, язык Ассемблера или С для отражения одной и той же функциональности. Достаточно представить себе два приложения, имеющих одни и те же функциональные возможности (те же экраны, отчеты, таблицы базы данных), но реализованные на разных языках, например, С и Clarion. Очевидно, что существует обратная взаимосвязь между уровнем языка и производственной выработкой программиста;
- фактическое число SLOC остается неизвестным до тех пор, пока проект не будет почти завершен. Поэтому SLOC нельзя использовать для предварительной оценки усилий на разработку и построения план-графика проекта;
- в программистском сообществе не достигнуто соглашения о методе подсчета строк кода. Языковые конструкции, используемые, например, в Visual C++, Ассемблере, Коболе или SQL, абсолютно различны. Метод же остается общим для любых приложений, в том числе использующих комбинацию различных языков;
- заказчику сложно понять, каково соотношение указанных им функциональных и технических требований к ПС и объемов программистской работы.

Нужна была новая мера размера, применяемая унифицировано по всему ЖЦ ПС, понятная заказчику и легко интерпретируемая в терминах прикладной области его деятельности. И такая мера появилась в конце 70-х годов.

Однако нужно сразу отметить, что несмотря на проблемы, SLOC по-прежнему остается той единицей измерения размера, к которой «приводятся» (в которую конвертируются) результаты вычисления размера, выполненные другими методами, для последующего использования в «старых» моделях трудоемкости, надежности, тестирования, стоимости и др. Ее вполне можно также применять и для определения размера ПС на завершающих стадиях разработки и при сопровождении (разумеется, учитывая вышеуказанные замечания) [1].

Первой и наиболее удачной альтернативой методу подсчета исходных строк кода стала разработанная в 1979 году Алланом Альбрехтом из IBM методология, названная «Анализ показателей функциональности» (FPA, от Function Points Analysis). В ее основе лежит взгляд на ПС извне, с позиций пользователя системы, а не

«со стороны» ее внутренних свойств (таких, как SLOC). В результате анализа исходных требований к ПС и выяснения реальных потребностей пользователей определяется объем *функциональных возможностей системы*, показателями которых служат функции обработки информации настолько низкого уровня, насколько они укладываются в систему мышления пользователей, а кроме функций – данные, которые эти функции обрабатывают. Таким образом, методология FPA базируется на идее декомпозиции функций и данных до предельно допустимого (с точки зрения пользователя) уровня[1]. Объем функциональных возможностей ПС (далее просто *функциональный размер*) определяется в так называемых *условных единицах функциональности, УЕФ* (или FP – от Function Points).

В течение пяти лет с момента появления, методология FPA и метод расчета размера отшлифовывались А. Альбрехтом и проходили практическую апробацию, а в середине 80-х годов была создана *Международная группа пользователей показателей функциональности* (IFPUG, от International Function Points User Group), которая и поддерживает дальнейшую эволюцию метода.

Текущая версия метода (версия 4.2), называемого *базовым методом* определения функционального размера (FP-методом) зафиксирована в практическом руководстве по расчету размера в единицах функциональности CPM (Counting Practices Manual, 4.2), выпущенном летом 2004 и доступном на сайте IFPUG: www.ifpug.org или Дэвида Лонгстрита www.softwaremetrics.com/freemanual.htm. Подробнее положения FP-метода рассматриваются далее в этом разделе.

Со времени опубликования FP-метода, появилось много его разновидностей, основанных на той же концепции (таблица 8.1). Наиболее известные из них представлены в разделе 8.2. Однако, различия в интерпретации изначальной концепции, привели к *несовместимости* методов определения размера и несопоставимости результатов их применения, что снизило привлекательность каждого из них.

Таблица 8.1. **Методы определения размера ПС, основанные на FPA**

Название	**Автор**	**Организация**	**Рекомендации по применению**
Function Points (FP)	A. Albrecht	IBM (1979). Сопровождается IFPUG (International Function Point User Group) (1986)	Автоматизированные информационные системы, системы организационного управления и др., оперирующие большими объемами информации в базах данных.
Feature Points	C. Jones	Software Productivity Research (1985)	Системы со сложными вычислениями и умеренными объемами данных
Mark-II Function Points	C.Symons	UKSMA (UK Software Metrics Association) (1988) Сопровождается UFPUG (United Kingdom Function Point User Group)	Те же, что и для FP-метода. Применяется преимущественно в Великобритании (стандартизован в SSADM)
3D Function Points (3D_FP)	S.Whitmire	Boeing (1992)	ПС любого класса, в частности системы реального времени.

[1] Методология не замыкается на конкретных методах декомпозиции ПС. Авторам книги известны, например, работы по использованию метода Саати для указанных целей [2].

Название	Автор	Организация	Рекомендации по применению
Object Points	S.Whitmire	ASMA OO-SIG (1993)	Объектно-ориентированные распределенные системы.
Highly Constrained Systems	G.Rule	EFPUG FPE-SIG (1993)	Системы, состоящие из четко размежеванных прикладных подсистем.
Object Points for ICASE	R.Banker	Carlson School of Management (University Minisota (1994)	ПС, разрабатываемые в среде интегрированных CASE.
PRICE Object Points	A.Minkiewicz	PRICE Systems, Lockheed Martin Corporation (1996)	ПС, при разработке которых применяется объектно-ориентированный подход. Модель учитывает число классов верхнего уровня, средневзвешенное число методов в классе, среднюю глубину дерева наследования, среднее число потомков в классе.
Bang Metric	T.De Marco	Atlantic Systems Guild (1982)	Метод подобен 3D_FP-методу.

В 1992 Группы Пользователей (User Groups)[2] из Австралии, Великобритании, Нидерландов и США, занимающихся измерением программных средств, создали рабочую группу WG12 в рамках подкомитета ISO/IEC/JTC1/SC7 и предложили концепцию *измерения функционального размера* (FSM, от Functional Size Measurement) и проект одноименного стандарта измерения размера - ISO/IEC 14143. Этот стандарт поможет решить проблему несовместимости методов и создать эталонную модель метода. Структура стандарта вкратце рассматривается в разделе 8.4.

8.1.2. Программный компонент. Взгляд пользователя

Согласно базовому FP-методу размер программной системы вычисляется путем суммирования размеров ее отдельных программных компонентов (ПК). Здесь *программный компонент* определяется как часть программного обеспечения системы (программное приложение, пакет, комплекс программ), реализующая отдельную задачу (комплекс задач) пользователя системы.

Поскольку базовый FP-метод ориентирован для применения при разработке *информационных систем*, процесс проектирования которых является в большинстве случаев процессом, «управляемым данными», размер и функциональная сложность измеряемого ПК оценивается исходя из оценок *сложности данных*, являющихся входными, выходными и обрабатываемыми данными *для каждой функции* ПК.

---

[2] Группы Пользователей (User Groups) – добровольные национальные или международные профессиональные сообщества пользователей, создаваемые для анализа состояния проблемы, представляющей общий интерес, и выработки согласованных коллективных решений (или стандартов). Примеры – IFPUG (International FP) , UFPUG (UK FP), LUG (Linux), APCUG (Association of Personal Computer) и др.

Исходная информация, необходимая для выполнения оценки размера на ранних стадиях ЖЦ, может быть получена из документации на ПС (описания требований, проекта, постановок задач, диаграмм потоков данных, описаний моделей данных и др.), а также путем интервьюирования конечных пользователей каждого ПК.

Метод оценивания не требует знания технологии реализации ПК и особенностей среды его функционирования. Оцениванию подлежат запрошенные пользователем *функции* ПК по обработке данных и связанные с ними *информационные объекты*. Степень детализации функций и информационных объектов ограничивается тем уровнем декомпозиции функциональных требований к ПК, который *приемлем* для понимания *конечным пользователем ПК* функциональных возможностей ПК по преобразованию информации в ходе выполнения запрошенной функции (рисунок 8.1).

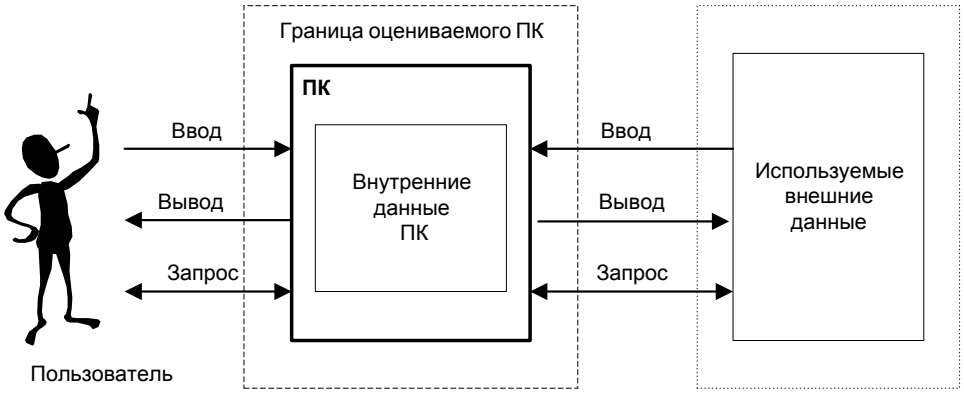


Рис. 8.1. **Взгляд пользователя на процесс функционирования ПК**

Определения. Важными для правильного применения метода являются определения следующих понятий:

Граница ПК - установленная *пользователем* граница между информационно-функциональными объектами *измеряемого* ПК и других (внешних) ПК или предметной области пользователя.

Внутренние данные ПК - идентифицируемая конечным пользователем группа логически связанных данных или управляющая информация, сохраняемая, обновляемая, добавляемая или удаляемая в пределах границы ПК.

Внешние данные ПК - идентифицируемая конечным пользователем группа логически связанных данных или управляющая информация, используемая (но не изменяемая) в пределах границы ПК, и являющаяся внутренними данными в пределах границы другого ПК.

Вход - данные или управляющая информация, попадающая извне (от пользователя либо другого ПК) в пределы границы измеряемого ПК.

Выход - данные или управляющая информация, покидающая границы ПК (и направляемая пользователю или другому ПК).

Запрос - запрашиваемая и в ответ получаемая пользователем информация. Это выход, непосредственно связанный со входом и получаемый без выполнения каких-либо действий над внутренними или внешними данными ПК (кроме выборки и элементарного редактирования).

Размер ПК – суммарное число условных единиц функциональности, приписываемых каждой функции в зависимости от количества и сложности информационных объектов, с которыми она работает.

Процесс подсчета УЕФ (FP) по методу FPA можно условно разделить на два подпроцесса:

- измерение объема и сложности *информационных объектов*, с которыми работает ПК;
- измерение объема и сложности *элементарных функций* по обработке данных в ПК.

8.1.3. Оценка размера и сложности объектов данных

Процесс измерения данных ПК включает перечисленные ниже шаги.

Шаг 1. Анализ документации ПС и определение перечня всех логически связанных групп данных, с которыми должен работать ПК (например, больничная карта, договор, ведомость, командный файл, диагностическое сообщение и др.).

Шаг 2. Определение границы измеряемого ПК и классификация данных на:

- внутренние логические объекты;
- внешние интерфейсные объекты.

Внутренний логический объект (ВЛО) - идентифицируемая конечным пользователем группа логически связанных данных и управляющая информация, сохраняемая и сопровождаемая (добавляемая, обновляемая или удаляемая) в пределах границы ПК.

Внешний интерфейсный объект (ВИО) - идентифицируемая конечным пользователем группа логически связанных данных или управляющая информация, используемая измеряемым ПК, но сопровождаемая в пределах границы другого ПК. ВИО является внутренним логическим объектом в другом ПК. Например, к ВИО могут относиться классификаторы, нормативно-справочная информация, вспомогательная информация (помощь) по работе с ПК и др.

Правила отнесения логической группы данных или управляющей информации к ВЛО таковы:

- группа данных выделяется в соответствии с требованиями *пользователя* к данным ПК;
- группа данных *сопровождается* в пределах границы ПК;
- модификация группы данных выполняется в рамках *элементарного процесса*, идентифицируемого пользователем.

Элементарный процесс – это элементарная единица действия, идентифицируемая (воспринимаемая) конечным пользователем в предметной области. Это самостоятельная технологическая операция в ПК, по завершении выполнения которой предметная область остается в целостном состоянии;

- группа данных не может быть отнесена к ВИО для данного ПК.

Правила отнесения логической группы данных или управляющей информации к ВИО таковы:

- группа данных выделяется в соответствии с требованиями пользователя к данным ПК;
- группа данных используется в ПК;
- группа данных не сопровождается в измеряемом ПК;

- группа данных является ВЛО по крайней мере в одном из других ПК;
- группа данных не может быть отнесена к ВЛО измеряемого ПК.

Шаг 3. *Определение сложности каждого ВЛО и ВИО.*

Уровень сложности может квалифицироваться как «низкий», «средний» и «высокий». Отнесение объекта к тому или иному уровню сложности производится исходя из числа *подгрупп данных* и числа *элементарных данных* ВЛО или ВИО.

Подгруппа данных объекта (ПДО) - идентифицируемая пользователем подгруппа группы данных ВЛО или ВИО. ПДО *не является самостоятельной* группой данных. Например, объект «больничная карта» может иметь ПДО «информация о болезни» (диагноз, назначение и др.) и «информация о больном» (паспортные данные, адрес) и др.

Элементарное данное объекта (ЭДО) - уникальное (неповторяющееся) идентифицируемое пользователем данное (поле) ВЛО или ВИО.

Матрица оценки уровня сложности ВЛО и ВИО представлена в таблице 8.2.

Таблица 8.2. **Матрица оценки сложности данных ПК**

Число подгрупп данных объекта (ПДО)	**Число элементарных данных объекта (ЭДО)**		
	1 - 19	**20 – 50**	**> 50**
0 – 1	Низкий	Низкий	Средний
2 – 5	Низкий	Средний	Высокий
> 5	Средний	Высокий	Высокий

Если необходимый уровень детализации данных ПК в документации ПС отсутствует - присваивание уровня сложности может производиться на основе экспертных оценок.

Шаг 4. *Взвешивание ВЛО и ВИО по уровням сложности.*

Веса, присваиваемые ВЛО и ВИО в зависимости от принятого уровня сложности, представлены в таблице 8.3.

Таблица 8.3. **Веса ВЛО и ВИО**

Тип Объекта	**Веса по уровням сложности**		
	Низкий	**Средний**	**Высокий**
ВЛО	7	10	15
ВИО	5	7	10

Шаг 5. *Подсчет условных единиц функциональности по всем ВЛО и ВИО.*

Для подсчета условных единиц функциональности информационных объектов ($УЕФ_о$) необходимо определить количество ВЛО и ВИО по каждому уровню сложности и полученные значения просуммировать.

Например, если выявлено 2 ВЛО низкого уровня сложности, 3 - среднего, а также 1 ВИО среднего уровня сложности и 2 - высокого, то

$$УЕФ_0 = 2 \cdot 7 + 3 \cdot 10 + 1 \cdot 7 + 2 \cdot 10 = 71 \text{ условная единица.}$$

8.1.4. Оценка размера и сложности функций обработки данных

Каждое функциональное требование к ПК представляет собой совокупность *элементарных функций* обработки данных, поддерживаемых элементарными процессами.

Процесс измерения элементарных функций обработки данных ПК включает перечисленные ниже шаги.

Шаг 1. *Анализ функциональных требований к ПК и выделение технологических процессов, поддерживающих их реализацию*. Технологический процесс может включать несколько технологических операций (элементарных процессов), каждая их которых реализует элементарную функцию ПК.

Шаг 2. *Отнесение каждой элементарной функции в пределах одного функционального требования к одной из трех категорий:*

- внешний ввод;
- внешний вывод;
- внешний запрос.

Внешний ввод (ВВД) – элементарный процесс обработки входа в ПК. ВВД выполняет функцию управления ПК или сопровождения (добавления, обновления или удаления) данных в ВЛО. Например, ввод данных в поле экрана, выбор данных из списка, обработка щелчка на кнопке.

Внешний вывод (ВЫВ) - элементарный процесс генерации выхода из ПК. Например, формирование и печать отчета, формирование и вывод данных на экран.

Внешний запрос (ЗАП) - элементарный процесс непосредственной выборки запрашиваемых пользователем по определенному критерию данных из ВЛО или ВИО без какого-либо изменения и без сопровождения данных в ВЛО. Например, получение тематической справки, получение контекстной помощи, вывод классификатора, многокритериальный поиск информации.

Правила отнесения функции к внешнему вводу:

- данные или управляющие воздействия, обрабатываемые функцией, поступают извне границы ПК;
- данные сохраняются в ВЛО, и производится сопровождение ВЛО;
- функция реализуется элементарным процессом;
- для функции, отождествляемой с элементарным процессом, выполняется одно из следующих правил:
  - логика обработки данных функцией отличает ее от других ВВД;
  - данные, обрабатываемые функцией, отличаются от данных в других ВВД.

Правила отнесения функции к внешнему выводу:

- функция посылает данные или управляющую информацию за пределы границы ПК;
- функция реализуется элементарным процессом;
- для функции, отождествляемой с элементарным процессом, выполняется одно из следующих правил:
  - логика обработки данных функцией отличает ее от других ВЫВ;
  - данные, обрабатываемые функцией, отличаются от данных в других ВЫВ

Правила отнесения функции к внешнему запросу:

- данные, составляющие критерий запроса, получены извне границы ПК;
- данные, составляющие результат запроса, покидают границу ПК;
- результирующие данные являются выборкой (данные в ВЛО не изменяются, логические или математические действия над данными не производятся);
- функция реализуется элементарным процессом;

- для функции, отождествляемой с элементарным процессом, выполняется одно из следующих правил:

- логика обработки данных (в части ввода и в части вывода) отличает функцию от других ЗАП;
- данные, которые обрабатывает функция (в части ввода и в части вывода), отличаются от данных в других ЗАП.

Шаг 3. *Определение сложности каждого ВВД, ВЫВ* и *ЗАП.*

Уровень сложности функции может квалифицироваться как «низкий», «средний» и «высокий». Отнесение функции к тому или иному уровню сложности производится исходя из числа *ссылочных объектов функции* и числа *элементарных данных* функции, представляющих вход, выход или выборку.

Ссылочный объект функции (СОФ) - внутренний логический объект, который сопровождается функцией, или внешний интерфейсный объект, который используется функцией измеряемого ПК.

Элементарное данное функции (ЭДФ) - уникальное (неповторяющееся) идентифицируемое пользователем данное (поле), относящееся ко входу, выходу или выборке функции. Например, вводимое поле, список, управляющее воздействие, отождествляемое со щелчком на управляющем объекте экрана (кнопке).

Элементарные данные функции связываются с элементарными данными объекта в ВИО и ВЛО, которые используются или сопровождаются функцией (рисунок 8.2).

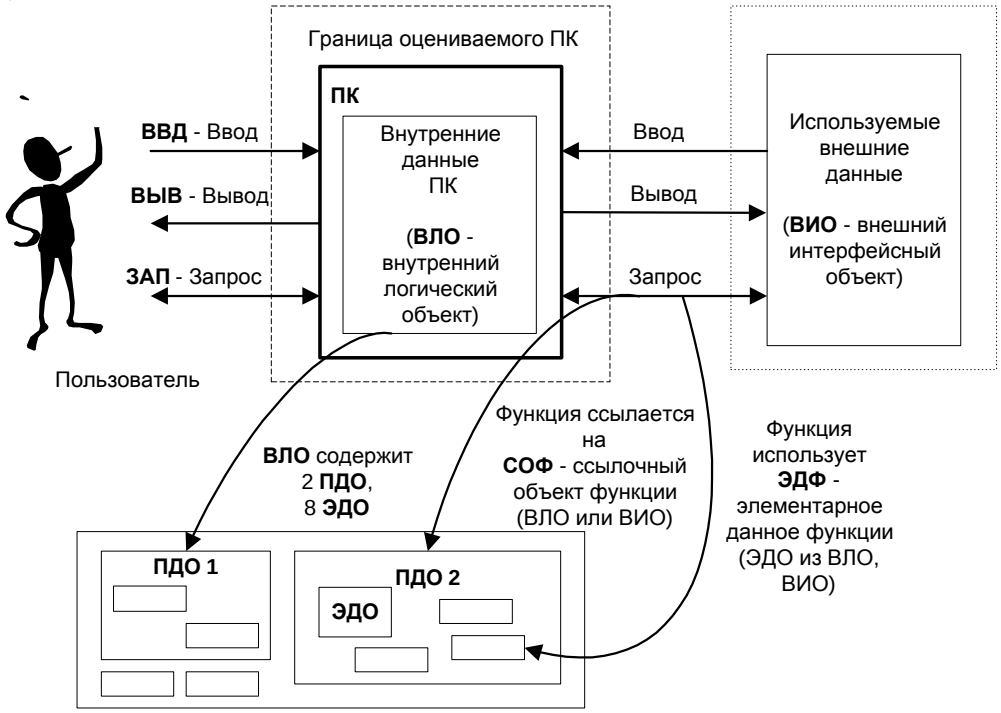


Рис. 8.2. **Взаимосвязь понятий в FPA**

Матрица оценки уровня сложности ВВД представлена в таблице 8.4.

Таблица 8.4. **Матрица оценки уровня сложности ВВД**

Число ссылочных объектов функции (СОФ)	**Число элементарных данных функции (ЭДФ)**		
	1 – 4	**5 – 15**	**> 15**
0 - 1	Низкий	Низкий	Средний
2 - 3	Низкий	Средний	Высокий
> 3	Средний	Высокий	Высокий

Оценка уровня сложности каждого внешнего запроса выполняется отдельно в части ввода и в части вывода. Результирующим значением считается высшая из двух оценок.

Матрица оценки уровня сложности ВЫВ представлена в таблице 8.5.

Таблица 8.5. **Матрица оценки уровня сложности ВЫВ**

Число ссылочных объектов функции (СОФ)	**Число элементарных данных функции (ЭДФ)**		
	1 – 5	**6 – 19**	**> 19**
0 - 1	Низкий	Низкий	Средний
2 - 3	Низкий	Средний	Высокий
> 3	Средний	Высокий	Высокий

Шаг 4. *Взвешивание ВВД, ВЫВ и ЗАП по уровням сложности.*

Веса, присваиваемые ВВД и ВЫВ, представлены в таблице 8.6.

Таблица 8.6. **Веса ВВД**

Категория функции	**Веса по уровням сложности**		
	Низкий	**Средний**	**Высокий**
ВВД	3	4	6
ВЫВ	4	5	7
ЗАП	3	4	6

Шаг 5. *Подсчет условных единиц функциональности по всем функциям обработки данных ПК.*

Для подсчета условных единиц функциональности для функций ($УЕФ_ф$) необходимо определить количество ВВД, ВЫВ и ЗАП по каждому уровню сложности для каждого функционального требования и просуммировать значения.

Шаг 6. *Определение размера (объема) функциональных возможностей или просто показателя функционального размера (ПФР)* выполняется по формуле:

$$ПФР = УЕФ_о + УЕФ_ф$$

Значение ПФР при необходимости может быть конвертировано в эквивалентное число строк исходного кода (SLOC) ПК на соответствующем языке программирования.

8.1.5. Определение эквивалентного числа строк кода

Конвертирование значения показателя функционального размера в эквивалентное число строк исходного кода ПК (SLOC) на соответствующем языке программирования обычно выполняется для применения вычисленного размера ПК в формулах расчета стоимости, производительности или надежности ПС. Относительное количество единиц SLOC, приходящихся на одну единицу УЕФ для неко-

торых языков программирования, представлено в таблице 8.7[3].

Таблица 8.7. **Эквивалентное число строк исходного кода для УЕФ**

Язык программирования	**SLOC/УЕФ**
Basic Assembly	575
JCL	400
Macro Assembly	400
C	225
Cobol 74 (Cobol I)	220
FORTRAN	210
Cobol 85 (Cobol II)	175
Pascal	160
PL/1	126
RPG I	120
RPG II/III	110
Natural	100
C++	80
Java	80
dBase III	60
Focus	60
Clipper	60
Oracle	60
Sybase	60
dBase IV	55
Perl	50
JavaScript	50
VBScript	50
Shell Script	50
SAS	50
APL	50

8.1.6. Учет нефункциональных требований к программной системе

Рассчитанное значение ПФР характеризует объем и сложность только *функциональных требований* и не учитывает сложности технических (нефункциональных) требований к разрабатываемой ПС. Для их учета необходимо оценить 14 *общесистемных характеристик*, отражающих *влияние* сложности этих требований на реализационные особенности ПС[4] (таблица 8.8).

Нужно отметить, что если организация разрабатывает однотипные программные средства в однородной среде, общесистемные характеристики можно считать постоянными. Для того чтобы узнать, какой – «чистый» или скорректированный - показатель функционального размера служит лучшим предсказателем трудоемкости разработки, нужно проводить эксперименты, пробуя сочетать чистый

[3] Данные David Consulting Group, адрес в Интернете:www.davidconsultinggroup.com/indata.htm . См. также п.8.4.3, где обсуждается проблема подготовки и использования эталонных данных

[4]Информация получена на сайте www.softwaremetrics.com.

ПФР с другими метриками. Так, например, многие методологии оценки трудоемкости ПС не допускают использования скорректированного ПФР (например, COCOMO).

Таблица 8.8. **Общесистемные характеристики ПС**

Общесистемная характеристика		Краткое описание
1	Передача данных	Какие средства связи используются для передачи или обмена информацией с приложением или системой?
2	Распределенная обработка данных	Каким образом осуществляется управление распределенными данными и функциями их обработки?
3	Производительность	Указана ли в требованиях пользователя необходимая продолжительность ответа на запросы или пропускная способность?
4	Конфигурация и занятость технических средств	Насколько плотно будет использоваться аппаратная платформа, на которой будет работать приложение?
5	Интенсивность транзакции	Как часто должны выполняться транзакции - ежедневно, еженедельно, ежемесячно и т.д.?
6	Оперативный ввод данных	Какой процент информации вводится оперативно (в он-лайне)?
7	Эффективность работы конечного пользователя	Ставится ли во главу угла при разработке приложения обеспечение эффективности работы конечного пользователя?
8	Оперативность модификации данных (в он-лайне)	Сколько внутренних логических объектов (ВЛО) обновляется транзакцией в он-лайне?
9	Сложность обработки данных	Предназначено ли приложение для выполнения обширной логической или математической обработки данных?
10	Возможность повторного использования приложения	Ориентируется ли приложение на удовлетворение требований одного конкретного потребителя или многих потребителей?
11	Простота установки	Насколько сложен перенос и инсталляция приложения?
12	Простота использования	Насколько понятны ручные и/или автоматизированные процедуры запуска, резервирования и восстановления?
13	Распространенность	Предназначено ли приложение для работы во многих организациях (территориальное распределение)?
14	Простота внесения изменений	Разрабатывается ли приложение специально так, чтобы облегчить внесение изменений?

Оценка общесистемных характеристик выполняется экспертным путем по 5 уровням сложности от 0 до 5:

0 – не оказывает никакого влияния;
1 – оказывает несущественное влияние;
2 - оказывает умеренное влияние;
3 - оказывает среднее влияние;
4 - оказывает существенное влияние;
5 – оказывает сильное влияние.

Если потребитель (заказчик) не может принять четкого однозначного решения относительно оценки указанных общесистемных характеристик, нужно пересматривать спецификацию требований или архитектурный проект системы.

Оценка влияния требований к передаче данных. Данные и управляющие воздействия посылаются приложению или получаются им через средства связи. Обмен информацией регулируется определенным протоколом. Примером служит протокол TCP/IP, который предоставляет общий язык для взаимодействия сетей, использующих множество локальных протоколов (Ethernet, Netware, AppleTalk, DECnet и др.). Рейтинги влияния характеристики передачи данных на сложность приложения представлены в таблице 8.9.

Таблица 8.9. **Учет требований по передаче данных**

Рейтинг	Основания для определения степени влияния характеристики
0	Приложение - пакет или программа для локального компьютера
1	Приложение - пакет, но имеет удаленные ввод данных *или* печать
2	Приложение - пакет, но имеет удаленный ввод данных *и* удаленную печать
3	Приложение осуществляет внешний интерфейс с информационной средой – выполняет оперативный сбор данных или предоставляет услуги предварительной телеобработки (TP, от teleprocessing) для пакетированного процесса или системы с запросами
4	Приложение - больше чем внешний интерфейс (или препроцессор), но поддерживает только один вид коммуникационного протокола TP
5	Приложение - больше чем внешний интерфейс и поддерживает более чем один вид протокола TP

Примечание. Приложение, которое поддерживает web-запросы и локальный доступ, должно получить оценку 3, а приложение, которое допускает модификацию объектов ВЛО через Интернет и локальное обновление, должно получить оценку 5.

Оценка влияния требований к распределенной обработке данных. Рейтинги влияния характеристики на сложность приложения представлены в таблице 8.10.

Таблица 8.10. **Учет требований по распределенной обработке данных**

Рейтинг	Основания для определения степени влияния характеристики
0	Приложение не предназначено для передачи данных или делегирования функций обработки другим компонентам системы
1	Приложение подготавливает данные, для того чтобы конечный пользователь обрабатывал их с помощью другого компонента системы, как, например, электронные таблицы и СУБД
2	Средствами приложения данные подготавливаются для передачи, затем переносятся и обрабатываются на другом компоненте системы (данные не для обработки конечным пользователем)
3	Распределенная обработка и передача данных являются оперативными и только в одном направлении
4	Распределенная обработка и передача данных являются оперативными и осуществляются в обоих направлениях
5	Функции обработки динамически выполняются на наиболее подходящем для этого компоненте системы

Оценка влияния требований к производительности. Требования к производительности устанавливаются в виде целевых показателей продолжительности ожидания ответа на запросы пользователя или пропускной способности средств

передачи данных. Рейтинги влияния характеристики на сложность приложения представлены в таблице 8.11.

Таблица 8.11. **Учет требований по производительности**

Рейтинг	Основания для определения степени влияния характеристики
0	Никакие специальные требования относительно производительности пользователем не установлены
1	Требования к производительности установлены, но никаких специальных действий предпринимать не нужно
2	Время ответа или пропускная способность являются критическими в течение часов пик. Никаких специальных проектных решений об использовании CPU не требуется. Обработка данных, которая не может быть завершена до конца сеанса работы, откладывается на следующий рабочий день (отложенная обработка)
3	Продолжительность ожидания ответа или пропускная способность является критической в течение всего рабочего времени. Никаких специальных проектных решений об использовании CPU не требуется. Порядок отложенной обработки должен согласовываться со взаимодействующими системами.
4	Кроме того, пользователем предъявляются достаточно жесткие требования к производительности, что требует включения в проект задач анализа производительности.
5	Кроме того, при проектировании и разработке нужно использовать инструментальные средства анализа производительности, чтобы удовлетворить требования пользователя

Оценка влияния требований к конфигурации и занятости технических средств. Разработка приложения может потребовать принятия специальных решений о рациональном использовании существующих или специально приобретаемых технических средств, применяемых автономно или в режиме разделения, с защитой информации или без, эксплуатируемых непрерывно и продолжительно или нет. Рейтинги влияния данной характеристики на сложность приложения представлены в таблице 8.12.

Таблица 8.12. **Учет требований по конфигурации и занятости технических средств**

Рейтинг	Основания для определения степени влияния характеристики
0	Никакие оперативные ограничения в требования не включены
1	Оперативные ограничения существуют, но не требуется никаких специальных усилий, чтобы их удовлетворить
2	В требования включены положения относительно обеспечения защиты информации или распределения времени использования среды
3	Включаются специальные требования к процессору для определенной части приложения
4	Заявленные в требованиях операционные ограничения на работу приложения требуют введения специальных ограничений на обработку приложения центральным или другим (назначенным) процессором
5	Кроме того, есть специальные ограничивающие условия на работу приложения с распределенными компонентами

Оценка влияния требований к интенсивности транзакции. Высокая интенсивность транзакций может повлиять на проектные решения и реализацию приложения. Рейтинги влияния данной характеристики на сложность приложения представлены в таблице 8.13.

Таблица 8.13. **Учет требований по интенсивности транзакций**

Рейтинг	Основания для определения степени влияния характеристики
0	Никакого пикового периода транзакции не предвидится
1	Предвидится пиковый период транзакции (например, ежемесячно, ежеквартально, сезонно, ежегодно)
2	Предвидится еженедельный пиковый период транзакции
3	Предвидится ежедневный пиковый период транзакции
4	В соответствии с требованиями приложение должно поддерживать высокую интенсивность транзакций, что влечет за собой решение задач анализа производительности на стадии проектирования
5	Кроме того, необходимо использовать инструментальные средства анализа производительности.

Оценка влияния требований к оперативному вводу данных. Рейтинги влияния данной характеристики на сложность приложения представлены в таблице 8.14.

Таблица 8.14. **Учет требований по оперативному вводу данных**

Рейтинг	Основания для определения степени влияния характеристики
0	Все транзакции обрабатываются в пакетном режиме
1	От 1% до 7% транзакций представляют собой интерактивный ввод данных
2	От 8% до 15% транзакций представляют интерактивный ввод данных
3	От 16% до 23% транзакций представляют интерактивный ввод данных
4	От 24% до 30% транзакций представляют интерактивный ввод данных
5	Более чем 30% транзакций представляют интерактивный ввод данных

Оценка влияния требований к эффективности работы конечного пользователя. Для обеспечения эффективной работы пользователей в режиме оперативной обработки информации в проекте приложения должны предусматриваться следующие возможности:

- функции навигации (например, функциональные клавиши, переходы по ссылке, динамически генерируемые меню);
- меню;
- оперативные подсказки и справочная информация;
- автоматическое перемещение курсора;
- скроллинг;
- удаленная печать (через оперативные транзакции);
- закрепление функциональных клавиш;
- пакетные задания, сформированные из оперативных транзакций;
- выделение данных на экране курсором;
- использование индикаторов - негативного изображения, выделения, цветного подчеркивания и др.;

- распечатка пользовательской документации оперативных транзакций;
- интерфейс мыши;
- всплывающие окна;
- минимум экранов для реализации деловой функции;
- двуязычная поддержка (учитывается при подсчете как четыре элемента);
- многоязычная поддержка (учитывается при подсчете как шесть элементов).

Рейтинги влияния данной характеристики на сложность приложения представлены в таблице 8.15.

Таблица 8.15. **Учет требований по эффективности работы пользователя**

Рейтинг	Основания для определения степени влияния характеристики
0	Ни одной из возможностей, упомянутых в списке
1	От 1 до 3 возможностей, упомянутых в списке
2	От 4 до 5 возможностей, упомянутых в списке
3	6 или больше возможностей из упомянутых в списке, но нет никаких специальных требований потребителя, связанных с эффективностью.
4	6 или больше возможностей, упомянутых в списке. Есть достаточно четкие требования потребителя относительно эффективности работы конечных пользователей, что влечет за собой включение в проект задач учета человеческих факторов (например, минимизация последовательностей нажатий клавиш, максимизация типовых (умалчиваемых) значений, использование шаблонов)
5	Шесть или больше возможностей, упомянутых в списке. Есть жесткие требования относительно эффективности работы конечного пользователя. Их реализация требует использования специальных инструментальных средств и процессов для демонстрации того, что цели по эффективности достигнуты

Оценка влияния требований к оперативности модификации данных. Рейтинги влияния данной характеристики на сложность приложения представлены в таблице 8.16.

Таблица 8.16. **Учет требований по оперативности модификации данных**

Рейтинг	Основания для определения степени влияния характеристики
0	Нет
1	Оперативная модификация 1 – 3 объектов (файлов). Объем обновляемых данных небольшой. Не сложное восстановление данных
2	Оперативная модификация 4 и более объектов. Объем обновления небольшой. Не сложное восстановление данных
3	Предполагается оперативная модификация основных внутренних логических объектов
4	Кроме того, весьма важна защита от потери данных. Она должна быть специально спроектирована и запрограммирована в системе
5	Кроме того, большие объемы данных требуют рассмотрения вопросов стоимости процесса восстановления. Нужны высоко автоматизированные процедуры восстановления с минимальным вмешательством оператора

Оценка влияния требований к сложности обработки данных. Если к приложению предъявляется требование сложной обработки информации, в проекте должны рассматриваться следующие возможности:

- тщательный контроль (например, специальная обработка входных данных) и/или специальная реализация защиты информации;
- обширная логическая обработка;
- обширная математическая обработка;
- большое количество обрабатываемых исключительных ситуаций, что может приводить к неполным транзакциям, которые должны обрабатываться повторно;
- сложная обработка, связанная с управлением множеством возможностей ввода/вывода (в том числе, например, мультимедиа).

Рейтинги влияния данной характеристики на сложность приложения представлены в таблице 8.17.

Таблица 8.17. **Учет требований по сложности обработки данных**

Рейтинг	Основания для определения степени влияния характеристики
0	Ни одна из упомянутых выше возможностей
1	Любая одна из упомянутых выше возможностей
2	Любые две из упомянутых выше возможностей
3	Любые три из упомянутых выше возможностей
4	Любые четыре из упомянутых выше возможностей
5	Все пять из упомянутых выше возможностей

Оценка влияния требований к повторному использованию приложения. Рейтинги влияния данной характеристики на сложность приложения представлены в таблице 8.18.

Таблица 8.18. **Учет требований по повторному использованию**

Рейтинг	Основания для определения степени влияния характеристики
0	Не повторно используемый код
1	Повторно используемый код для применения в пределах приложения
2	Меньше чем 10% приложения может быть востребовано более чем одним потребителем
3	10% или больше может быть востребовано более чем одним потребителем
4	Приложение было специально укомплектовано и/или документировано так, чтобы облегчить повторное использование, и настраивается потребителем на уровне исходного кода
5	Приложение было специально укомплектовано и/или документировано так, чтобы облегчить повторное использование, и настраивается для использования по известному параметру потребителя

Оценка влияния требований к простоте установки приложения. Если приложение должно быть переносимым и легко устанавливаемым, план его ввода в действие и/или инструментальные средства конверсии должны быть предоставлены и проверены в течение стадии системного тестирования. Рейтинги влияния данной характеристики на сложность приложения представлены в таблице 8.19.

Таблица 8.19. **Учет требований по простоте установки**

Рейтинг	Основания для определения степени влияния характеристики
0	Потребителем не были заявлены никакие специальные требования и никакой специальной программы инсталляции для установки приложения не нужно
1	Потребителем не были заявлены никакие специальные требования, но для установки приложения необходима специальная инсталляция
2	Требования к преобразованию и установке были заявлены потребителем, и руководства по установке должны быть предоставлены и проверены. Отражение вопросов конверсии в проекте не считается важным
3	Требования к преобразованию и установке были заявлены потребителем, и руководства по установке должны быть предоставлены и проверены. Отражение вопросов конверсии в проекте считается важным
4	В добавление к 2, автоматизированные инструменты конверсия и инсталляции должны быть предоставлены и проверены
5	В добавление к 3, автоматизированные инструменты конверсии и инсталляции должны быть предоставлены и проверены

Оценка влияния требований к простоте использования приложения. Если простота использования – одно из требований к приложению, на стадии системного тестирования должны быть предоставлены и проверены эффективные процедуры запуска, резервирования и восстановления. Приложение должно минимизировать действия, выполняемые вручную. Рейтинги влияния данной характеристики на сложность приложения представлены в таблице 8.20.

Таблица 8.20. **Учет требований по простоте использования**

Рейтинг	Основания для определения степени влияния характеристики
0	Потребителем не были заявлены никакие специальные требования по оперативному использованию, кроме обычных процедур резервирования
1 - 4	Одна, несколько, или все следующие возможности должны предусматриваться в приложении. Каждая возможность имеет вес 1, кроме тех, которые отмечены особо
	Эффективные процессы запуска, резервирования и восстановления, однако вмешательство оператора необходимо
	Эффективные процессы запуска, резервирования и восстановления, но вмешательство оператора не требуется (учитывается при подсчете как два элемента)
	Минимизирована необходимость монтирования ленты
	Минимизирует необходимость управления бумагой
5	Приложение разрабатывается для автоматического функционирования. Автоматическое функционирование означает, что *никакое вмешательство оператора не* требуется для управления системой, кроме запуска или останова приложения. Автоматическое резервирование – одно из свойств приложения

Оценка влияния требований к распространенности приложения. Если приложение должно устанавливаться во многих местах (многих организациях), в проекте должны отражаться вопросы эффективного функционирования приложения в различных аппаратно-программных средах. Рейтинги влияния данной характеристики на сложность приложения представлены в таблице 8.21.

Таблица 8.21. **Учет требований по распространенности приложения**

Рейтинг	Основания для определения степени влияния характеристики
0	В требованиях потребителя не оговаривается возможность использования приложения более чем на одном рабочем месте / более одной установки
1	Потребности установки приложения во многих пунктах предусматриваются в проекте, однако приложение разрабатывается так, что будет работать только в *идентичных* аппаратно-программных средах
2	Потребности установки приложения во многих пунктах предусматриваются в проекте, но приложение разрабатывается так, что будет работать только в *однотипных* аппаратно-программных средах
3	Потребности установки приложения во многих пунктах предусматриваются в проекте, и приложение разрабатывается так, что будет работать *в различных* аппаратно-программных средах
4	Должны разрабатываться и проверяться документация и план поддержки приложения во многих пунктах и приложение таково, как описано в 1 или 2
5	Должны разрабатываться и проверяться документация и план поддержки приложения во многих пунктах и приложение таково, как описано в 3

Оценка влияния требований к простоте внесения изменений. Если приложение специально разрабатывается так, чтобы в него было просто вносить изменения, в его проекте должны предусматриваться следующие возможности:

- предоставление гибких справок и отчетов, получением которых можно управлять с помощью *простых* запросов; например, логика «И/ИЛИ» применима только к *одному* ВЛО (учитывается при подсчете как один элемент);
- предоставление гибких справок и отчетов, получением которых можно управлять с помощью запросов средней сложности; например, логика «И/ИЛИ» применима к более чем одному ВЛО (учитывается при подсчете как 2 элемента);
- предоставление гибких справок и отчетов, получением которых можно управлять с помощью сложных запросов; например, комбинация логики «И/ИЛИ» применима к одному или более ВЛО (учитывается при подсчете как 3 элемента);
- эталонные (контрольные) бизнес-данные хранятся в таблицах, которые поддерживаются потребителем с помощью оперативных интерактивных процессов, но изменения вступают в действие только на следующий рабочий день;
- эталонные (контрольные) бизнес-данные хранятся в таблицах, которые поддерживаются потребителем с помощью оперативных интерактивных процессов, и изменения вступают в силу немедленно (учитывается как два элемента).

Рейтинги влияния данной характеристики на сложность приложения представлены в таблице 8.22.

Таблица 8.22. **Учет требований по простоте внесения изменений**

Рейтинг	Основания для определения степени влияния характеристики
0	Ни одна из упомянутых выше возможностей
1	Любая одна из упомянутых выше возможностей
2	Любые две из упомянутых выше возможностей
3	Любые три из упомянутых выше возможностей
4	Любые четыре из упомянутых выше возможностей
5	Все пять из упомянутых выше возможностей

Показатель сложности технических требований. После того, как определена степень влияния всех 14 общесистемных характеристик на размер и сложность разработки ПС, может быть рассчитан поправочный (настроечный) коэффициент сложности ПС (*Ксложж*):

$$Ксложж = 0.65 + [(\sum_{i=1}^{14} C_i)/100]$$

где C_i - степень влияния каждой общесистемной характеристики.

Как правило, этот коэффициент модифицирует значение ПФР на ±35%.

Нужно отметить, также, что различные организации могут не использовать все 14 характеристик сложности технических требований к ПС. Например, Software Productivity Research, Inc. рекомендует применение только трех.

Общий функциональный размер. Общий размер *разрабатываемой ПС*[5], вычисляемый в УЕФ, определяется по формуле:

$$FP = Ксложж \cdot ПФР$$

8.2. Методы измерения, основанные на концепции FSM

8.2.1. Метод Feature Points

Метод *Feature Points (FeP)*, предложенный К. Джоунсом в 1985[6] году, представляет собой развитие FP-метода, предпринятое с целью улучшения точности оценок функционального размера программных систем, обладающих вычислительной сложностью, в частности, систем реального времени, операционных систем, встроенных систем, систем телекоммуникации и программных систем управления процессами [3].

Наряду с пятью базовыми логическими элементами FP-модели (ВЛО, ВИО, ВВД, ВЫВ, ЗАП), FeP-модель использует один дополнительный элемент - а*лгоритм* (АЛ). Этот элемент имеет по умолчанию вес 3, но может варьироваться от 1 до 10, в зависимости от сложности алгоритма. Все остальные элементы имеют по одному фиксированному значению веса сложности, что отражает снижение важности учета данных для определения размера ПК и устраняет проблему классификации сложности элементов по данным (таблица 8.23).

Таблица 8.23. **Веса базовых логических элементов модели FeP**

Вес элемента	Значение	Вес элемента	Значение
АЛв	1-10 (по умолчанию 3)	ЗАПв	4 (фиксированный)
ВВДв	4 (фиксированный)	ВЛОв	7 (фиксированный)
ВЫВв	5 (фиксированный)	ВИОв	7 (фиксированный)

ПФР вычисляется так же, как и в FP-методе.

Для учета сложности технических требований к ПС метод FeP предлагает

---

[5] FP-метод может применяться для определения размера ПС не только на стадиях ее разработки, но и при сопровождении и развитии. Однако мы ограничились только формулой расчета размера разрабатываемой ПС.

[6] Методы в обзоре упорядочены по годам их разработки.

оценивать 3 общесистемные характеристики по уровням сложности от 1 до 5 и по полученному суммарному уровню сложности характеристик (СУС) определять Кслож, пользуясь специальной таблицей, предлагаемой Software Productivity Research, Inc (таблица 8.24).

Таблица 8.24. **Значения *Кслож по* уровням сложности**

СУС по трем характеристикам	Значение Кслож	СУС по трем характеристикам	Значение Кслож
2	0.6	7	1.1
3	0.7	8	1.2
4	0.8	9	1.3
5	0.9	10	1.4
6	1.0	11	1.5

Общий функциональный размер ПС определяется по формуле:

$$FeP = Кслож \cdot ПФР$$

8.2.2. Метод Mark-II Function Points

В отличие от FP-метода, метод *Mark-II Function Points (MK-II)*, предложенный Ч.Саймонсом в 1988 году, основан на выделении только одного (а не пяти, как в FP-методе) базового логического элемента - *логической транзакции*. Функциональные требования рассматриваются как совокупность логических транзакций, а функциональный размер приложения представляется суммой размеров всех логических транзакций. Метод ориентирован на измерение размера и сложности обработки информации *приложений с базами данных* [4].

Каждая логическая транзакция - эквивалент элементарного процесса обработки данных. Она включает три компонента:

- *входные данные*, поступающие извне границ приложения;
- *выходные данные*, покидающие границы приложения,
- *обработку* хранимых в приложении данных, представленных в виде типов сущностей (реляционных таблиц) в третьей нормальной форме.

Каждый тип сущности трактуется как независимый, а ссылки на типы сущностей в каждой транзакции подсчитываются при определении размера приложения. Для сравнения, в FP-методе типы сущностей группировались в ВЛО и ВИО, а ссылки на типы сущностей подсчитывались как СОФ и ЭДФ по каждому ВВД, ВЫВ и ЗАП. Кроме того, в МК-II-методе различают *основные сущности* (primary entities), имеющие непосредственное отношение к прикладной области, и так называемую единую *системную сущность* (system entity) - совокупность таблиц, содержащих информацию исключительно реализационного характера.

Сложность входов и выходов транзакции определяется числом элементов данных (типов полей), которые они содержат (аналогичное понятие в FP-методе - ЭДО для ВЛО и ВИО).

Показатель функционального размера приложения рассчитывается по формуле:

$$ПФР = Nвх \cdot Wвх + Nвых \cdot Wвых + Nобр \cdot Wобр\ ,$$

где *N* - число входных данных, выходных данных и ссылок на сущности (основные и системную, соответственно), а *W* - относительные веса, присваиваемые элементам в зависимости от их «вклада» в сложность приложения. Значения этих весов - *Wвх = 0.58, Wвых = 0.26* и *Wобр = 1.66* - получены экспериментальным путем.

Учет сложности технических требований к приложению выполняется по 19 характеристикам (14 из которых взяты из метода FP). Оценивается уровень сложности каждой характеристики (от 0 до 5) и по полученному суммарному *СУС* рассчитывается *Ксложж* по формуле:

$$Ксложж = (СУС \cdot 0.005) + 0.65$$

Результирующее значение функционального размера приложения определяется как:

$$MkII\_FP = Ксложж \cdot ПФР$$

Коэффициент *Ксложж* модифицирует значение ПФР на ±12,5%.

Нужно отметить, что методы FP и Mk-II дают сопоставимые результаты оценивания размера на относительно небольших приложениях с базами данных (примерно 400 УЕФ). Для более крупных приложений метод Mk-II дает более точные значения, чем FP-метод. Результаты сопоставительного анализа Mark-II и FP представлены на сайте http://www.measuresw.com/library/Papers/Rule/MK2IFPUG.html.

Текущая версия метода – версия 1.3.1, датированная сентябрем 1998 - доступна на сайте UKSMA: www.uksma.co.uk/.

Методы FP и Mk-II имеют обширную инструментальную поддержку, в частности, в CASE-инструментах (например, в Oracle Designer'2000 (приложение 4).

8.2.3. Метод 3D Function Points

Метод *3D Function Points (3D_FP)*, разработан С. Вайтмайером для компании Boeing в 1992 году. Основная цель разработки метода - решение проблем, связанных с измерением функционального размера любых приложений, не попадающих в категорию информационных [5]. Предлагаемый метод расчета основан на предположении, что сложность любого разрабатываемого приложения можно рассматривать с трех точек зрения (или в трех измерениях, откуда и название модели - 3-Dimensions) - сложности *данных*, *функций* и *управления* (поведенческий аспект). В зависимости от назначения системы в приложении может преобладать то или иное измерение размера и сложности, например, данные - в информационных системах, функции - в программах инженерных расчетов и научных исследований, управление - в системах реального времени и т.д. Могут также существовать приложения, не имеющие явно выраженной доминанты, то есть смешенные приложения.

Для получения общего размера приложения модель 3D_FP оценивает уровень сложности одной-двух характеристик по каждому измерению, независимо от проблемной области или технологии реализации.

Для оценки сложности данных (*СлД*) непосредственно используется FP-метод (определяются УЕФ для ВЛО, ВИО, ВВД, ВЫВ и ЗАП).

Для оценки сложности функций (СлФ) устанавливается число *трансформаций* (*Nтранс*) - совокупность шагов процесса обработки данных (то есть число операций, преобразующих вход в выход) и семантических утверждений, управляющих выполнением операций. Каждая трансформация «взвешивается» по трем уровням

сложности («низкая», «средняя» и «высокая»), подобно тому, как это делалось в FP-методе.

Для оценки сложности управления (*СлУ*) определяется множество уникальных состояний (*Nc*), в которых может находиться процесс, а также множество переходов (*Nn*) между состояниями. Для состояний и переходов устанавливается только один уровень сложности - 1. Значение *СлУ* определяется по формуле:

$$СлУ = Nc + Nn$$

Общий функциональный размер приложения вычисляется по формуле:

$$3D\_FP = СлД + СлФ + СлУ$$

8.2.4. Метод Use Case Points для объектно-ориентированных приложений

Идея применения методологии FPA для определения размера объектно-ориентированных ПС, разрабатываемых по методологии А.Джекобсона OOSE (от Object Oriented Software Engineering), принадлежит С. Вайтмайеру (1992 г.).

Подход С. Вайтмайера базировался на диаграммах классов, включая посылку сообщений между классами [6]. Каждый класс рассматривался как внутренний логический объект, а сообщения, посылаемые за пределы границ системы, трактовались как транзакции. Существование внешних интерфейсных объектов отвергалось, поскольку к ним не могло быть непосредственного доступа. Таким образом, допущения этого подхода приводили к отклонению от концепции FPA. Во-первых, в FPA посылаемое сообщение необязательно является транзакцией (с точки зрения пользователя), а во-вторых, к ВИО есть доступ, по крайней мере, для чтения.

В 1993 году Г.Кернер из Objectory Systems inc. (ныне Rational Software) предложил новую меру размера объектно-ориентированных ПС, называемую *Use Case Points (UCP)* и основанную на анализе сущностей в Модели сценариев использования ПС [7], [8]. С помощью этой модели в OOSE описываются функциональные требования к ПС.

Модель сценариев использования UCM (от Use Case Model) имеет два вида элементов – акторы и собственно функциональные сценарии использования. Акторы – это субъекты, находящиеся вне системы, взаимодействующие с системой или обменивающиеся с ней информацией. Последовательность действий акторов с системой определяется сценарием использования.

Показатели *UCP* вычисляются в ходе анализа модели сценариев использования следующим образом.

Сначала выполняется классификация *акторов* – на «простые», «средние» и «сложные» - и каждому типу акторов приписывается вес (таблица 8.25).

Таблица 8.25. **Классификация акторов**

Тип актора	Назначение	Вес
Простой	Другая система (приложение) с определенным API-интерфейсом	1
Средний	Другая система (приложение), взаимодействующая через протокол, например, TCP/IP	2
Сложный	Человек, взаимодействующий с приложением через GUI или Web-страницу	3

Количество показателей функциональности для акторов ($УЕФ_a$) вычисляется по формуле

$$УЕФ_a = \sum_{i=1}^{3} N_i \cdot W_i$$

где N – число простых, средних или сложных акторов, а W - относительные веса, присваиваемые им в зависимости от «вклада» в сложность приложения.

Сценарии использования также делятся на «простые», «средние» и «сложные». Существует три способа определения сложности сценариев:

1) в зависимости от *числа транзакций* в описании сценария, включая вторичные сценарии (таблица 8.26, колонка «Транзакции»);

Таблица 8.26. **Классификация сценариев использования**

Тип сценария	Транзакции	Классы	Сложность реализации	Вес
Простой	<= 3	<= 5	Простой интерфейс пользователя, одна таблица БД	5
Средний	4 - 7	5 -10	Интерфейс пользователя средней сложности, 2 и более таблицы БД	10
Сложный	>= 7	>= 10	Очень сложный интерфейс пользователя или более 3 таблиц БД	15

2) в зависимости от количества классов, реализующих определенные сценарии (таблица 8.26, колонка «Классы»);

3) в зависимости от уровня сложности работ по реализации (таблица 8.26, колонка «Сложность реализации»).

«Взвешивание» сценариев использования производится так же, как и акторов. В результате определяется количество показателей функциональности для сценариев ($УЕФ_c$).

Показатель функционального размера объектно оиентированного приложения (*ПФРО*) вычисляется по формуле

$$ПФРО = УЕФ_a + УЕФ_c$$

Для учета *сложности реализации общесистемных характеристик* и *нефункциональных требований* к приложениям используются факторы, представленные в таблице 8.27.

Таблица 8.27. **Общесистемные характеристики**

№	Фактор (характеристика системы)	Вес
1	Распределенная обработка информации	2
2	Высокая производительность	1
3	Эффективность работы конечного пользователя	1
4	Сложная внутренняя обработка данных	1
5	Возможность повторного использования приложения	1
6	Простота установки	5
7	Простота использования	5
8	Переносность	2
9	Простота внесения изменений	1
10	Параллельная обработка	1
11	Включение специальных свойств защиты информации	1
12	Предоставление непосредственного доступа для третьей стороны	1
13	Предоставление специальных возможностей обучения пользователей	1

Каждому фактору приписана оценка сложности (вес), отражающая «вклад» соответствующего фактора в сложность приложения.

Рейтинг влияния факторов оценивается экспертным путем по уровням сложности от 0 до 5, где 0 – означает, что фактор не оказывает никакого влияния на выполнение работ по проекту системы, а 5 – оказывает очень сильное влияние.

Поправочный (настроечный) коэффициент, учитывающий сложность общесистемных характеристик *К1слож*, рассчитывается по формуле

$$К1слож = 0.6 + 0.01 \cdot (\sum_{i=1}^{13} W_i C_i)$$

где W_i – вес *i*-го фактора, C_i – экспертная оценка влияния *i*-го фактора на сложность реализации приложения.

Для учета условий среды разработки приложений используются факторы, представленные в таблице 8.28. Каждому фактору приписана оценка зрелости среды разработки (вес), отражающая «вклад» соответствующего фактора в успех проекта.

Таблица 8.28. **Характеристики среды разработки**

№	Фактор (характеристика среды разработки)	Вес
1	Владение основами методологии RUP	1.5
2	Знание проблемной области	0.5
3	Опыт объектно-ориентированной разработки	1
4	Лидерские задатки аналитика	0.5
5	Мотивация коллектива	1
6	Стабильность требований	2
7	Частичная занятость специалистов	-1
8	Сложность языка программирования	-2

Рейтинг влияния факторов среды разработки, указанных в таблице 8.28 оценивается таким же образом, как и факторов сложности приложений.

Поправочный (настроечный) коэффициент, учитывающий условия среды разработки *К2слож*, рассчитывается по формуле

$$К2слож = 1.4 + (-0.03 \cdot (\sum_{i=1}^{8} W_i C_i)$$

где W_i – вес *i*-го фактора, C_i – экспертная оценка влияния *i*-го фактора на успех проекта.

Общий функциональный размер объектно-ориентированного приложения рассчитывается по формуле

$$UCP = ПФРО \cdot К1слож \cdot К2слож$$

Нужно отметить, что структура UCP, предложенная Г.Кернером, не основывается на элементах, представляющих размер в FPA, что не позволяет сопоставлять результаты измерений приложений.

8.2.5. Отображение элементов объектно-ориентированного подхода на FPA

Способ «примирения» концепций OOSE и FPA предложил Т.Фецке [9], построив отображение элементов базовых моделей подхода OOSE на элементы подхода FPA

В OOSE модель сценариев использования является основой для построения других моделей – *модели объектов предметной области (домена)* и *модели анализа* (на самых ранних стадиях ЖЦ); модели проекта, модели реализации и модели тестирования (на более поздних стадиях ЖЦ).

Для построения отображения элементов подхода OOSE на FPA Т.Фецке использовал первые две модели – модель предметной области и модели анализа.

Модель предметной области включает объекты ПрО, между которыми устанавливаются взаимосвязи наследования и агрегации.

Модель анализа основана на типах объектов – объект-сущность, объект-интерфейс и объект-воздействие (управление). Объекты-сущности – это объекты ПрО, моделирующие информацию в системе и связанные отношениями наследования и агрегации. Объекты-интерфейсы моделируют поведение системы и информацию, которая представляет систему во внешней среде. Объекты-воздействия моделируют функциональность, не охватываемую другими типами объектов (например, вычисления, в которые вовлечены несколько объектов-сущностей). Такая типизация объектов поддерживает создание структуры, адаптируемой к изменениям требований к отдельным типам объектов (например, только к интерфейсу системы). Модель анализа строится по модели сценариев использования. Функциональность каждого сценария декомпозируется и назначается типизированным объектам.

Процедура построения отображения включает следующие этапы.

Этап 1. Отображение понятия «граница приложения». Граница приложения в OOSE устанавливается с помощью модели сценариев использования UCM, поскольку акторы находятся вне приложения, а сценарии определяют функциональность приложения. В сумме все акторы формируют исчерпывающий взгляд на измеряемое приложение. Однако понятие «актор» шире, чем «пользователь» и «внешнее приложение» в FPA, поскольку как акторы могут рассматриваться *функциональные* части (подсистемы) измеряемого приложения (например, функции печати или процедуры работы с БД). Эти акторы не должны учитываться при измерениях.

Правила отображения для этапа 1 таковы:

1.1) рассматривать каждого актора – человека - как пользователя системы;

1.2) рассматривать каждого актора- не человека, который является отдельной системой, не предназначенной для решения задач измеряемой системы, как внешнее приложение;

1.3) отвергать акторы (не являющиеся людьми), решающие задачи системы.

Этап 2. Идентификация элементов внутри границ приложения. В FPA внутри границ приложения идентифицируются логические объекты и элементарные функции.

Кандидатами на группы элементарных функций (ввода, вывода и запросы) являются группы сценариев использования, где один сценарий может учитываться как одна или более функций в зависимости от выполняемых в нем задач. В целом,

модель сценариев использования не дает достаточно информации для принятия решения о том, как учитывать каждый сценарий при расчете размера.

Т. Фецке предлагает следующую процедуру отображения для этапа 2:

2.1) в качестве кандидата на одну или несколько функций выбрать сценарии, имеющие непосредственное отношение к акторам, отобранным по правилам 1.1 и 1.2 на этапе 1;

2.2) выбрать все сценарии, расширяющие каждый сценарий, выбранный в 2.1. Эти расширения могут включать взаимодействие с пользователем или внешним приложением;

2.3) не выбирать никакие другие сценарии.

Понятию логических объектов соответствует в OOSE понятия объектов ПрО в модели ПрО (домена) и типов объектов в модели анализа, причем объекты-сущности отображаются непосредственно на логические объекты, объекты-интерфейсы связываются с представлением данных актору, а объекты-воздействия моделируют внутренние процессы.

Если используется модель анализа, множество анализируемых объектов ограничивается объектами-сущностями и применяются приведенные ниже правила отображения 2.4 и 2.5:

2.4) выбрать каждый объект-сущность в качестве кандидата на логический объект (если это не противоречит правилам 2.8 - 2.10);

2.5) не выбирать никакие другие объекты.

Если используется только модель ПрО, применяются правила 2.6 и.2.7:

2.6) выбрать каждый объект ПрО в качестве кандидата на логический объект (если это не противоречит правилам 2.8 – 2.10);

2.7) не выбирать никакие другие объекты.

Для объектов, связанных отношениями агрегации и наследования, Т.Фецке предлагает такие правила отображения:

2.8) объект ПрО или объект-сущность, являющийся частью другого объекта (агрегированный в другой объект), не может быть кандидатом на логический объект FPA, но может быть кандидатом на подгруппу данных объекта (ПДО), связанного с объектом верхнего уровня в агрегации;

2.9) абстрактный объект не может быть кандидатом на логический объект, но может быть кандидатом на ПДО каждого объекта, который наследует его свойства;

2.10) подобъекты конкретного объекта могут быть кандидатами на логический объект или его ПДО.

В FPA рассматриваются также логические объекты, которым может не быть соответствия в модели ПрО, хотя они и нужны пользователю. Это, например, сообщения об ошибках, подсказки. В этом случае применяется следующее правило:

2.11) если в сценарии предполагается использование логических объектов, которые не представлены в модели объектов, такие объекты все равно должны быть учтены в измерениях.

Этап 3. Определение типов элементов. На этом этапе элементы, выбранные на этапе 2, классифицируются в соответствии с требованиями FPA: кандидаты функции – как функции ВВД, ВЫВ и ЗАП, а кандидаты в логические объекты – как ВЛО и ВИО, и к ним применяются правила расчета УЕФ, установленные в FPA.

Этап 4. Определение весовых коэффициентов. Используется руководство IFPUG по расчету размера в FPA. Подсчитывается число подгрупп данных объек-

тов (ПДО), элементарных данных объектов (ЭДО) и функций (ЭДФ), ссылочных объектов функций (СОФ). Аналогом элементарных данных являются атрибуты объектов. Применяется следующие правила отображения:

4.1) атрибуты объектов являются кандидатами в элементарные данные объектов и функций, используемых для их чтения или ведения;

4.2) кандидаты в подгруппы данных объектов определяются по правилам 2.8-2.10;

4.3) каждый объект, читаемый или сопровождаемый в сценарии использования, учитывается как ссылочный объект функции для соответствующих функций тогда и только тогда, когда объект может быть идентифицирован как логический объект на этапе 3.

Таким образом, применение правил отображения элементов приложений, разрабатываемых по объектно-ориентированной методологии OOSE, позволяет расширить сферу традиционного использования FPA, обеспечить согласованность измерений объектно-ориентированных приложений в проектах ПС и получать сопоставимые результаты.

Парадигма «трансформации» подходов в FPA в дальнейшем нашла отражение в работах Дж.Кэммилера [10] (метод COP – *Component object points*), Г.Тиологли [11] (метод POP – *Predictive object points*), С.Эбрехи и О.Пастора [12] (метод OOmFP - *OO-Method Function Points*) и др.

8.2.6. Метод Object Points for ICASE

Метод *Object Points for ICASE (I_FP)*, разработанный Д. Бенкером в 1994 году, не обязательно связывать с традиционным объектно-ориентированным подходом. Под объектами здесь понимаются *экранные формы* (или экраны), *формы отчетов* (или отчеты) и *модули (*компоненты языка 3-го поколения). Это стандартные элементы приложений при разработке средствами интегрированных CASE-инструментов.

Расчет функционального размера производится следующим образом [13].

Определяется число объектов каждого вида. Далее для каждого из этих объектов устанавливается уровень сложности (как «простой», «средний» и «сложный») в зависимости от значений характеристик, представленных в таблице 8.29.

Таблица 8.29. **Матрица оценки сложности экранов и отчетов**

Для экранов				**Для отчетов**			
Число полей экрана	**Число и источник таблиц данных**			**Число полей отчета**	**Число и источник таблиц данных**		
	Всего <4: <2 серверов <3 клиентов	**Всего <8:** 2 или 3 сервера 3-5 клиентов	**Всего 8+:** >3 серверов >5 клиентов		**Всего <4:** <2 серверов <3 клиентов	**Всего <8:** 2 или 3 сервера 3-5 клиентов	**Всего 8+:** >3 серверов >5 клиентов
<3	простой	простой	средний	**0 / 1**	простой	простой	средний
3-7	простой	средний	сложный	**2 / 3**	простой	средний	сложный
>8	средний	сложный	сложный	**4+**	средний	сложный	сложный

Затем определяются относительные веса элементов в зависимости от уровня сложности (таблица 8.30)

Таблица 8.30. **Веса объектов**

Тип объекта	**Вес по сложности**		
	Простой	Средний	Сложный
Экран	1	2	3
Отчет	2	5	8
Модуль	-	-	10

И, наконец, функциональный размер приложения вычисляется как сумма полученных значений по всем объектам.

8.2.7. Метод Early Function Points. Раннее прогнозирование размера

Метод раннего прогнозирования функционального размера (EFP, от Early Function Points) предложен Роберто Мели (Италия) в 1997 году. EFP – это не альтернатива базовому FP-методу (IFPUG 4.0), а его адаптация для быстрого и раннего оценивания размера.

Метод основывается на идентификации объектов будущего приложения - логических данных и функциональности - на различных *уровнях* их детализации. Степень неопределенности оценки, представляемой диапазоном «минимум-максимум», тем выше, чем выше уровень, на котором останавливается процесс идентификации объектов приложения. Однако многоуровневый подход делает возможным использование любых знаний (в любой форме и любой глубины), оставляя на время некоторые вопросы о деталях без ответа, и наоборот, не используя существующей детальной информации о части системы.

Ключевыми объектами оценивания являются: *функциональные примитивы*, *микрофункции*, *функции*, *макрофункции* и *логические элементы данных*. Каждому из этих объектов может быть назначено множество значений УЕФ, основанных на статистических данных.

Логические элементы данных (ЛЭД) - группы связанных элементов данных, соответствующих ВЛО и ВИО в базовом FP-методе. Они классифицируются по пяти уровням сложности: «низкий», «средний», «высокий», «сложный», «сверхсложный». Первые три оценки выводятся по правилам FP-метода, вторые две – предназначены для оценивания крупных (составных) объектов данных без их детализации (например, объектов, состоящих из нескольких взаимосвязанных таблиц). На стадии раннего прогнозирования размера различия между ВЛО и ВИО не делаются. Матрица оценки сложности составных объектов представлена в таблице 8.31.

Таблица 8.31. **Матрица оценки сложности составных объектов данных**

Количество отдельных объектов в составном	**Уровень сложности**
2 - 4	Сложный
5 - 8	Сверхсложный

Четыре основные категории объектов-функций, характеризующих функциональность приложения таковы:

- *функциональные примитивы (ФПр)* – мельчайшие элементарные процессы, которые не могут быть далее детализированы. Это процессы ввода (ПрВВД), вывода (ПрВЫВ) и запроса (ПрЗАП), соответствующие объектам ВВД, ВЫВ и ЗАП в базовом FP-методе;
- *микрофункции (МиФ)* - набор из четырех функциональных примитивов -

создание, выборка, обновление и удаление, выполняемых над элементарными данными одного или нескольких объектов данных. То есть, это функции управления (ведения) данных;

- *функции (Фун)* - набор функциональных примитивов средней сложности, которые могут быть ассоциированы с операционными потребностями пользователей приложения. Матрица оценки их сложности представлена в таблице 8.32;

Таблица 8.32. **Матрица оценки сложности функций**

Уровень сложности	Количество функциональных примитивов
Низкий	5 – 12
Средний	13 – 19
Высокий	20 – 25

- *макрофункции (МаФ)* - набор функций средней сложности. Их можно ассоциировать с фрагментом приложения, с целым приложением или со всей системой потребителя. Матрица оценки их сложности представлена в таблице 8.33.

Таблица 8.33. **Матрица оценки сложности макрофункций**

Уровень сложности	Количество функций
Низкий	2 – 3
Средний	4 – 7
Высокий	8 – 12

Все объекты EFP-метода далее связываются с тремя весовыми категориями – «минимум», «среднее» и «максимум» (таблица 8.34).

Таблица 8.34. **Веса объектов EFP-метода по уровням сложности**

ЛЭД	Минимум	Среднее	Максимум	МиФ	Минимум	Среднее	Максимум
Низкий	5	6	7	Микроф.	16	18	20
Средний	8	9	10	**Фун**	**Минимум**	**Среднее**	**Максимум**
Высокий	13	14	15	**Низкий**	45	56	67
Сложный	14	18	22	**Средний**	73	91	109
Сверхсложн.	27	39	51	**Высокий**	106	133	160
ФПр	**Минимум**	**Среднее**	**Максимум**	**МаФ**	**Минимум**	**Среднее**	**Максимум**
ПрВВД	4	5	7	**Низкий**	151	215	280
ПрВЫВ	5	6	8	**Средний**	302	431	560
ПрЗАП	4	5	7	**Высокий**	603	861	1119

Эти значения весов могут корректироваться по мере сбора и обработки данных в реальных проектах.

Для определения прогнозируемого размера приложения в УЕФ необходимо определить количество объектов всех видов по каждому уровню сложности и полученные значения просуммировать.

Не существует достаточно четкого описания метода идентификации объектов – можно «отталкиваться» как от объектов-данных, так и от объектов-функций (деловых процессов пользователя). Многое зависит от знания и опыта экспертов, выполняющих анализ предметной области, в которой будет работать система. Некоторые рекомендации можно найти в работе Л.Сантилло и Р.Мели [14].

8.2.8. Метод Full Function Points и его разновидности

Метод *Full Function Points (FFP)* - наиболее интересное развитие методологии FPA. Он предложен в 1997 А.Эбреном из UQAM (университет в Квебеке, Монреаль, Канада) и поддерживается Международным Консорциумом COSMIC (Common Software Measurement International Consortium) [15], [16].

Метод в большей степени ориентирован на системы *реального времени* (в том числе системы телекоммуникации, системы управления процессами) и *встроенные системы* (системы управления устройствами), но может использоваться и в универсальном контексте (системы организационного управления, включая банковские, бухгалтерские и др., обеспечивающие обработку больших объемов данных). Однако он не предназначен для определения размера компонентов ПО, имеющих большую математическую сложность или поддерживающих аудио- и видео-образы (компьютерные игры, программы музыкальных инструментов и др.).

Главное преимущество COSMIC-FFP - использование процессо-ориентированного взгляда на функции программного обеспечения. В соответствии с моделью ПО, предложенной в FFP-методе, функциональные требования пользователей отображаются в совокупность функциональных процессов. Каждый из этих процессов представляет собой упорядоченное множество подпроцессов, выполняющих либо перемещение данных, либо их обработку (рисунок 8.3).

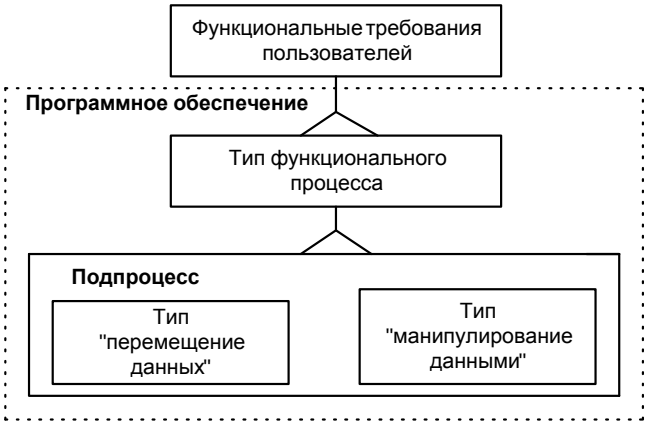


Рис.8.3. **Модель ПО в интерпретации FFP**

Модель COSMIC-FFP различает четыре типа подпроцессов перемещения данных – *вход*, *выход*, *чтение* и *запись*. Каждый из этих подпроцессов перемещает данные, принадлежащие только к одной группе данных (в рамках границ группы). Подпроцессы- входы перемещают данные извне – в границы приложения пользова-

теля, подпроцессы-выходы – изнутри границ приложения – наружу (пользователям или устройствам, которые обслуживаются программно), а подпроцессы чтения и записи перемещают данные из и в хранилища (рисунок 8.4).

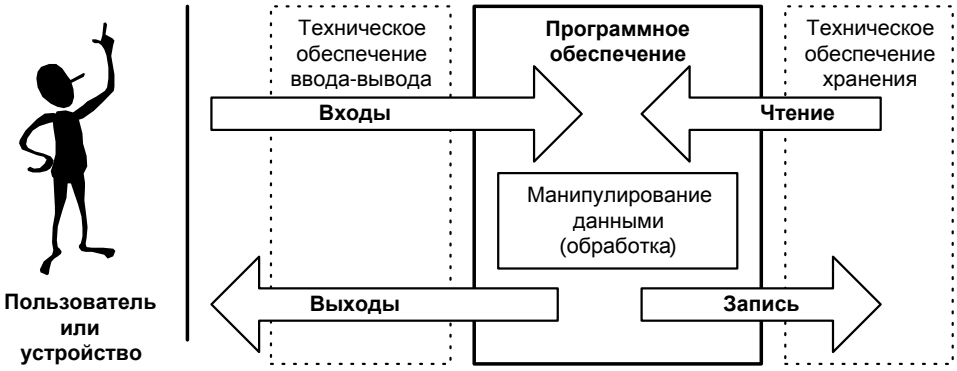


Рис.8.4. **Основной поток данных с позиций их функциональной обработки**

Подпроцессы манипулирования данными в данной версии (2.2) метода FFP не определены, поскольку концепция измерения процедур обработки данных остается пока не согласованной [17].

Программное обеспечение (на рисунке 8.4) включает все программные средства (и разрабатываемые, и не разрабатываемые). В зависимости от того, каких программных средств касаются требования пользователя, ПО разделяется на *слои*, с каждым из которых связывается свой набор функциональных процессов.

Например, если пользователь определяет функциональные требования к приложению и функциональные требования к графическому интерфейсу пользователя (GUI), но не предъявляет требований к другим компонентам ПО (например, драйверам устройств), модель ПО может включать два слоя (рисунок 8.5).

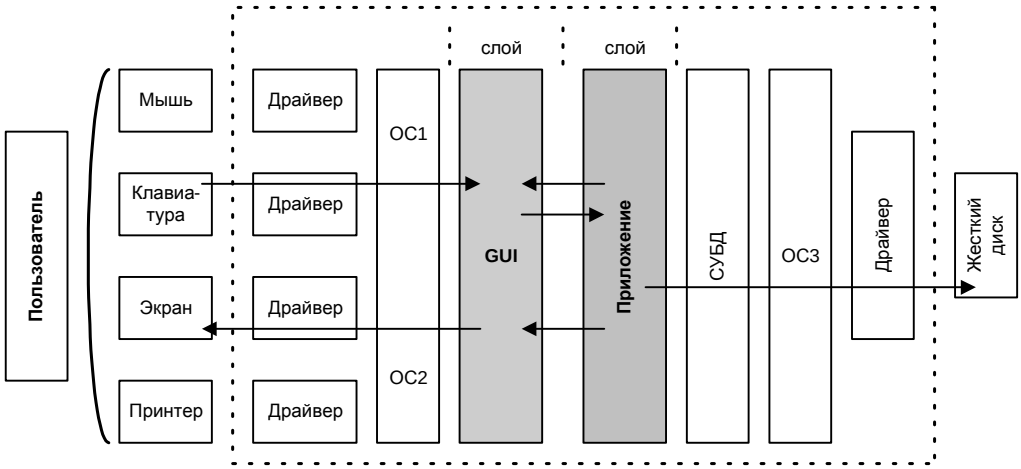


Рис.8.5. **Пример модели ПО для определения функционального размера**

Процесс измерения функционального размера ПО включает следующие шаги, детально описанные в [17]:

- проанализировать требования к ПО и взаимодействию ПО с техническими средствами, установить границы того фрагмента (части) ПО, размер которого будет определяться;
- идентифицировать все возможные функциональные процессы, триггеры событий (осуществляющих «запуск» этих процессов из среды слоя) и группы данных;
- из идентифицированных объектов построить модель ПО в контексте требований FFP (отобразить функциональные требования на модель FFP) и выделить слои;
- идентифицировать все подпроцессы в каждом функциональном процессе;
- оценить функциональные процессы в единицах CFSU (Cosmic Functional Size Unit) из расчета – одна единица размера на один подпроцесс перемещения данных:

$$РазмерCFSU(функционального\_процесса_i) = \sum_{\text{j}} Размер(входы_{\text{j}}) +$$

$$\sum_{\text{k}} Размер(выходы_{\text{k}}) + \sum_{\text{l}} Размер(чтение_{\text{l}}) + \sum_{\text{m}} Размер(запись_{\text{m}});$$

- Оценить объем *изменений* в каждом функциональном процессе по формуле:

$$РазмерCFSU(Изменения(функционального\_процесса_i)) =$$

$$\sum_{\text{j}} Размер(добавленные\_перемещения_{\text{j}}) +$$

$$\sum_{\text{k}} Размер(модифицированные\_перемещения_{\text{k}}) +$$

$$\sum_{\text{l}} Размер(удаленные\_перемещения_{\text{l}});$$

- оценить размер каждой части ПО (внутри слоя) и затем размер каждого слоя, агрегируя значения для всех новых и измененных функциональных процессов.

Очевидно, что метод COSMIC-FFP может эффективно применяться только в том случае, когда полностью сформирована спецификация требований пользователя, что редко бывает на ранних стадиях ЖЦ. Поэтому неудивительно, что для целей раннего прогнозирования функционального размера систем реального времени (для которых не мог быть применен метод Early Function Points) Р. Мели и А. Эбрен предложили новый метод - *Early & Quick COSMIC-FFP* (E&Q) [18].

Как и в методе EFP, в E&Q-методе ПО моделируется в виде иерархии функций - *функциональных процессов (ФП*). Они классифицируются по уровням сложности – «низкий», «средний» и «высокий» в зависимости от прогнозируемого количества подпроцессов. Далее вводится понятие *главный процесс (ГП)* – как множество средних ФП. Уровень сложности ГП, в зависимости от количества ФП, может также быть «низким» (6-12 ФП), «средним» (13 – 19 ФП) или «высоким» (20 - 25 ФП). По аналогии вводится *макропроцесс (МП)* как совокупность ГП средней сложности. Уровни сложности МП - «низкий» (2 - 3 ГП), «средний» (4 – 7 ГП) или «высокий» (8 -12 ГП). И, наконец, 4-й вид процессов – *типовые процессы (ТП)* – множество из четырех наиболее часто используемых функциональных процессов –

создание, выборка, обновление, удаление. Это разновидность главных процессов. Веса сложности указывают количество единиц CFSU, которое должно быть присвоено процессу.

Одна из важнейших концепций COSMIC FFP – *контекст измерения* системы, формируемый в результате определения *Целей* (назначения) измерения, *Сферы*[7] (масштаба) измерения и *Угла зрения* (взгляда) на измеряемый объект. Именно Угол зрения определяет уровень абстракции и детализации рассматриваемых элементов системы. Результаты измерений, выполненных по одной методологии, но под разными углами зрения, не могут сопоставляться и агрегироваться.

В отличие от методологий измерения размера первого поколения (FPA, Mark-II и др.), в COSMIC FFP объекты могут рассматриваться под углом зрения не только конечного *пользователя* (или системы, взаимодействующей с измеряемой системой), но и *разработчика*. Разработчик может, например, принять решение об использовании разных технологий реализации отдельных частей системы, что, в свою очередь, приведет к выделению тех или иных слоев системы, о которых пользователь и не подозревает. Поэтому, сравнивая результаты оценок размера, полученные по методу COSMIC FFP и другим, нужно обращать внимание на установленный Угол зрения. Например, расчеты размера бизнес-приложений по FPA и COSMIC FFP, выполненные под углом зрения пользователей дают вполне сопоставимые результаты.

Вероятно, ввиду наибольшей востребованности методов оценивания размера в сфере бизнес приложений, в 2005 году появилась адаптация COSMIC FFP для этой сферы [19]. В ней в качестве исходных элементов для анализа системы могут служить диаграммы «сущность-связь», диаграммы классов или сценарии использования системы.

В последнее время метод COSMIC FFP получил существенную инструментальную поддержку, в частности, в CASE-средах, предлагаемых Rational Software inc. (приложение 4).

8.3. Определение размера Web-приложений

8.3.1. Характеристика программного обеспечения для Web и подход к его измерению

В последнее десятилетие Интернет занял прочное место во многих сферах деятельности. По данным, опубликованным Cutter Consortium, уже к 2000 году количество Web-сайтов во «всемирной паутине» перешагнуло рубеж 196 миллионов [20].

Темпы программной индустрии для Web значительно превысили темпы формирования основ инженерии качества ПО для Web и, прежде всего, методов измерения и оценивания ПО – его размера, трудоемкости и стоимости.

Изменения, произошедшие за последние годы в практике разработки программного обеспечения, подытожены в таблице 8.35 [21].

---

[7] Например, слой, повторно используемый объект, пакет, приложение, решение для предприятия и др.

Таблица 8.35. **Сопоставление традиционной и Web-разработки ПО**

Характеристики	Традиционная разработка	Web-разработка
Основные цели	Разработка качественного продукта с минимальными затратами	Разработка качественного продукта на рынок за возможно наиболее короткий срок
Размер команды проекта	Средний и большой (сотни человек)	Малый (3-5 человек в команде)
Продолжительность	12-18 месяцев	3-6 месяцев
Применяемый подход к разработке	Основан на: требованиях; выпуске по фазам и/или приращениям; сценариях использования. Документированный	Основан на: RAD, сборке из блоков, прототипировании, RUP. Модельно-ориентированная разработка
Парадигмы и методологии	Объектно-ориентированные методы, генераторы, современные языки программирования (C++ и другие), CASE-средства	Компонентные методы, языки 4 и 5 поколения (HTML, Java и др.), визуализация (движение, анимация) и др.
Процессы	CMM-подобные	Не упорядочены
Программные продукты	В основном новые системы, основанные на коде, с небольшим количеством повторно используемых компонентов, со сложными внешними интерфейсами	Относительно простые объектно-ориентированные системы с большим количеством повторно используемых компонентов, с не сложным внешним интерфейсом
Персонал	Профессиональные разработчики с достаточным опытом	Дизайнеры графики, неопытные программисты, инженеры и др.
Технологии оценивания	SLOC и FPA-подобные модели, COCOMO и др.	До 2000 года не упорядочены

Методологии измерения, широко применяемые до 2000 года, ориентировались преимущественно на *традиционную разработку*. Действительно, создаваемые модели проверялись и уточнялись в течение более 20 лет; параметры моделей многократно подвергались калибровке по данным, полученным в результате анализа большого количества проектов; постепенно определялись, уточнялись и оптимизировались процессы, в которых применялись эти модели для планирования и управления разработкой.

Однако очевидные отличия Web-проектов от традиционных (представленные в таблице 8.35), обусловили сложность адаптации и применения существующих методологий к Web-разработке.

Проблемы оценивания Web-проектов, с которыми столкнулись специалисты, подытожены в таблице 8.36. Для сравнения указаны подходы, традиционно используемые для оценивания.

Первые приемлемые практические решения в данной проблематике появились только в 2000-м и последующих годах, после накопления и обработки достаточного количества эмпирических данных, лежащих в основе построения необходимых метрик и создания новых моделей и методов. Однако пока не достигнуто согласие специалистов о том, каким образом разрабатывать оценки для Web-проектов.

Таблица 8.36. **Проблемы оценивания Web**

Область Проблем	Традиционный подход	Проблемы для Web-разработки
Процесс оценивания	Использование аналогов и уроков, извлеченных из прошлого опыта. Общая концепция – по стандартам ISO/IEC 14598, ISO/IEC 15539	Оценка работы исключительно на основе мнений разработчиков (как правило, слишком оптимистичных)
Оценивание размера	Поскольку системы основаны на требованиях, используются методы SLOC или FPA-подобные	ПО использует множества Web-объектов (html, xml, апплетов и др.). Нет согласованных методов
Оценивание усилий	Усилия оцениваются с помощью регрессионных моделей. Для определения соотношений между переменными строятся графики данных по проекту	Усилия оцениваются путем разбиения работы на отдельные задачи, требующие приложения усилий. История по проектам практически отсутствует
Оценивание продолжительности	Рассчитывается как кубический корень от усилий	Расчеты с помощью корня кубического дают отклонения на порядок выше
Калибровка моделей	Для калибровки моделей с целью повышения их точности используются измерения по прошлым проектам	Измерения по прошлым проектам фиксируются как прецеденты (их слишком мало, чтобы использовать)
Анализ риска	Модели оценивания используются для анализа риска, а также для вычисления коэффициента отдачи от инвестиций и анализа затрат/ выгод	Анализ риска выполняется без применения моделей оценивания. Коэффициент отдачи от инвестиций не вычисляется, анализ затрат/выгод не производится

Программное обеспечение, разрабатываемое для Web, можно классифицировать по четырем категориям:

- Web-приложения,
- Web-интерфейсы к существующим приложениям,
- динамические Web-сайты,
- статические Web-сайты.

Web-приложения в полной мере предоставляют пользователю функциональные возможности программных приложений через Web-браузер. К Web-приложениям относят версии существующих приложений, предназначенные для работы в Web-среде, а также новые, ранее не существовавшие, типы приложений.

При их реализации используются HTML-формы, встроенные скрипты и динамически генерируемые HTML-страницы для ввода и отображения данных, возможности серверов по выполнению функций приложения, а также Web-интерфейс, как стандартная оболочка, обрамляющая приложение. Примеры Web-приложений – приложение, предоставляющее услуги электронной почты на почтовом сервере Hotmail (http://www.hotmail.com), расчетное приложение-форма с Web-оболочкой (Java-апплет), предоставляющее услуги Интернет-калькулятора (“Constructive Reals Calculator” - http://www.hpl.hp.com/personal/Hans_Boehm/crcalc/CRCalc.html) и др.

Web-интерфейсы к существующим приложениям обеспечивают для широкого круга пользователей удаленный доступ к ним посредством Web-браузера.

При их реализации используются HTML-формы, встроенные скрипты и динамически генерируемые HTML-страницы для взаимодействия с приложениями и предоставления пользователю услуг, подобных тем, которые обеспечиваются существующими интерфейсами приложений. Примеры Web-интерфейсов – регистрационные формы, предоставляющие услуги выбора туристических агентств и доступа к их программным приложениям, осуществляющим работу с клиентами (например, http://www.maxi.com.au).

Динамические Web-сайты предоставляют по запросам пользователей регулярно изменяемую и/или пополняемую информацию об организациях и лицах или по определенной проблематике. Используют динамически генерируемые сервером HTML-страницы и сопровождают данные в БД, подлежащие отображению на экране пользователя. Примеры – сайты новостей (например, http://www.theage.com.au).

Статические Web-сайты предоставляют пользователям предопределенную, редко изменяемую, информацию об организациях и лицах или по определенной проблематике. Организованы как гипертекстовые документы. Реализованы как коллекции жестко запрограммированных HTML-страниц с гипертекстовыми ссылками на другие страницы в пределах сайта и указателями на близкие по тематике сайты. Могут включать как текстовые, так и не текстовые элементы, а также HTML-страницы, поддерживающие элементарные функции взаимодействия пользователя с информационной средой. Разрабатываются с помощью простых инструментов, например, Microsoft FrontPage или вручную с помощью текстовых редакторов. Примеры статических Web-сайтов – сайты организаций (ISO, IFPUG, Microsoft и др.).

Все представленные категории ПО для Web обычно используют такие конструктивные элементы, которых нет в традиционных приложениях, а именно:

- не текстовые (мультимедиа) элементы – графика, видео и аудио;
- элементы с внешними источниками - публикации и отчеты, получаемые по сети Web;
- гипертекстовые ссылки к другим Web-сайтам.

Эффективность определения размера ПО для Web зависит от категории ПО и назначения получаемых оценок размера.

Для оценивания размера ПО первых трех категорий может применяться концепция *измерения функционального размера FSM*, а полученные оценки использоваться для определения трудоемкости разработки этого ПО.

FSM концептуально объединяет все представленные ранее методы определения размера ПО (раздел 8.2), полагая в основу оценивания, прежде всего, объем *функциональных возможностей* программного обеспечения, программная реализация которых требует затрат времени и средств.

Для статических Web-сайтов подход FSM эффективен только в следующих случаях:

- если изменения, которые необходимо сделать в Web-сайте, касаются его функциональности, например, добавляется новый запрос или расширяется существующий (запрос о новом виде продукции, уточняющая информация в запросе о продукции и др.),
- если возрастающие объемы данных или потребность в получении динамически изменяемой информации требуют преобразования типа Web-сайта на динамический.

Такие ситуации возникают, когда по структуре Web-сайт становится похожим на информационную систему.

В большинстве же случаев FSM-подход мало эффективен для статических Web-сайтов. Размер и трудоемкость создания таких сайтов в основном определяется количеством, размером и сложностью HTML-страниц, а не функциональностью, предоставляемой пользователям.

Двумя приемлемыми метриками для статических Web-сайтов являются:

1) размер и сложность *HTML-страницы*, вычисляемая исходя из
   - размера страницы в «словах» или «байтах»,
   - количества нетекстовых элементов на странице и элементов с внешними источниками, с учетом типа этих элементов,
   - количества и типа гиперссылок в рамках страницы и за ее пределы;
2) общее количество и тип *уникальных* нетекстовых элементов и элементов с внешними источниками, используемыми в рамках всего Web-сайта.

Метод определения размера статических Web-сайтов, основанный на использовании указанных метрик, называемый *Web Points* (*WP*), рассматривается в п.8.3.3.

Работы же над проблемой определения *функционального* размера ПО для Web остальных категорий (не статических Web-сайтов) проводятся в двух направлениях:

- создание новых методов в рамках FSM-подхода, как, например метод *Web Objects*, рассматриваемый в п.8.3.2.
- интерпретация правил IFPUG для Web-приложений (п.8.3.4).

8.3.2. Метод Web Objects. Определения размера Web-приложений

Для оценивания размера Web-приложений Д.Рифер в 2000 году предложил метрику W*eb Objects (WO)*, которая основывается на оценке сложности множества *конструктивных элементов*, составляющих Web-приложение [21].

Метрика использует уравнение Холстеда (см. главу 5) для оценивание размера *программы* по независимым от языка программирования параметрам - *операндам* и *операторам*, причем под программой в данном случае понимается конструктивный элемент Web-приложения:

$$S^* = N \log_2(n) = (N_1^* + N_2^*) \cdot \log_2(n_1^* + n_2^*)$$

где N – общее число Операндов/Операторов,

n – число разных Операндов/Операторов,

N_1^* – оценка общего числа Операндов,

N_2^* – оценка общего числа Операторов,

n_1^* – оценка числа разных Операндов,

n_2^* – оценка числа разных Операторов,

S^* – объем (сложность) работы по созданию программы.

Д.Рифер выделил несколько типов конструктивных элементов Web-приложения, оценки сложности которых «по Холстеду», используются в качестве *предсказателей* размера (таблица 8.37).

Таблица 8.37. **Предсказатели размера Web-приложения**

№ пп	Предсказатели	Примеры Операндов (объектов)	Примеры Операторов (действий)	Веса сложности[8]		
				Н	**С**	**В**
1	Блоки	Компоненты ActiveX, DCOM, OLE	Create, Apply, Call, Interface, Terminate	1	2	4
2	COTS-компоненты (включая код упаковщиков)	Коммерческие программы, библиотеки процедур	Initiate, Terminate, Apply, Bind, Customize, Export	2	4	6
3	Графические файлы	Шаблоны, рисунки, фотографии и др.	Apply, Align, Import, Export, Insert	2	4	6
4	Компоненты мультимедиа	Текст, видео, звук, 3D-объекты, плагины, мета-теги	Create, Cut, Paste, Clear, Edit, Animate	1	2	4
5	Компоненты, для которых возможен расчет FPA-подобными методами	Число таблиц на сервере, число таблиц на клиенте, % реюза	Transform (входы в выходы), Access, Generate, Interface	-[9]	-	-
6	Запросы на языке запросов	Число операторов языка запросов	Browse, Find, Search, Retrieve, Optimize	5	5	8
7	Повторно используемые компоненты	Алгоритмы, проекты, программы и др.	Create, Aggregate, Apply, Call, Interface, Terminate	2	4	6
8	Скрипты и макросы языка визуального программирования	Агрегации, контейнеры, поддержка реального времени	Create, Store, Distribute, Serialize, Edit, Generate	1	2	3
9	Другие			2	4	6

Каждый конструктивный элемент-предсказатель любого типа представлен множеством операндов (объектов) и операторов (действий над объектами) Примеры операндов и операторов также приведены в таблице 8.37.

С каждым типом предсказателей связан вес, приписываемый в зависимости от оцененного уровня сложности операндов и операторов Web-приложения. Значения весов получены Д.Рифером по реальным данным из 46 проектов [21].

Процедура определения размера Web-приложения такова.

Прежде всего, нужно выбрать пригодные для приложения элементы-предсказатели и составить их общий список. Если не найден подходящий тип предсказателя – использовать «другой».

Далее необходимо *для каждого конструктивного элемента из списка*:

- определить множество уникальных операндов и операторов;
- оценить «вклад» операндов и операторов в сложность разработки конструктивного элемента приложения, используя градацию по уровням сложности («низкий», «средний», «высокий») и установить вес по таблице 8.37;
- определить количество элементов каждого уровня сложности;

---

[8] Уровень сложности: Н – «низкий», С – «средний», В – «высокий».

[9] Размер элемента может быть учтен другим методом (см. разделы 8.1 и 8.2).

- применить известную процедуру агрегирования оценок и получить оценку размера конструктивных элементов одного типа в единицах web objects (WO_i, $i \in (1,9)$):

$$WO_i = \sum_{j=1}^{3} N_{ij} W_{ji}$$

где N_j – число простых, средних или сложных элементов, а W_j – относительные веса, присваиваемые им в зависимости от «вклада» в сложность приложения.

Наконец, просуммировать оценки для всех типов элементов приложения.

8.3.3. Метод Web Points. Определение размера статических Web-сайтов

Метод определения размера статических Web-сайтов в единицах *Web Points* (*WP*) был представлен Д.Клиэри на конференции IFPUG в 2000 году [22].

В качестве предсказателей размера сайта автор использует количество и сложность HTML-страниц. Сложность каждой страницы классифицируется как «низкая», «средняя» и «высокая» в зависимости от ее размера в «словах» и суммарного количества гиперссылок внутри страницы и за ее рамки, плюс количество не текстовых элементов на странице (таблица 8.38). Диапазоны значений, указанные в таблице, получены Д.Клиэри в результате анализа данных фирмы Charismatek и в ходе практического применения метода могут быть откалиброваны по имеющимся у разработчиков сайтов данным.

Таблица 8.38. **Оценивание сложности HTML-страницы**

Сложность		**Число гиперссылок**		
		0-5	**6-15**	**>15**
Количество слов	**0-300**	Низкая	Низкая	Средняя
	301-500	Низкая	Средняя	Высокая
	>500	Средняя	Высокая	Высокая

Уровням сложности приписывается вес: 4 – для «низкой», 6 – для «средней» и 7 – для «высокой» сложности каждой HTML-страницы. Общий размер сайта в единицах Web Points рассчитывается по формуле:

$$WP = \sum_{i=1}^{3} N_i \cdot W_i$$

где N – число простых, средних или сложных страниц, а W - относительные веса, присваиваемые им в зависимости от «вклада» в сложность Web-сайта.

Метод эффективен в тех случаях, когда, во-первых, функционально Web-сайт существенно отличается от информационной системы, а во-вторых, его изменения в основном касаются *содержимого* HTML-страниц (например, добавления блока текста или графического элемента).

8.3.4. Адаптация методов оценивания размера Web-приложений к методологии FPA

Представленные методы *Web Objects*, *Web Points* и некоторые другие, например, *Internet Points, Data Web Points* [23], обладают рядом недостатков, в частности:

- измерения применимы на *завершающих стадиях* разработки ПО,
- методы ориентированы на Web-приложения и статические Web-сайты,

- оценки получены эмпирическим путем в условиях применения определенных *технологий* реализации ПО для Web (например, реализация с помощью html или xml) и нуждаются в калибровке.

Поскольку ISO пока не предложила стандартизованных методов, разработанных в рамках концепции FSM и ориентированных на атрибуты, свойственные Web-приложениям, многие специалисты адаптируют ранее предложенные ими методы определения размера к методологии FPA.

Метод Web Objects (уточненная версия). В 2002 году Д.Рифер развил метод Web Objects, взяв за основу методологию FPA. Он уточнил спектр и описания конструктивных элементов Web-приложений, сократил общее количество предсказателей размера, специфичных для Web, до четырех и добавил их к показателям ПФР, принятым в FPA (таблицы 8.39 и 8.40) [24].

Таблица 8.39. **Предсказатели размера по методологии FPA**

Предсказатели	**Описание**	**Веса сложности**		
		Н	**С**	**В**
ВЛО	Внутренние логические объекты, используемые Web-приложением для хранения информации и сопровождаемые в нем	7	10	15
ВИО	Внешние логические объекты, на которые ссылается Web-приложение, но сопровождаемые другим программным приложением	5	7	10
ВВД	Логические элементарные бизнес-процессы вне границ приложения, направленные “внутрь” приложения с целью ведения его ВЛО, доступа к Объектам Мультимедиа, запуска Скриптов и доступа к Связям	3	4	6
ВЫВ	Логические элементарные бизнес-процессы, в результате выполнения которых данные покидают границы приложения в затребованном пользователями виде (отчеты, экраны)	4	5	7
ЗАП	Логические элементарные бизнес-процессы, в результате выполнения которых формируется запрос и выборка данных, покидающих границы приложения (например, простой просмотр данных)	3	4	6

Калибровки в новом варианте метода выполнены Д.Рифером по 64 Web-проектам всех категорий (в упомянутой ранее классификации).

Сначала определяется количество и сложность операндов и операторов первых трех видов конструктивных элементов Web-приложения (мультимедиа, блоков и скриптов) и подсчитываются Web Objects с учетом весов. Используется процедура оригинального метода Web Objects 2000 года.

Далее определяется сложность Связей (xml, html и запросов). Для подсчетов может использоваться методология, разработанная институтом SEI для учета строк кода - SLOC [1].

Определяется количество и сложность объектов Web-приложения, присущих традиционным программным приложениям (таблица 8.39). Может использоваться наиболее подходящий метод, совместимый с FPA (разделы 8.1 и 8.2).

И, наконец, путем суммирования всех полученных оценок, определяется размер Web-приложения в условных единицах функциональности – Web Objects.

Таблица 8.40. **Уточненные предсказатели размера Web-приложений**

Предсказатели	Описание	Веса			Рекомендации
		Н	**С**	**В**	
Объекты мультимедиа	Физические объекты Web-приложения, используемые для вывода информации в мультимедиа формате	4	5	7	Учитывают сложность встраивания в приложения аудио- и видео-файлов, графических образов (JPEG и др.), создания Web-страниц, публикуемых документов (на стороне клиента и сервера). Примеры уровней сложности: JPEG – *низкий*; A2b music, Microsoft-картинка – *средний*; PCX Image, XIF Image, AIFF Audio – *высокий*. Подсчет операторов выполняется отдельно для каждого объекта
Блоки (включая повторно используемые)	Логические объекты, используемые для конструирования Web-приложений и автоматизации их функций	3	4	6	Учитывают сложность разработки (приобретения) компонентов и библиотек, включая новые активные элементы – ActiveX, апплеты, агенты и др., статические элементы – COM, DCOM, OLE и др., повторно используемые элементы – логотипы и др., и исключая стандартные библиотеки среды разработки (на стороне клиента и сервера) Каждый уникальный блок в библиотеке учитывается отдельно. Уровень сложности устанавливается исходя из количества объектов: 1-50 – *низкий*, 51-250 – *средний*, более 250 – *высокий* Подсчет уникальных операторов выполняется отдельно для каждого блока
Скрипты	Логические объекты, используемые Web-приложением для связи внутренних объектов с блоками по определенным шаблонам	2	3	4	Учитывают сложность связывания html/xml-данных и генерации отчетов (автоматически), организации обращения к ODBC-источникам, интеграции и анимации приложений по установленной логике (с помощью GIF), и управления динамическим Web-контентом посредством настраиваемых паллет, масок, окон и команд (на стороне клиента и сервера) Каждый скрипт или сценарий использования учитывается отдельно. Уровень сложности устанавливается исходя из количества акторов: 1-3 – *низкий*, 4-6 – *средний*, более 6 – *высокий* Подсчет уникальных операторов выполняется отдельно для каждого скрипта
Связи (линки)	Логические объекты (xml, html и запросы), поддерживаемые Web-приложением с целью установления связей с внешними приложениями	3	4	6	Учитывают сложность разработки средств динамического связывания приложений, их интеграции, обращения к БД и другим приложениям Подсчитываются отдельно по числу строк (а не операндов и операторов). Может использоваться методика SEI для учета SLOC. Уровень сложности устанавливается так: html – *низкий*, запросы – *средний*, xml – *высокий*

Д.Рифер уточнил также коэффициенты расширения кода с учетом современных языков и средств разработки Web-приложений [24].

Конвертирование значения показателя функционального размера в эквивалентное число строк исходного кода Web-приложения (SLOC) на соответствующем языке программирования может выполняться с использованием данных, представленных в таблице 8.41.

Таблица 8.41. **Значения коэффициента расширения кода**[10]

Язык программирования	**SLOC/УЕФ**
Языки программирования 1-го поколения	320
C	128
Языки программирования 2-го поколения	107
COBOL (ANSI85)	91
FORTRAN 107	107
PASCAL	91
Языки программирования 3-го поколения	80
C++	53
Java для Web	32
LISP	64
ORACLE	38
Visual Basic	40
Visual C++	34
Визуальные языки для Web (по умолчанию)	35
Объектно-ориентированные языки (по умолчанию)	29
EIFFEL	20
PERL	22
Smalltalk	20
Объектно-ориентированные языки для Web (по умолчанию)	25
Языки программирования 4-го поколения	20
Crystal Reports	20
Генераторы программ (по умолчанию)	16
HTML	15
SQL для web	10
Электронные таблицы (по умолчанию)	6
Excel	6
Screen Painter	6
Языки программирования 5-го поколения	5
XML	6
MATHCAD	5

Метод OOmFPWeb. Для оценивания размера ПО для Web на ранних стадиях его создания О.Пастор и С.Эбрехи предложили метод OOmFPWeb, являющийся адаптацией ранее разработанного ими метода OOmFP для расчета размера объектно-ориентированного ПО [25].

Метод базируется на пользовательских требованиях к Web-приложению, представленных в *Концептуальной модели приложения*.

---

[10] См. также п.8.4.3, где обсуждается проблема подготовки и использования эталонных данных

Предполагается, что приложение разрабатывается с применением модельно-ориентированного подхода *OOWS* (от Object-Oriented Web-Solutions). Подход OOWS был предложен этими же авторами как расширение объектно-ориентированного подхода к разработке ПО с целью автоматической генерации кода по Концептуальной модели приложения, которая строится на ранних стадиях ЖЦ ПО с использованием UML-диаграмм [26].

Процесс концептуального моделирования включает два вида деятельности – проектирование информационного контента и проектирование навигации (структуры Web-приложения и сценариев его использования) и завершается построением Концептуальной модели, включающей ряд моделей, в частности *Модель объекта*, *Модель навигации* и *Модель представления (презентации)* приложения, используемые далее для определения размера ПО.

Все абстракции в моделях специфицируются в терминах классов, их структуры, поведения и функциональности.

Объектная модель в графической форме отображает классы, их атрибуты, сервисы (методы) и отношения между классами (устанавливаемые с помощью механизмов агрегации и наследования). В иерархии агрегации нужно указать тип агрегации – ассоциация или композиция. В иерархии наследования нужно специфицировать, будет ли специализация постоянной или временной. В первом случае отношение характеризуется с помощью соответствующего условия на атрибуты-константы. Во втором случае специфицируется условие на атрибуты-переменные или сервис типа «захват», который активизирует дочерний класс.

Навигационная модель получается путем дополнения Объектной модели навигационной семантикой Web-приложения, базирующейся на точках зрения каждого агента (определяющего функциональность заданного типа пользователя), идентифицированного в Объектной модели.

Примитивы Навигационной модели:

- *Навигационная карта* (Navigational Map) – отражает глобальный взгляд на Web-приложение с точки зрения определенного вида агентов. Включает:

Навигационный контекст (Navigational Context) – определяет базовые элементы взаимодействия пользователя с ПО, включая множество Навигационных классов и Навигационных ссылок в Объектной модели. Навигационная ссылка (Navigational relationship) определяет ненаправленную связь между двумя навигационными классами (отношение типа агрегации или наследования);

Навигационная связь (Navigational Link) – определяет попарные отношения, специфицирующие возможность достижения связи между двумя навигационными контекстами;

- *Навигационный класс* (Navigational Class) – включает набор атрибутов и набор операций, которые доступны определенному типу пользователей;
- *Контекстные ссылки* (Context Relationships) – указывают направления навигации между контекстами, представляющими заданный (целевой) навигационный контекст;
- *Контекстно зависимая ссылка* (Contextual Dependency Relationship) – обеспечивает предоставление дополнительной информации в текущем узле (без указания дальнейшей навигации).

Презентационная модель охватывает требования, касающиеся *представляемой информации*. Связывает следующие элементы построения презентации с навигационными контекстами:

- *Информационный макет* (Information layout). Например: реестр, таблица, дерево, «мастер-деталь» и др.;
- *Критерий упорядочения* (Ordering criteria), определяющий порядок выбора взгляда на запрошенную информацию (упорядочение по возрастанию/убыванию);
- *Организация информации* (Information organization), а именно:

Кардинальность (насыщенность) страницы (Page cardinality) – число прецедентов (instances), приходящихся на один «логический блок». Может быть постоянной (статической) или переменной (динамической);

Режим доступа (Access mode). Доступ к логическим блокам может быть последовательным или случайным (прямым).

Процесс оценивания размера Web-приложения основан на *отображении* концепции FPA на примитивы Концептуальной модели.

Отображение включает следующие шаги:

Шаг 1. Концептуальная модель анализируется с целью определения границ приложения и сферы измерения (отдельно измеряемых фрагментов приложения, например, отдельных Web-сервисов, сценариев для каждого агента и др.).

Шаг 2. Концептуальная модель анализируется с целью определения кандидатов в логические объекты-данные и функции.

Выделяются следующие типы объектов измерения:

- *классы*, реализующие навигацию (кандидаты в ВЛО),
- *взгляды* на объекты-данные вне рассматриваемого приложения (кандидаты в ВИО);
- *сервисы* (методы навигационного класса) (кандидаты в ВВД),
- *контекстные (гипертекстовые) ссылки* между навигационными классами (кандидаты в ЗАП),
- *навигационные контексты* (кандидаты в ВЫВ или ЗАП).

В качестве ВЛО учитывается каждый класс, ассоциируемый с агентом. Класс инкапсулирует множество элементов данных (атрибутов), представляющих состояние объектов класса. В качества ВИО учитываются взгляды, образующие фильтры на классы существующей (внешней) системы. В качестве ВВД учитывается каждый сервис, который представляет элемент функциональной логики объекта и активизируется для (по требованию) агента. В качестве ЗАП учитывается каждый навигационный контекст, определенный в Модели навигации. Контекстная информация выбирается без внешнего вмешательства в поведение системы.

Шаг 3. Определяется сложность каждого идентифицированного объекта (для каждого навигационного класса) в терминах уровней сложности (низкий, средний, высокий).

Так же, как и в FPA, сложность структур данных и функций устанавливается по количеству выделенных ПДО, СОФ, ЭДО и ЭДФ, правила учета (отображения) которых представлены в таблице 8.42. Для классификации объектов по сложности используются таблицы методологии FPA (см. раздел 8.1).

Таблица 8.42. **Правила отображения объектов**

Фактор сложности	Правила отображения
ЭДО для классов	По одному ЭДО для каждого атрибута (данных) класса
	По одному ЭДО для каждого атрибута в функции идентификации класса или взгляда, на который есть однозначная агрегированная ссылка
	По одному ЭДО для каждого атрибута в функции идентификации суперкласса данного класса
ПДО для классов	Одна ПДО для класса
	По одной ПДО для каждой многозначной ссылки типа агрегации (классов или взглядов)
ЭДО для взглядов	По одному ЭДО для каждого атрибута (данных) взгляда (не производного)
	По одному ЭДО для каждого атрибута в идентификационной функции, на которую ссылается класса с помощью однозначной ссылки типа агрегации
ПДО для взглядов	Одна ПДО для взгляда
	По одной ПДО для каждой многозначной ссылки типа агрегации с классом
ЭДО для сервисов	По одному ЭДО для каждого аргумента сервиса, связанного с данными
	Один ЭДО для способности приложения посылать сообщения
	Один ЭДО для действия в ходе выполнения сервиса (Accept/Cancel)
СОФ для сервисов	Один СОФ для класса, в котором определен сервис
	По одному СОФ для каждого нового класса, на который есть ссылка в аргументе сервиса, связанном с объектом
	Если определены ограничения целостности, учитывается один СОФ для нового класса, на который есть ссылка в формуле
	Если сервис представляет собой транзакцию, учитывается по одному СОФ для каждого класса, на который есть ссылка в формуле транзакции
	Если определена специализация по условию, учитывается по одному СОФ для каждого нового класса, к которому указан доступ в формуле специализации
	Если определена специализация по событию (событие захвата / освобождения (carrier/liberator)), учитывается по одному СОФ для каждого нового класса, для которого событие состоит в захвате/освобождении
	По одному СОФ для каждого нового класса, на который есть ссылка в формуле контрольного условия, установленного в *Диаграмме переходов состояний*
	.По одному СОФ для каждого нового класса, на который есть ссылка в формуле триггера, установленного в *Диаграмме взаимодействия*
	По одному СОФ для нового класса, на который есть ссылка в формуле предусловия, установленного в *Диаграмме перехода состояний*
ЭДО для навигационных контекстов	По одному ЭДО для каждого атрибута навигационных классов
	По одному ЭДО для каждой контекстно зависимой ссылки
	По два ЭДО для каждой контекстной ссылки
	По одному ЭДО для каждого сервиса, определенного в навигационных классах
	По одному ЭДО для каждой сервисной связи, ассоциированной с сервисом
	По одному ЭДО для каждого атрибута, определенного в формуле фильтра популяции. Фильтр популяции обеспечивает селекцию объектов класса
	По одному ЭДО для каждого атрибута, определенного в формуле фильтра доступа к информации
	По одному ЭДО для каждого атрибута, определенного в формуле индекса

Фактор сложности	Правила отображения
	По четыре ЭДО для каждого режима последовательного доступа к блокам данных (предыдущий, следующий, первый, последний, по номеру)
	По пять ЭДО для каждого режима случайного доступа к блокам данных (предыдущий, следующий, первый, последний, указанный номером)
	По одному ЭДО для кардинальности страницы (динамической)
	По одному ЭДО для каждого критерия упорядочения контекста навигации
	По одному ЭДО для каждой возможности указать выполняемое действие
	По одному ЭДО для способности приложения отображать сообщения (ошибки, диагностика и др.)
СОФ для навигационных контекстов	По одному СОФ для каждого навигационного класса
	По одному СОФ для каждого нового класса, на который есть ссылка в формуле фильтра популяции
	По одному СОФ для каждого нового класса, на который есть ссылка в формуле фильтра доступа к информации
	По одному СОФ для каждого нового класса, на который есть ссылка в формуле индекса

Шаг 4. Уровням сложности приписываются веса (в соответствии с FPA) и рассчитывается функциональный размер (отдельно данных и функций) Web-приложения в единицах OOmFPWeb, а затем и суммарный размер приложения по формулам

$$OOmFP_o = \sum_{i=1}^{n} OOmFP_{class_i} + \sum_{j=1}^{m} OOmFP_{views_i}$$

$$OOmFPWeb_{ф} = \sum_{i=1}^{n} OOmFP\text{Web}_{service_i} + \sum_{j=1}^{m} OOmFPWeb_{\text{NC}_i}$$

$$OOmFPWeb = OOmFPWeb_o + OOmFPWeb_{ф},$$

где $OOmFP_o$ – количество единиц размера объектов-данных, $OOmFP_{class}$ - классов, $OOmFP_{views}$ - взглядов, $OOmFPWeb_{ф}$ - функций, $OOmFPWeb_{service}$ - сервисов, $OOmFPWeb_{NC}$ – навигационных контекстов.

Шаг 5. Значения в единицах OOmFPWeb уточняются с учетом общесистемных нефункциональных требования к приложению.

8.4. Стандартизация методологии измерения размера

8.4.1. Базовый стандарт по измерению объема функциональности

Со времени опубликования методологии FPA появилось множество методов, основанных на концепции определения объема (размера) функциональности ПО. Однако различия в интерпретации оригинальных понятий и правил FPA привели к несовместимости этих методов, что снизило привлекательность каждого из них и стало препятствием на пути использования в качестве стандарта по определению функционального размера.

В конце 1992 года сообщества специалистов по измерению ПО из Австралии, Великобритании, Нидерландов и США создали рабочую группу WG12 при ISO/IEC JTC1/SC7, которая предложила проект стандарта, призванного устранить несовмес-

тимость методов и создать более строгую методологию измерения функционального размера – FSM (Functional Size Measurement). По мере выполнения этого проекта к нему подключались специалисты из ASMA (Australian Software Metrics Association), NESMA (Netherlands Software Metrics Association), UKSMA (UK Software Metrics Association), IFPUG и других организаций и сообществ.

Теперь это проект международного стандарта ISO/IEC 14143 «Information technology - Software measurement - Functional size measurement», который включает шесть частей, находящихся на разных стадиях разработки и согласования:

1. Часть 1. Определение концепции (в 1998 году принят как стандарт ISO/IEC 14143-1:1998) [27].

2. Часть 2. Оценивание соответствия методов определения размера программных средств стандарту ISO/IEC 14143-1:1998 (в 2002 году принят как стандарт ISO/IEC 14143-2:2002) [28].

3. Часть 3. Верификация метода измерения размера (объема) функциональных возможностей (в 2003 году утвержден как технический отчет ISO/IEC TR 14143-3:2003) [29].

4. Часть 4. Эталонная модель измерения размера (объема) функциональных возможностей (в 2002 году утвержден как технический отчет ISO/IEC TR 14143-4:2003) [30].

5. Часть 5. Определение функциональных доменов, для которых может применяться метод определения размера (в 2004 году утвержден как технический отчет ISO/IEC TR 14143-5:2004) [31].

6. Часть 6. Руководство по использованию серии стандартов ISO/IEC 14143 и связанных с ними международных стандартов (в 2006 году утвержден как стандарт ISO/IEC 14143-1:2006) [32].

В *части 1 стандарта* описаны общие основные особенности методов измерения функционального размера и определяется перечень *обязательных требований*, которым должен отвечать метод, претендующий называться методом FSM. Назначение этой части стандарта - обеспечить однозначную интерпретацию принципов FSM и облегчить сравнение мер функционального размера. Предполагается, что главными потребителями части 1 стандарта будут лица, которые привлекаются к разработке методов измерения функционального размера и нуждаются в знании основных характеристик этих методов, а также те, кто хочет проверить, отвечает ли конкретный метод определения размера ПС основным концепциям FSM.

Часть 2 стандарта предоставляет схему для оценивания степени *соответствия* определенного метода измерения функционального размера обязательным требованиям части 1 этого стандарта. Схема включает требования к составу и квалификации экспертных групп, входным документам для выполнения оценивания, процедурам оценивания и к оформлению результатов. Выполнение оценивания по этой схеме гарантирует объективность, беспристрастность, последовательность и повторяемость оценок.

Часть 3 стандарта нужна, прежде всего, тем, кто хочет *выбрать метод* оценивания размера, пригодный для использования по отношению к программным системам определенного *класса*, разрабатываемым для конкретной *функциональной области*. Для этого метод-кандидат на использование должен быть протестирован (верифицирован) путем сопоставления результатов вычислений этим методом и другим (ранее утвержденным) методом. Оцениваются такие показатели эффектив-

ности метода: *проверяемость* метода, *повторяемость* результатов, *воспроизводимость* процедур, *точность* оценок, *конвертируемость* меры в другую меру размера, *порог распознаваемости* изменений в функциональных требованиях пользователя, *применимость* в рамках определенной функциональной области.

Потребители различных методов измерения функционального размера иногда имеют претензии к разработчикам методов FSM, поскольку на практике эти методы дают плохие результаты. Это связано с тем, что не действует никакого соглашения о стандартных *эталонных наборах требований пользователя* (RUR, Reference User Requirements), применительно к которым могли бы рассматриваться претензии потребителей.

Цель *части 4 стандарта* - дать потребителям ориентиры, относительно которых можно было бы оценить эффективность методов FSM для различных классов программного обеспечения в различных программных средах. Используя эту часть стандарта:

- разработчики метода FSM будут иметь возможность проверить его соответствие *функциональные области*, в которой он может эффективно использоваться, и отшлифовать метод на эталонных RUR;
- оценщики методов FSM будут располагать эталонными объектами, по отношению к которым FSM может быть применен и сопоставлен;
- пользователи прекратят неправомерное использование непригодных для их классов ПС методов измерения размера;
- менеджеры проектов смогут использовать улучшенные эталонные данные для контрольного тестирования качества ПС и продуктивности проекта.

Эталонные требования пользователя включают функциональные требования пользователя (FUR, от Functional User Requirements), требования к качеству (QR, от Quality Requirements) и технические требования (TR, от Technical Requirements).

Стандарт содержит примеры эталонных требований пока только для двух крупных классов ПС – деловых (офисных) информационных систем и систем реального времени/управления.

Схема использования эталонной модели FSM показана на рисунке 8.6.

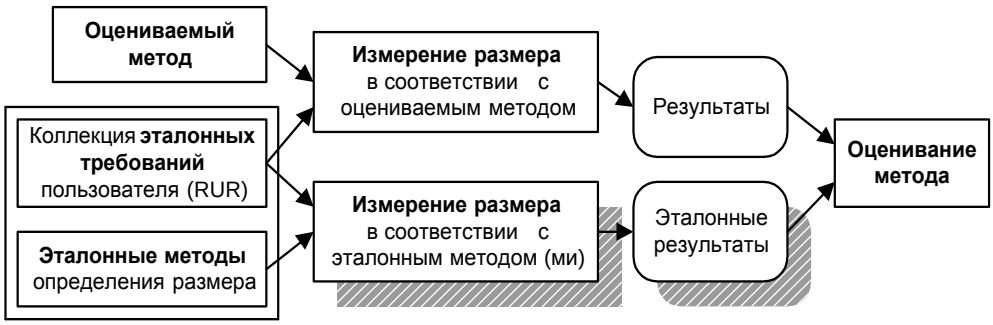


Рис.8.6. **Оценивание метода FSM по эталонам**

Цель *части 5 стандарта* – определить подход к *классификации* функциональных требований потребителя для применения в FSM. Используя эту часть стандарта:

- потребители FSM смогут оценить особенности функциональных требований к собственным ПС и классифицировать их по принадлежности к одной или больше функциональным областям;
- потребители смогут выбрать метод FSM, зарекомендовавший себя как подходящий для функциональных областей, отвечающих тем FUR, по которым оценивается размер;
- разработчики метода FSM будут способны четко определить функциональные области, для которых их метод может использоваться эффективно;
- оценщики методов и эксперты по методам FSM будут обеспечены четкими руководствами для определения функциональных областей, применительно к которым нужно определять эффективность методов;
- разработчики ПС смогут использовать эталонные требования в качестве ориентира для проверки полноты набора исходных функциональных и нефункциональных требований пользователя разрабатываемых ПС определенного класса.

Часть 6 стандарта – дает краткий обзор смежных стандартов, касающихся определения функционального размера по методологии FSM, и устанавливает взаимосвязь между:

- серией стандартов ISO/IEC 14143, предоставляющих определения и понятия в области измерения функционального размера и правила обеспечения совместимости и проверки методов, разработанных в рамках методологии FSM, и

- собственно стандартизованными FSM-методами, представленными отдельными стандартами - ISO/IEC 19761, ISO/IEC 20926, ISO/IEC 20968 и ISO/IEC 24570.

Эта часть определяет также процесс, предназначенный для оказания помощи потребителям в выборе Метода FSM, отвечающего их требованиям, и, кроме того, обеспечивает руководство по использованию оценок функционального размера.

Предполагается, что часть 6 стандарта будет востребована как потребителями оценок функционального размера, так и разработчиками методов FSM.

Стандарт ISO/IEC 14143 продолжает эволюционировать. О его текущем состоянии (а также состоянии других проектов стандартов) можно узнать на сайте подкомитета SC7: http://www.sqi.gu.edu.au/sc7/mirror/index_e_frameset.html.

8.4.2. Стандартизованные методы измерения объема функциональности

Наряду с разработкой стандарта ISO/IEC 14143 рабочая группа WG12 провела процедуры установления соответствия ряда методов определения функционального размера программного обеспечения требованиям ISO/IEC 14143 и стандартизации этих методов.

Нужно отметить, что стандартизованы методы расчета только *функционального* размера без учета сложности реализации нефункциональных требований (т.е. стандартизованы методы получения так называемых *не скорректированных* (unadjusted) оценок).

Первым методом, который признан полностью соответствующим требованиям стандарта ISO/IEC 14143, стал метод COSMIC Full Function Points (версия 2.2), представленный в разделе 8.2. Он оформлен как следующий стандарт

- ISO/IEC 19761:2003 “COSMIC-FFP - A Functional Size Measurement Method”.

За ним последовали стандарты методов IFPUG FPA (версия 4.1), Mark II FPA (версия 1.3.1) и NESMA FPA (версия 2.1).

- ISO/IEC 20926:2003 “IFPUG 4.1 Unadjusted functional size measurement method - Counting practices manual”
- ISO/IEC 20968:2002 “Mk II FPA - Counting Practices Manual”.
- ISO/IEC 24570:2005 “Software Engineering - NESMA FSM- method version 2.1 - Definitions and counting guidelines for the application of FPA”.

8.4.3. Использование эталонных данных для определения размера

С применение методов FSM связаны три проблемы: *квалификация специалистов*, выполняющих оценки размера (или зрелость организации в целом), *использование данных других фирм*-разработчиков ПО, если отсутствуют собственные данные об измерении размера ПС в УЕФ, и *конвертирование в SLOC полученных оценок* в УЕФ для их применения в моделях оценивания затрат, надежности и др.

Решить первую проблему достаточно сложно, так как, в отличие от многих других стран, в Украине пока не создана соответствующая учебная база (нет организаций и сообществ пользователей (наподобие национальных FPUG), имеющих официальные права готовить и аттестовать специалистов по методам FSM). Хотя *освоить* FSM-методы вполне можно самостоятельно, используя доступные в Интернет материалы, в частности полные руководства по IFPUG 4.1 и 4.2, COSMIC 2.2 и MK II FPA 1.3.1, ссылки на которые были даны в соответствующих разделах данной главы.

Для решения второй проблемы можно воспользоваться обширными сведениями об опыте определения размера, размещенными в Интернет известными организациями, центрами и лабораториями. Однако это отрывочные сведения, пользоваться которыми можно только в том случае, если есть уверенность в *идентичности характеристик проектов*, по которым они получены, и конкретных отечественных проектов, для которых они должны применяться. Данные о характеристиках проектов обычно размещаются (наряду с полученными оценками) в *базах данных* фирм, которые их публикую. Доступ к этим БД строго ограничен. В любом случае все используемые модели оценивания нуждаются в калибровке по *историческим данным*, собираемым организацией-разработчиком.

Нужно заметить, что существует международная группа по стандартизации эталонных данных о разработке ПО ISBSG (от International Software Benchmarking Standards Group), которой создан репозиторий проектных данных и регулярно выполняется их специальных анализ (http://www.isbsg.org/isbsg.nsf/weben/Special %20Analyses). Однако отчеты предоставляются только по предоплате.

Третья проблема касается *коэффициента расширения кода* для различных языков программирования, то есть соотношения оценок в УЕФ и SLOC. Со времени появления FPA было опубликовано немало таблиц пересчета оценок, составленных известными специалистами в данной проблематике, например, Дж. Кейперсом (1996 год, адрес http://www.theadvisors.com/langcomparison.htm). Даже в этой главе представлено две таких таблицы (таблица 8.7 и таблица 8.41) [11]. Таблицы пересчета

[11] Для уточнения адресов используйте строку запроса информации на поисковом сервере в Интернет: “Function point” “Languages table”

составляются на основе анализа данных большого числа завершенных проектов (как правило, более 500). Однако, данные в них все ж существенно отличаются.

С недавних пор регулярное опубликование соотношений FP/SLOC начал *Quality Software Management*, Inc.. Так, в апреле 2005 года QSM представил третью версию подготовленной им таблицы пересчета [33].

Таблица составлена по данным 2597 завершенных проектов, сведения о которых хранятся в базе данных QSM. Чтобы обеспечить высокое качество оценок FP и SLOC, для анализа были выбраны только одно- языковые проекты (в которых не использовались смеси языков программирования). Наряду с *усредненными* оценками коэффициентов расширения кода QSM опубликовал *диапазоны значений*, чтобы можно было увидеть разброс оценок по проектам, а также *медиану*, чтобы точнее отразить основную тенденцию в изменении оценок.

Кроме собственных данных QSM в таблице представлены «свежие» данные David Consulting, Inc. (таблица 8.43).

Таблица 8.43. **Коэффициенты расширения кода. QSM, версия 3.0**

Язык программирования	**SLOC/FP (данные QSM)**				**SLOC/FP (данные David Consulting)**
	Среднее	**Медиана**	**От**	**До**	
Access	**35**	38	15	47	-
Ada	**154**	-	104	205	-
APS	**86**	83	20	184	-
APP	**69**	62	32	127	-
Assembler	**172**	157	86	320	575-базовый, 400 макро
C	**148**	104	9	704	225
C++	**60**	53	29	178	80
C#	**59**	59	51	66	-
Clipper	**38**	39	27	70	60
COBOL	**73**	77	8	400	175
Dbase III	**-**	-	-	-	60
Dbase IV	**52**	-	-	-	55
Easytrive+	**33**	34	25	41	-
Excel	**47**	46	31	63	-
Focus	**43**	42	32	56	60
FORTRAN	**-**	-	-	-	210
FoxPro	**32**	35	25	35	-
HTML	**43**	42	35	53	-
Ideal	**66**	52	34	203	-
IEF/Cool, Gen	**38**	31	10	180	-
Informix	**42**	31	24	57	-
J2EE	**61**	50	50	100	-
Java	**60**	59	14	97	80
JavaScript	**56**	54	44	65	50
JCL	**60**	48	21	115	400
JSP	**59**	-	-	-	-
Lotus Notes	**21**	22	15	25	-
Mantis	**71**	27	22	250	-
Mapper	**118**	81	16	245	-
Natural	**60**	52	22	141	100
Oracle	**38**	29	4	122	60

Язык программирования	SLOC/FP (данные QSM)				SLOC/FP (данные David Consulting)
	Среднее	Медиана	От	До	
Oracle Developer 2K	**41**	30	21	100	-
Oracle FORMS	**42**	30	23	100	-
Pacbase	**44**	48	26	60	-
PeopleSoft	**33**	32	30	40	-
Perl	**60**	-	-	-	50
PL/1	**59**	58	22	92	126
PL/SQL	**46**	31	14	110	-
PowerBuilder	**30**	24	7	105	-
REXX	**67**	-	-	-	-
PRG II/III	**61**	49	24	155	120
SabreTalk	**80**	89	54	99	-
SAS	**40**	41	33	49	50
Siebel Tools	**13**	13	5	20	-
Slogan	**81**	82	66	100	-
SmallTalk	**35**	32	17	55	-
SQL	**39**	35	15	143	-
VBScript	**45**	34	27	50	50
Visual Basic	**50**	42	14	276	-
VPF	**96**	95	92	101	-
Web Scripts	**44**	15	9	114	-

Литература к главе 8

1. *Park R.* Software Size Measurement: A Framework for Counting Source Statements // Tech. Report CMU/SEI-92-TR-020. – Software Eng. Inst., Pittsburgh, 1992. – 242 p.
2. *Santillo L.* Early FP Estimation and the Analytic Hierarchy Process // ESCOM-SCOPE 2000, Munich, Germany, April 18-20. – 2000. – 9 p.
3. *Jones C.*, A short history of Function Points and Feature Points // Cambridge, Massachusetts: Software Productivity Research, Inc.- 1988.- 45 p.
4. *Symons C.*, Software Sizing and Estimating: Mk II FPA // Chichester: John Willey &Sons Ltd.- 1991.-57 p.
5. *Whitmire S.*, An Introduction to 3D Function Points // Sottware Development - 1995.- V.3, N.4.- P. 43-53.
6. *Whitmire S.*, Applying function points to object-oriented software models // in Software Engineering productivity handbook, ed. J.Keyes.- McGraw-Hill.-1992.- P. 229-244.
7. *Karner G*. Resource estimation for objectory projects // Objectory Systems.-1993.
8. *Use* case points // www2.fiit.stuba.sk/~bielik/courses/msi-slov/reporty/use_case_po-ints.pdf
9. *Fetcke T.*, Abran A., Nguyen T-H. Mapping the OO-Jacobson approach into FPA // in Proc. of TOOLS-23'97,August 1997 - http://www.fetcke.de/papers/Fetcke1997.pdf
10. *Kammelar J*. A Sizing Approach for OO-environments - http://www.iro.umontreal.ca/~sahraouh/qaoose/papers/Kammelar.pdf
11. *Teologlou G*. Measuring object oriented software with predictive object points // proceedings of ESCOM SCOPE 99 Conference, Herstmonceux, England, Shaker Publishing.- 1999.
12. *Abrahão S*, Poels G., Pastor O. A Functional Size Measurement Method for Object-Oriented Conceptual Schemas: Design and Evaluation Issues // www.feb.ugent.be/Fac/Research/WP/Papers/wp_04_233.pdf

13. *Banker D. et al.*, Automation Output Size and Reuse Metrics in a Repository-Based CASE Environment // IEEE Trans. Soft. Eng. - 1994.- SE - 20, N.3.- P. 169-186.
14. *Santillo L.,Meli R.* Early Function Points: some practical experiences of use // ESCOM-ENCRESS 98, Project Control for 2000 and Beyond, Rome, Italy, May 27-29. –1998. – 13 p.
15. *Full* Function Points: Counting Practices Manual / St-Pierre D., Maya M., Abran A., Desharnais J.-M., Bourque P.// Technical Report 1997-04, Quebec, Montreal, Canada.-1997.
16. *The* Common Software Measurement International Consortium (COSMIC) - http://www.cosmicon.com
17. *Measurement* Manual, The COSMIC Implementation Guide for ISO/IEC 19761:2003, Version 2.2, January 2003. COSMIC Measurement Practices Committee. - http://www.gelog.etsmtl.ca/cosmic-ffp/private/CFFP_MM_v2.2e.pdf
18. *Meli R.*, Abran A. et al. On the applicability of COSMIC-FFP for measuring software throughout its life cycle // ESCOM-SCOPE 2000, April 18-20. – 2000. – 10 p.
19. *Applying* COSMIC FFP to Business Application Software, version 1.0 – 2005. - http://www.gelog.etsmtl.ca/cosmic-ffp/updates/b.pdf
20. *Pastor O*. Functional size measurement for Web Application - http://www.ifi.uni-klu.ac.at/IWAS/HM/Staff/Heinrich.Mayr/Teaching/FSM.pdf
21. *Reifer D.J.* Web development: Estimating Quick-to-Market Software // IEEE Software. – nov/dec. – 2000. – P. 57-64.
22. *Cleary D*. Web-Based Development and Functional Size Measurement // in Proc. of IFPUG Annual Conf., San Diego, USA, 2000 – www.charismatek.com.au/_public1/pdf/ webfsm.pdf
23. *Abrahão S*. On the Functional Size Measurement of Object-Oriented Conceptual Schemas: Design and Evaluation Issues // www.dsic.upv.es/~sabrahao/PhDThesis_SilviaAbrahao.pdf
24. *Reifer D.J.* Estimating Web Development Costs: There Are Differences //www.stsc.hill.af.mil/Crosstalk/2002/06/reifer.html
25. *Abrahão, S.*, Pastor, O. Measuring the Functional Size of Web Applications // International Journal of Web Engineering and Technology.-2003.-V. 1, N. 1, P. 5-16.
26. *Pastor O*., Gomez J., Insfran E., Pelechano V. The OO-Method Approach for Information Systems Modeling: From Object-Oriented Conceptual Modeling to Automated Programming // Information Systems.-2001.-V. 26.-N.7.- P. 507–534.
27. *ISO/IEC 14143-1:1998*. Information technology -- Software measurement -- Functional size measurement -- Part 1: Definition of concepts.
28. *ISO/IEC 14143-2:2002*. Information technology -- Software measurement -- Functional size measurement -- Part 2: Conformity evaluation of software size measurement methods to ISO/IEC 14143-1:1998.
29. *ISO/IEC TR 14143-3:2003*. Information technology -- Software measurement -- Functional size measurement -- Part 3: Verification of functional size measurement methods.
30. *ISO/IEC TR 14143-4:2002.* Information technology -- Software measurement -- Functional size measurement -- Part 4: Reference model.
31. *ISO/IEC TR 14143-5:2004*. Information technology -- Software measurement -- Functional size measurement -- Part 5: Determination of functional domains for use with functional size measurement.
32. *ISO/IEC 14143-6:2006*. Information technology -- Software measurement -- Functional size measurement -- Part 6: Guide for use of ISO/IEC 14143 series and related International Standards.
33. *QSM* function point programming languages table. Version 3.0, april 2005 - http://www.qsm.com/FPGearing.html.

Глава 9. ОЦЕНКА ЗАТРАТ НА РАЗРАБОТКУ ПРОГРАММНЫХ СИСТЕМ

9.1. Обзор основных методов оценки затрат

9.1.1. Классификация методов и моделей оценки затрат

Хотя существует немало моделей и методов, предназначенных для прогнозирования трудоемкости, продолжительности и стоимости создания ПС, разработчики программных систем все еще часто руководствуются одним из двух законов:

- законом Паркинсона, который гласит: «любая работа стремится захватить *все* доступные ресурсы» (сколько бы много их не было), или
- законом конкурентных цен, согласно которому «если не воспользоваться предложением выполнить работу в заданные сроки и по заданной цене - этим предложением воспользуются *другие*» (независимо от того, приемлемы ли условия).

Основные категории методов оценивания затрат были названы Б. Боэмом еще в 1981 году [1] и кратко охарактеризованы ниже.

Методы экспертных оценок. В этих методах решение о факторах, влияющих на затраты ПС, и степени этого влияния принимается *экспертом* или группой экспертов. Примерами методов служат метод *Delphi* и его расширение (Wideband Delphi). Их основное преимущество - учет опыта предыдущих разработок, а недостаток - большая зависимость от компетентности экспертов. Эти методы применяются для экспертных оценок в любых областях, связанных с неопределенностями, в частности, для оценки размера и трудозатрат в тех случаях, когда отсутствуют исторические данные о соответствующих измерениях в прошлых проектах.

Метод оценивания «сверху-вниз». Этот метод основывается на учете характеристик проекта ПС в целом и применяется на ранних стадиях планирования ПС. Он прост в использовании и требует минимум знания деталей проекта. Его преимущества - тщательное оценивание общесистемных работ, связанных с интеграцией, документированием, контрольными функциями, управлением конфигурацией и др. Однако метод страдает неточностью и приводит к недооценке компонентов на нижних уровнях проекта, а, кроме того, не содержит механизмов для настройки параметров оценок в ходе ЖЦ ПС.

Метод оценивания «снизу-вверх». Этот метод предусматривает идентификацию и оценивание каждого отдельного программного компонента (или отдельной задачи) и последующее комбинирование результатов для получения оценок всего проекта ПС. Преимущества метода - возможность тщательного оценивания, недостатки - недооценка общесистемных работ, большая трудоемкость и сложность применения на ранних стадиях ЖЦ из-за отсутствия данных.

Метод аналогий. Представляет собой сопоставление характеристик предложенного к разработке проекта ПС и завершенных программных проектов-аналогов. Оценка по аналогам возможна как на уровне всего проекта ПС, так и на уровне отдельных компонентов. Основные достоинства метода - использование реальных данных проектов и накопленного опыта разработки и оценивания. Сложность применения обусловлена трудностью выявления и оценки отличий нового проекта от

завершенных проектов. Точность метода зависит от достоверности исторических данных об аналогах.

Методы параметрических уравнений - наиболее распространенные методы оценки затрат на ПС, основанные на использовании *математических моделей*.

Параметрами этих моделей являются, с одной стороны, исторические данные о классе проектов, к которым принадлежит оцениваемый проект, а с другой - рассчитанные значения объема ПС и числа выполняемых им функций, а также такие факторы стоимости, как язык программирования, методология проектирования, опыт разработчиков и другие.

Достоинства методов этой категории - возможность повторения оценок (выполнение расчетов по моделям), простота модификации входных данных и настройки формул с учетом условий разработки конкретных проектов (калибровка моделей). Однако результаты оценок по существующим моделям проблематичны при оценке проектов, разрабатываемых по новым технологиям и в новых условиях. В этом случае оценке обязательно должна предшествовать калибровка моделей.

По классификации Б.Боэма [1] ***математические модели*** подразделяются на:

- *линейные* - модели, в основе которых уравнение прямой, построенной по входным данным модели:

$$Расходы = a_0 + a_1 \cdot x_1 + a_2 \cdot x_2 + ... + a_n \cdot x_n ,$$

где $x_1...x_n$ - переменные факторов стоимости, а $a_0...a_n$ - последовательность коэффициентов, полученных по данным о завершенных проектах;

- *мультипликативные* - модели, в которых оценки стоимости имеют вид:

$$Расходы = a_0 \cdot a_1^{x_1} \cdot a_2^{x_2} \cdot ... \cdot a_n^{x_n} ,$$

где $x_1...x_n$ - переменные стоимостных факторов, а $a_0...a_n$ - последовательность коэффициентов, полученных по данным о завершенных проектах;

- *аналитические* - модели, имеющие вид некоторых функций (*f*), не являющихся ни линейными, ни мультипликативными:

$$Расходы = f(x_1, x_2, ..., x_n)$$

Особенностью аналитических моделей является то, что они содержат небольшое количество переменных и, следовательно, не могут учесть все разнообразие факторов, влияющих на стоимость ПС;

- *табличные* - модели, представляющие зависимость между факторами стоимости и расходами в виде матриц;

- *композиционные* - модели, использующие комбинацию всех или некоторых из упомянутых методов.

Достоинство композиционных моделей состоит, в первую очередь, в том, что с их помощью можно охватить широкий спектр ситуаций при разработке ПС. Кроме того, эти модели позволяют определить четкие алгоритмы действий по оцениванию ПС, и, таким образом, обеспечить инструментальную поддержку определения входных данных и расчетов расходов на разработку ПС.

Помимо перечисленных выше методов и моделей, классифицированных Б.Боэмом, для оценки затрат применяются ***современные методы:***

- нейронные сети,
- байесовские сети,
- методы имитационного моделирования и др.[2].

В этой главе кратко представлены наиболее известные методы и модели оценивания затрат на разработку ПС, в частности, *семейство моделей COCOMO* (раздел 9.2), а также указаны ссылки на доступные источники для получения дополнительной информации об этих и других методах и моделях. Поскольку многие методы реализованы в инструментах оценивания затрат (как коммерческих, так и бесплатных), в разделе 9.3 приведен перечень основных *инструментов* и их изготовителей.

9.1.2. Математическая модель SLIM

Математическая модель SLIM (Software Life Cycle Model) была разработана Л. Патнамом в 1978 году [3]. В этой модели предполагается, что трудозатраты на разработку ПО распределяются по закону, описываемому кривой лорда *Рэлея* (Lord Rayleigh).

Количество персонала, занятого в проекте по месяцам, представляется производной от функции общих затрат на проект и выражается уравнением вида:

$$\frac{dy}{dt} = 2 \cdot K \cdot a \cdot t \cdot \exp(-a \cdot t^2)$$

где $\frac{dy(t)}{dt}$ интенсивность использования персонала (количество занятого персонала по месяцам), K – общие трудозатраты на проект от его начала до завершения (включая сопровождение), a – параметр (константа), определяющий наклон кривой, t – общая продолжительность ЖЦ проекта.

На рисунке 9.1 изображен график кривой Рэлея, на котором область под кривой представляет *функцию общих трудозатрат* и, соответственно, может быть выражена в виде:

$$y(t) = \int_0^t \frac{dy(t)}{dt} = K \times (1 - e^{-at^2})$$

Очевидно, что при $t \to \infty$, $y(t)=K$.

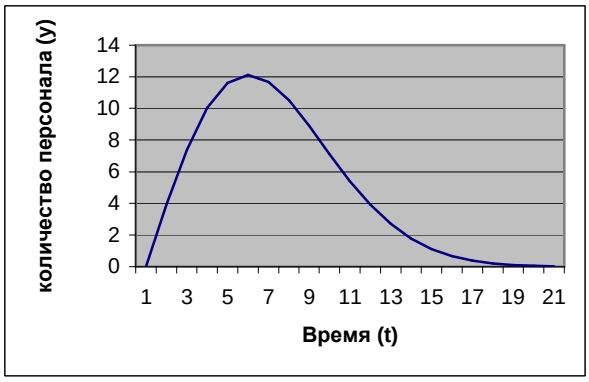


Рис. 9.1. **Форма кривой Рэлея (*K=1, a=0.02*)**

Поскольку ЖЦ проекта обычно делится на фазы, каждая из которых может моделироваться кривой Рэлея, общие результаты будут представлять сумму частных кривых (рисунок 9.2).

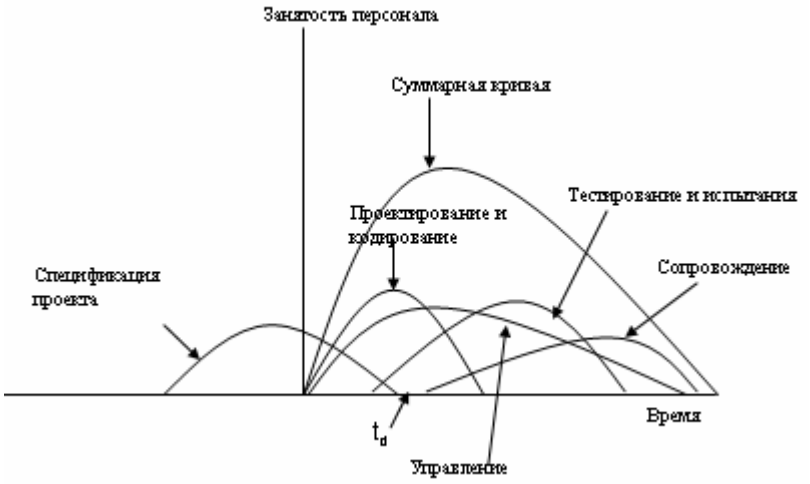


Рис. 9.2. **Семейство кривых Рэлея по стадиям проекта**

Точка на оси времени, в которой $\frac{dy(t)}{dt}$ максимальна, соответствует

$$a = \frac{1}{2 \cdot t_d},$$

где t_d – время, истекшее от начала проекта.

В модели SLIM *производительность персонала собственно разработки P* определяется из уравнения

$$P = \frac{S}{E}$$

где S – размер ПС (в SLOC), E – трудозатраты *на разработку*.

По эмпирическим данным реальных проектов в модели определено, что E примерно равно 40% от K, т.е. $E = 0.4K$.

Степень загрузки персонала ассоциируется со *сложностью (трудностью) проекта D*, определяемой по формуле:

$$D = \frac{K}{(t_d)^3}$$

Связь между *производительностью персонала P* и сложностью проекта D описывается уравнением:

$$P = aD^{(-2/3)}$$

После несложных преобразований легко получить уравнение условного «объема» (*размера проекта)*:

$$S = 0.4\ a\ (K)^{1/3}(t_d)^{4/3},$$

а из этого уравнения - *общие трудозатраты* на жизненный цикл ПС:

$$K = \frac{S^3}{C^3 (t_d)^4}$$

где $C = 0.4\alpha$ – константа, называемая индексом производительности, которая используется для учета стоимостных атрибутов - таких как методы разработки и управления проектом, инструменты разработки, квалификация и опыт персонала, сложность проекта и др. Ее значения могут варьироваться от 610 до 57314.

Тогда *трудозатраты на разработку*: $E = 0.4K$ или:

$$E = 0.4 \left[\frac{S}{C} \right]^3 \cdot \frac{1}{(t_d)^4}$$

Как видно из этого уравнения, трудозатраты увеличиваются в *кубической* зависимости от размера S при постоянном времени. При фиксированном размере, трудозатраты изменяются в обратной зависимости от времени.

Оптимальное *время разработки* (выпуска ПС) определяется следующим уравнением:

$$t_d = 2.4E^{1/3}$$

Описанная модель реализована в наборе инструментов управления проектом под общим названием SLIM (Software Lifecycle Management), в состав которого входят [4]:

– Slim-Estimate – планирование и оценивание проекта;
– SLIM-Control - отслеживание проекта;
– SLIM-Metrics - репозиторий для хранения метрик и исторических (эталонных) данных о проектах.

9.1.3. Математическая модель SEER-SEM

Математическая модель SEER-SEM (System Evaluation and Estimation of Resources – Software Estimating Model) была разработана на основе моделей Jensen и COCOMO в начале 90-х годов [5].

Подобно другим моделям, основными исходными данными для нее служат размер оцениваемого проекта (в строках кода или единицах функционального размера). Оценки размера переводятся во внутреннюю унифицированную меру размера S_e, называемую «эффективным размером», которая вычисляется по следующей общей формуле:

$$S_e = NewSize + ExistingSize \cdot (0.4 * Redesign + 0.25 \cdot Reimpl + 0.35 \cdot Retest)$$

где *NewSize* – размер нового исходного кода, *ExistingSize* – размер повторно используемого кода, *Redesign, Reimpl, Retest* – размеры изменяемого кода в связи с повторным проектированием, повторным кодированием и повторным тестированием, соответственно.

Как видно из этой формулы, S_e линейно растет с ростом нового кода и объема переделок существующего кода. Оценки размера в этом случае указываются в строках исходного кода (SLOC).

Модель позволяет использовать (взамен SLOC) меру размера в условных единицах функционального размера ПС. Предполагается, что размер определяется по методологии FPA и учитываются *только функциональные* требования (т.е. про-

изводится расчет не откорректированного функционального размера в единицах UFP (unadjusted function points)). В этом случае, для оценки *размера* S_e применяется формула:

$$S_e = L \cdot (F \cdot UFP)^{E/1.2}$$

где L – коэффициент расширения кода для определенного языка программирования, F – мультипликативный параметр, учитывающий влияние на затраты сложности и среды разработки, UFP – функциональный размер в UFP, E – параметр, учитывающий сложность разработки ПО определенного типа (варьируется от 1.04 до 1.2).

Для оценки *трудозатрат* применяется следующая формула:

$$K = D^{0.4} \cdot (S_e / C_{te})^{1.2}$$

где S_e – размер ПС, D – сложность управления персоналом, рейтинг, отражающий динамику увеличения количества персонала, участвующего в разработке, C_{te} – эффективность технологии, комплексная метрика, учитывающая атрибуты затрат, влияющие на производительность работы (эффективность работы разработчиков, атрибуты процесса разработки и характеристики продукта).

Для оценки *продолжительности* разработки применяется следующая формула:

$$t_d = D^{-0.2} (S_e / C_{te})^{0.4}$$

Область применения модели охватывает все фазы ЖЦ ПС, в том числе, сопровождение. Модель учитывает разные условия среды и конфигурации, такие как клиент-серверные приложения, настольные, распределенные, графические и т.п., а также наиболее распространенные методы разработки и модели ЖЦ.

Эта модель получила развитие в одноименном инструменте, разработанном Galorath Associates, Inc., SEER Technologies Division (http://www.gaseer.com/).

9.1.4. Методы экспертных оценок

Как уже упоминалось, экспертные методы широко применяются в областях с высокой степенью неопределенности.

В литературе по оценке затрат наибольшее внимание уделено описанию двух методов, классического метода Delphi и его расширения (WD - Wideband Delphi) [2,6,7]. Это «ручные» методы, их отличие заключается только в том, что расширенный метод Delphi предполагает проведение групповых дискуссий экспертов, а классический – нет. Расширенный метод Delphi использовался, в частности, для получения начальных оценок параметров при построении модели COCOMO II и ее расширений [2].

Далее кратко рассмотрен ***расширенный метод Delphi***. Процесс оценивания по этому методу подобен процессу инспекций и включает следующие шаги:

Шаг 1. Планирование. Формирование экспертной группы и назначение ее руководителя (модератора). Рекомендуемый состав группы - от 3 до 5 человек. Требования к экспертам – знание предметной области проекта, опыт разработки и оценивания. Экспертами могут выступать и члены проектной группы.

Шаг 2. Проведение предварительного совещания. В ходе совещания производится представление проблемы на рассмотрение экспертов. В данном случае,

проблема состоит в том, чтобы оценить затраты на проект. При проведении оценивания используются специально разработанные стандартные опросные формы, в которых эксперты указывают 3 оценки (пессимистическую, оптимистическую и наиболее вероятную).

Шаг 3. Индивидуальная подготовка. На этой стадии эксперты *независимо* формулируют предварительные результаты оценивания затрат.

Шаг 4. Совещание по оцениванию. В ходе совещания эксперты пытаются достичь согласия по результатам оценивания. Совещание проводится в несколько раундов, в каждом из которых в ходе дискуссии оценки уточняются.

На первом совещании модератор собирает анонимные ответы экспертов, выполняет первоначальную оценку и определяет отклонения. На последующих совещаниях в ходе групповых дискуссий оценки уточняются.

Итерации повторяются до достижения согласия. Результирующую оценку получают следующей формуле:

$$E_n = (E_{min} + 4E_i + E_{max})/6$$

где E_n – номинальная оценка, E_{min} – минимальная, E_i - наиболее вероятная, E_{max} - максимально возможная.

Другим популярным экспертным методом, который использует разделение проблемы на составные части, считается и ***метод декомпозиции работ WBS*** (Work Breakdown Structure). Он хорошо сочетается с методом Delphi для определения состава работ проекта и их оценки.

Имеются разные варианты применения этого метода. Согласно [2, 7] структурная декомпозиция работ проекта включает две составляющие: иерархическую структуру ПС и множество поддерживающих действий, требуемых для ее реализации (например, управление проектом, обеспечение качества и др.). С каждым элементом иерархии связываются стоимостные характеристики (трудозатраты, продолжительность и стоимость) как прогнозируемые, так и фактические. При последовательном применении этого метода для всех проектов формируется собственная база данных оценок распределения затрат по проектам организации. Эти оценки могут использоваться для разработки собственной модели оценивания, а также для калибровки параметров известных моделей.

9.1.5. Метод аналогий

В настоящее время существует ряд моделей и программных инструментов оценивания, использующих в разной степени метод аналогий. Этот метод получает за рубежом все большее распространение благодаря наличию и постоянному пополнению баз исторических данных о проектах.

Математический метод оценивания по аналогии реализован в инструменте *ANGEL* (ANaloGy softwarE tooL) [8, 9, 10], в состав которого входит БД характеристик промышленных проектов.

Процесс применения метода аналогий в системе ANGEL состоит из следующих шагов:

1. Выбор аналогов.
2. Оценка сходств и отличий.
3. Оценка качества аналогов.

4. Рассмотрение отдельных ситуаций.
5. Оценивание.

Выбор аналогов состоит в анализе БД оценок завершенных проектов-аналогов, имеющих подобные характеристики (размер, затраты, среду и прочее).

Сходство нового проекта с аналогами определяется на основании характеристик проекта, содержащихся в БД. Количество сравниваемых характеристик зависит от объема БД с характеристиками проектов-аналогов. Характеристики проекта представляются в виде n-мерного евклидового пространства, в котором каждая характеристика – это отдельное измерение. Наиболее сходные по характеристикам проекты будут иметь меньшее расстояние в пространстве.

Для *оценки качества* аналогов, выбранных для предсказания, могут использоваться такие метрики:

- абсолютная ошибка предсказаний (E-A);
- относительная ошибка (*RE*, от Relative Error)

$$RE = \frac{A - E}{A},$$

- отклонение относительной ошибки (MRE, от Magnitude of Relative Error)

$$MRE = \frac{A - E}{A} 100\%$$

- среднее отклонение относительной ошибки (MMRE, от Median Magnitude of Relative Error)

$$MMRE = \frac{1}{n} \sum_{i=1}^{n} MRE_i$$

- качество предсказаний (PRED, от Prediction)

$$PRED(q) = \frac{k}{n},$$

где A – реальное значение анализируемого показателя, E – оцененное (прогнозное) значение показателя, n – количество проанализированных проектов, q – допустимая погрешность предсказания, k – количество проектов (из общего числа n проектов), для которых $MMRE \le q$.

Необходимость рассмотрения отдельных *ситуаций* может возникнуть в том случае, если выбранный проект-аналог имеет характеристики, которые нужно исключить из рассмотрения.

Метод аналогий применяется также в инструменте *SPR KnowledgePLAN* [11]. В нем оценивание затрат и планирование работ по проекту также основано на обширной базе знаний о проектах, которая ежегодно обновляется. Инструмент позволяет оценивать затраты, проводить анализ причин (анализ "what if"), распределение сроков по фазам разработки и видам работ (в частности, по методологии RUP (Rational Unified Process)). Инструмент поддерживает оценку размера в строках кода, единицах функционального размера (по FPA), а также расчет размера методом аналогий. Имеет интерфейс с Microsoft Project и другими промышленными инструментами управления разработкой.

9.1.6. Нейронные сети

Для оценки стоимостных характеристик проектов могут использоваться известные теоретические методы, изначально применявшиеся для извлечения знаний, в частности, нейронные сети [2].

Согласно определению, приведенному в [12], нейронные сети представляют «класс аналитических методов, построенных на (гипотетических) принципах обучения мыслящих существ и функционирования мозга и позволяющих прогнозировать значения некоторых переменных в новых наблюдениях по данным других наблюдений (для этих же или других переменных) после прохождения этапа так называемого обучения на имеющихся данных».

Разработка сетевой модели включает определение *узлов* сети, количества *слоев* (layers) *связей* (нейронов) между узлами, количества связей в каждом слое и способа связывания узлов. Кроме того, определяются алгоритмы переходов между узлами, а также алгоритмы обучения сети на собранных реальных данных.

Пример нейронной сети представлен на рисунке 9.3 (из [12]).

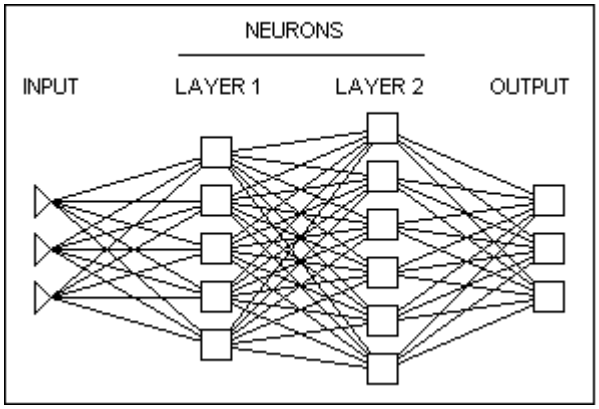


Рис. 9.3. **Пример нейронной сети**

Нейронная сеть для оценивания одной или нескольких стоимостных характеристик проекта содержит узлы, представляющие все возможные взаимосвязанные атрибуты процессов, продуктов и ресурсов проекта, и взвешенные функции оценивания стоимостных характеристик.

После построения архитектуры сети для проекта проводится ее *обучение* по историческим данным о реальных завершенных проектах.

Эти данные представляют *входные параметры модели (*например, размер, сложность, язык программирования, опыт персонала и др.) и *выходные параметры модели* (значения трудоемкости, продолжительности, стоимости). Параметры сети уточняются до получения приемлемой разницы между реальными и прогнозными данными [13].

В целом, процесс построения сети для оценивания крупного проекта достаточно сложен, однако, существуют программные инструменты для его поддержки, например, пакет STATISTICA Neural Networks [14].

Аналогичным образом для оценивания затрат могут применяться и байесовские сети [15].

Применение аппарата нейронных и байесовских сетей для оценивания затрат в настоящее время ограничено лишь экспериментальными исследованиями [2].

9.1.7. Динамические методы

Динамические методы используют предположение о том, что факторы, влияющие на стоимость и продолжительность проекта (например, опыт разработчиков, исходные требования, необходимость обучения и др.), изменяются на протяжении разработки (т.е. являются динамическими). Это существенно отличает динамические методы от других методов оценивания.

В основе методов лежит классический аппарат имитационного моделирования систем. Графически имитационная модель представляется в виде модифицированной сети (переменные величины в узлах которой определяют характеристики проекта (процессов, продуктов, ресурсов), используемых для моделирования) с положительными и отрицательными циклами обратной связи [2], а математически описывается системой дифференциальных уравнений первого порядка. Так, динамическая модель, предложенная в [16], позволяет предсказывать изменения стоимости, потребности в персонале и продолжительности в течение разработки проекта на основании первоначальных значений этих оценок. Эта модель была применена также для оценки влияния на затраты наличия повторно используемого ПО.

В [2] отмечено, что динамические методы хороши для планирования и контроля, но их трудно настраивать (калибровать).

9.1.8. Неформальные эмпирические методы

К неформальным эмпирическим методам оценивания трудозатрат, получившим распространение благодаря популярности методологии Extreme Programming (глава 11), можно отнести метод *Planning Game*.

Это неформальный метод планирования проекта, основанный на итеративном выполнении комбинации действий - экспертного оценивания текущего состояния проекта и идентификации той области проекта и задач в ней, которые должны быть решены в первую очередь [17].

В процессе планирования принимают участие все члены команды проекта (как заказчики (пользователи), так и разработчики), которые вовлекаются в обсуждение требований к ПС, облекаемых в форму так называемых «пользовательских историй» (“user stories”). Функциональные аспекты этих историй описывают заказчики, а разработчики оценивают требуемые усилия на их реализацию и риски.

Область проекта, для которой выполняется текущее планирование, устанавливается перечислением тех историй, программная поддержка которых (в очередном выпуске (релизе) ПС) может, с одной стороны, обеспечить наибольшую отдачу для бизнеса заказчика, а с другой, - быть реализована группой разработчиков за 1 - 3 недели. Каждая история, «планируемая» к выпуску, декомпозируется на несколько задач для разработчиков, и определяется трудоемкость их программирования и тестирования. Этот процесс может быть «проигран» неоднократно до достижения компромисса заказчика и разработчиков.

Процесс планирования по методу Planning Game охватывает весь ЖЦ проекта, повторяясь при подготовке каждого нового выпуска ПС.

9.2. Семейство моделей оценивания затрат COCOMO

9.2.1. Обзор моделей

В настоящее время на базе известной иерархической модели COCOMO[1] [1] разработано несколько моделей, применимых для разных условий разработки и типов программных систем. Поэтому сегодня можно говорить о семействе композиционных моделей COCOMO, основанных на общем подходе, и использующих сочетание экспертного и алгоритмического методов оценивания [18].

В таблице 9.1 перечислены основные модели семейства и их назначение.

Таблица 9.1. **Семейство моделей COCOMO**

Краткое название	Полное название	Назначение
COCOMO II	Constructive Cost Model	Полная трехуровневая модель оценки затрат для новых проектов ПС (только ПО)
COINCOMO	Constructive Incremental COCOMO	Учет и распределение затрат при итеративной разработке с интеграцией оценок для проекта
COQUALMO	Constructive Quality Model	Оценка количества оставшихся дефектов в программных продуктах
COPROMO	Constructive Productivity-Improvement Model	Оценка эффективности вложения ресурсов в новые технологии для улучшения производительности
COPLIMO	Constructive Product Line Investment Model	Поддержка оценивания стоимости линии продуктов и анализ отдачи от инвестиций
iDave	Information Dependability Attribute Value Estimation	Оценка и отслеживание отдачи от инвестиций в обеспечение надежности ПС (включая безопасность, безотказность, устойчивость и др.)
COPSEMO	Constructive Phased Schedule & Effort Model	Распределение оценок затрат по стадиям разработки. Применяется совместно с CORADMO
CORADMO	Constructive Rapid Application Development Model	Оценка и распределение затрат для небольших проектов, разрабатываемых по модели быстрой разработки (RAD)
COPROMO	Constructive Productivity-Improvement Model	Прогнозирование эффективности инвестирования ресурсов в новые технологии для увеличения производительности
COSYSMO	Constructive Systems Engineering Cost Model	Оценка затрат на протяжении полного ЖЦ (в соответствии с ISO/IEC 15288 [19])
COCOTS	Constructive Commercial-Off-The-Shelf Cost Model	Оценка затрат, связанных с оценкой и интеграцией готовых программных продуктов (COTS - Commercial-Off-The Shelf) в единую ПС
COSOSIMO	Constructive System of Systems Integration Cost Model	Оценка затрат, связанных с определением и интеграцией крупных интегрированных распределенных ПС

[1] Эта модель представлена в книге Б.Боэма, переведенной на русский язык и опубликованной в 1985 году. Там эта модель называется КОМОСТ. Мы же используем ее оригинальное название - COCOMO.

Модели COCOMO II и COINCOMO используют одну и ту же математическую модель (базовые уравнения затрат и продолжительности), но учитывают разные условия разработки (модель распределения затрат).

Независимые модели (COCOTS, COSYSMO, COSOSIMO) могут применяться как самостоятельно, так и с COCOMO II.

Рекомендации по выбору той или иной модели семейства для практического применения вкратце даны в таблице 9.2 [18].

Таблица 9.2. **Рекомендации по применению моделей для оценивания затрат**

Применять модель…	**Если оцениваемые работы включают…**
COCOMO II	Разработка программных компонентов и ПС в целом
COCOTS	Выбор, настройка и интеграция готовых к использованию продуктов (COTS-продуктов)
COSYSMO	Системная инженерия (проектирование, спецификация и интеграция независимых системных компонентов в единую систему)
COSOSIMO	Спецификация, приобретение и интеграция двух и более отдельно разработанных систем (включая сетевые решения, аппаратные и программные)
COCOMO II и COCOTS	Разработка программных компонентов и ПС с применением COTS-продуктов.
COSYSMO и COCOMO II	Системная инженерия и разработка ПО для одной системы
COSYSMO и COSOSIMO	Системная инженерия отдельных систем и интеграция многих систем
COCOMO II, COSYSMO, COCOTS, COSOSIMO	Системная инженерия, разработка ПО, интеграция крупномасштабных систем и COTS-продуктов

Остальные модели представляют расширения базовой модели COCOMO II и применяются вместе с ней.

Основное отличие всех моделей семейства – учет разных видов работ, факторов, влияющих на затраты (атрибутов стоимости и масштаба), распределение затрат по стадиям и видам работ, в зависимости от модели ЖЦ. Поскольку модели концептуально схожи и частично перекрываются по составу параметров, в настоящее время ведутся работы по их интеграции в единую модель оценивания [18].

Более подробно в главе рассмотрена базовая модель семейства - модель оценивания затрат COCOMO II. Описание назначения остальных моделей можно найти на сайте по адресу: *http*://www.sunset.usc.edu/research/

9.2.2. Модель COCOMO II. Общая характеристика

Модель COCOMO II учитывает только те затраты, которые связаны с разработкой *программного обеспечения* системы или отдельной *программной системы*[2] (и не учитывает затраты, связанные с поставкой и монтажом технических средств и другими общесистемными работами).

---

[2] Далее для удобства изложения используется понятие программная система.

Модель ориентирована на порционность поступления информации для оценивания на протяжении всего периода разработки ПС и является трехуровневой:

1. *Предварительная модель (Application Composition Model).* Обеспечивает предварительную оценку трудозатрат на ПС на *ранних* стадиях разработки. Модель предназначена для оценки трудоемкости прототипирования, а также разработки ПС с использованием интегрированных сред (ICASE).

2. *Предпроектная модель (Early Design Model).* Обеспечивает предварительную оценку трудозатрат на разработку как ПС в целом, так и отдельных программных компонентов на предпроектных стадиях ЖЦ. Может применяться для технико-экономического обоснования затрат на создание ПС, а также для распределения затрат по стадиям разработки.

3. *Детальная модель (Post Architecture Model).* Уточняет оценку, выполненную по Предпроектной модели. Обеспечивает поуровневую оценку трудозатрат на разработку ПС - от программных компонентов до программных модулей. Может применяться на стадиях проектирования и разработки ПС, а также при сопровождении.

В числе достоинств модели COCOMO II следует отметить:

1. Определенность. Модель включает определения основных понятий и количественных характеристик без ограничения типов ПС.

2 Точность. Точность модели согласована с большим количеством фактических данных, которые использовались для построения уравнения номинальных трудозатрат и определения параметров модели. Точность получения оценок по модели зависит от точности оценивания входных данных (размера ПС и параметров модели).

3. Объективность. Идеальная модель оценки трудозатрат характеризуется одним атрибутом - сложностью - и каждому завершенному проекту приписывается уровень сложности. Однако поскольку этот уровень сложности субъективен, невозможно определить его пригодность для прогнозирования оценок новых проектов. Данная модель повышает объективность оценок путем применения шкалы оценок, учитывающих влияние разных атрибутов на оценку трудозатрат.

4. Детальность. Модель позволяет последовательно уточнять оценки трудозатрат по мере разработки (от предварительных, грубых оценок на уровне ПС до детальных оценок на уровне модулей).

5. Область применения. Модель применима для оценки трудозатрат на разработку средних и крупных проектов ПС любого функционального назначения.

6. Простота применения. Модель относительно проста для понимания и применения. Входные данные для расчета определяются экспертным путем.

9.2.3. Оценка трудозатрат по Предварительной модели

Оценка трудозатрат по модели выполняется на уровне программной системы в целом. Прогнозируемый размер ПС определяется методом Object Points for ICASE (см. главу 8).

Расчет трудозатрат на разработку ПС производится по следующему алгоритму.

1. Вычисляется функциональный размер ПС. Для этого сначала методом Object Points for ICASE [20] определяется *общий* функциональный размер ПС (OP) по

всем составляющим ее информационно-функциональным объектам (экранам, отчетам, модулям), включая все объекты, которые будут использоваться повторно.

Затем определяется функциональный размер *разрабатываемых компонентов* ПС (NOP) по формуле

$$NOP = (OP - (100 - \%Reuse))/100$$

где %*Reuse* – доля (в процентах) повторно используемых объектов (экранов, отчетов и модулей).

2. Оценивается уровень производительности, PROD, как среднее значений атрибутов «Опыт работы и квалификация разработчика» и «Зрелость и возможности ICASE» (таблица 9.3).

3. Трудозатраты на разработку вычисляются в человеко-месяцах (чел-мес.) по формуле $T = NOP/PROD$

Таблица 9.3. **Связь стоимостных атрибутов и производительности**

Опыт работы и квалификация разработчика	Очень низкий	Низкий	Номинальный	Высокий	Очень высокий
Зрелость и возможности ICASE	Очень низкий	Низкий	Номинальный	Высокий	Очень высокий
PROD	4	7	13	25	50

9.2.4. Общие уравнения номинальных затрат

Рассматриваемые далее Предпроектная и Детальная модели COCOMO II используют *общие* уравнения оценки номинальных трудозатрат и отличаются *параметрами настройки* и *уровнем детализации* оцениваемых элементов ПС.

Номинальные трудозатраты на разработку ПС (средние) рассчитываются в человеко-месяцах при среднем продуктивном рабочем времени, равном 152 часа в месяц, по формуле (1).

$$T_{ном} = A \cdot V^{B} \qquad (1)$$

где $A = 2.45$ константа, полученная по результатам статистического анализа фактических данных более 80 реальных проектов[3],

V - предполагаемый размер ПС или программного компонента ПС в тысячах строк исходного кода KSLOC.

B - показатель степени при размере V, учитывающий изменение эффективности процесса разработки (производительности труда) при увеличении размера ПС. Определяется по формуле (2):

$$B = 0.91 + 0.01 \cdot \sum_{j=1}^{5} \Phi_j \qquad (2)$$

где Φ_j - значения соответствующих *коэффициентов атрибутов масштаба*.

Для оценок $\boldsymbol{T_{ном}}$ в других единицах измерения необходимо:

- для расчета в человеко-годах - разделить $T_{ном}$ на 12;
- для расчета в человеко-днях - умножить на 19.

---

[3] Здесь и далее значения числовых коэффициентов атрибутов соответствуют калибровке модели COCOMO II 2000 года [21].

Номинальная (средняя) продолжительность разработки рассчитывается по формуле (3):

$$Д_{ном} = 3.67 \cdot T_{ном}^{(0.38+0.2\cdot(B-1.01))} \quad (3)$$

9.2.5. Интегральные атрибуты масштаба разработки

Коэффициенты масштаба разработки ПС определяются в результате анализа соответствующих *интегральных атрибутов масштаба*, которые характеризуют сложность и условия разработки ПС и вычисляются на уровне ПС как для Предварительной, так и для Детальной модели.

Выбор представленных ниже интегральных атрибутов обоснован тем, что они являются основной причиной экспоненциального характера изменения трудоемкости разработки при увеличении масштаба разработки. Например, «Очень высокий» уровень оценок (рейтинг) этих атрибутов характеризует проект ПС, разрабатываемый относительно небольшим коллективом, обладающим большим опытом работы с аналогичными проектами, и в относительно стабильных условиях работы в организации. Кроме того, существует минимальная потребность в принятии новых решений по проблемам обработки данных и не требуется параллельная разработка аппаратных и общесистемных средств.

В таблице 9.4 приведены интегральные атрибуты масштаба, упорядоченные по шкале оценок от «Очень низкий» до «Очень высокий», и даны уровни ранжирования этих атрибутов.

Таблица 9.4. **Рейтинги интегральных атрибутов масштаба**

Атрибут масштаба	**Уровень оценки**					
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх высокий**
Новизна проекта (**НОВ**)	Полностью новый	Во многом новый	В какой-то степени новый	В основном известный	В большой степени известный	В полной мере известный
Жесткость проекта (согласованность) (**СОГЛ**)	Строгая согласованность	Допускаются некоторые компромиссы	Значительная	Относительная	Незначительная	При необходимости
Управление риском/ архитектурой (**УР**)	Слабое 20%	В какой-то мере 40%	Зачастую 60%	Как правило 75%	В основном 90%	Полное 100%
Коллективизм (**КОЛЛ**)	Сложное взаимодействие	Затруднено в некоторой степени	Зачастую коллективная работа	В основном коллективная работа	Высокая степень взаимодействия	Непрерывное взаимодействие
Технологическая зрелость разработки (**ТЗР**)	CMM Уровень 1 (ниже среднего)	CMM Уровень 1 (выше среднего)	CMM Уровень 2	CMM Уровень 3	CMM Уровень 4	CMM Уровень 5

Каждому уровню оценок каждого интегрального атрибута масштаба соответствует числовой коэффициент масштаба Φ_j (таблица 9.5). Шкала оценок атрибутов масштаба построена в порядке уменьшения влияния атрибута на трудозатраты.

Таблица 9.5. **Значения коэффициентов масштаба**

Φ_j Интегральный атрибут масштаба	Уровень оценки					
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх высокий**
НОВ	6.20	4.96	3.72	2.48	1.24	0.00
СОГЛ	5.07	4.05	3.04	2.03	1.01	0.00
УР	7.07	5.65	4.24	2.83	1.41	0.00
КОЛЛ	5.48	4.38	3.29	2.19	1.20	0.00
ТЗР	7.8	6.24	4.68	3.12	1.56	0.00

Для учета атрибутов масштаба в показателе степени *B* (формула 1) необходимо выполнить следующую процедуру.

Процедура определения коэффициентов масштаба разработки:

Шаг 1. Экспертным методом оценить каждый интегральный атрибут масштаба.

Если какой-либо интегральный атрибут определить сложно, - следует оценить его единичные атрибуты, а затем определить субъективные средневзвешенные оценки интегрального атрибута.

Шаг 2. Для каждого интегрального атрибута по таблице 9.5 определить соответствующий коэффициент масштаба

Шаг 3. Просуммировать коэффициенты и вычислить параметр B.

Так как атрибуты масштаба, перечисленные в таблице 9.4, являются интегральными, они могут оцениваться по совокупности *единичных атрибутов*.

Описание интегральных атрибутов масштаба.

***Новизна проекта* (*НОВ*)**- отражает степень влияния на трудозатраты уровня понимания целей и задач разрабатываемой ПС, а также опыта работы коллектива в предметной области (таблица 9.6).

Таблица 9.6. **Единичные атрибуты интегрального атрибута НОВ**

Новизна проекта	**Уровень оценки**		
	Очень низкий	**Номинальный-высокий**	**Очень высокий**
Понимание целей ПС в организации	Общее	Значительное	Полное
Опыт в разработке данного типа ПС	Умеренный	Значительный	Большой
Необходимость разработки новых архитектур и методов (алгоритмов) обработки данных	Большая	Умеренная	Минимальная
Необходимость одновременной разработки нового аппаратного обеспечения и операционных процедур	Большая	Умеренная	Незначительная

***Жесткость (согласованность) проекта* (*СОГЛ*)**- этот атрибут отражает степень влияния на трудозатраты необходимого уровня согласованности проекта. Оценивать этот атрибут можно непосредственно либо по единичным атрибутам, представленным в таблице 9.7.

Таблица 9.7. **Единичные атрибуты интегрального атрибута СОГЛ**

Жесткость (согласованность) проекта	**Уровень оценки**		
	Очень низкий	**Номиналь-ный-высокий**	**Очень высо-кий**
Необходимость строгой согласованности с предопределенными требованиями к ПС	Полная	Значительная	Небольшая
Необходимость строгого соответствия спецификациям внешних интерфейсов	Полная	Значительная	Небольшая
Материальное стимулирование досрочного выполнения работы	Высокое	Среднее	Малое

***Управление риском/архитектурой* (*УР*)**- этот атрибут отражает влияние на трудозатраты уровня управления риском проекта ПС (таблица 9.8).

Таблица 9.8. **Единичные атрибуты интегрального атрибута УР**

Управление риском/архитектурой	**Уровень оценки**					
	Очень низ-кий	**Низ-кий**	**Номи-наль-ный**	**Высо-кий**	**Очень высо-кий**	**Сверх высо-кий**
Идентифицирует ли план управления риском (при наличии) критические к риску элементы ПС, устанавливает ли контрольные точки в ЖЦ ПС для проверок и устранения риска	Нет	Слабо	В некоторой степени	В значительной степени	Почти полностью	Полностью
Соответствуют ли плану управления риском продолжительность, бюджет и сроки проверок и устранения риска	Нет	Слабо	В некоторой степени	В значительной степени	Почти Полностью	Полностью
Доля (процент) в продолжительности разработки, приходящаяся на обоснование архитектуры ПС (проектных решений) при заданных общих целях проекта	5%	10%	17%	25%	33%	40%
Доля (процент) разработчиков высокой квалификации (аналитиков), которыми располагает проект	20%	40%	60%	80%	100%	120
Существует ли инструментальная поддержка для устранения рисков элементов, верификации проектных решений	Нет	Неболь шая	Весо-мая	Хоро-шая	Почти полная	Полная

Управление риском/архитектурой	Уровень оценки					
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх высокий**
Уровень неопределенности при выборе ключевых элементов архитектуры проекта (интерфейсов, технических средств, технологии и др.)	Очень большой	Значительный	Существенный	Определенный	Малый	Совсем малый
Число и критичность элементов, подверженных риску	>10 критичных	5-10 критичных	2-4 критичных	1 критичный	>5 не критичных	<5 не критичных

Оценка «Сверх высокий» не будет учитывать влияние этого атрибута на трудозатраты. Значения коэффициентов данного атрибута убывают по мере увеличения уровня оценки по шкале.

***Коллективизм* (*КОЛЛ*)**- этот атрибут отражает степень влияния на трудозатраты слаженности работы коллектива разработчиков и пользователей (таблица 9.9).

Таблица 9.9. **Единичные атрибуты интегрального атрибута КОЛЛ**

Коллективизм	Уровень оценки					
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх высокий**
Согласованность целей коллектива	Минимальная	Незначительная	Относительная	Значительная	Большая	Полная
Способность и готовность членов коллектива приспосабливаться к целям других членов	Малая	Незначительная	Относительная	Значительная	Большая	Полная
Опыт участников работы в составе данного коллектива	Нет	Малый	Незначительный	Относительный	Значительный	Большой
Формирование коллектива по принципу доверия и взаимодействия	Нет	В очень малой степени	В незначительной степени	Относительно	В значительной степени	В большой степени

***Технологическая зрелость процесса разработки* (*ТЗР*)**- это интегральный атрибут, отражающий зависимость трудозатрат от эффективности организации и управления процессами разработки ПС в организации.

Процедура определения значения атрибута основана на использовании модели оценки технологической зрелости СММ, которая подробно описана в главе 12.

Для оценивания атрибута «Технологическая зрелость процесса разработки» нужно определить *уровень соответствия* деятельности по разработке ПС ключе-

вым направлениям процесса разработки (КРА). Для этого (в простейшем случае) определяется примерный процент случаев, в которых обеспечивается деятельность по каждому направлению (таблица 9.10.).

Таблица 9.10. **Оценка уровня зрелости по КРА**

Наименование направления КРА	**Почти всегда >90% случаев**	**Часто 60-90% случаев**	**Почти поровну 40-60% случаев**	**От случая к случаю 10-40%**	**Крайне редко < 10%**	**Никогда или не знаю**
Направление i						

Далее уровень согласованности с КРА взвешивается и вычисляется значение атрибута *Кз* (уточняющее данные в таблице 9.10) по формуле:

$$Кз = 5 - \sum_{i=1}^{18} \left[(KPA\%_i) / 100 \cdot 5 / 18 \right] \qquad (4)$$

9.2.6. Оценка трудозатрат по Предпроектной модели

Уравнение номинальных трудозатрат (1) не учитывает множества факторов, влияющих на трудоемкость разработки ПС *независимо от ее размера*. Поэтому результаты оценки могут оказаться очень неточными. Для устранения этого недостатка в Предпроектную модель введено семь дополнительных параметров - *стоимостных атрибутов*, оценки которых могут быть получены на ранних этапах разработки ПС. Эти атрибуты являются комбинацией стоимостных атрибутов Детальной модели. Каждому атрибуту соответствует набор оценок, упорядоченных по шкале оценивания, и набор соответствующих числовых коэффициентов. Эти коэффициенты используются в качестве множителей при получении уточненных оценок трудозатрат по Предпроектной модели.

Уточнение оценок Предпроектной модели.

Уточненное уравнение *оценки трудозатрат* $T_п$ имеет вид:

$$T_п = \prod_{i=1}^{7} (K_i) \cdot T_{ном} \qquad (5)$$

где $T_{ном}$ - номинальная трудоемкость, вычисленная по формуле (1), $\prod_{i=1}^{7} (K_i)$ - произведение коэффициентов стоимостных атрибутов.

Требуемая *продолжительность разработки* $Д_п$ рассчитывается по формуле:

$$Д_п = 3.67 \cdot [T_п^{0.28+0.2\bullet(B-1.01))}] \cdot OCP\% / 100 \qquad (6)$$

где *OCP%* - коэффициент сжатия продолжительности разработки по отношению к требуемой (в процентах).

Стоимостные атрибуты Предпроектной модели объединены в 4 группы (таблица 9.11). Оценки стоимостных атрибутов упорядочены по шкале оценок. Каждой оценке соответствует числовой коэффициент (таблица 9.12).

Значение коэффициента меньше 1 означает снижение трудозатрат, 1 - не изменяет номинальную оценку, больше 1 - увеличивает.

Таблица 9.11. **Стоимостные атрибуты Предпроектной модели**

Группа	Стоимостной атрибут	K_i
Атрибуты разработчиков	1.Квалификация разработчиков (**ПЕРС**) 2. Опыт работы (**ОРАБ**)	K_1 K_2
Атрибуты разработки	1. Инструментальная поддержка и среда проекта (**ИСРП**) 2. Ограничение сроков разработки (**ОСР**)	K_3 K_4
Атрибуты среды	Сложность программно-аппаратной среды (**СПАС**)	K_5
Атрибуты программного продукта	1. Сложность и надежность (**СЛНД**) 2. Требуемое повторное использование (**ПИСП**)	K_6 K_7

Таблица 9.12. **Коэффициенты стоимостных атрибутов Предпроектной модели**

Стоимостной атрибут	Уровень оценки						
	Сверх низкий	Очень низкий	Низкий	Номинальный	Высокий	Очень высокий	Сверх высокий
Квалификация разработчиков K_1	2.12	1.62	1.26	1.00	0.83	0.63	0.50
Опыт работы K_2	1.59	1.33	1.12	1.00	0.87	0.71	0.62
Инструментальная поддержка и среда проекта K_3	1.43	1.30	1.10	1.00	0.87	0.73	0.62
Ограничение сроков разработки K_4	-	1.43	1.14	1.00	1.00	1.00	
Сложность программно-аппаратной среды K_5	-	-	0.87	1.00	1.29	1.81	2.61
Сложность и надежность K_6	0.73	0.81	0.98	1.00	1.30	1.74	2.38
Требуемое повторное использование K_7	-	-	0.95	1.00	1.07	1.15	1.24

Алгоритм оценивания стоимостных атрибутов аналогичен оцениванию атрибутов масштаба и включает следующие шаги.

1. Экспертным методом оценить каждый стоимостной атрибут. Если какой-либо атрибут определить сложно, - следует выбрать номинальную оценку.

2. Для каждого атрибута по таблице 9.12 определить коэффициент K_i.

3. Перемножить коэффициенты и вычислить $\prod_{i=1}^{7}(K_i)$.

Атрибуты разработчиков - характеризуют влияние на трудоемкость особенностей коллектива разработчиков в целом (таблица 9.13).

Таблица 9.13. **Атрибуты разработчиков**

Атрибуты разработчиков	**Уровень оценки**						
	Сверх низкий	**Очень низкий**	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх высокий**
Квалификация разработчиков	20 проц	39 проц	45 проц	55 проц	65 проц	75 проц	85 проц
Опыт работы	<3 мес	5 мес	9 мес	1 год	2 года	4 года	6 лет

Оценки уровня квалификации разработчиков выражаются в *процентилях*[4] распределения квалификации среди всех разработчиков организации. Оценивание квалификации производится по следующим факторам:

- способность к анализу и программированию;
- эффективность и тщательность выполнения работы;
- способность к общению и сотрудничеству.

Оценке квалификации подвергается коллектив в целом, а не отдельные специалисты. Опыт работы определяется средним стажем работы коллектива в данной предметной области, среде разработки и с языками программирования.

Атрибуты разработки - характеризуют влияние на трудозатраты уровня инструментальной поддержки разработки и изменения установленных ограничений на сроки разработки (таблица 9.14).

Таблица 9.14. **Атрибуты разработки**

Атрибуты разработки	**Уровень оценки**				
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**
Инструментальная поддержка и среда проекта	Простые инструменты редактирования, кодирования и отладки	Простые CASE-инструменты, слабая интеграция	Базовые инструменты ЖЦ, умеренно интегрированные	Мощные инструменты для всего ЖЦ, умеренно интегрированные	Мощные интегрированные среды разработки для всего ЖЦ
Ограничение сроков разработки	75%	85%	100%	130%	160%

Высокие оценки означают возможное снижение трудозатрат и увеличение производительности.

---

[4] Процентиль - единица относительной градации на участке шкалы от 1 до 100.

Оценки ограничения сроков разработки (OCP) задаются в процентах от номинальной продолжительности разработки. Этот атрибут используется для оценки трудозатрат при сокращении сроков по сравнению с вычисленными номинальными значениями. Устанавливать это ограничение менее 75% от номинальной продолжительности не рекомендуется, поскольку это может привести к ухудшению характеристик ПС.

Атрибуты среды - оценивают совокупное влияние на трудозатраты сложности программно-аппаратной среды в части ограничений по времени выполнения и требуемой оперативной памяти, а также частоты изменения среды разработки (таблица 9.15).

Таблица 9.15. **Атрибуты среды**

Сложность программно-аппаратной среды	**Уровень оценки**			
	Низкий	**Номинальный**	**Высокий**	**Очень высокий**
Ограничение по времени и памяти	< 50%	50%	65%	80%
Изменчивость среды разработки	Очень стабильная	Стабильная	Слегка изменчива	Изменчива

Атрибуты программного продукта - оценивают влияние на трудозатраты сложности ПС и требований к его надежности (затраты на достижение высокого уровня надежности), а также степени, в которой необходимо обеспечить последующее повторное использование компонентов.

Атрибут сложность и надежность - это интегральный атрибут, в целом учитывающий такие характеристики, как требуемая надежность, сложность, ожидаемый размер баз данных (БД), требуемый уровень документирования ПС. В таблице 9.16 представлены единичные атрибуты для его оценивания.

Таблица 9.16. **Единичные атрибуты для атрибута Сложность и надежность**

Сложность и надежность	**Уровень оценки**						
	Сверх низкий	**Очень низкий**	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх высокий**
Сумма оценок единичных атрибутов	5, 6	7, 8	9-11	12	13-15	16-18	19-21
Акцент на надежности, документации	Очень слабый	Слабый	В какой-то степени	Достаточный	Сильный	Очень сильный	Чрезвычайный
Сложность	-	Простой	В какой-то степени сложный	Средне сложный	Сложный	Очень сложный	Сверх-сложный
Размер БД	Малый	Малый	Малый	Средний	Большой	Очень большой	Чрезвычайно большой

Оценки интегрального атрибута получаются как средневзвешенная сумма оценок единичных атрибутов.

Уровень документирования определяется исходя из того, насколько полно должны охватываться документами стадии ЖЦ. Номинальная оценка означает, что состав документов соответствует потребностям ЖЦ.

Оценка сложности ПС производится по критериям, представленным в таблице 9.17. Поскольку сложность обработки данных учитывается при определении размера ПС, данный атрибут учитывает связь рейтинга сложности и типа ПС.

Для оценки сложности по таблице 9.17 следует выбрать нужный критерий, который характеризует ПС (или группу критериев - в этом случае оценка сложности - это средневзвешенное по критериям). Атрибут «Размер БД» введен для того, чтобы учесть дополнительные трудозатраты на подготовку тестовых данных при больших объемах данных.

Таблица 9.17. **Критерии сложности**

Значения оценки	**Логическая сложность**	**Сложность интерфейса пользователя**	**Сложность вычислений**
Простой	Простые алгоритмы обработки информации, простые запросы	Простые входные формы, простые генераторы отчетов	Простые арифметические выражения
В какой-то степени сложный	ПС сравнительно несложной логической структуры, характеризуется простой формой входных и выходных документов, простые запросы	Использование простых построителей графического пользовательского интерфейса	Умеренно сложные выражения с использованием стандартных функций
Среднесложный (номинальный)	ПС несложной логической структуры, характеризуется сложностью входных и выходных документов и простотой получения данных	Простое использование готовых элементов интерфейса	Стандартные математические и статистические процедуры
Сложный	ПС сравнительно сложной логической структуры, характеризуются разнообразием форм входных и выходных документов	Разработка и расширение набора элементов интерфейса. Простой звуковой ввод	Численный анализ: многовариантная интерполяция. Операции усечения и округления
Очень сложный	Программы с очень сложной логикой обработки данных, сложные запросы, многосекционные отчеты	Умеренно сложная динамическая графика, мультимедиа	Сложный структурированный численный анализ. Простая параллелизация
Сверх сложный	Сложные запросы, распределенная обработка данных	Сложные мультимедиа	Сложный неструктурированный численный анализ. Сложное распараллеливание

Атрибут *Требуемое повторное использование* учитывает дополнительные трудозатраты, необходимые для конструирования компонентов, предназначенных для повторного использования в данном или следующих проектах.

Эти трудозатраты расходуются на обеспечение большей универсальности компонентов, более подробной документации и более тщательного тестирования, гарантирующего повторную используемость в других компонентах (таблица 9.18).

Таблица 9.18. **Шкала оценок атрибута Требуемое повторное использование**

	Уровень оценки				
	Низкий	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх высокий**
Требуемое повторное использование	Нет	Для данного проекта	Для однотипных продуктов	Для одного семейства продуктов	Для любого продукта

Процедура оценки трудозатрат на разработку ПС по Предпроектной модели включает следующую последовательность шагов.

1. Рассмотреть все атрибуты масштаба и экспертным методом определить коэффициенты масштаба Φ_j по таблице 9.5.
2. Вычислить показатель степени по формуле (2).
3. Вычислить размер V в показателях функционального размера (УЕФ) и конвертировать его в строки кода (см. главу 8).
4. Вычислить номинальную трудоемкость $T_{ном}$ по формуле (1).
5. Оценить 7 атрибутов стоимости экспертным путем и определить корректирующие коэффициенты этих атрибутов K_i по таблице 9.12.
6. Вычислить уточненную трудоемкость T_n по формуле (5).
7. Вычислить требуемую продолжительность разработки в месяцах $Д_n$ по формуле (6).

9.2.7. Оценивание трудозатрат по Детальной модели

Оценивание трудозатрат по Детальной модели отличается от Предпроектной модели тем, что стоимостные атрибуты и размер определяются *отдельно для каждого* модуля (или программного компонента) [21].

Атрибуты масштаба определяются один раз на уровне всей ПС.

Характеристика стоимостных атрибутов модели.

Для более точной оценки трудозатрат в Детальной модели введено 17 стоимостных атрибутов, сгруппированных в четыре группы, как и в Предпроектной модели (таблица 9.19). Числовые коэффициенты представлены в таблице 9.20.

Таблица 9.19. **Стоимостные атрибуты Детальной модели**

Группа	**Стоимостной атрибут**	**K_i**
Атрибуты разработчиков	1.Квалификация аналитика (**КАНЛ**)	K_1
	2. Квалификация программиста (**КПРГ**)	K_2
	3. Опыт работы в предметной области (**ОРАБ**)	K_3
	4. Опыт работы в среде разработки (**ОСРЗ**)	K_4
	5. Опыт работы с языками программирования (**ОРЯП**)	K_5
	6. Постоянство коллектива в проекте (**ПКОЛ**)	K_6
Атрибуты разработки	1. Инструментальная поддержка (**ИНСТ**)	K_7
	2. Ограничение сроков разработки (**ОСР**)	K_8
	3. Распределение разработки (**РАСР**)	K_9

Группа	Стоимостной атрибут	K_i
Атрибуты среды	1. Ограничения по времени выполнения (**ОВРМ**) 2. Ограничения по оперативной памяти (**ОПАМ**) 3. Изменчивость среды разработки (**ИЗМС**)	K_{10} K_{11} K_{12}
Атрибуты программного продукта	1. Требуемая надежность (**ТНАД**) 2. Сложность (**СЛОЖ**) 3. Размер базы данных (**РАБД**) 4. Соответствие документации потребностям ЖЦ (**ДОКУ**) 5. Требуемое повторное использование (**ПИСП**)	K_{13} K_{14} K_{15} K_{16} K_{17}

Таблица 9.20. **Коэффициенты стоимостных атрибутов Детальной модели**

Стоимостной атрибут	Уровень оценки					
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх высокий**
Квалификация аналитика (K_1)	1.42	1.19	1.00	0.85	0.71	-
Квалификация программиста (K_2)	1.34	1.15	1.00	0.88	0.76	-
Опыт работы в предметной области (K_3)	1.22	1.10	1.00	0.88	0.81	-
Опыт работы в среде разработки (K_4)	1.19	1.09	1.00	0.91	0.85	-
Опыт работы с языками программирования (K_5)	1.20	1.09	1.00	0.91	0.84	-
Постоянство коллектива в проекте (K_6)	1.29	1.12	1.00	0.90	0.81	-
Инструментальная поддержка (K_7)	1.17	1.09	1.00	0.90	0.78	-
Ограничение сроков разработки (K_8)	1.43	1.14	1.00	1.00	1.00	-
Распределение разработки (K_9)	1.22	1.09	1.00	0.93	0.86	0.80
Ограничения по времени выполнения (K_{10})	-	-	1.00	1.11	1.29	1.63
Ограничения по оперативной памяти (K_{11})	-	-	1.00	1.05	1.17	1.46
Изменчивость среды разработки (K_{12})	-	0.87	1.00	1.15	1.30	-
Требуемая надежность (K_{13})	0.82	0.92	1.00	1.10	1.26	-
Сложность (K_{14})	0.73	0.87	1.00	1.17	0.34	1.74
Размер базы данных (K_{15})		0.90	1.00	1.14	1.28	-
Соответствие документации (K_{16})	0.81	0.91	1.00	1.11	1.23	-
Требуемое повторное использование (K_{17})		0.95	1.00	1.07	1.15	1.24

Атрибуты разработчиков - позволяют уточнить различия, связанные с разными группами исполнителей: их квалификацией, опытом работы и т.д. Рейтинги этих атрибутов представлены в таблице 9.21.

Квалификация аналитиков. Оценивание должно основываться на квалификации аналитиков как единой группы и производиться по следующим факторам:

- способность к анализу;

- эффективность и тщательность выполнения работы;
- способность к общению и сотрудничеству.

Таблица 9.21. **Рейтинги атрибутов разработчиков**

Атрибуты разработчиков	**Уровень оценки**				
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**
Квалификация аналитика	15 проц	35 проц	55 проц	75 проц	90 проц
Квалификация программиста	15 проц	35 проц	55 проц	75 проц	90 проц
Опыт работы в предметной области	2 мес	6 мес	1 год	3 года	6 лет
Опыт работы в среде разработки	2 мес	6 мес	1 год	3 года	6 лет
Опыт работы с языками программирования и инструментами	2 мес	6 мес	1 год	3	6
Постоянство коллектива в проекте	48%	24%	12%	6%	3%

Квалификация программистов. Оценивание должно основываться на квалификации программистов как единой группы и производиться по следующим факторам:

- способность к программированию;
- эффективность и тщательность выполнения работы;
- способность к общению и сотрудничеству (в составе группы).

Оценки квалификации не должны включать опыт работы, поскольку он учитывается отдельно.

Опыт работы в предметной области - определяет влияние на трудоемкость уровня опыта работы группы в данной предметной области. Рейтинги атрибута определяются в терминах эквивалентного уровня опыта группы в разработке данного типа систем или подсистем.

Опыт работы в среде разработки - учитывает влияние на производительность труда использования мощных сред поддержки разработки, таких как графический интерфейс пользователя, СУБД, средства распределенных сред и сетевые возможности.

Опыт работы с языками программирования и инструментами. Разработка ПС включает применение инструментальных средств для анализа требований и проекта, управление конфигурацией, документирование, управление библиотеками, проверку непротиворечивости и т.п. Опыт работы на языке и в среде программирования также оказывает влияние на продолжительность разработки.

Постоянство коллектива - отражает негативное влияние на затраты текучести кадров в группе разработчиков и оценивается в процентном отношении годовых изменений.

Атрибуты среды - отражают влияние на трудозатраты обеспечения требований, связанных с производительностью ПС, а также изменения программно-

аппаратной среды разработки. Рост трудозатрат может быть связан с более высокой стоимостью обнаружения ошибок, связанных с производительностью ПС и их исправлением на этапах системного тестирования. В Детальной модели эти атрибуты рассматриваются отдельно. Рейтинги этих атрибутов представлены в таблице 9.22.

Таблица 9.22. **Рейтинги атрибутов среды**

Атрибуты среды	**Уровень оценки**				
	Низкий	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх высокий**
Ограничения по времени выполнения	< 50%	50%	70%	95%	
Ограничения по оперативной памяти	-	<=50%	70%	85%	95%
Изменчивость среды разработки	Большие изменения каждые 12 месяцев, небольшие – каждый месяц	Большие изменения каждые 6 месяцев, небольшие – через 2 недели	Большие изменения – 2 месяца, небольшие – каждую неделю	Большие – 2 недели, небольшие – каждые 2дня	-

Атрибуты разработки. Перечень атрибутов представлен в таблице 9.23.

Таблица 9.23. **Рейтинги атрибутов разработки**

Атрибуты разработки	**Уровень оценки**				
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**
Инструментальная поддержка	Простые инструменты редактирования, кодирования и отладки	Простые CASE-инструменты, слабая интеграция	Базовые инструменты ЖЦ, умеренно интегрированные	Мощные инструменты для всего ЖЦ, умеренно интегрированные	
Ограничение сроков разработки	75%	85%	100%	130%	160%
Распределение разработки	Телефон, электронная почта	Телефон, FAX	Эл. почта через узкополосный канал связи	Связь через широкополосный канал связи	Широкополосный канал, видеоконференции

В Детальной модели COCOMO II введен новый атрибут «Распределение разработки», который позволяет учитывать возможные негативные влияния взаимодействия территориально рассредоточенных групп исполнителей (например, соисполнителей) на трудозатраты.

Атрибуты программного продукта. В Детальной модели рассматривается отдельно влияние на трудозатраты каждого из факторов продукта (таблица 9.24). Это позволяет рационально распределить затраты на разработку между модулями.

Таблица 9.24. **Рейтинги атрибутов продукта в Детальной модели**

Атрибуты продукта	**Уровень оценки**					
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх высокий**
Требуемая надежность	Некоторое неудобство	Небольшие, легко устранимые потери	Средние, легко устранимые потери	Большие финансовые потери	Риск человеческой жизни	-
Сложность	По таблице 9.17					
Размер базы данных	-	РАБД<10	10<РАБД <100	100<=РАБД <1000	РАБД>= 1000	-
Соответствие документации потребностям ЖЦ	Многие этапы не охвачены	Некоторые потребности этапов не охвачены	Соответствует потребностям ЖЦ	Избыточная	Сверх избыточная для ЖЦ	-
Требуемое повторное использование	Нет	Нет	Для данного проекта	Для однотипных продуктов	Для одного семейства продуктов	Для любого продукта

Требуемая надежность. Необходимость учета требуемого уровня надежности при оценке затрат связана с тем, что для достижения установленного уровня надежности нужны дополнительные затраты на тестирование, документирование и оценку надежности. Кроме того, требования к надежности могут быть распределены среди модулей неравномерно. Номинальная оценка означает, что отказы при функционировании модуля легко устранимы и не приводят к большим потерям.

Оценка сложности выполняется по критериям, представленным в табл. 9.17.

Атрибут «Размер БД» введен для того, чтобы учесть дополнительные трудозатраты на подготовку тестовых данных при больших объемах данных.

РАБД = Размер БД(*байт*)/*SLOC*

Алгоритм оценивания трудозатрат по Детальной модели подобен алгоритму оценивания Предпроектной модели (за исключением количества стоимостных атрибутов и уровня оценивания) и описан ниже.

1. Оценить атрибуты масштаба и определить коэффициенты масштаба Φ_j.
2. Вычислить показатель степени B по формуле (2).
3. Определить состав оцениваемых модулей и оценить размер каждого модуля V_m в показателях функционального размера, где $m = \{1,M\}$- M - количество модулей в ПС, и конвертировать его в строки кода (SLOC).
4. Вычислить номинальные трудозатраты каждого модуля $T_{номт}$.
5. Оценить 17 атрибутов стоимости и экспертным путем определить корректирующие коэффициенты этих атрибутов K_i для каждого модуля.
6. Вычислить уточненную трудоемкость T_m каждого модуля по формуле (7):

$$T_m = \prod_{i=1}^{17} (K_i) \cdot T_{номт} \qquad (7)$$

где $T_{номт}$ - номинальная трудоемкость модуля, вычисленная по формуле (1).

7. Вычислить требуемую продолжительность разработки модуля в месяцах $Д_m$ подстановкой T_m в формулу (6).

Соотнесение стоимостных атрибутов Предпроектной и Детальной моделей

Большинство стоимостных атрибутов Предпроектной модели являются комбинацией стоимостных атрибутов Детальной модели (таблица 9.25).

Таблица 9.25. **Стоимостные атрибуты Предпроектной и Детальной моделей**

Предпроектная модель		**Детальная модель**
Сложность и надежность	СЛНД	ТНАД, СЛОЖ, РАБД, ДОКУ
Требуемое повторное использование	ПИСП	ПИСП
Сложность программно-аппаратной среды	СПАС	ОВРМ, ОПАМ, ИЗМС
Квалификация разработчиков	ПЕРС	КАНЛ, КПРГ, ПКОЛ
Опыт работы	ОРАБ	ОРАБ, ОСРЗ, ОРЯП
Инструментальная поддержка и среда проекта	ИСРП	ИНСТ, РАСР
Ограничение сроков разработки	ОСР	ОСР

Оценки значений интегрального атрибута вычисляются как субъективно средневзвешенные оценки единичных атрибутов. При этом шкала оценок интегрального атрибута может быть шире, чем шкала единичных атрибутов. Если оценка стоимостного атрибута попадает между уровнями рейтингов, она должна округляться в сторону номинального рейтинга, например, если рейтинг атрибута попадает между «Очень низкий» и «Низкий», следует выбирать «Низкий».

Для вычисления этих оценок применяется следующий алгоритм.

1. Каждому рейтингу оценки каждого единичного атрибута присваивается вес в такой последовательности: оценке «Очень низкий» - 1, «Низкий» - 2 и т.д.

2. Суммируются веса единичных атрибутов и размещаются по шкале интегрального атрибута.

3. Вычисляются усредненные значения коэффициентов интегрального атрибута.

Ниже для иллюстрации рассмотрен ***пример оценки интегрального атрибута ПЕРС***.

Интегральный атрибут «Квалификация разработчика» (ПЕРС) является комбинацией трех единичных атрибутов, приведенных в таблице 9.26, каждый из которых имеет масштаб ранжирования от «Очень низкий» до «Очень высокий».

1. Рейтингам единичных атрибутов присваиваются веса (таблица 9.26).

Таблица 9.26. **Единичные атрибуты интегрального атрибута ПЕРС**

Единичные атрибуты	**Уровень оценки**				
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**
Вес единичного атрибута	1	2	3	4	5
Квалификация аналитика	15	35	55	75	90
Квалификация программиста	15	35	55	75	90
Постоянство коллектива в проекте	48%	24%	12%	6%	3%

2. Суммируются веса единичных атрибутов и в результате получаются значения от 3 до 15. Эти веса переносятся на масштаб интегрального атрибута таким образом: оценка «Номинальный» 9 отвечает сумме весов (3+3+3), а соответствующий им коэффициент равен 1.

Оценка «Сверхнизкий» соответствует сумме весов (1+1+1) или (1+1+2), и т.д. (таблица 9.27).

Таблица 9.27. **Сумма весов единичных атрибутов**

Интегральный атрибут	**Уровень оценки**						
	Сверх-низкий	**Очень низкий**	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх-высокий**
Сумма весов трех единичных атрибутов	3, 4	5,6	7,8	9	10,11	12,13	14,15

3. Усредненные значения коэффициентов интегрального атрибута (таблица 9.28) вычисляются следующим образом:

Таблица 9.28. **Усредненные значения коэффициентов интегрального атрибута ПЕРС**

Коэффициенты	**Уровень оценки**						
	Сверх-низкий	**Очень низкий**	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**	**Сверх-высокий**
Сумма весов трех единичных атрибутов	3, 4	5,6	7,8	9	10,11	12,13	14,15
Комбинация КАНЛ, КПРГ	20 проц	39	45	55	65	75	85
ПКОЛ	45%	30%	20%	12%	9%	5%	4%
Коэффициенты интегрального атрибута	2.5	1.62	1.26	1.00	0.83	0.66	0.50

a) определяются комбинации всех весов, сумма которых составляет от 3 до 4;

b)- вычисляется произведение коэффициентов для этих последовательностей (таблица 9.29):

(1.42×1.34×1.29)/2 =1.23 (1.19×1.34×1.29)/2 =1.03

(1.42×1.34×1.12)/2 =1.06 (1.42×1.15×1.29)/2 =1.05

Таблица 9.29. **Коэффициенты единичных атрибутов**

Коэффициенты единичных атрибутов	**Уровень оценки**				
	Очень иизкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**
Веса	1	2	3	4	5
K_1	1.42	1.19	1.00	0.85	0.71
K_2	1.34	1.15	1.00	0.87	0.76
K_6	1.29	1.12	1.00	0.90	0.81

c) вычисляются значения интегрального коэффициента:

1.23+(1.06+1.03+1.05)/3=2.5

Для определения всех интегральных коэффициентов шаги a)-c) выполняются применительно ко всем оценкам.

9.2.8. Корректировка оценок модели

Предпроектная и Детальная модели дают скалярные оценки трудозатрат и продолжительности разработки. В большинстве случаев, при использовании подобных моделей предпочтительнее получать интервальную оценку, внутри которой можно выбрать приемлемое значение (от самого оптимистического до самого пессимистического) для принятия более обоснованного решения о продолжительности и трудозатратах.

Путем статистического моделирования (применяя нормальное распределение) были получены коэффициенты для вычисления граничных оценок интервалов доверия (таблица 9.30).

Таблица 9.30. **Оценки интервалов доверия оценок**

Модель	Оптимистическая оценка	Пессимистическая оценка
Предварительная модель	$0.50\times T_n$	$2\times T_n$
Предпроектная модель	$0.67\times T_n$	$1.5\times T_n$
Детальная модель	$0.8\times T_n$	$1.25\times T_n$

Эти же коэффициенты рекомендуется использовать для оценки продолжительности разработки.

9.2.9. Распределение трудозатрат по стадиям разработки и видам работ

Модель COCOMO II позволяет выполнить распределение трудозатрат по стадиям разработки в условиях применения *стандартной (каскадной) модели*.

В таблице 9.31 указаны укрупненные виды работ, а в таблице 9.32 – примерное долевое распределение затрат и сроков разработки по стадиям и работам, выполняемым на стадиях (не соответствуют ГОСТ 34). Серым фоном выделены работы и стадии, которые непосредственно не учитываются в модели COCOMO II.

Таблица 9.31. **Виды работ, учитываемые в модели COCOMO II**

Виды работ	Оценивание
Определение требований	Нет
Проектирование	Да
Реализация	Да
Интеграция и тестирование	Да
Управление проектом	Да
Управление конфигурацией	Да
Обеспечение качества	Да
Внешнее руководство проектом	Нет

Виды работ «Определение требований» и «Внешнее руководство проектом» непосредственно не учитываются в модели COCOMO II, но для них указана рекомендуемая доля общих затрат (в %). В связи с этим, в таблице 9.32 общий объем трудозатрат составляет больше 100% [22]

Указанное в таблице 9.32 распределение зависит от *размера ПС*, представленного в тысячах строк исходного кода (KSLOC, далее K).

Таблица 9.32. **Распределение трудозатрат и продолжительности по каскадной модели ЖЦ**

Стадия разработки	Оценивание	Процент общей оценки	
		Трудозатраты	Продолжительность
Планирование и анализ требований (Plan&Requirement)	нет	7% (2% – 15%)	Обычно от 16% - 24% (2% - 30%)
Проектирование (Product Design)	Да	17%	24% - 28%
Программирование (Programming)	Да	64%-52%	56%-40%
Интеграция и тестирование Integration & Test	Да	19%-31%	20%-32%
Передача (Transition)	нет	12% (0%-20%)	12.5% (0%-20%)

Пример. В таблице 9.33 в качестве примера приведено *распределение трудозатрат в зависимости от размера ПС*.

Изменение процентного соотношения затрат с ростом размера объясняется тем, что большие ПС требуют большей доли затрат на интеграцию и тестирование. Если вычисленный размер ПС находится между значениями, представленными в таблице 9.33, соответствующее долевое распределение получается путем линейной интерполяции данных из таблицы по формуле (8):

$$y=y_0 + [(x-x_0)/(x_1-x_0)]\cdot(y_1-y_0) \qquad (8)$$

где y - искомое распределение в процентах, x - вычисленный размер, (x_0,y_0) и (x_1,y_1) - точки таблицы, между которыми находится вычисленный размер.

Таблица 9.33. **Распределение трудозатрат по стадиям разработки (в %)**

Стадия разработки	Размер (тыс. строк)							
	8К	16К	32К	64К	128К	256К	512К	1024К
Планирование и анализ требований	7	7	7	7	7	7	7	7
Проектирование	17	17	17	17	17	17	17	17
Программирование, из них:	60	60	58	57	55	54.9	54.8	54.4
- детальное проектирование	25.3	25.7	24.5	24.7	24	23.9	23.9	23.8
- кодирование и автономное тестирование	34.7	34,3	33.5	32.3	31	30.9	30.9	30.6
Интеграция и тестирование	24	23	25.8	26	28.1	28.1	28.2	28.6

Для ***моделей ЖЦ итеративного типа*** (спиральной, с приращениями) предназначена ранее упоминавшаяся модель COINCOMO. Это обобщенная модель распределения трудозатрат по стадиям разработки, которая использует в качестве входных данных результаты оценки затрат и продолжительности, полученные по модели COCOMO II.

Кроме COINCOMO для итеративного ЖЦ можно использовать модель MBASE (Model-Based Architecting & Software Engineering), предложенную Б.Боэмом в [23].

В таблице 9.34 представлены результаты распределения трудозатрат и продолжительности для моделей ЖЦ этого типа. Стадии разработки модели MBASE соотнесены со стадиями, принятыми в итеративной модели с приращениями, используемой в RUP (Rational Unified Process) [24]. Указанное распределение (как и для каскадной модели) зависит от размера ПС (применительно к функциональным возможностям итерации).

Таблица 9.34. **Распределение трудозатрат и продолжительности для моделей ЖЦ итеративного типа**

Стадия Разработки	Оценивание	Процент общей оценки			
		MBASE		RUP	
		Трудозатраты	Продолжительность	Трудозатраты	Продолжительность
Начало (Inception)	Нет	6% (2% - 15%)	12.5% (2% -30%)	5%	10%
Уточнение (Elaboration)	Да	24% (20% - 28%)	37.5% (24% -28%)	20%	20%
Конструирование (Construction)	Да	76% (72% - 80%	62.5% (58% - 67%)	65%	50%
Передача (Transition)	Нет	12% (0% - 20%)	12.5% (0% - 20%)	10%	10%
Всего		118% (102% - 135%)	125% (102% - 150%)	100%	100%

В таблице 9.34 серым фоном выделены стадии, которые непосредственно не учитываются в модели COCOMO II. В связи с этим общая оценка по модели MBASE превышает 100%.

Сопоставление терминов в русско- и англоязычных источниках.

Для удобства использования материала, представленного в этом разделе, а также дополнительной информации (из англоязычных источников) в таблицах 9.35-9.37 сопоставлены русские и английские наименования и обозначения атрибутов модели COCOMO II.

Таблица 9.35. **Сопоставление наименований атрибутов масштаба**

№ пп	Русское наименование	Обозначение	Английское наименование	Обозначение
1	Новизна проекта	НОВ	Precedentedness	PREC
2	Жесткость проекта (согласованность)	СОГЛ	Development Flexibility	FLEX
3	Управление риском/ архитектурой	УР	Architecture / Risk Resolution	RESL
4	Коллективизм	КОЛЛ	Team Cohesion	TEAM
5	Технологическая зрелость процесса разработки	ТЗР	Process Maturity	PMAT

Таблица 9.36. **Сопоставление наименований атрибутов стоимости в Предпроектной модели**

№ пп	Русское наименование	Сокращение	Английское наименование	Сокращение
1	Квалификация разработчиков	ПЕРС	Personnel Capability	PERS
2	Опыт работы	ОРАБ	Personnel Experience	PREX
3	Инструментальная поддержка и среда проекта	ИСРП	Facilities	FCIL
4	Ограничение сроков разработки	ОСР	Schedule	SCED
5	Сложность программно-аппаратной среды	СПАС	Platform Difficulty	PDIF
6	Сложность и надежность	СЛНД	Product Reliability and Complexity	RCPX
7	Требуемое повторное использование	ПИСП	Developed for Reusability	REUSE

Таблица 9.37. **Сопоставление наименований атрибутов стоимости в Детальной модели**

№ пп	Русское наименование	Сокращение	Английское наименование	Сокращение
1	Квалификация аналитика	КАНЛ	Analyst Capability	ACAP
2	Квалификация программиста	КПРГ	Programmer Capability	PCAP
3	Опыт работы в предметной области	ОРАБ	Applications Experience	APEX
4	Опыт работы в среде разработки	ОСРЗ	Platform Experience	PLEX
5	Опыт работы с языками программирования	ОРЯП	Language and Tool Experience	LTEX
6	Постоянство коллектива в проекте	ПКОЛ	Programmer Capability	PCAP
7	Инструментальная поддержка	ИНСТ	Use of Software Tools	TOOL
8	Ограничение сроков разработки	ОСР	Required Development Schedule	SCED
9	Распределение разработки	РАСР	Multisite Development	SITE
10	Ограничения по времени выполнения	ОВРМ	Execution Time Constraint	TIME
11	Ограничения по оперативной памяти	ОПАМ	Main Storage Constraint	STOR
12	Изменчивость среды разработки	ИЗМС	Platform Volatility	PVOL
13	Требуемая надежность	ТНАД	Required Software Reliability	RELY
14	Сложность	СЛОЖ	Product Complexity	CPLX
15	Размер базы данных	РАБД	Data Base Size	DATA
16	Соответствие документации потребностям ЖЦ	ДОКУ	Documentation Match to Life-Cycle Needs	DOCU
17	Требуемое повторное использование	ПИСП	Developed for Reusability	RUSE

9.3. Оценка затрат на разработку Web-приложений

Помимо перечисленных ранее моделей из семейства COCOMO, существуют и другие интерпретации данного подхода, учитывающие специфику ПС.

В работе [25] Д.Рифер предложил ***модель WEBMO*** для оценки трудозатрат и продолжительности разработки Web-приложений, разработанную с помощью экспертного оценивания и регрессионного анализа данных по 46 проектам.

Математическая формулировка этой модели базируется на уравнениях и комбинации параметров моделей COCOMO II и SoftCost-OO [26]. Соотношение для оценки трудоемкости T (в человеко-месяцах) представлено уравнением (9), а для оценки продолжительности D (в календарных месяцах) – уравнением (10).

$$T = A \cdot \prod_{i=1}^{8} k_i \cdot V^{p_1} \tag{9}$$

$$D = B \cdot T^{p_2} \tag{10}$$

где A и B – константы, k_i – коэффициенты стоимостных атрибутов, p_1, p_2 – показатели степени обоих уравнений, полученные в результате анализа оценок пяти типов Web-проектов, V – размер приложения, измеряемый методом *Web Objects* (см. п.8.3.2). Полученные эмпирически значения параметров A, B, p_1 и p_2 представлены в таблице 9.38.

Таблица 9.38. **Значения постоянных параметров модели WEBMO**

Тип Web-приложения	A	B	p_1	p_2
Электронная коммерция	2.3	2.0	1.03	0.5 или 0.32
Финансовые приложения	2.7	2.2	1.05	0.5 или 0.32
Бизнес-приложения	2.0	1.5	1.00	0.5 или 0.32
Web-порталы	2.1	1.8	1.00	0.5 или 0.32
Поиск и использование информации	2.1	2.0	1.00	0.5 или 0.32

В модели используется 8 стоимостных атрибутов Предпроектной модели COCOMO II. Следует отметить, что атрибут КОЛЛ (Коллективизм) в COCOMO II используется как атрибут масштаба, а атрибут ЭФПР (Эффективность процесса) по своему смыслу соответствует атрибуту масштаба ТЗР (Технологическая зрелость процесса разработки).

Перечень стоимостных атрибутов и их рейтинги представлены в таблице 9.39. Значения коэффициентов стоимостных атрибутов отличаются от значений модели COCOMO II.

Для определения трудоемкости разработки ***статических Web-сайтов*** может использоваться простой метод, представленный Д.Клиэри в работе [27]. Согласно этому методу размер сайта V определяется в единицах *Web Points* (см. п.8.3.3), а трудоемкость рассчитывается по формуле $T = PDR \cdot V$, где PDR – интенсивность разработки (количество Web Points, «выпускаемых» специалистами проекта за один час). Например, для проектов CHARISMATEK, inc. PDR=0.5. Учитывается только трудоемкость анализа требований, проектирования, кодирования и тестирования HTML-страниц. Трудоемкость разработки или приобретения контента страниц, не текстовых элементов на страницах, элементов из внешних источников (файлов) должна рассчитываться отдельно.

Таблица 9.39. **Рейтинги значения стоимостных атрибутов**

Атрибуты	**Уровень оценки**				
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**
Сложность и надежность продукта (приложения) (СЛНД)	Только клиентское приложение, не распределенное, простые вычисления и I/O, не требуется высокая надежность	Клиент-серверное, ограниченно распределенное, стандартные вычисления, управление файлами, потери легко восстановить	Клиент-серверное, полностью распределенное, интегрированное, с базами данных, требуется обычное восстановление	Клиент-серверное, глобально распределенное, сложные вычисления, высокая стоимость потерь из-за ошибок	Клиент-серверное, полностью распределенное, тесное взаимодействие, робота в режиме реального времени, последствия ошибок критичны
Значения	**0.63**	**0.85**	**1.00**	**1.3**	**1.67**
Сложность программно-аппаратной среды (изменчивость ОС и сетевых серверов) (СПАС)	Редкие изменения платформы, скоростная сеть, наилучшая возможная связь, нет ограничений на ресурсы (время и память)	Небольшие изменения платформы, быстрый сетевой сервис, незначительные проблемы с ресурсами	Стабильная платформа, приемлемая производительность сети; приемлемая связь, требуется контроль ресурсов	Частые изменения платформы, медленная сеть, плохая связь, недостаточные ресурсы приводят к проблемам	Нестабильная платформа, низкая производительность, плохая связь, ограниченные ресурсы
Значения	**0.75**	**0.87**	**1.00**	**1.21**	**1.41**
Квалификация разработчиков (ПЕРС)	15 проц Большие простои (текучесть кадров)	35 проц Небольшие простои	55 проц Очень маленькие простои	75 проц Редкие простои	90 проц Отсутствие простоев
Значения	**1.55**	**1.35**	**1.00**	**0.75**	**0.58**
Опыт работы (ОРАБ)	≤ 2 мес	≤ 6 мес	≤ 1 год	≤ 3 года	≤ 6 лет
Значения	**1.35**	**1.19**	**1.00**	**0.87**	**0.71**
Инструментальная поддержка и среда проекта (ИСРП)	Международная разработка, без взаимодействия, языковые средства, базовые методы	Несколько городов, частичное взаимодействие, базовые CASE- средства, базовые методы	Один город, методы для всего ЖЦ, хорошие инструменты, команды разработки	Одно здание, интегрированные методы и инструменты, объединенные команды	Одна рабочая среда, интегрированные методы и средства с поддержкой взаимодействия
Значения	**1.36**	**1.13**	**1.00**	**0.86**	**0.68**

Атрибуты	Уровень оценки				
	Очень низкий	**Низкий**	**Номинальный**	**Высокий**	**Очень высокий**
Ограничение сроков разработки (ОСР)	Требуется сокращение до 75% от номинального (на 25%)	Требуется сокращение до 85 % от номинального (на 15%)	Нет сокращения 100%	Можно увеличить до 130%	Можно увеличить до 160%
Значения	**1.35**	**1.15**	**1.00**	**1.05**	**1.10**
Требуемое повторное использование (ПИПС)	Не используется	Не используется	Не запланированное повторное использование	Планируемое повторное использование или библиотечные компоненты	Систематическое повторное использование, основанное на архитектуре
Значения	**-**	**-**	**1.00**	**1.25**	**1.48**
Коллективизм (степень сотрудничества) (КОЛЛ)	Отсутствие единого видения и сотрудничества	Некоторое взаимопонимание и взаимодействие	Во многом общее видение, группы по функциональным направлениям	Полное взаимопонимание, тесное взаимодействие	Исключительно хорошее взаимопонимание
Значения	**1.45**	**1.31**	**1.00**	**0.75**	**0.62**
Эффективность процесса (ЭФПР)	Полностью не управляемый (ad hoc) – «героический»	Настроенный на проект, основанный на роли лидера	Четкий процесс, основанный на процессном подходе	Эффективный процесс, применение эффективных практических приемов работы	Эффективный процесс, стремление рационально распределить время
Значения	**1.36**	**1.20**	**1.00**	**0.86**	**0.65**

9.4. Характеристика основных инструментов оценки затрат

В табл. 9.40 перечислены наиболее распространенные инструменты оценки затрат, как использующие собственные методы оценивания, так и основанные на рассмотренных выше методах.

Более полный список инструментов оценивания и их производителей можно найти, например, по адресу: http://www.laatuk.com/tools/effort_estimation_tools.html, а также с помощью поисковых серверов.

Таблица 9.40. **Инструменты оценки затрат на разработку ПС**

Название инструмента	Фирма-изготовитель и адрес в Интернет
PRICE S	Lockheed - Martin PRICE Systems http://www.pricesystems.com/
ESTIMACS	Computer Associates International Inc. http://www.cai.com/products/estimacs.htm/
ESTIMATE Professional	Software Productivity Center http://www.spc.com/
EstimatorPal	EstimatorPal http://www.effortestimator.com/
EZEstimate	Strive Logic http://www.openrage.com/main/ezestimate_full.htm/
KnowledgePlan	Software Productivity Research Inc. http://www.spr.com/
SEER-SEM (System Evaluation and Estimation of Resources - Software Estimation Model)	Galorath Associates, Inc., SEER Technologies Division http://www.galorath.com
SLIM (Software Life Cycle Model)	Quantitative Software Management, Inc. (QSM) http://www.qsm.com/
SOFTCOST-R	Reifer Consultants, Inc (RCI) www.reifer.com/
Function Point Workbench	Software Productivity Research Inc. http://www.spr.com
COSTAR	Softstar Systems Inc. http://www.softstarsystems.com/
SystemStar	SoftStar Systems http://www.softstarsystems.com/
SCOPE IT	SCOPE IT Inc. http://www.scopeit.com/

В работе [28] представлены результаты анализа применения разных методов и инструментов оценивания затрат для крупных проектов ПС, где перечислены такие *основные характеристики* современных инструментов:

– поддержка разных метрик размера: в строках кода, показателях функционального размера, функциях и др.;

– распределение затрат и продолжительности по стадиям и видам работ применительно к разным моделям разработки;

– настройка модели для разных типов проектов ПС;

– поддержка оценивания затрат на проекты новых ПС, а также на сопровождение и развитие ПС.

Кроме того, как указано в [28], в результате анализа многих проектов выявлено, что оценки затрат для *крупных проектов*, полученные с помощью инструментов, оказались более точными, чем полученные «ручными» методами оценивания. Особенно это касается оценки затрат на «общесистемные работы» (анализ требований, документирование, управление проектами и др.).

Для *небольших проектов* и работ по проектированию, программированию точность оценок, полученных с помощью инструментов и вручную, сравнительно одинакова.

Представляет определенный практический интерес предложенная в [28] ***структурная декомпозиция работ*** и процентное распределение затрат для *шести типов ПС*:

- *Web-проекты*. К ним относят приложения, разрабатываемые для поддержки корпоративных Web-сайтов,
- *информационные системы* (ИС). К ним относят ПС, поддерживающие стандартные функции ввода, обработки и хранения информации,
- *аутсорсинговые системы*. К ним автор относит ПС, разрабатываемые внешними организациями-разработчиками (вне страны-производителя ПС),
- *коммерческие ПС*. К ним относят ПС, разрабатываемые для продажи (а не по заказу). Это любые пакетные продукты (так называемые COTS-продукты), например, текстовые процессоры,
- *встроенное ПО*. Это ПО, разрабатываемое для управления физическими устройствами (в частности, средствами телекоммуникации),
- *военные ПС*. К ним отнесены любые разработки систем, выполняемые в соответствии с требованиями военных стандартов.

Структурная декомпозиция работ и процентное соотношение затрат для указанных выше типов проектов представлены в таблице 9.41. Эти данные могут служить ориентиром при оценивании проектов.

Как видно из таблицы 9.41 в разных типах проектов состав работ значительно варьируется, что необходимо учитывать при настройке и выборе модели оценивания.

Таблица 9.41. **Долевое распределение затрат на создание разных типов ПС в разрезе видов работ**

Виды работ	**Тип ПС**					
	Web-проекты	**ИС**	**Аутсорсинг**	**Коммерческие**	**Встроенные**	**Военные**
Определение требований	5.0%	7.5%	9%	4%	4%	7%
Прототипирование	10.0%	2.0%	2.5%	1.0%	2.0%	2.0%
Разработка архитектуры		0.5%	1.0%	2.0%	1.5%	1.0%
Планирование проекта		1.0%	1.5%	1.0%	2.0%	1.0%
Предварительное проектирование		8.0%	7.0%	6.0%	7.0%	6.0%

Виды работ	Тип ПС					
	Web-проекты	**ИС**	**Аутсорсинг**	**Коммерческие**	**Встроенные**	**Военные**
Техническое проектирование		7.0%	8.0%	5.0%	6.0%	7.0%
Проверки проекта			0.5%	1.5%	2.5%	1.0%
Программирование	30.0%	20.0%	16.0%	23.0%	20.0%	16.0%
Включение готовых компонентов	5.0%		2.0%	2.0%	2.0%	2.0%
Приобретение пакетов		1.0%	1.0%		1.0	1.0
Инспекции кода				1.5%	1.5%	1.0%
Независимая верификация и валидация						1.0%
Управление конфигурацией		3.0%	3.0%	1.0%	1.0%	1.5%
Формальная интеграция		2.0%	2.0%	1.5%	% 2.0%	1.5%
Документация пользователя	10.0%	7.0%	9.0%	12.0%	10.0%	10.0%
Автономное тестирование	30.0%	4.0%	3.5%	2.5%	5.0%	3.0%
Функциональное тестирование		6.0%	5.0%	6.0%	5.0%	5.0%
Интеграционное тестирование		5.0%	5.0%	4.0%	5.0%	5.0%
Системное тестирование		7.0%	5.0%	7.0%	5.0%	6.0%
Опытная эксплуатация				6.00%	1.5%	3.0%
Приемочное тестирование		5.0%	3.0%		1.0%	3.0%
Независимое тестирование						1.0%
Обеспечение качества			1.0%	2.0%	2.0%	1.0%
Инсталляция/обучение		2.0%	3.0%		1.0%	1.0%
Управление проектом	10.0%	12.0%	12.0%	11.0%	12.0%	13.0%
Всего	100%	100%	100%	100%	100%	100%
Количество работ	7	18	21	20	23	25

Литература к главе 9

1. *Боэм Б.У.* Инженерное проектирование программного обеспечения. М., Радио и связь. - 1985. – 511 с.
2. *Boehm B.*, Abts C., Chulani S. Software Development Cost Estimation Approaches – A Survey.-46p. /sunset.usc.edu/publications/TECHRPTS/2000/usccse2000-505/usccse2000-505.pdf
3. *Putnam L. A* General Empirical Solution to the Macro Software Sizing and Estimation Problem, IEEE Trans. Software Eng., SE-4, n.4, 1978, pp.345-361.
4. SLIM Suite. Tools for Software Project Management // http://www.qsm.com/SLIMSUITE.pdf
5. *Fischman L.*, McRitchie K., Galorath D. Inside SEER-SEM. -http://www.stsc.hill.af.mil/crossTalk/2005/04/0504Fischman.pdf
6. *Wiegers K.E.* Stop Promising Miracles - http://www.processimpact.com/
7. *Kavoussanakis K*, Terry Sloan UKHEC Report on Software Estimation. – 23p. //www.ukhec.ac.uk/publications/reports/estimation.pdf
8. *Cowderoy*, A.J.C., Jenkins J.O. Cost estimation by analogy as a good management practice. // Proc. Software Engineering 88, ed. Pyle I.C., Liverpool: IEE/BCS.- 1988.- p. 80-84.

9. *Automated* Project Cost Estimation: Using Analogies. The ANGEL Project - http://dec.bournemouth.ac.uk/ESERG/ANGEL/
10. *Shepperd M.*, Schofield C. Effort Estimation by Analogy: A Case Study // ESCOM'7, Wilmslow, 1996, p.4. - http://www.ecfc.u-net.com/cost/index.htm
11. *SPR* KnowledgePLAN - http://www.spr.com/
12. Электронный учебник. – Словарь Н - http://www.statsoft.ru/home/textbook/glossary/gloss_n.html
13. *Gray* A., MacDonell S. A Comparison of Techniques for Developing Predictive Models of Software Metrics // Information and Software Technology.- V.39.- Nov. 6, - June 1997, pp. 425-437.
14. *Neural* Network - //www.statsoft.ru/home/products/neuralnetwork/details/master/default.htm
15. *Stamelos I.*, Angelis L., Sakellaris E. On the Use of Bayesian Belief Networks for the Prediction of Software Productivity // Information and Soft. Techn. - V.45. - N.1.-2003.- P.51-60.
16. *Abdel-Hamid T.* Adapting, Correcting, and Perfecting Software Estimates: a Maintenance Metaphor // IEEE Computer.- 1993.- V.26.- N.3.
17. *Stellman A.*, Greene J. Applied Software Project Management. Estimation - http://www.stellman-greene.com/images/stories/LectureNotes/03%20estimation.pdf
18. *COCOMO* Suite Methodology and Evolution / Barry Boehm, Ricardo Valerdi, Jo Ann Lane, and Winsor Brown // University of Southern California Center for Software Engineering - http://www.stsc.hill.af.mil/crossTalk/2005/04/0504Boehm.html
19. *ISO/IEC 15288: 2002* Systems engineering. Systems life cycle processes (ДСТУ ISO/IEC 15288: 2005 Інженерія систем. Процеси життєвого циклу системи)
20. *Banker D. et al.*, Automation Output Size and Reuse Metrics in a Repository-Based CASE Environment // IEEE Trans. Soft.Eng. - 1994.- SE 20.- N.3.- P.169-186.
21. *COCOMO II* Model Definition Manual (Version 1.4), USC, 1997, 78 p.
22. *http*://www.sunset.usc.edu/research/COCOMOII/cocomo_main.htm
23. *Model*-Based Architecting & Software Engineering (MBASE) - http://sunset.usc.edu/research/MBASE
24. *Jacobson I.*, Booch G., Rumbaugh J. The Unified Software Development Process. // Addison-Wesley, 1999. – 463 p.
25. *Reifer D. J.* Web Development: Estimating Quick-to-Market Software // IEEE Software.- 2000.-V.17.- N.6.-P.57-64.
26. *Reifer, D. J.* SoftCost-OO Reference Manual, Reifer Consultants, Inc.- 1993.
27. *Cleary D.* Web-Based Development and Functional Size Measurement // Proc. of IFPUG Annual Conf., San Diego, USA, Sept. 2000 – www.charismatek.com.au/_public1/pdf/webfsm.pdf
28. *Jones C.* Software Cost Estimating Methods for Large Projects// http://www.stsc.hill.af.mil/crossTalk/2005/04/0504Jones.pdf

Глава 10. УПРАВЛЕНИЕ РИСКОМ ПРОЕКТОВ

10.1. Парадигма управления риском проекта

10.1.1. Элементы парадигмы управления риском SEI

Понятие *риск проекта* связывается с возможностью понести потери в ходе выполнения проекта. Эти потери могут проявляться в снижении качества конечного продукта, превышении стоимости его разработки, задержке окончания разработки или в срыве проекта (то есть, отказе заказчика от проекта).

Величина риска определяется произведением *серьезности последствий* нежелательного события в проекте (уровнем потерь) и *вероятности* наступления этого события.

Серьезность последствий оценивается с позиций влияния нежелательного события, с одной стороны, на характеристики ПС и сложность ее последующего сопровождения, а, с другой, - на эффективность, стоимость и продолжительность процесса разработки ПС.

Вероятность оценивается по степени определенности, с которой можно прогнозировать *проявление* каждого риска в проекте, заключающееся в *перерастании* этого риска в *проблему* для проекта.

Здесь мы использовали определение риска в трактовке института SEI (Software Engineering Institute), который предложил четкую парадигму, программу и ряд практических методов управления риском разработки проектов ПС [1].

Парадигма управления риском упорядочивает различные виды деятельности, касающиеся управления риском в области программной инженерии (рисунок 10.1).

Рис. 10.1. **Парадигма управления риском**

Представление парадигмы в виде круга подчеркивает ее суть, которая заключается в *непрерывности* процесса управления риском. Ориентация стрелок указывает направление логического потока информации, связывающего отдельные виды деятельности. Центральной частью парадигмы является коммуникация, в эффективности средств и методов осуществления которой залог успешного выполнения всех остальных видов деятельности и управления риском в целом.

Краткий обзор основных функций парадигмы представлен в таблице 10.1, а более подробное описание этих функций, а также методов и средств их реализации, содержится в разделе 10.4.

Таблица 10.1. **Функции парадигмы управления риском**

Функции парадигмы	**Цель**
Идентификация	Искать и найти риски проекта до того, как они перерастут в проблемы
Анализ	Преобразовать данные о рисках в информацию, которая может использоваться ЛПР для принятия адекватных решений: определить конкретный источник каждого риска, серьезность последствий, вероятность и время возможного проявления. Произвести классификацию рисков по областям риска и приоритезировать риски по степени важности.
Планирование	Использовать проанализированную информацию о рисках для выработки решений и плана действий по каждому риску. Интегрировать эти решения и планы в единый план управления риском проекта ПС.
Учет и контроль	Наладить учет и контроль состояния каждого риска и действий по его устранению (или снижению его величины), используя для этого метрики риска. Контролировать соблюдение представленного в плане графика действий по риску и эффективность самого плана действий.
Регулирование	На базе учета и контроля каждого риска произвести обоснованную, своевременную и эффективную коррекцию отклонений в запланированных действиях по риску.
Коммуникация	Обеспечить непрерывную эффективную передачу информации и обратную связь со всеми функциями и на всех уровнях управления риском (включая устраняемые, не устраняемые (находящиеся под наблюдением) и вновь появляющиеся риски). Учитывать как внутренние, так и внешние для проекта источники информации о риске.

10.1.2. Таксономия риска

Базис для организации данных и управления риском в парадигме SEI обеспечивает *таксономия риска*. Это средство выделения и изучения различных аспектов риска проекта ПС [2].

Таксономия риска охватывает наиболее общие области риска проекта, касающиеся характеристик ПС, среды и процессов разработки и ограничений проекта[1] (рисунок 10.2).

Эта таксономия имеет иерархическую структуру и систематизирует источники (области) риска по трем уровням (рисунок 10.3):

- класс,
- элемент класса,
- атрибут элемента[2].

---

[1] Таксономия риска SEI была проверена на более чем 30 проектах ПС. На практике она может частично видоизменяться с учетом специфики конкретного проекта.

[2] Сохранено название уровней, предложенное SEI

A. Технические аспекты разработки

1. Требования
- а) Стабильность
- б) Полнота
- в) Однозначность
- г) Достоверность
- д) Реализуемость
- е) Новизна
- ж) Масштабность

2. Проект
- а) Функциональность
- б) Сложность
- г) Интерфейсы
- д) Производительность
- е) Тестопригодность
- ж) Аппаратные ограничения
- з) Приобретаемое ПО

3. Кодирование и автономное тестирование
- а) Реализуемость
- б) Автономное тестирование
- в) Кодирование/реализация

4. Интеграция и интеграционное тестирование
- а) Среда
- б) Интеграция продукта
- в) Интеграция системы

5. *Нефункциональные характеристики*
- а) Удобство сопровождения
- б) Надежность
- в) Защищенность
- г) Безопасность
- д) Человеческие факторы
- е) Спецификации

B. Среда и технология разработки

6. Процесс разработки
- а) Формализованность
- б) Укомплектованность
- в) Контролируемость процесса
- г) Опыт применения
- д) Контролируемость продукта

7. Система поддержки разработки
- а) Мощность
- б) Укомплектованность
- в) Удобство применения
- г) Опыт применения
- д) Надежность
- е) Сопровождаемость
- ж) Поставка

8. Процесс управления
- а) Планирование
- б) Организация проекта
- в) Опыт управления
- г) Организация взаимодействия

9. *Методы управления*
- а) Мониторинг
- б) Управление персоналом
- в) Обеспечение качества
- г) Управление конфигурацией

10. *Рабочая обстановка*
- а) Качество работы
- б) Кооперация
- в) Коммуникация
- г) Моральный климат

C. Внешние ограничения проекта

11. Ресурсы
- а) Сроки разработки
- б) Штат проекта
- в) Финансирование
- г) Средства разработки

12. Условия договора
- а) Тип договора
- б) Ограничения договора
- в) Договорные зависимости

13. Интерфейсы проекта
- а) Заказчик
- б) Смежники
- в) Соисполнители
- г) Головной исполнитель
- д) Высшее руководство
- е) Продавцы
- ж) Политические принципы

Рис. 10.2. **Таксономия риска**

Класс определяет *сферу деятельности* по программной инженерии, с которой может быть связан тот или иной риск. Элемент класса указывает *конкретную область риска* в соответствующей сфере деятельности.

Атрибут элемента определяет *фактор риска* в определенной области риска, с которым может быть связано нежелательное событие, действие или факт, являющиеся источником риска.

Основные *классы* областей риска таковы:

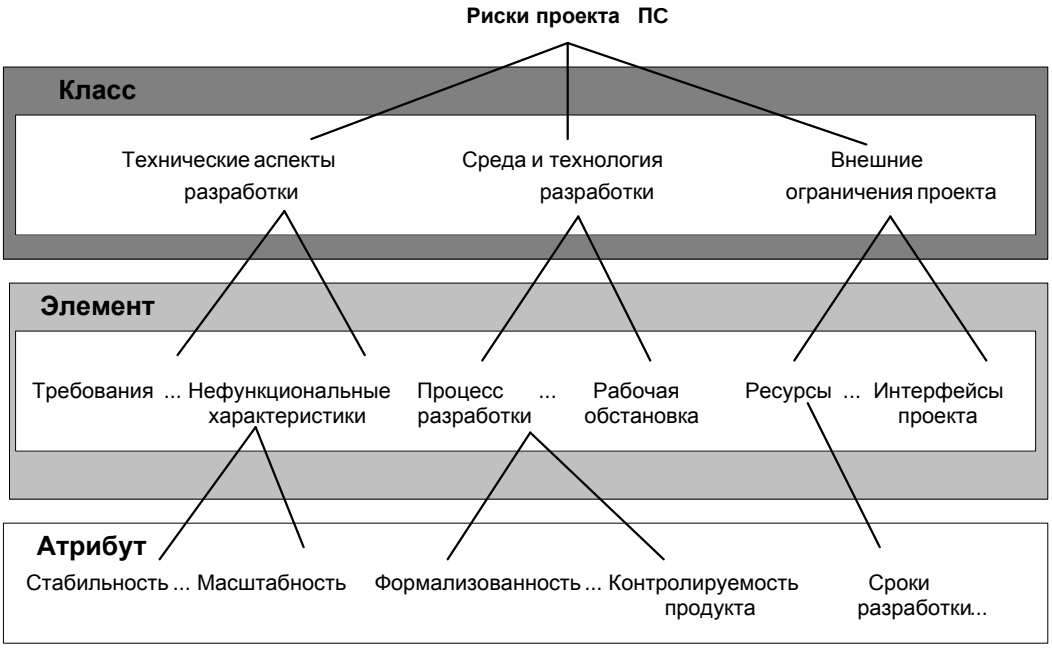


Рис. 10.3. **Структура таксономии риска**

1. *Технические аспекты разработки* (инженерия программного продукта). В этот класс входят области риска, отождествляемые с процессами (работами) на стадиях ЖЦ ПС (разработка требований, проектирование, кодирование, тестирование и др.) и характеристиками рабочих продуктов (требований, проекта, кода и др.) на этих стадиях;

2. *Среда и технология разработки*. В этот класс входят области риска, отождествляемые с методами, процедурами и инструментами, используемыми в ходе разработки ПС;

3. *Внешние ограничения проекта*. В этот класс входят области риска, отождествляемые с внешними для проекта факторами, такими как наличие ресурсов разработки, условия заключаемых договоров, формы и особенности взаимодействия организаций-участников проекта ПС и др.

Таксономия риска не только предлагает способ систематизации рисков по указанным в ней аспектам программной инженерии, но и служит основой для разработки методов идентификации источников риска путем интервьюирования членов проекта с использованием вопросника, согласующегося с этой таксономией.

Вопросник, основанный на таксономии риска (для краткости, TBQ, от Taxonomy-Based Questionnaire), является инструментом, применение которого гарантирует охват всех потенциальных областей риска благодаря наличию в нем вопросов, касающихся нижнего уровня таксономии риска - атрибутов. Количество и форма задаваемых вопросов может быть различной в зависимости от специфики проекта, выбранного метода интервьюирования и обработки его результатов. В любом случае они должны ориентироваться на максимально полное и эффективное извлечение знаний членов проекта (включая менеджеров, проектировщиков, технический

персонал и др.) о рисках конкретного проекта ПС.

Обобщенный вариант TBQ института SEI использует определенные соглашения и правила задания и интерпретации ответов на вопросы (касающиеся нумерации вопросов, их структуры и др.). В приложении 6 предлагается несколько отличный от используемого SEI вариант TBQ, разработанный с учетом удобства последующей автоматизации процесса интервьюирования и обработки ответов.

10.2. Обзор методологий управления риском проекта

10.2.1. Методология оценивания риска

SEI предлагает использовать три методологии управления риском ПС:

- оценивание риска,
- непрерывное управление риском,
- коллективное управление риском.

Методология оценивания риска SRE (от Software Risk Evaluation) включает формальный метод идентификации, анализа, контроля и устранения рисков ПС, который применяется первоначально на самой ранней стадии разработки проекта ПС (еще до заключения договора с разработчиком) и затем периодически в ходе всего ЖЦ проекта.

Предлагаемые SRE функции управления риском подразделяются на основные и обеспечивающие.

Основные функции. К основным функциям относятся обнаружение, спецификация, оценивание, структурирование (консолидация) и устранение рисков.

Выполнение функции *обнаружения рисков* обеспечивает систематический и полный охват всех потенциальных областей риска с применением адекватных инструментов и технологий, в частности, вопросника TBQ.

Выполнение функции с*пецификации риска* касается фиксации *всех* аспектов идентифицированного риска ПС. Форма-спецификация риска представляет собой структуру, которая упрощает решение задач приоритезации рисков, локализации источников и причин возникновения рисков, определения методов и усилий, предпринимаемых для устранения рисков в источниках их возникновения.

Существует много способов спецификации риска - от простого утверждения о риске до сложного высказывания с применением условных выражений вида «ЕСЛИ <условие> ТО <последствия>», связок «И» и др. Выбор того или иного способа определяется эффективностью его практического применения для конкретного проекта и особенностями инструментальной поддержки обработки рисков. Наряду с утверждением о риске в спецификации указывается источник риска, ассоциированный с категорией (номером) риска по таксономии, и другая информация.

Функция *оценивания риска* заключается в определении величины каждого риска, что служит основанием для присваивания приоритета риску и выявления наиболее важных на текущий момент рисков проекта.

Существуют различные механизмы оценивания величины риска - от простых субъективных оценок по 3-бальной шкале до строгих статистических измерений.

Примеры простых механизмов оценивания величины риска представлены в таблице 10.2 и на рисунке 10.4.

Таблица 10.2. **Шкала для оценивания рисков**

Величина риска	**Описание**
Критический	Высокая вероятность того, что есть серьезный риск для многих факторов проекта ПС (стоимости, эффективности, сопровождаемости и др.).
Высокий	Высокая вероятность того, что есть умеренный риск для одного или более факторов проекта.
Средний	Средняя вероятность того, что есть умеренный риск для одного или более факторов проекта.
Низкий	Низкая вероятность того, что есть умеренный риск для одного из многих факторов проекта.

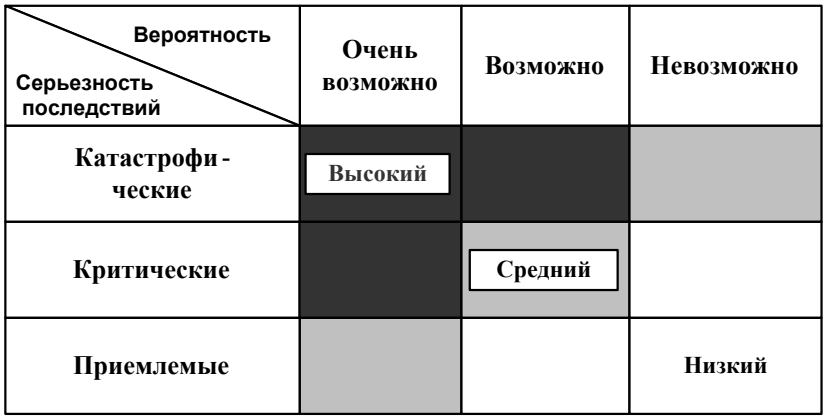


Рис.10.4. **Матрица трехуровневой оценки риска**

Функция *консолидации рисков* касается преобразования разрозненных данных о каждом риске в концентрированные информационные блоки для принятия решений по управлению риском. Преобразование данных основано на применении приемов объединения, структурирования и абстракции данных об отдельных рисках ПС (что, например, необходимо, в случае обнаружения идентичных рисков в разных сеансах интервьюирования, различных проявлений одного и того же риска, поглощающих рисков от одного источника, рисков, мало отличающихся по величине, и др.).

Функция *устранения* рисков состоит в определении областей проекта, на которых необходимо сосредоточить ресурсы (и называемых областями устранения риска), а также разработке *стратегий* и *рекомендуемых действий*, способных снизить и устранить угрозу успешного выполнения проекта. *Область устранения риска* представляет собой логическую группировку подобных рисков, удобную с точки зрения эффективного управления этими рисками. Результатом выполнения данной функции является *план управления риском проекта ПС,* а также элементы организационно-методической, технологической и инструментальной поддержки управления риском проекта ПС.

Собственно устранение каждого риска осуществляется в соответствии с конкретным планом его устранения и поддерживается такими функциями парадигмы управления риском, как планирование, учет, контроль и регулирование риска.

Обеспечивающие функции. К обеспечивающим функциям при оценивании риска, предлагаемым методологией SRE, относятся функции согласования действий по управлению риском, их планирования и координации, верификации и валидации, а также обучения и создания условий для эффективного ведения (хранения, обновления, удаления) информации о риске.

Организации, в которых внедряется процесс управления риском проектов ПС, как правило разрабатывают собственный или адаптируют приведенный метод оценивания риска к потребностям и инфраструктуре собственных проектов. Некоторые практические предложения по оцениванию риска разработки отечественных проектов представлены в разделе 10.5.

10.2.2. Непрерывное управление риском

Методология непрерывного управления риском CRM (от Continuous Risk Management) основана на определенных принципах управления риском в ходе всего ЖЦ проекта и не зависит от конкретных применяемых методов и инструментов оценки и устранения риска. Принципы управления риском таковы:

- *разностороннее (в разных проекциях) видение предмета*
  - формирование взгляда на предмет с разных позиций при сохранении общности целей, коллективной ответственности и распределении обязанностей;
  - концентрация внимания на результатах;
- *коллективная работа*
  - совместная работа для достижения общей цели;
  - оптимальные условия проявления таланта, знаний и опыта;
- *глобальные перспективы*
  - восприятие разработки ПС в контексте крупномасштабных работ по проекту системы;
  - допущение как благоприятных, так и неблагоприятных перспектив реализации спецификаций ПС (связанных с превышением сроков, стоимости, программными сбоями и др.);
- *дальновидность*
  - опережение событий, идентификация неопределенностей, предупреждение потенциальных проблем;
  - управление проектными ресурсами и действиями в режиме предвосхищения проблем;
- *открытое взаимодействие*
  - поощрение свободного тока информации между всеми уровнями проекта;
  - обеспечение возможности формального, неформального и импровизированного взаимодействия;

- обеспечение процесса достижения консенсуса с учетом мнений отдельных лиц (имеющих специальные знания и углубленный взгляд на проблему идентификации и управления конкретным риском);

- *интегрированное управление*
  - признание управления риском в качестве интегральной и жизненно важной части управления проектом;
  - адаптация методов и инструментов управления риском к инфраструктуре и культуре проекта;
- *непрерывность процесса*
  - постоянство бдительности;
  - непрерывная идентификация и управление рисками на всех стадиях ЖЦ проекта.

На этих принципах основаны базовые функции управления риском - идентификация, анализ, планирование, учет и контроль, регулирование и коммуникация, которые детально описаны в разделе 10.4.

10.2.3. Коллективное управление риском

Методология коллективного управления риском TRM (от Team Risk Management) определяет дополнительные действия в деятельности по управлению риском, которые связаны с осуществлением *совместного* управления риском со стороны заказчика проекта ПС и его исполнителя. Ее применение обеспечивает формирование среды, содержащей множество процессов, методов и инструментов, необходимых для совместной работы заказчика и исполнителя над непрерывным управлением риском в ходе ЖЦ проекта.

Методология TRM основана на семи рассмотренных выше принципах управления риском и философии бригадной (групповой) работы. Она развивает парадигму управления риском, предложенную SEI, путем добавления двух функций - *инициирование* коллективной работы и собственно *коллективное управление* риском. Эти функции выполняются последовательно применительно к каждому риску, но действия по управлению риском проекта в целом могут быть как последовательными, так и параллельными, и итеративными (например, планирование для одного риска может совмещаться с идентификацией другого).

Инициировать коллективную работу может либо заказчик, либо исполнитель. При этом оба коллектива должны осознавать необходимость соблюдения принципов коллективной работы и ответственность за ее результаты.

Коллективное управление предполагает формализацию отношений заказчик-исполнитель и формирование обобщенных взглядов на риски ПС. Систематическое и периодическое совместное применение методов управления риском обеспечивает единообразие в понимании рисков проекта и их относительной важности и получение обобщенной информации о рисках, приоритетах, метриках и планах действий.

Основные положения трех методологий, представленных выше, послужили базисом для формирования подхода к управлению риском проектов ПС в терминах процесса управления риском, его организационной структуры и стадий выполнения, а также необходимой инструментальной поддержки модели принятия решений при оценивании риска.

10.3. Процесс управления риском проекта

10.3.1. Требования к процессу управления риском

В стандарте ISO/IEC 12207 указывается, что «назначение процесса управления риском состоит в *непрерывной* идентификации и уменьшении рисков проекта в течение всего жизненного цикла проекта. В результате успешного осуществления процесса:

- будет определена сфера управления риском;
- будут определены и внедрены надлежащие стратегии управления риском;
- кроме рисков проекта, оговоренных в стратегии управления риском проекта, будут идентифицироваться риски, возникающие в ходе выполнения проекта;
- риски будут анализироваться, и ресурсы для мониторинга этих рисков будут выделяться в соответствии с приоритетами, назначаемыми рискам;
- методы мониторинга риска будут выбираться таким образом, чтобы была возможность выявлять изменения в состоянии риска и контролировать действия по мониторингу;
- будут приниматься надлежащие меры по устранению и предупреждению влияния риска» [3].

Нужно отметить, что процесса управления риском не было среди процессов ЖЦ в первой редакции стандарта ISO/IEC 12207:1995, и его появление в проекте новой версии стандарта свидетельствует о признании важности проблемы управления риском проекта для разработки высококачественных систем. Концепция *непрерывности* управления риском, лежащая в основе парадигмы SEI, позволяет построить процесс управления риском, соответствующий требованиям стандарта ISO/IEC 12207.

Процесс управления риском проекта ПС, предлагаемый SEI, включает следующие четыре этапа [4]:

- *согласование целей* - определение нужд и целей проекта, достижение соглашений по управлению риском,
- *подготовка работ* - планирование и координация предстоящих работ по оцениванию риска проекта ПС,
- *оценивание риска* - выполнение функций SRE и получение рекомендаций по управлению риском проекта ПС,
- *подготовка к устранению риска* - разработка рекомендаций по устранению рисков по всем областям устранения риска,
- *разработка плана управления риском* и приведение его в действие.

Процесс управления риском применяется *независимо* от стадии ЖЦ, на которой находится проект, и независимо от конкретного сценария и форм организации взаимодействия действующих лиц при разработке проекта - заказчика, исполнителя, соисполнителей, сторонних экспертов и др.

Один из возможных примеров использования процесса управления риском представлен на рисунке 10.5.

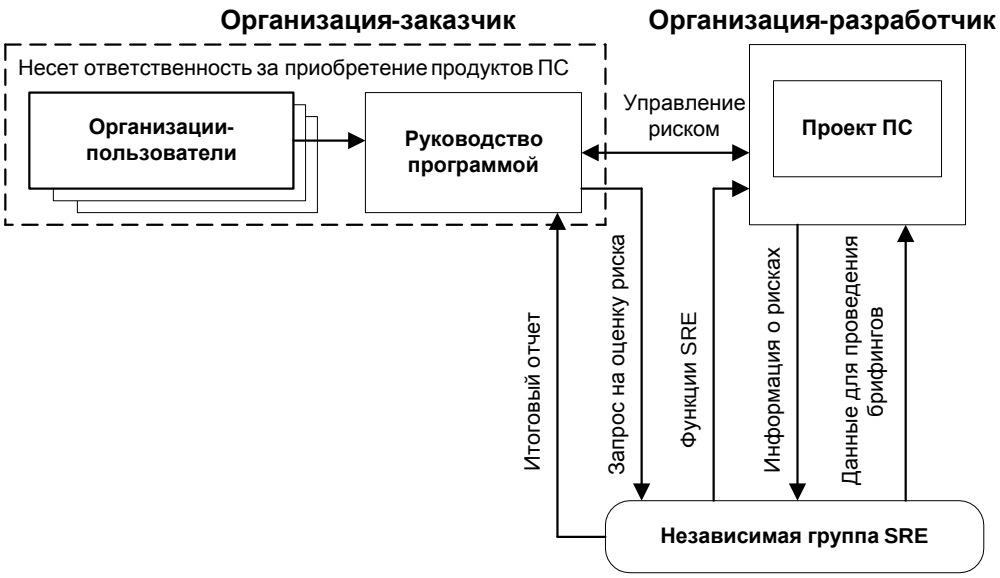


Рис. 10.5. **Пример применения процесса управления риском**

В этом примере *заказчиком* проекта ПС, в разработке которой заинтересована широкая общественность, является государственная организация-посредник, ответственная за выполнение программы информатизации, включающей ряд проектов. Эта организация заключает договор с определенной *организацией-разработчиком* на выполнение проекта. Далее, для внедрения процесса управления риском и выполнения оценок риска срыва этого проекта она обращается к независимой организации, оказывающей услуги по управлению риском проектов, и нанимает *независимую группу экспертов*, становясь ее *клиентом*.

Независимая группа выполняет функции SRE (см. п.10.2.1), обеспечивает подготовку и проведение *брифингов* и совещаний по оценке риска на территории организации-разработчика и анализ полученной информации о рисках. Совместно с заинтересованными организациями независимая группа разрабатывает стратегии устранения риска и составляет рекомендации по управлению риском с учетом особенностей конкретного проекта.

Результаты оценки риска проекта в виде итогового отчета передаются клиенту. Этот отчет содержит описание всех найденных рисков и предложения по их устранению.

Руководствуясь полученным отчетом, организация-заказчик совместно с организацией-исполнителем составляет *план управления риском* проекта ПС (п.10.3.8) и в дальнейшем периодически контролирует его выполнение.

Организация-разработчик обеспечивает выполнение функций учета, контроля и регулирования указанных в отчете рисков по соответствующим частным *планам устранения риска* для каждого риска (или группы рисков из одной области риска). План устранения риска разрабатывается для наиболее важных рисков, установленных в процессе идентификации и анализа рисков.

Группа проекта в ходе процессов учета и контроля риска должна отслеживать с одной стороны выполнение плана устранения риска, а с другой - его эффек-

тивность в снижении конкретного риска. Если план не выполняется, он может быть заменен новым планом, либо в действие может быть введен заранее подготовленный альтернативный план выхода из исключительных ситуаций. Если план устранения риска успешно выполнен - риск можно считать *закрытым*.

Закрытый риск удаляется из списка наиболее важных рисков, а его место в списке занимается следующим риском, который получает статус важного риска с определенным приоритетом.

Для нового риска составляется план устранения и этот риск включается в сферу управления риском.

Клиентом независимой организации, оказывающей услуги в управлении риском, может стать не только организация-заказчик, но и организация-разработчик ПС, если эта организация по собственной инициативе внедряет процесс управления риском, стремясь усовершенствовать процесс разработки ПС.

Процесс управления риском проекта ПС начинается с осознания руководством необходимости определения угроз и оценки риска проекта и с создания *координационной группы (совета)* по управлению риском проекта. Численность группы может колебаться от одного человека с частичной занятостью до нескольких человек с полной занятостью. Кандидатами в координационную группу могут быть члены группы проекта, представители пользователей, исполнителей и соисполнителей. *Лидером* этой группы является представитель нанимаемой независимой группы экспертов, который несет ответственность за выполнение обеспечивающих функций планирования и координации в рамках метода SRE. Интерфейс организации-разработчика с независимой группой (в лице ее лидера) осуществляется *координатором* действий из организации-разработчика. Он несет ответственность за подбор (в среде проекта) специалистов для интервьюирования и проведение брифингов при посещении организации-разработчика независимой группой экспертов.

Собственно деятельность по управлению риском проекта ПС должна осуществляться квалифицированным персоналом, включенным в состав проекта (*группой проекта*), а распределение ролей и ответственности за управление риском - отражаться в плане управления риском. Один из возможных способов распределения ответственности при управлении риском представлен на рисунке 10.6.

В данном примере каждый *член группы* несет ответственность за непрерывный процесс идентификации, оценивания и классификации рисков и формирование планов устранения рисков. Утверждают эти планы *технические лидеры* группы проекта. Ответственность за отслеживание состояния рисков проекта может быть возложена и на отдельное лицо или группу лиц в проекте.

Менеджер проекта производит ревизию и интеграцию информации о рисках, назначение ответственных за риски и планы устранения, а также обеспечивает взаимодействия с внешним окружением проекта.

Группа проекта периодически предоставляет отчеты о своей деятельности *руководству организации-заказчика*. Эти результаты касаются состояния группы рисков наибольшей важности (например, первой десятки или, в общем случае, первой Н-ки рисков) и планов по их устранению. Кроме того, предоставляется информация об эффективности процесса управления риском (например, данные об интенсивности идентификации новых рисков по отношению к интенсивности устра-

нения рисков и др.). Детальная ревизия этой информации производится менеджером проекта на периодической и событийной основе.

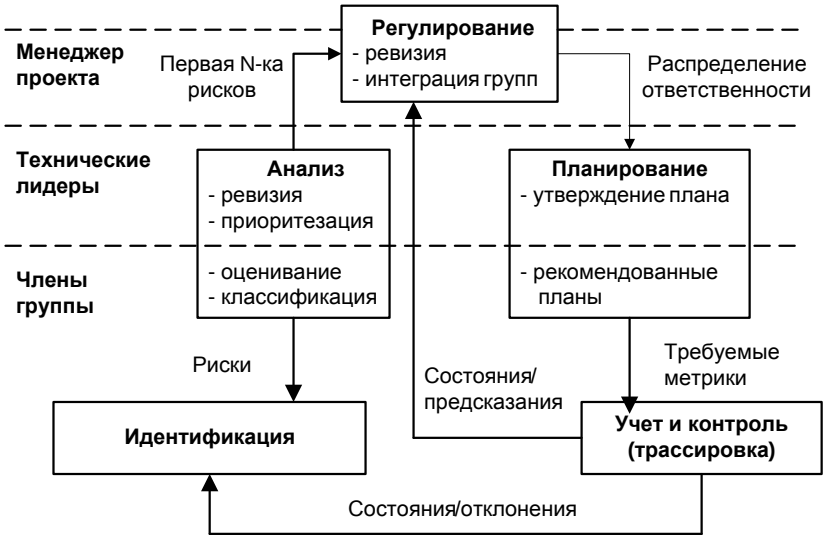


Рис. 10.6. **Распределение ролей при управлении риском**

Лица, осуществляющие функции управления риском проекта ПС, должны обладать знаниями в области программной инженерии, проблемной области, для которой разрабатывается ПС, а также знать принципы управления риском и владеть методами и инструментами управления риском.

Считается, что человек обладает необходимыми знаниями для управления риском, если:

- он принимал участие в управлении, по крайней мере, одним проектом,
- применял методологии управления риском, по крайней мере, для одного проекта и
- имеет опыт в соответствующей проблемной области проекта.

Если не выполняется хотя бы одно из перечисленных условий, член группы проекта должен иметь возможность пополнить свои знания путем обучения.

Способы приобретения необходимых знаний таковы:

- формальное обучение. Занятия с членами проекта проводятся организацией-заказчиком или независимой организацией по соответствующей программе;
- неформальное обучение. Этот вид обучения предполагает самостоятельное посещение курсов или обучение по месту работы по соответствующей программе.

10.3.2. Этап согласования целей

На этапе согласования целей проводится не менее двух совещаний, по результатам которых заключается соглашение с организацией-разработчиком на выполнение оценки риска проекта ПС.

На *первом совещании*, проводимом высшим *руководством* организации-разработчика совместно с *лидером независимой группы экспертов* (из независимой

организации-эксперта) и *представителями группы проекта*, обсуждаются установленные цели проекта и возможности их достижения.

В результате совещания:

- должны быть определены функции всех групп, представители которых принимают участие в совещании;
- группа проекта должна получить достаточное представление о целях и задачах SRE и быть способной определить контекст применения SRE в своем проекте;
- независимая организация-эксперт должна хорошо понять стратегический контекст оценивания и получить ответы на вопросы о том, с какими стратегическими проблемами сталкивается организация? Каковы отдаленные перспективы завершения проекта? Каковы ожидаемые результаты оценивание риска? Каковы ближайшие перспективы организации? Какова степень критичности оценивания риска? Каким рискам подвергается проект с точки зрения менеджеров? Каковы оценки времени их перерастания в проблемы? Какого рода действия планирует руководство в связи с отмеченными рисками?
- независимая организация-эксперт и бригада проекта должны взвесить возможности выполнения оценивания риска, готовность к совместному выполнению работ и потребность в согласовании своих действий с высшим руководством организации;
- должна быть назначена дата второго совещания.

Целью *второго совещания* является получение заключений и утверждение договоренностей о проведении оценивания риска ПС. Наряду с участниками первого совещания на второе совещание должны быть приглашены группы технической поддержки из организации-разработчика.

На этом совещании рассматриваются проблемы, с которыми может столкнуться разработчик проекта, и подвергаются ревизии уже утвержденные цели проекта в части оценки риска. Здесь также поднимаются вопросы обеспечения ресурсами процедур управления риском.

В результате совещания:

- представители организации-разработчика должны понимать, в чем заключается SRE, как долго выполняется оценка риска, каковы ресурсы, роли и типичные результаты проведения SRE, каковы требования к конфиденциальности информации о рисках;
- независимая организация-эксперт должна понимать все проблемы клиента, связанные с угрозами проекту, и должна разрешать те из них, которые касаются оценивания риска;
- независимая организация-эксперт должна четко понимать, какие цели преследует клиент, проводя оценивание риска, каков контекст оценивания в терминах стратегических направлений и целей организации-разработчика;
- клиент и независимая организация-эксперт должны достичь соглашения по вопросам логистики SRE: определить сроки и необходимые ресурсы, утвердить координатора и т.д.;
- клиент должен осознать важность роли координатора;
- должен быть определен следующий шаг процесса управления риском.

10.3.3. Этап подготовки работ

Этап подготовительных работ состоит в адаптации метода оценивания риска к нуждам организации-разработчика, подборе независимой группы управления риском из организации-эксперта, подборе групп, с которыми будет проводиться интервьюирование, и планировании оценивания риска.

Проведение подготовки к оцениванию риска возлагается на лидера независимой группы экспертов и координатора (со стороны организации-разработчика).

Цель *адаптации метода SRE* к особенностям и целям проекта ПС состоит в том, чтобы обеспечить эффективное применение предлагаемых SRE средств в условиях среды разработки проекта. Адаптация предполагает соотнесение целей организации-разработчика с таксономией риска, которая была согласована на предыдущем этапе, определение вида и числа групп, которые должны интервьюироваться, и приведение TBQ в соответствие с целями проекта.

Независимая группа управления риском комплектуется из 3 - 4 экспертов из независимой организации, оказывающей услуги в управлении риском, и 2 - 3 человек из организации-клиента, непосредственно не связанных с разрабатываемым проектом. Участники группы со стороны организации-клиента не должны иметь отношений подчиненности с интервьюируемыми лицами и не должны вовлекаться в повседневную деятельность по разработке проекта. Однако они должны обладать надлежащими знаниями организационных, технических и финансовых аспектов проекта, а также иметь представление о процессе разработки ПС, используемых технологиях и прикладной области проекта. Независимая группа должна пройти формальное обучение до того, как приступит к исполнению обязанностей по управлению риском проекта ПС. Ответственность за подбор группы и распределение обязанностей в ней несет лидер группы.

Интервьюируемые группы могут формироваться исходя из различных критериев, например, по принадлежности к организации или сфере деятельности, в контексте определенного класса продуктов, по спектру заказчиков или исполнителей проекта, а также в разрезе функций разработки ПС. Обычно формируются следующие группы интервьюирования:

- группа, включающая аналитиков, проектировщиков, тестировщиков,
- группы различной профессиональной (функциональной) ориентации: независимой верификации и валидации, обеспечения качества, управления конфигурацией и др.,
- группа менеджеров проекта ПО системы,
- группа инженеров-системотехников и программистов,
- группа заказчиков и пользователей,
- группа соисполнителей проекта и др.

Все члены одной группы интервьюирования не должны иметь отношений подчиненности, что даст им возможность свободно излагать свои мысли по поводу рисков, проблем и особенностей проекта. Обычная численность группы интервьюирования – 5 - 6 человек. Ответственность за ее формирование и функционирование при проведении сеансов интервьюирования несет координатор, работающий под руководством лидера независимой группы экспертов и использующий в своей работе инструктивно-методические материалы этой группы.

Деятельность всех групп интервьюирования должна *тщательно планироваться* с учетом не одновременности проведения сеансов интервьюирования и занятости специалистов по основному виду деятельности, наличия свободных помещений для проведения работы и других ресурсов.

10.3.4. Этап оценки риска

Этот этап процесса управления риском охватывает совокупность действий, выполняемых «на месте» - в ходе посещения независимой группой экспертов организации, разрабатывающей проект ПС (рисунок 10.7).

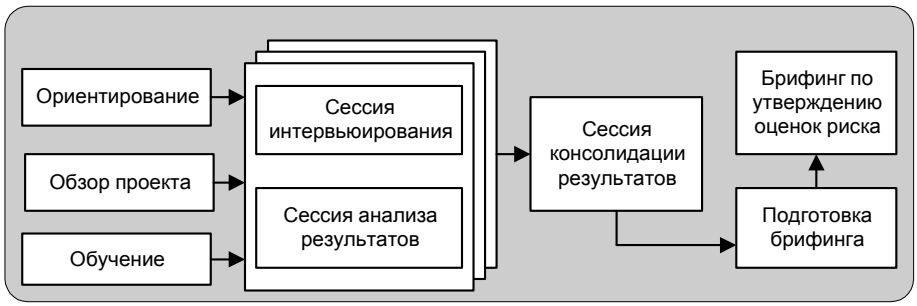


Рис. 10.7. **Действия по оценке риска «на месте»**

Назначение *ориентирования*, ответственность за проведение которого возлагается на лидера независимой группы, - ознакомить всех участников этой стадии процесса управления риском с целями, методами и средствами оценивания риска, а также ролью каждого участника в оценивании. В заседании принимают участие руководство, персонал, имеющий отношение к разрабатываемому проекту, и независимая группа экспертов.

Заседание по *обзору проекта* проводится с целью ознакомления независимой группы экспертов со структурой, характеристиками, терминологией и функциональными аспектами разрабатываемого проекта ПС. Это заседание открыто для всех участников процесса оценивания риска.

Обучение состоит в овладении группой независимых экспертов практическими приемами выполнения действий по оцениванию риска проекта путем проведения деловой игры, в основе которой лежит моделирование функций парадигмы риска, использования таксономии риска и TBQ.

Целью *сессий интервьюирования* является обнаружение и спецификация рисков. В основе организованной дискуссии лежит подготовленный (и адаптированный) вопросник TBQ, хотя строгое следование структуре вопросника и необязательно. Утверждение о каждом идентифицированном риске документируется в терминах лексики участников. Наряду с утверждением о риске фиксируется контекст риска и наиболее важные моменты дискуссии. Члены независимой группы должны гарантировать, что в результате сессии интервьюирования:

- ни один потенциальный риск или проблема не опущены;
- источник риска установлен и может быть отнесен к соответствующему классу таксономии риска;
- группа располагает достаточной информацией для выполнения объективной оценки каждого обнаруженного риска.

Цель *сессии анализа результатов* - завершение функций спецификации и оценивания риска. Сессия проводится независимой группой экспертов сразу же по окончании каждой сессии интервьюирования, пока контекст риска еще свеж в умах членов группы. Каждый член группы получает копию записей о рисках и обсуждает только те риски, формулировки по которым вызывают разночтения и нуждаются в изменении. Далее члены группы независимо друг от друга дают оценку риска, используя заранее утвержденный механизм оценивания, а затем обсуждают те риски, оценки которых существенно отличаются у разных экспертов, и пытаются достичь консенсуса. Каждый конкретный риск обретает свое место в таксономии риска. Всевозможные корректировки утверждений о риске, а также оценки рисков и место рисков в таксономии документируются наряду с обоснованиями принятых решений.

Цель *сессии консолидации результатов* состоит в совместном рассмотрении всего множества рисков и его структурировании. Она проводится независимой группой экспертов по завершении всех сессий интервьюирования и анализа. Каждый эксперт получает копию описания всех рисков, отсортированных по уровням оценок величины в рамках каждой категории по таксономии. Далее группа совместно рассматривает риски в каждой категории с целью выявления рисков-кандидатов для консолидации, достигает консенсуса по окончательной формулировке и оценке консолидированного риска. Описания консолидированных рисков документируются наряду с обоснованиями принятых решений.

Цель заседания по *подготовке брифинга* состоит в оформлении полученных результатов оценок для их краткого представления «на месте». На этом этапе формируется список всех рисков, отсортированных по категориям и величине. По первым «Н» особо важным рискам готовится наиболее полная информация. С целью обеспечения конфиденциальности источников данных производится ревизия подготовленных сообщений и наглядных пособий (например, слайдов), представляемых на брифинге, и вносятся необходимые поправки.

Брифинг по утверждению оценок риска проводится с целью ознакомления руководства организации-разработчика и всех участников оценивания риска с результатами работы. Таким образом, путем открытого взаимодействия обеспечивается обратная связь в управлении риском. Полученные организацией-разработчиком результаты оценивания проверяются и утверждаются. По завершении брифинга копии слайдов предоставляются организации-разработчику. О любых неточностях в слайдах сообщается лидеру группы оценивания, который несет ответственность за внесение всех необходимых поправок в документацию о рисках.

10.3.5. Этап подготовки к устранению риска

Этот этап охватывает действия по подготовке итогового отчета для клиента. Действия выполняются совместно независимой группой экспертов и представительной группой лиц-участников проекта в ходе одного-двух посещений организации-разработчика.

Последовательность действий на этапе подготовки к устранению риска такова:

- подготовка предварительного отчета о рисках,

- совещание по результатам ревизии рисков клиентом,
- разработка мероприятий по устранению риска,
- подготовка итогового отчета.

Подготовка предварительного отчета выполняется независимой группой экспертов по завершении стадии оценивания риска. Ее цель состоит в том, чтобы представить результаты анализа рисков и предложить стратегии их устранения. Предварительный отчет должен содержать описания консолидированных рисков и областей устранения риска, проекции рисков на соответствующие области устранения и цели клиента, а также стратегии устранения или общие рекомендации по устранению риска.

Подготовленный предварительный отчет подвергается ревизии со стороны руководства клиента и представителей проекта, а затем обсуждается на *совещании по результатам ревизии*.

В ходе обсуждения предварительного отчета устанавливается, отражает ли он проблемы/риски организации? Соответствует ли он целям успешного завершения проекта? Оправдывает ли он ожидания/цели, поставленные до начала оценивания риска?

Обсуждение рисков происходит в рамках областей устранения риска, указанных в отчете. В результате дискуссии определяется, связаны ли каким-либо образом идентифицированные области устранения риска с целями и задачами организации? Достаточно ли практичны рекомендации в контексте целей и задач проекта? Остались ли за рамками отчета какие-либо вопросы?

В результате совещания клиент должен установить приоритеты важности рекомендаций, определить наличие ресурсов и необходимое время для устранения риска, выяснить, какого рода действия должно предпринять руководство для устранения риска.

Цель *разработки мероприятий по устранению риска* - облегчить процесс создания плана устранения риска проекта ПС путем выработки рекомендаций, детальных процедур, оптимальных последовательностей действий с расчетами стоимости работ и указанием измеряемых характеристик.

Для выработки необходимых мероприятий по устранению риска независимая группа экспертов совместно с представителями проекта проводит серию совещаний (как правило, на территории организации-разработчика). В ходе совещаний для каждой области устранения риска идентифицируются ограничения среды разработки проекта, которые потенциально могут стать препятствием на пути реализации плана устранения риска в данной области риска, и вырабатываются соответствующие рекомендации по преодолению этих препятствий.

Для каждой области устранения риска выясняется, каково текущее состояние проекта по данной области устранения риска? Каково целевое состояние должно быть достигнуто? Что может быть сделано для достижения этого состояния и какова общая стратегия действий? Какова последовательность конкретных действий? При наличии четких рекомендаций по составу плана устранения риска в каждой области устранения риска руководству клиента в дальнейшем останется принять решения, касающиеся ресурсов, сроков выполнения и ответственных исполнителей работ.

Цель *подготовки итогового отчета* - довести результаты оценивания риска до руководства организации-клиента. Итоговый отчет формируется на базе предварительного отчета, но содержит уточненные оценки, прошедшие ревизию клиента, а также копии всех представленных на брифинге слайдов и другие материалы, которые использовались при оценке риска.

10.3.6. Разработка плана управления риском

Управление риском конкретного проекта разработки ПС осуществляется членами проекта (представителями заказчика, исполнителя, соисполнителей) в соответствии с планом управления риском, составленным с учетом рекомендаций, полученных от независимой группы экспертов и зафиксированных в итоговом отчете о проведении оценки риска проекта ПС.

План управления риском проекта ПС определяет, каким образом процесс управления риском будет выполняться в рамках конкретного проекта разработки ПС, и указывает процессы, действия, этапы работ, методы и инструменты, используемые членами группы проекта, ответственными за управление риском проекта ПС. Этот план может быть частью плана управления риском системного уровня, частью плана управления проектом, или самостоятельным планом. Выбор того или иного способа планирования зависит от объема и сложности проекта, состава и размеров группы проекта, а также общих принципов планирования, которым следует организация, осуществляющая управление риском.

Примерная структура ***плана управления риском*** такова:

Введение

Раздел определяет цели и сферу действий плана, а также содержание плана. Здесь должны быть указаны любые предположения, ограничения и принципы реализации процессов, а также любые связанные с этим планы, документы и стандарты.

Обзор процессов

В этом разделе приводится краткое описание действий по управлению риском и их взаимосвязь. Определяются все потоки управления и данных, указывается, каким образом действия по управлению риском интегрируются с другими действиями по управлению проектом.

Организация

Этот раздел плана включает информацию об организации работ по управлению риском проекта и распределении ответственности (обязанностей) в проекте между заказчиком, поставщиком ПС (головным исполнителем проекта) и соисполнителями проекта. Указывается организационная структура управления риском, которая проецирует действия по управлению риском на роли, исполняемые членами проекта по управлению риском. Определяется перечень лиц, ответственных за управление риском со стороны организации-заказчика, организации-разработчика и организаций-соисполнителей, а также перечень выполняемых ими действий (процедур) и описание конкретных ожидаемых результатов этих действий.

Детали процесса

В этом разделе подробно описываются процессы и процедуры, необходимые для систематического управления риском. Эта часть плана также включает:

1) методы и инструменты, которые выбираются для поддержки каждой функции, а также критерии их выбора;

2) ссылки на другие планы, руководства и учебные материалы по любому методу или инструменту, который документирован в другом месте (например, в соответствующих документах проекта, документах уровня организации или документах заказчика);

3) все показатели (метрики) улучшения процесса, которые должны собираться и фиксироваться в отчетах (например, число открытых рисков, число рисков по классам, тренд (изменения) величины риска от момента его идентификации до закрытия, число успешных устранений, число неуспешных устранений и др.);

4) процесс, обеспечивающий оценивание и улучшение основного процесса управления риском (например, ежеквартальное оценивание эффективности применяемых методов, периодическая ревизия отчетов заказчика с целью определения их полезности и др.).

Ресурсы и графики

Этот раздел документирует ресурсы (например, стоимость, трудоемкость, оборудование и расходные материалы, используемое ПО), необходимые для поддержки процесса управления риском. Специфицируется утвержденный бюджет, а также источники финансирования устранения риска. В этот раздел плана включается проекция действий по управлению риском на план-график и этапы разработки проекта ПС, а также перечень всех выпускаемые рабочих продуктов, касающихся управления риском, включая итоговые отчеты по риску, планы устранения и др.

Документирование риска

В этом разделе должна быть специфицирована инструментальная поддержка управления риском, указаны структуры таблиц базы данных о рисках, методы, средства и процедуры ведения базы данных, перечень входных и отчетных форм и др.

Здесь также должны быть определены все процедуры и требования по составлению, обработке, контролю и ведению (хранению) относящихся к риску документов и форм.

План управления риском может подвергаться модификации при появлении новых инструментов, методов или других элементов управления риском, необходимых для управления риском конкретного проекта.

Текущий список рисков и планы по их устранению не рекомендуется включать в план управления риском проекта ПС, чтобы не пришлось непрерывно его корректировать.

10.4 Функции, методы и средства управления риском проекта

10.4.1. Идентификация риска

Для успешной реализации процесса управления риском проекта жизненно важны функции *идентификации, анализа, планирования, учета и контроля, а также консолидации* риска [5].

Идентификация риска представляет собой процесс, в ходе которого неопределенности, связанные с проектом, трансформируются в реальные риски, которые можно описать и измерить.

Основные задачи идентификации риска таковы.

Формулирование высказывания о риске. Включает определение и фиксацию условий, являющихся причиной возможных потерь для проекта, а также предполагаемых последствий, связанных с проявлением этих условий.

Формулирование контекста риска. Включает фиксацию дополнительной информации, касающейся обстоятельств, событий и взаимосвязей в проекте, которые дополняют высказывание о риске.

В таблице 10.3 представлены некоторые методы и средства, которые могут быть использованы группой проекта при выполнении идентификации риска.

Таблица 10.3. **Методы и средства идентификации риска**

Задача	**Рекомендуемые методы и средства**	**Описание**
Все задачи	Информационный лист риска (рисунок 10.8)	Форма, которая документирует информацию о риске (схожая с отчетом об аномалиях в ПО). Пополняется по мере сбора информации о риске.
Формулирование высказывания о риске	Метод мозгового штурма	Метод коллективного выявления рисков, при котором персонал проекта высказывает предположения о рисках, что дает возможность, основываясь на разных мнениях, сформулировать утверждения о конкретных рисках.
	Периодический отчет о рисках	Метод, требующий от персонала проекта обязательного и регулярного отчета о рисках.
	Адаптация вопросника к особенностям проекта	Средство (инструмент), позволяющий адаптировать вопросник TBQ, рекомендованный SEI, к характеристикам конкретного проекта.
	Регистрационная форма риска (рисунок 10.9)	Форма, используемая для первичного документирования новых рисков по мере их идентификации.
	Краткий вопросник	Укороченный TBQ, который может использоваться при произвольном или периодическом отчете о риске. Может также использоваться на совещаниях или при индивидуальном интервьюировании как вспомогательное средство для идентификации рисков.
	Вопросник риска TBQ	Перечень вопросов для интервьюирования, составленных в соответствии с таксономией риска проекта ПС.
	Интервьюирование по TBQ	Метод, при котором проводится структурированное коллективное интервьюирование с использованием TBQ.
	Произвольный отчет о риске	Метод, при котором персонал проекта в инициативном порядке заполняет регистрационные формы риска при появлении новых рисков.
Формулирование контекста риска	Все приведенные выше методы и средства	Пригодны все упомянутые методы и средства, поскольку контекст риска (более подробная информация, дополняющая утверждение (высказывание) о риске) всегда формулируется при идентификации риска.

Идентификатор	**Информационный лист риска**		Идентифицирован:__/__/__
Приоритет _____	Утверждение о риске		
Вероятность _____			
Последствия _____			
Время наступления	Источник	Класс	Передан
Контекст			
Стратегия устранения			
Альтернативный план и условия применения			
Состояние		Дата учета состояния __/__/__	
Утверждение	Дата закрытия ___/____/_____		Критерий закрытия

Рис. 10.8. **Информационный лист риска**

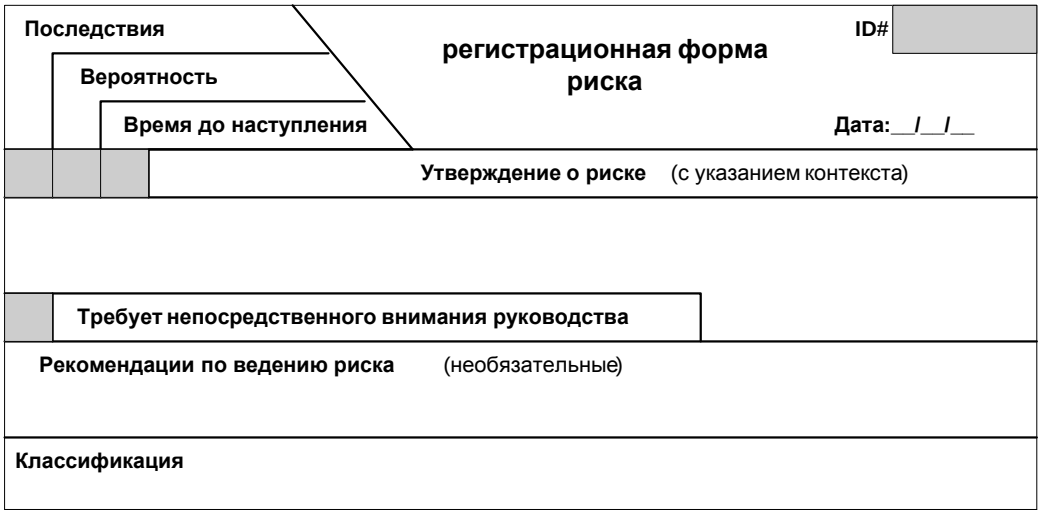


Рис.10.9. **Регистрационная форма риска**

10.4.2. Анализ риска

Анализ риска представляет собой процесс, в ходе которого устанавливаются детали рисков. Цель состоит в определении величины рисков, их взаимосвязи и степени важности.

Персонал, имеющий надлежащие знания, опыт и условия для эффективной обработки информации о риске, несет ответственность за оценивание, классификацию и приоритезацию рисков.

Задачи анализа риска таковы.

Оценивание. Оценивание атрибутов риска включает определение:

- *последствий* - потерь или отрицательных изменений в проекте в случае перерастания риска в проблему;
- *вероятности* - возможности перерастания риска в проблему;
- *резерва времени* - периода времени, который будет необходим для выполнения действий по устранению риска до его перерастания в проблему.

Классификация. Предполагает группирование рисков на основе общности их характеристик. Ее цель - облегчить идентификацию дублирующих рисков и упростить список рисков.

Приоритезация. Предполагает распределение отдельных рисков или наборов рисков по их критичности для выделения *наиболее важных* рисков, а также ранжирование этих наиболее важных рисков или наборов рисков по критерию или множеству критериев, установленных в проекте. Результатом приоритезации рисков является список рисков, часто называемых списком «первой Н-ки».

В таблице 10.4 представлены некоторые методы и средства, которые могут быть использованы группой проекта при выполнении анализа риска.

Таблица 10.4. **Методы и средства анализа риска**

Задачи	**Рекомендуемые методы и средства**	**Описание**
Все задачи	Информационный лист риска	Приведено в таблице 10.3.
Оценивание	Двухуровневое оценивание	Метод, при котором каждый риск оценивается по двухуровневой шкале с точки зрения: *серьезности последствий* (значительные, незначительные), *вероятности появления* (вероятно, невероятно), *времени наступления* (скоро, не скоро).
	Регистрационная форма риска	Может использоваться для отображения результатов двухуровневого или трехуровневого оценивания риска.
	Трехуровневое оценивание	Метод, при котором каждый риск оценивается по трехуровневой шкале с точки зрения: *серьезности последствий* (катастрофические, предельно допустимые, допустимые), *вероятности появления* (очень возможно, возможно, невозможно), *времени наступления* (незамедлительно, скоро, не скоро).
Классификация	Группирование по сходству	Метод, при котором естественным образом связанные риски группируются. Принцип группирования идентифицируется и фиксируется.

Задачи	Рекомендуемые методы и средства	Описание
	Гистограмма	Средство графического представления числа рисков в каждой классификационной категории.
	Регистрационная форма риска	Может использоваться для отображения результатов при применении методов группирования рисков или классификации рисков по таксономии.
	Классификация по таксономии	Метод, при котором риски группируются в соответствии с аспектами разработки ПС с использованием структуры «класс-элемент-атрибут», предлагаемой таксономией риска разработки ПС.
Приоритезация	Сравнительное ранжирование рисков	Метод, при котором риски ранжируются. Состоит в попарном сравнении рисков на основе одного или множества критериев. Сравнение продолжается до тех пор, пока не будут перебраны все риски.
	Мультиголосование	Метод голосования для подготовки списка наиболее важных рисков. Если список слишком длинный - производится серия голосований с целью его сокращения до разумных пределов. Каждый участник получает несколько голосов (чем больше голосов у участника, тем больше альтернатив упорядочения списка он может представить). Участники голосуют индивидуально и голоса подсчитываются одним из членов группы.
	Определение первой Н-ки по Парето	Метод, при котором наиболее важные риски проекта выбираются на основе результатов трехуровневого оценивания атрибутов.
	Первая пятерка	Метод, при котором эксперты выбирают первую пятерку рисков проекта в результате коллективного анализа, проводимого одним из методов, например методом потенциальной N-ки. Цель состоит в сборе индивидуальных мнений о наиболее важных рисках для проекта.
	Потенциальная Н-ка	Метод, при котором наиболее важные риски проекта выбираются на основе учета индивидуальных мнений, получаемых в результате применения метода первой пятерки. Все представленные участниками риски под номером 1 группируются; затем группируются риски с номером 2 и т.д. до тех пор, пока все риски из списков первых пятерок рисков, представленных участниками, не будут сгруппированы. Если риск появляется более чем в одной группе, он исключается из всех групп кроме той, в которой у него наибольший ранг. В результате получается неупорядоченный список наиболее важных рисков проекта.

10.4.3. Планирование устранения риска

Планирование устранения риска представляет собой процесс, в ходе которого принимаются решения о мерах, предпринимаемых по отношению к рискам. Результатами планирования являются планы действия по рискам для каждого риска или множества связанных рисков.

Ответственность за разработку соответствующих планов несет персонал, имеющий надлежащую компетенцию, знания, опыт и возможности (ресурсы) для эффективной обработки информации о рисках. В целом, целью планирования является ответ на следующие вопросы:

- мой ли это риск? (сфера ответственности),
- что я могу предпринять? (подход к устранению),
- как долго и что именно я должен делать? (масштабность действий).

Задачи планирования риска таковы.

Распределение ответственности. Распределение ответственности за каждый риск (группу рисков) требует от менеджера проекта или назначенных лиц понимания сути соответствующего риска и способности определить стратегию управления этим риском. Существует три альтернативы для определения ответственности за риск:

- включить риск в сферу собственного управления (поддержки);
- передать риск выше по инстанции внутри организации или в другую организацию;
- делегировать риск внутри организации (передать ответственность ниже по инстанции).

Определение подхода. Определение подхода к планированию устранения риска предполагает принятие решения о виде плана, приемлемого для данного риска, путем ответа на вопросы:

- *достаточно ли информации о риске*? Если ответ – «нет» - разработать план исследований для получения необходимой информации;
- *если риск перерастет в проблему, будет ли приемлемым масштаб последствий*? или, *может ли рассмотрение риска быть отложено на будущее время*? Если ответ – «да» - принять риск в сферу своей ответственности, документировать причины принятия решения о приемлемости риска;
- *если с риском нельзя смириться, нужно ли предпринимать немедленные действия*? Если ответ – «да» - взяться за устранение риска путем разработки и реализации плана по устранению;
- *существует ли (возможно ли) такое действие по устранению, которое можно или необходимо предпринять*? Если ответ – «нет» - риск нужно взять на учет и разработать требования по наблюдению за ним. При планировании должны быть определены метрики, сбор которых необходим для контроля состояния риска.

Определение масштаба риска и действий. Определение масштаба риска и действий по его устранению включает ответ на следующие вопросы при разработке плана устранения риска:

- насколько сложным будет устранение риска?
- как этот процесс должен быть документирован?
- какова стратегия устранения риска?

- каковы задачи?

Как правило, в зависимости от природы риска, выделяют два вида планов устранения риска, различающихся по сложности и объему требуемых ресурсов:

- план в виде перечня элементарных действий при менее сложной процедуре устранения риска (одно или несколько действий);
- план в виде алгоритма решения задачи с указанием графика и денежных средств на выполнение работ при сложной процедуре устранения риска (множество действий).

В таблице 10.5 представлены некоторые методы и средства, которые могут быть использованы группой проекта при выполнении планирования устранения риска.

Таблица 10.5. **Методы и средства планирования устранения риска**

Задача	**Рекомендуемые методы и средства**	**Описание**
Все Задачи	Блок-схема принятия решения при планировании устранения риска (рисунок 10.10)	Аппарат, который может использоваться для напоминания планировщику о возможных подходах, а также критериях выбора этих подходов.
	Информационный лист риска	Используется для документирования выбранной стратегии и действий по устранению риска, а также ответственности за разработку плана.
Определение подхода	Целевые метрики	Метод, при котором идентифицируются метрики для отслеживания хода устранения риска и изменений в состоянии риска. Разрабатывается перечень вопросов, которые используются при проведении сессии мозгового штурма с целью идентификации подходящих метрик.
Определение масштаба риска и действий	Список элементарных действий	Ведение списка, в котором перечисляются одно или более действий, необходимых для устранения риска. При использовании этого средства обеспечивается отслеживание и фиксация состояния дел по устранению риска.
	Рабочий листок планирования устранения риска (рисунок 10.11)	Форма, которая используется для идентификации, анализа и документирования альтернативных действий и решений по устранению риска. Она также служит для ведения истории альтернатив, которые рассматривались перед тем, как был выбран конкретный план устранения риска.
Определение масштаба риска и действий	Комплексное планирование	Метод, при котором разрабатываются планы действий по устранению сложного риска или множества взаимосвязанных рисков, зависимости между которыми высоки и устранение может быть дорогостоящим.

Задача	Рекомендуемые методы и средства	Описание
Определение масштаба риска и действий	Комплексное планирование	Для введения в действие таких планов, как правило, необходимо одобрение руководства. Зачастую для разработки детальных планов и графиков требуется проведение групповой экспертизы и в этом случае метод ориентируется на групповую работу. Комплексное планирование предполагает применение следующих приемов и средств: группирование по сходству, мозговой штурм, причинно-следственный анализ, анализ экономической эффективности, диаграммы Ганта, целевые метрики, сокращение списка, мультиголосование, диаграммы PERT (от Program Evaluation and Review Technique) (сетевые диаграммы) , структура распределения работы WBS (от work breakdown structure)
	Регистрационная форма риска	Используется для документирования рекомендованного действия по устранению риска

10.4.4. Учет и контроль состояния риска

Учет и контроль состояния риска представляет собой процесс, в ходе которого собираются, компилируются и фиксируются данные о риске лицами, несущими ответственность за наблюдение за «законсервированными» и устраняемыми рисками. Метрики, собираемые в процессе наблюдения, определяются при планировании, а их наблюдаемые значения предоставляются эксперту в отчетах о наблюдениях или в виде наглядных пособий (на презентациях). Информация затем используется для принятия решений о контроле законсервированных рисков и о планах устранения.

Задачи учета состояния риска таковы.

Регистрация данных о риске. Регистрация данных о риске включает все шаги, связанные со сбором информации о риске и обновлением значений метрик для наблюдаемых и устраняемых рисков. Целью сбора информации является контроль изменений в состоянии наблюдаемых рисков и планов по устранению рисков.

Компиляция. Компиляция (или преобразование) данных о риске предполагает анализ, комбинирование, вычисление (расчет) и структурирование данных о заданных рисках с целью облегчения управления процессом устранения риска и контроля эффективности плана устранения, а также учета изменений в наблюдаемых рисках. Способ, который используется персоналом проекта для компиляции данных, должен быть зафиксирован документально.

Фиксация. Фиксация предполагает доведение информации о состояниях рисков и планов устранения до экспертов и членов группы.

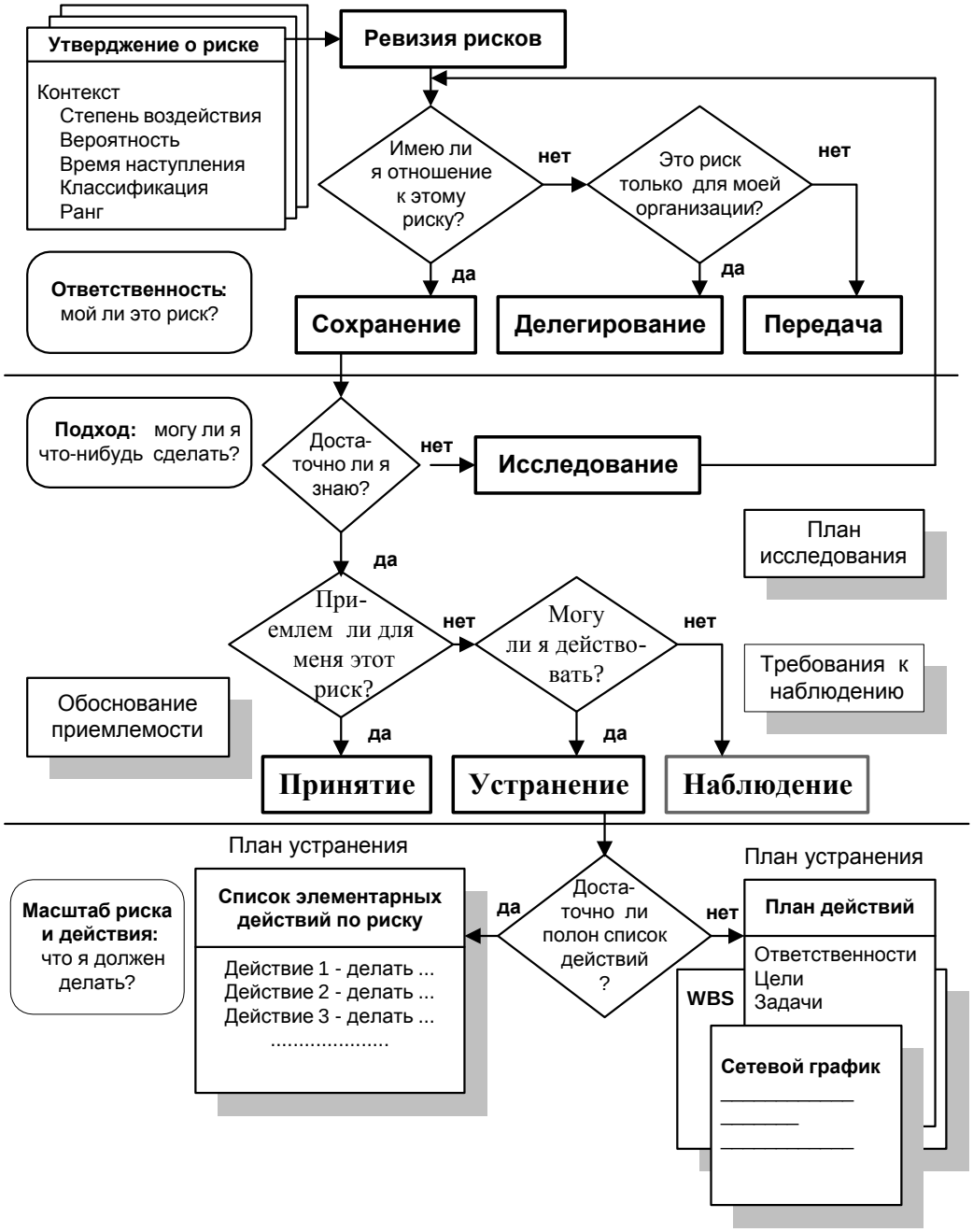


Рис.10.10. **Блок-схема принятия решения при планировании устранения риска**

Доведение информации о риске может быть выполнено путем представления письменного отчета (на бумажном или электронном носителе) или в виде устного представления. Полученные отчеты и проведенные презентации подытоживают данные, которые были проанализированы и структурированы, и используются экспертами в ходе регулирования риска.

Рабочий листок планирования устранения риска	
Идентификатор риска	**Ответственный**
Описание риска	
Цели и ограничения по устранению (в обозримые сроки)	
Дополнительные сведения (исходные причины, затрагиваемые элементы)	
Зависимые риски	
Альтернативные стратегии/действия	
Сопутствующие планы устранения	
Стратегический критерий оценивания	
Выбранная стратегия/действия	**Критерий успеха**
Альтернативная стратегия	**Условие применения**

Рис. 10.11. **Рабочий листок планирования устранения риска**

Трассирование рисков. Контроль состояния рисков предполагает и постоянное трассирование. Существует не так уж много средств, специально для этого предназначенных. Тем не менее, есть много *подходов* к трассированию рисков, использующих известные общие методы и средства учета рисков. Краткий обзор некоторых из них представлен в таблице 10.6.

Таблица 10.6. **Подходы к учету и трассированию рисков**

Задача	Подходы к учету риска	Описание
Сбор данных о риске	Повторное оценивание атрибутов риска	При использовании данного подхода лицо, несущее ответственность за риск, должно периодически переоценивать атрибуты риска для определения изменений в оценке серьезности последствий, вероятности появления и времени наступления последствий, с которыми связан риск. Этот подход обеспечивает получение информации о текущем состоянии наблюдаемых рисков и планов устранения риска. Для оценивания атрибутов риска используются следующие методы: двухуровневое оценивание атрибутов, трехуровневое оценивание атрибутов.
	Непосредственное взаимодействие	Этот подход предполагает неформальное взаимодействие с персоналом, тесно связанным с риском или занимающимся его устранением. Зачастую производится интервьюирование разработчиков проекта или других лиц, непосредственно отвечающих за риск или выполнение плановых действий по его устранению.
	Ревизия рабочих продуктов проекта	Этот подход предполагает изучение технических аспектов осуществления проекта. Ревизируемые документы могут быть полезны для отслеживания технических рисков, а также углубленного изучения общих вопросов проекта. Подход может также использоваться для поиска новой информации о риске.
	Ревизия отчетов о ходе выполнения проекта	Этот подход предполагает ревизию документации, которая составляется по результатам регулярных совещаний о ходе выполнения и состоянии проекта. Эти ревизии могут способствовать получению информации о текущем состоянии наблюдаемых рисков и планов их устранения.
	Автоматический сбор данных	Этот подход предусматривает использование имеющихся в продаже инструментов для трассирования и учета метрик проекта ПС и показателей качества, что обеспечивает получение достоверных количественных данных о риске. Собираемые данные могут использоваться для наблюдения за рисками и прогрессом в их устранении.
Компиляция	Подведение итогов хода устранения риска	Этот подход предполагает подытоживание положительных (отрицательных) сдвигов в устранении риска с целью принятия решений об эффективности плана устранения риска. Для получения информации о состоянии плана устранения риска используется метод: составление отчета о ходе устранения риска
	Подведение итогов о состоянии риска	Этот подход использует итоговые таблицы - краткое табличное представление ключевых элементов данных. Для создания и использования табличных форматов предназначаются следующие методы и средства:

Задача	Подходы к учету риска	Описание
Компиляция	Подведение итогов о состоянии риска	информационный лист риска, электронная таблица трассирования риска, цветовая диаграмма. Анализ данных о текущем состоянии может показать необходимость изменения приоритета риска или привлечения дополнительных сил на его устранение. В результате анализа могут быть также обнаружены новые риски для проекта.
	Подведение итогов наблюдения риска	Этот подход использует графические представления компилированных данных о риске. Для представления данных о риске в виде графиков или диаграмм используются следующие средства: столбиковая диаграмма, временная диаграмма, временной график.
Фиксация	Устный отчет	Этот подход использует неформальные способы взаимодействия для сообщения данных о риске. Лица, отвечающие за риски, отдают устные рапорты об общем состоянии своих рисков. Такого рода рапорты могут также использоваться для информирования руководства о критических проблемах в случае их возникновения.
	Письменный отчет	Этот подход может использовать формальные или неформальные меморандумы или документы (например, электронную почту, отчеты и др.). Виды отчетов должны согласовываться с действующими в организации механизмами отчетности. Для поддержки этого процесса могут использоваться следующие методы: отчет о ходе устранения риска, информационный лист риска, электронная таблица трассирования риска, цветовая диаграмма.
	Формальные презентации	Этот подход требует выбора приемлемых средств и форм презентаций. Формальные презентации зачастую подкрепляются письменными отчетами и содержат дополнительный материал, который мог не попасть в отчеты.

В таблице 10.7 представлены некоторые рекомендуемые методы и средства, используемые в рамках вышеописанных подходов к учету и трассированию риска.

Таблица 10.7. **Методы и средства учета риска**

Задача	Методы и средства	Описание
Сбор данных о риске	Двухуровневое оценивание	Метод описан в таблице 10.4.
	Трехуровневое оценивание	Метод описан в таблице 10.4.

Задача	Методы и средства	Описание
Компиляция и фиксация	Отчет о ходе устранения риска	Этот метод требует компилирования данных с использованием текстовой информации и графиков (например, временных диаграмм, столбиковых диаграмм и др.) для документирования детализированной информации о планах устранения рисков. Отчеты о ходе устранения риска используются для поддержки принятия решений. Формат отчета и информация, включаемая в отчет, должны соответствовать требованиям организации к отчетности.
	Информационный лист риска	Описан в таблице 10.3.
	Электронная таблица трассировки риска (рисунок 10.12)	Используются для подытоживания текущих состояний всех рисков и мониторинга рисками проекта. Обновление и ревизия рисков производится периодически (например, еженедельно или ежемесячно). Отчеты о трассировке риска обычно включаются в состав справочных материалов для проведения совещаний по проекту.
	Цветовая диаграмма (рисунок 10.13)	Инструмент, используемый для подытоживания состояний наиболее важных рисков и усилий по их устранению. С каждым планом устранения риска связывается один из трех цветов: *зеленый* - указывает, что план работает должным образом, и действия по его корректировке не требуются, *желтый* - указывает, что план не работает должным образом, тем не менее, действия по его корректировке не нужны, *красный* - указывает, что план не работает и нужны управляющие воздействия.
	Столбиковая диаграмма	Отражают изменения в количественном составе рисков по отдельным категориям и могут использоваться для идентификации существующих тенденций.
	Диаграмма временной зависимости	Показывает зависимость одного показателя (метрики) риска от другого во времени. Диаграммы временной зависимости полезны для идентификации тенденций в соотношении двух показателей с течением времени.
	Временной график	Иллюстрирует изменения показателя риска во времени. Временные графики полезны для идентификации тенденций в изменении показателя с течением времени.

Электронная таблица трассировки риска						**Дата**
Идентификатор риска	**Приоритет**	**Утверждение о риске**	**Описание состояния**	**Вероятность**	**Последствия**	**Ответственный**

Рис. 10.12. **Электронная таблица трассировки риска**

Цветовая диаграмма						
Состояние (красный/ желтый/ зеленый)	**Идентификатор риска**	**Утверждение о риске**	**Ответственный**	**План действий**	**Основные вехи** контролируемые события	**Комментарии**

Рис. 10.13. **Цветовая диаграмма**

10.4.5. Регулирование состояния риска

Регулирование состояния риска представляет собой процесс, в ходе которого лицо, принимающее решение, анализирует данные, содержащиеся в отчетах, принимает решения о дальнейших мерах, предпринимаемых в отношении риска, и осуществляет эти решения на практике.

Задачи регулирования состояния риска таковы.

Анализ учетных данных о риске. Предполагает изучение данных проекта о тенденциях, отклонениях и аномалиях. Цель состоит в достижении ясного понимания текущего состояния каждого риска и плана по его устранению.

Принятие решения. Основывается на использовании учетных данных о риске для определения мер, которые должны быть предприняты в отношении рисков проекта. Известны четыре возможных решения по риску:

- перепланировать;
- закрыть;
- активизировать план действий на случай исключительных ситуаций;
- продолжить наблюдение и выполнение текущего плана.

Перепланирование необходимо в тех случаях, когда результаты анализа данных показывают, что план устранения риска не работает, а специальный альтернативный план отсутствует.

Закрытие риска означает, что риск перестает существовать либо его дальнейшее отслеживание экономически не оправдано. Такая ситуация может произойти в том случае, если:

- вероятность появления или воздействие риска опускается ниже установленного порога;
- риск трансформируется в проблему и эта проблема решается способами, не входящими в сферу управления риском.

Специальные планы - это заранее сформированные альтернативные планы действий на случай неработоспособности основного плана устранения риска.

Если анализ учетных данных показывает, что план устранения риска выполняется успешно, может быть принято решение продолжить использование того же плана.

Выполнение решения. Осуществление принятых решений может потребовать внесения изменений в планы или инициации действия специальных планов. Внесение изменений может, в свою очередь, привести к *возврату к планированию*,

а активизация предусмотренных исключительными ситуациями действий и продолжение отслеживания рисков - к *возврату к учету* состояния рисков.

Методы, применяемые для регулирования риска, используют базовые общеизвестные методологии анализа, принятия, документирования и выполнения решений (таблица 10.8).

Таблица 10.8. **Методы и средства регулирования риска**

Задача	Рекомендуемые методы и средства	Описание
Анализ	Причинно-следственный анализ	Применяется для анализа взаимозависимости между рисками и причинами их возникновения.
	Анализ экономической эффективности	Применяется для повторного оценивания стоимости и эффективности определенной стратегии устранения риска, если установлено, что использование этой стратегии не приносит ожидаемых результатов. Обеспечивает получение информации, необходимой ЛПР для принятия решения о продолжении предпринимаемых действий по риску или изменении стратегии.
	Отчет о ходе устранения риска	Предполагает компиляцию собранных данных о риске с использованием текстовых и графических средств документирования детализированной информации о выполнении планов устранения рисков. Отчеты о ходе устранения риска обеспечивают ЛПР необходимыми данными для определения адекватных действий по регулированию состояния риска (путем введения альтернативного плана, перепланирования и т.д.). Формат отчета и информация, включаемая в отчет, должны соответствовать требованиям организации к отчетности.
	PERT диаграммы	Диаграммы PERT (Program Evaluation and Review Technique) - сетевые графики, которые могут использоваться для анализа влияний изменений в состоянии риска и планов устранений.
	Электронная таблица трассировки риска	Описана в таблице 10.7.
	Цветовая диаграмма	Описана в таблице 10.7.
Решение	Закрытие риска	Это метод формального документирования информации о риске, который находится в стадии закрытия по одной из причин: ввиду успешного устранения, перерастания в проблему или стабилизации (и перехода в разряд приемлемых и наблюдаемых). Любые уроки, извлеченные из наблюдения или устранения риска или множества рисков, а также критерии закрытия риска или множества рисков, должны фиксироваться при закрытии.

Задача	Рекомендуемые методы и средства	Описание
	Сокращение списка	Этот метод используется при большом числе рисков, стратегий управления риском или других факторов. Он особенно полезен в приложении к методу мозгового штурма, в ходе которого участники сессии голосуют по каждому элементу списка для определения необходимости его сохранения в списке.
	Мультиголосование	Описан в таблице 10.4.
Выполнение	Закрытие риска	См. задачу «Решение».
	Отчет о ходе устранения риска	См. задачу «Анализ».
	Информационный лист риска	Описан в таблице 10.3.
	Электронная таблица трассировки риска	Описана в таблице 10.7.
	Цветовая диаграмма	Описана в таблице 10.7.

10.4.6. Методы и средства коммуникации

Организация взаимодействия членов проекта при управлении риском касается этических аспектов их деятельности. Менеджерами должна быть внедрена такая культура общения, при которой идентификация риска и действия по его отслеживанию стали бы неотъемлемой частью ежедневной работы, а получение любой полезной информации о риске оценивалось положительно и поощрялось. Хорошее взаимодействие всех участников управления риском обеспечивает своевременное устранение существующих и потенциальных проблем благодаря «прозрачному» обмену информацией внутри и между всеми уровнями проекта, приданию значимости и весомости каждому отдельному «голосу», созданию атмосферы доверия ко всей информации о риске.

Функция коммуникации служит «стимулятором» выполнения всех других функций парадигмы риска и гарантирует, что:

- риски и планы их устранения интерпретируются однозначно,
- информация о риске является доступной для всех членов проекта;
- любой информации о риске уделяется надлежащее внимание;
- существует эффективный диалог между менеджером и бригадой проекта.

10.5. Рекомендации по оценке риска проектов

Анализ методологий оценки риска, предлагаемых такими зарубежными организациями, как NASA Lewis Research Center, Defense Systems Management College, DSDC (Defense Logistics Agency Systems Design Center), BMPC (Best Manufacturing Practices Center), SEI и др., показал, что:

все они схожи в подходах к оцениванию риска,

расходятся в принципах ранжирования рисков и

используют таксономию риска, подобную предложенной SEI.

Для оценки риска отечественных проектов ПС предлагается использовать таксономию риска SEI [2], модифицированный вопросник, представленный в приложении 6, и подход к ранжированию рисков, применяемый фирмой BMPC и описанный ниже.

Вопросник содержит *несколько* вопросов к одному атрибуту таксономии, способных прояснить угрозы качеству, стоимости и срокам разработки продукта, которые связаны с указанным атрибутом.

Каждому вопросу приписан максимальный *вес*, отражающий важность данного вопроса для снижения общего риска по соответствующему атрибуту. Чем выше вес, тем существеннее вопрос с точки зрения обнаружения риска.

Вопросы для проведения интервьюирования сформулированы в такой форме, чтобы на каждый из них можно было дать однозначный ответ: *«Да»*, *«Нет»*, *«Частично»*, *«Не знаю»* или *«Не применим»*.

Ответ *«Да»* свидетельствует об *отсутствии* риска, суть которого сформулирована в вопросе.

Ответ *«Нет»* или *«Не знаю»* свидетельствует о *наличии* риска.

Ответ *«Частично»* также свидетельствует о наличии риска, но дает возможность эксперту, оценивающему риск, указать вес вопроса, отличный от предлагаемого максимального веса.

Ответ *«Не применим»* соответствует неправомерности задания вопроса для данного проекта и не учитывается при оценке риска.

Процедура идентификации и анализа рисков не отличается от представленной в предыдущих разделах этой главы.

Оценка риска выполняется снизу-вверх по каждому атрибуту таксономии с последующим вычислением комбинированного риска по элементам и классам.

Метод оценки риска BMPC включает следующие шаги.

Шаг 1. Вычисление *ранга риска* по одному атрибуту. Ранг риска атрибута вычисляется исходя из количества вопросов, на которые был дан ответ «Да» или «Частично». Определяется сумма весов этих ответов. Затем вычисляется общая сумма весов всех вопросов для атрибута (вес атрибута), за исключением вопросов с ответами «Не применим». После этого находится процентное соотношение этих сумм по следующей формуле

$$R = (\sum_{i=1}^{k} Vд_i \; / \; (\sum_{j=1}^{n} V_j - \sum_{j=1}^{t} Vн_j)) \cdot 100$$

где $Vд_i$ – вес вопроса с ответом «Да» или «Частично»,

$к$ – количество таких ответов,

V_j – вес каждого вопроса,

n – общее количество вопросов, ассоциируемых с атрибутом,

$Vн_j$ – вес вопроса с ответом «Не применим»,

t – количество таких ответов.

Шаг.2. Вычисление *уровня риска* атрибута. Зависимость ранга и уровня риска представлена в таблице 10.9.

Таблица 10.9. **Уровни риска**

	Уровень риска		
	Низкий	Средний	Высокий
Ранг	100%-70%	69%-30%	29%-0%

Шаг 3. Вычисление ранга и уровня риска каждого атрибута.

Шаг 4. Вычисление *комбинированного риска элемента* таксономии путем определения соотношения рисков атрибутов этого элемента и составление шкалы (или столбиковой диаграммы).

Например, элемент «Требования» состоит из 7 атрибутов. Если по пяти атрибутам был получен высокий риск, а по 2 – средний, то соотношение рисков на шкале будет 2/7 для среднего и 5/7 для высокого риска (рисунок 10.14).

Рис. 10.14. **Столбиковая диаграмма риска**

Шаг 5. Вычисление комбинированного риска по всем элементам.

Шаг 6. Аналогичным образом может быть установлен комбинированный риск проекта в целом путем определения соотношения рисков всех атрибутов таксономии.

Литература к главе 10

1. *Higuera R.,* Haimes Y. Software Risk Management // CMU/SEI-96-TR-012, Pittsburg, Pa.: Software Engineering Institute, Carnegie Mellon University. –1996. – 49 p.
2. *Carr M.* et al. Taxonomy-Based Risk Identification // CMU/SEI-93-TR-006, Pittsburg, Pa.: Software Engineering Institute, Carnegie Mellon University. – 1993. –90 p.
3. ISO/IEC 12207:1995 / Amd.2:2004 Information technology – Software life cycle processes
4. *Sisti F.,* Joseph S. Software Risk Evaluation Method Version 1.0 // CMU/SEI-94-TR-19, Pittsburg, Pa.: Software Engineering Institute, Carnegie Mellon University. – 1994. – 175 p.
5. *Gallagher B* et al. Software Acquisition Risk Management Key Process Area (KPA)- A Guidebook Version 1.0 // CMU/SEI-97-HB-002, Pittsburg, Pa.: Software Engineering Institute, Carnegie Mellon University. – 1997. – 102 p.

Глава 11. МЕТОДОЛОГИИ ПОВЫШЕНИЯ КАЧЕСТВА В СОВРЕМЕННОЙ ПАРАДИГМЕ

11.1. Цикл управления качеством

11.1.1. Подход к управлению качеством в старой парадигме. Система Ф.Тейлора

Известно, что цикл управления качеством промышленной продукции включает:

- *планирование* качества;
- *выполнение работы* по плану;
- *проверку* качества и
- *регулирование* (корректирующие действия).

Несмотря на то, что этот цикл, называемый циклом PDCA (Plan-Do-Check-Action, План-Работа-Проверка-Корректировка), присутствует и в старой (продукто-ориентированной), и в новой (процессо-ориентированной) парадигме качества, механизмы его осуществления различны.

В *старой парадигме качества*, представленной ***системой Ф. Тейлора***, доминирующей в управлении производством в первой половине XX века, цикл управления выглядел, упрощенно, следующим образом. Инженер (конструктор) устанавливал требования к качеству в виде допусков на соответствующие параметры изделий. Рабочий изготавливал образцы изделий (иногда с дефектами). Технический контролер производил выборочный контроль образцов из партии изделий и фиксировал нарушение допусков («избыток» дефектных образцов). Администратор «принимал меры» (применял санкции) в виде наказаний или поощрений, утилизации брака, изменения допусков на качество и др. В результате, суть отношений составлял конфликт сторон, представляющих *разные* элементы цикла производства и имеющих *разные* интересы. Критерии качества определял, фактически, конструктор. Ответственность за качество нес контролер. Администратор заботился о количественных показателях выпуска, а рабочий – о выполнении личного плана. При такой системе управления, многие причины дефектов (плохая конструкция изделия, плохой технологический процесс, плохое снабжение, плохая обученность) оставались не выявленными.

В.А. Лапидус в [1] так характеризует этот порок системы Тейлора: «Система Тейлора предполагает, что если ударить по последнему элементу – исполнителю, то он либо сам все исправит, либо вытянет всю цепь причин. Но это и есть самое большое заблуждение. Да, если результат – это ошибка или халатность…самого исполнителя, то, получив удар, он, как собака, которую дрессируют таким способом, постарается не повторить ошибку. Но если причина не в нем, то ему наносится травма, которая лишь усугубляет дело…Когда он (исполнитель) начинает показывать на реальные, как ему кажется, причины, например, на не отремонтированный станок, плохой инструмент, плохое освещение, менеджеры приходят в ярость. Им кажется, что исполнитель просто пытается уйти от ответственности и переложить ее на других, в том числе и на них самих. А это надо пресекать в корне. Система Тейлора неэффективна, потому что наказывать исполнителей не просто бесполезно,

но и вредно, т.к. они лишаются интереса к работе…, их труд превращается в наказание».

Несмотря на недостатки, система Ф. Тейлора «была великолепна для своего времени… Такие ее элементы, как допуски, наверное, переживут еще многие поколения новых менеджеров».

К сожалению, в сфере отечественного производства программной продукции (как и, от части, в промышленности), система Тейлора продолжает существовать, почему и приходится вспоминать о ней в данной работе.

11.1.2. Подход к управлению качеством в новой парадигме. Статистическое управление процессами

В основе *новой парадигмы управления качеством* лежит процессный подход и статистическое мышление. Этот подход был предложен в 1924 году В. Шухартом и далее развивался Э. Демингом, Дж. Джураном, К. Исикавой, Ф. Кросби и др.

Центральным объектом управления качеством в новой парадигме становится производственный процесс, а цель – попасть в допуск – заменяется новыми: 1) *обеспечить стабильность* (устойчивость) процесса; 2) непрерывно *уменьшать вариации* стабильного процесса, являющиеся источниками дефектов и несоответствий.

Почти все характеристики процессов и продуктов демонстрируют изменчивость при многократном измерении. Эта изменчивость имеет два источника:

- явления, свойственные выполняемому процессу, результаты которых общие для всех измерений заданного атрибута процесса (общие причины);
- явления, причины которых могут быть объяснены и устранены (особые причины).

Вариации параметров продуктов на выходе процесса представляют собой реализации устойчивого случайного процесса, функция распределения которого остается постоянной во времени (модель нормального распределения).

Вариации, связанные с общими причинами (нормальные вариации процесса) существуют из-за взаимодействия различных по природе компонентов процесса (людей, машин, материалов, методов). Они случайны, однако находятся в предсказуемых пределах (рисунок 11.1 а). Предсказуемость здесь является синонимом контролируемости (подконтрольности).

Особые причины, вызывающие вариации процесса, находятся за пределами процесса - неправильный вход процесса, неподготовленные исполнители, непригодные материалы, нарушение условий среды, отказывающие инструменты и др. Эти причины оказывают существенное влияние на характеристики продуктов и процессов и изменяют параметры кривой распределения (рисунок 11.1 б). Если все особые причины устранены, и предупреждается их повторное появление, процесс считается стабильным.

Стабильность процесса по отношению к любому заданному атрибуту определяется с помощью измерения атрибута и слежения за результатом во времени. Если одно или более измерений выходят за диапазон случайной вариации или наблюдается систематическое изменение кривой, процесс не стабилен.

Предложив перенести акцент с отдельных дефектов продукции на вариации процессов, В.Шухарт указал тем самым на два важных момента:

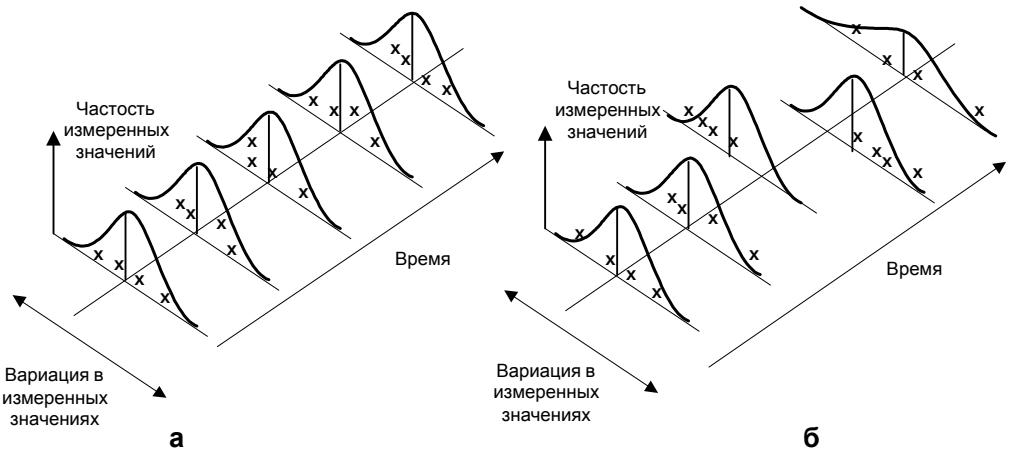


Рис. 11.1. **Контролируемые и неконтролируемые вариации процесса**

- нужно искать не виновных в дефектах, а определять причины дефектов, устранять их и предотвращать появление в будущем;
- источниками дефектов являются вариации процессов. Уменьшение вариаций – главная цель управления процессами. Для ее достижения необходимо сотрудничество (командная работа) участников цикла PDCA.

Один из методов, который часто используется для учреждения контрольных пределов допустимой вариации процесса, - ***статистическое управление процессом***, *SPC* (от Statistical Process Control). Этот метод также был предложен Шухартом в 20-годы и далее использовался его последователями как основа для повышения качества продукции (см., например, работы К.Исикавы [2], Х.Кумэ [3], а также электронный учебник по промышленной статистике компании StatSoft, Inc. [4], доступный в Интернет).

В настоящее время метод SPC применяется в инженерии качества ПС для контроля стабильности и улучшения отдельных процессов ЖЦ ПС, о чем свидетельствуют работы Э. В. Флорака [5], Веллера [6] и др.

Нужно отметить, что задачи *управления* качеством, связанные с применением SPC и других методов *управления процессами* и совершенствования процессов, решаются не только в интересах конкретного проекта ПС, но и всей организации. Поэтому, их решение находится в сфере ответственности, как группы качества, так и *группы инженерии процесса разработки*.

11.2. Всеобщее управление качеством

11.2.1. Философия TQM

Всеобщее управление качеством, TQM (от Total Quality Management) – это одновременно и философия, и набор руководящих принципов, которые составляют основу для *непрерывного совершенствования* организации.

Американское Сообщество контроля качества (ASQC, American Society for Quality Control) определяет TQM как такой подход к управлению, который *приводит* производителя продукции к продолжительному успеху благодаря ориентации

на удовлетворение потребностей потребителя. TQM базируется на участии *всех* членов организации в совершенствовании процессов, продуктов, услуг и собственной профессиональной культуры, и извлечении пользы для *каждого* ее члена.

В рамках философии TQM традиционные цели максимизации дохода или выгоды, минимизации затрат и достижения контролированного роста объемов продукции превращаются в цели обеспечения готовности к выпуску нужной для потребителя продукции, повышения его удовлетворенности изготовленной продукцией, улучшения ее качества и снижения сроков производства.

Философия TQM внесла фундаментальное изменение в определение и трактовку качества продукции. Она заменила лозунг *«все, что ни будет сделано – купят»* другим: *«будет делаться то, что купят»*. А это значит, что качество определяется и оценивается потребителем. Стратегия управления процессами обеспечивает достижение «всеобщего качества» во всех аспектах бизнеса, а не только применительно к изготавливаемой продукции. Непрерывное усовершенствование бизнеса, будь то улучшение проекта продукции, используемых материалов или применяемых технологий, помогает повысить «всеобщее качество» работы организации и качество конечной продукции.

Для того чтобы подчеркнуть отличия основных принципов традиционного подхода к качеству от принципов TQM, мы сопоставили их в одной таблице (таблица 11.1).

Таблица 11.1. **Отличие традиционных и современных подходов к качеству**

Принципы традиционного подхода	**Принципы TQM**
Продуктивность и качество - конфликтующие цели	Продуктивность и качество не есть конфликтующими целями
Качество определяется степенью соответствия продукта техническим спецификациям или стандартам	Качество определяется способностью продукта удовлетворять установленные или предполагаемые потребности потребителя в данном продукте
Качество измеряется степенью не соответствия продукта спецификации и стандартам	Качество измеряется степенью удовлетворения потребителя, а также способностью поставщика непрерывно совершенствовать процессы и продукты
Качество продукта достигается благодаря его интенсивной инспекции	Качество достигается благодаря эффективному проекту продукта и процессам его изготовления
Если продукт отвечает минимальным стандартам качества, отдельные дефекты допускаются и остаются не устраненными	Появление дефектов в продуктах предотвращается посредством применения технологий управления процессами
Обеспечение качества - это отдельный вид деятельности (функция), связанный с оцениванием продукта	Обеспечение качества – это составляющая всех видов деятельности (каждой функции) на всех стадиях жизненного цикла продукта
В плохом качестве продуктов виновны исполнители	За качество продукта ответственно руководство
Деловые отношения между членами коллектива производителей продукта непродолжительны и основаны на личной финансовой заинтересованности	Командная организация труда. Команду составляют специалисты разных функциональных профилей. Их деловые отношения основаны на разделении общих взглядов на обеспечение качества продукта

11.2.2. Метод управления Э. Деминга

Философия TQM связывается с именем Э. Деминга, а впервые аббревиатура TQM была использована NAVAIR (Naval Air Systems Command, Командование системами морской авиации) в 1985 году при описании японского стиля управления качеством продукции. Деминг предложил руководству NAVAIR четырнадцать пунктов обязательств по управлению производственным процессом.

Теперь они известны как положения ***метода управления Деминга***:

1) Обеспечьте постоянство намерений усовершенствования продукции и услуг.

2) Примите новую философию всеобщего качества.

3) Устраните зависимость качества от массовой инспекции продукта.

4) Покончите с практикой поощрения за выполненную работу, исходя исключительно из той *ценности*, которую она представляет в данный момент.

5) Совершенствуйте систему производства и обслуживания непрерывно и на все времена.

6) Внедрите систему производственного обучения.

7) Назначайте руководителей исходя из принципа реального лидерства в коллективе.

8) Искорените страх потери рабочего места.

9) Поломайте преграды между организационными подразделениями.

10) Не используйте лозунгов, увещеваний и понуканий рабочей силы.

11) Не устанавливайте количественных норм и стандартов работы.

12) Поощряйте гордость за собственное мастерство и качество выполненной работы.

13) Учредите мощную программу образования и индивидуальной переподготовки.

14) Подвигните каждого на то, чтобы преобразить свой труд.

Практические подходы к достижению TQM разнообразны. Например, в Hughes Aircraft Company для этого последовательно используются такие методологии, как QFD (см. главу 3), планирование промышленных экспериментов и SPC [7].

В основе выполнения процесса в рамках TQM лежит расширенный цикл PDCA:

определить *потребности* потребителя;

идентифицировать *важные производственные факторы* для реализации потребностей потребителя;

провести *цикл усовершенствования;*

зафиксировать успехи (предотвратить отступление назад, обеспечить повторяемость);

предоставить результаты потребителю и убедиться в его удовлетворенности.

Подходы PDCA и TQM направлены на повышение качества продукта посредством совершенствования производственного процесса - *единственного процесса* или производственной линии (т.е. варианта модели процессов, включающей один процесс).

11.3. Лестницы восхождения к качеству

Подходы, условно названные здесь «лестницами восхождения», принципиально отличаются от ранее представленных подходов PDCA и TQM, для которых предметом исследования были характеристики *конкретных продуктов* и показатели производственного процесса.

В данном случае предполагается, что существует множество *идеализированных процессов*, следование которым позволит достичь высокого качества ПС.

Оцениваются не характеристики продуктов, а степень *приверженности* установленным процессам. Постепенно внедряя определенное множество процессов (и подтверждая их внедрение), организация как бы «поднимается» по ступенькам к вершинам качества – бездефектной разработке ПС и непрерывному совершенствованию процессов.

Примерами таких «лестниц» могут служить модели зрелости:

- CMM (включая SW-CMM, SE-CMM, CMMI) (разработчик SEI, США) (глава 12),
- BOOTSTRAP (Bootstrap Institute, Великобритания) [8],
- TRILLIUM (Bell Canada, Канада) [9] и др.

Все они предлагают 5-уровневую шкалу зрелости и отличаются спектром процессов и принципами их группирования по уровням зрелости.

Нужно отметить, что «с возрастом» эти модели несколько видоизменились, адаптируясь к моделям процессов и способам оценивания, предлагаемым стандартом ISO/IEC 15504. Однако и сам стандарт, представляющий собой стандартизованный вариант модели усовершенствования и оценивания процессов в проекте SPICE (Software Process Improvement and Capability determination), продолжает развиваться рабочей группой ISO/IEC JTC1/SC7/WG10 по оцениванию процессов разработки ПО (адрес ее сайта в Интернете: http://www.sqi.gu.edu.au/sc7/wg10).

Наиболее известная и широко применяемая модель зрелости – CMM - уже была представлена в главе 1, а ее обобщение – CMMI рассматривается в главе 12.

11.4. Гибкие технологические линии

11.4.1. Технологическая подготовка разработки

Цель построения гибких технологических линий (ТЛ) – создание условий быстрой разработки каждой конкретной ПС.

Архитектура ТЛ определяется на этапе технологической подготовки разработки (ТПР) совместно группой инженерии процесса разработки, группой качества, руководством проекта и заказчиком ПС исходя из знания *потребностей пользователей*, *целевых характеристик* создаваемой ПС, *особенностей инфраструктуры* проекта и *накопленного коллективного опыта* разработки подобных систем в организации [10]. По существу, качество создаваемого продукта определяется качеством проведения ТПР (в частности, возможностями приобретения знаний и повторного использования опыта).

В задачи ТПР входит:

- определение состава используемых *процессов* ЖЦ (из числа регламентированных эталонной моделью процессов ЖЦ из ISO/IEC 12207) и *минимального* набора *действий* и *задач*[1] в этих процессах,
- определение модели ЖЦ и *методологии* разработки, адекватной потребностям проекта,
- построение *инструментально-технологических модулей*, поддерживающих выполнение процессов (или отдельных действий в процессах).

Инструментально-технологический модуль (ИТМ) - это совокупность операций, которые:

- выполняются по установленной *технологии*, реализующей соответствующий *метод* выполнения действий;
- объединены *технологическим маршрутом* их выполнения (*операционным* сценарием);
- имеют известные *предпосылки* выполнения, *входы* и *выходы*;
- обеспечены *информационной*, *программной* и *методической* поддержкой выполнения.

Общая структура ИТМ показана на рисунке 11.2.

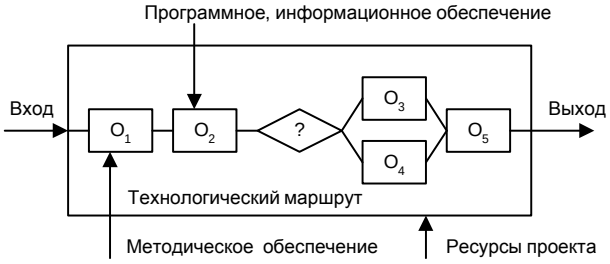


Рис. 11.2. **Структура ИТМ**

ИТМ конкретизируют общие *профили процессов* применительно к отдельным действиям (задачам), выполняемым в рамках конкретного проекта.

На практике ИТМ представляют собой программные комплексы и методики, разрабатываемые группой инженерии разработки и/или группой качества в инициативном порядке (вне договоров с заказчиками). Пример ИТМ для поддержки регистрации дефектов в процессе Верификации показан в таблице 11.2 (описание ТМ) и на рисунке 11.3 (форма для регистрации дефектов в программном комплексе «Менеджер_Якості»).

Таблица 11.2. **Состав ИТМ для поддержки процесса верификации**

Состав ИТМ	Описание составляющих ИТМ
Базовый метод	Сквозной контроль, коллегиальная проверка, метод Ортогональной классификации дефектов (ODC).
Информационная поддержка	Формы для регистрации дефектов, планирования проверок, сбора данных о выполнении проверок, взгляды (view) для формирования итоговых отчетов о дефектах. Методики по методам.

[1] В терминологии ДСТУ 3918-99 (и, соответственно, ISO/IEC 12207-95) внутренняя структура процесса определяется множеством *выполняемых действий* и *решаемых задач*, при выполнении каждого действия

Состав ИТМ	Описание составляющих ИТМ
Инструментальная поддержка	СУБД Oracle. Oracle Designer'2000. Книга Excel с собственной панелью инструментов (с макросами на VBA), API-функции для работы с репозиторием Oracle Designer.
Вход	Планы проверок в период выполнения текущего этапа проекта.
Выход	Данные о дефектах (в БД). Печатные отчеты о верификации.
Операции	1. Сбор данных о дефектах. 2. Регистрация дефектов по шаблонам форм и сохранение в БД. 3. Получение регламентных отчетов о дефектах. 4. Получение регламентных отчетов о выполнении проверок. 5. Оценка эффективности проверок по метрикам.

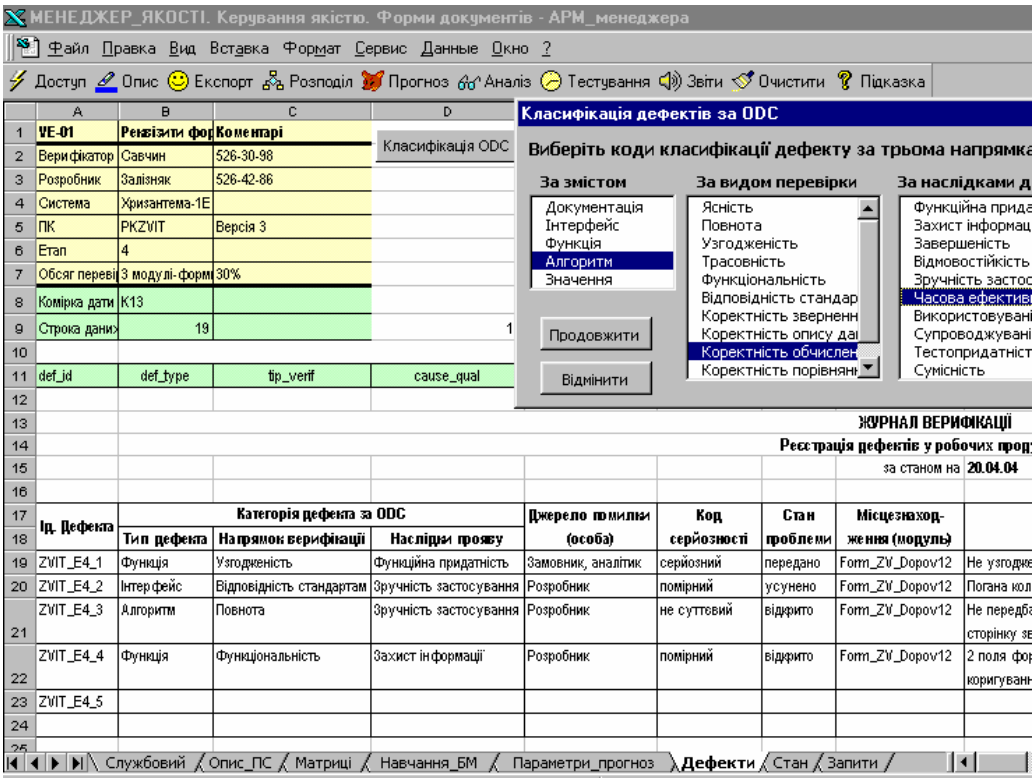


Рис. 11.3. **Фрагмент ИТМ для поддержки процесса Верификации**

Рассматриваемый подход лежит в основе таких методологий разработки, как, например, LSD [11] и QIP [12].

Метод LSD (Lean Software Development) (метод «суженной» разработки ПС) заключается в отсеивании «мало значимых» видов деятельности по разработке ПС и концентрации внимания на таких действиях, выполнение которых внесет, с точки зрения руководства проекта, существенный вклад в достижение целей качества продукта, установленных заказчиком.

Метод QIP (Quality Improvement Paradigm) подобен LSD тем, что также основывается на избирательном подходе к выбору множества выполняемых действий. Однако кроме цели разработки конкретного программного продукта с необ-

ходимыми характеристиками качества, ставит цель (и предлагает соответствующие механизмы) для непрерывного совершенствования организации.

11.4.2. Методы улучшения качества QIP и IDEAL. Фабрика опыта

Метод QIP развивается в SEL (Software Engineering Laboratory) центра GSFC (Goddard Space Flight Center) в NASA уже в течение 20 лет. Его применение позволяет контролировать эволюцию целей организации и оценивать статус текущих процессов по отношению к этим целям. В отличие от «лестниц восхождения», здесь используется внутреннее оценивание процессов относительно *собственных* целей организации, а не идеализированных процессов.

Для повышения качества программных продуктов и совершенствования процессов разработки метод использует парадигму GQM и методологию Experience Factory (Фабрика Опыта) [13]. Построение ТЛ разработки конкретной ПС основывается на *анализе опыта* выполнения подобных проектов, представленного в базе знаний о проектах организации в виде «пакета моделей», и построении цепочки процессов и соответствующей инфраструктуры, пригодных для получения продукта, удовлетворяющего целям проекта и организации. Иными словами, если

$$\textit{Процессы}\ (P_X, Q_Y, R_Z,\ldots)\ \text{---}>\ \textit{Продукты}\ (X, Y, Z,\ldots)$$

и нужно разработать продукт V, то, основываясь на понимании *взаимосвязи* между процессами $P_X, Q_Y, R_Z,\ldots$ и продуктами $X, Y, Z,\ldots$ и *целей* для продукта V, мы выбираем соответствующую смесь подпроцессов $p_i,\ q_j, r_k,\ldots$, формируя процесс, адаптированный для целей V:

$$\textit{Процесс}\ (P_V)\ \text{---}>\ \textit{Продукты}\ (V)$$

Метод QIP включает 6 шагов:

Шаг 1. Охарактеризовать текущий проект и его среду *в терминах применяемых моделей и метрик.*

Располагая знаниями о том, что собой представляет будущий проект (особенности проблемной области, факторы среды разработки, цели и ограничения проекта), нужно отнести его к классу проектов с подобными характеристиками для того, чтобы воспользоваться существующим опытом разработки и выбрать оптимальную стратегию его выполнения. При сопоставлении проектов учитываются следующие группы факторов:

- человеческий фактор – количество исполнителей, профессиональный уровень, организационная структура коллектива, наличие опыта в соответствующей проблемной области, опыт выполнения процессов и др.;
- особенности проблемной области – тип приложений, новизна проблематики, изменчивость, внешние ограничения и др.;
- факторы процесса разработки - модель жизненного цикла, методы, технологии, инструментальные средства, язык программирования и др.;
- факторы продукта – релизы (очереди поставки) системы, объем системы, необходимые характеристики качества (например, надежность, переносность) и др.;
- факторы ресурса – компьютеры в целевой среде и среде разработки, график разработки, финансовые средства, используемое существующее программное обеспечение и др.

Шаг 2. Установить цели в количественном измерении *для успешного исполнения и усовершенствования проекта.*

Используя известные механизмы определения измеримых целей, например, QFD или GQM, установить цели для процессов и продуктов. Цели могут быть определены для любых объектов с учетом используемых моделей качества и различных точек зрения (пользователя, заказчика, менеджера проекта, организации).

Шаг 3. Выбрать подходящую модель процессов *и инструментально-методологические средства поддержки проекта.*

Все процессы должны иметь определения и описываться в терминах целей, которые они должны удовлетворять. Это поможет понять, при каких условиях эффективны различные процессы, и выбрать исходную модель процесса, соответствующую определенному контексту применения, среде, особенностям и целям проекта, а также любым целям, поставленным для организации (например, проведение экспериментов с различными процессами, моделями или другими объектами).

После того, как выбрана определенная модель процесса, ее нужно адаптировать к условиям проекта и выбрать интегрированный набор подходящих методов, технологий и инструментов.

На практике, выбор процессов – итеративная процедура, с переопределением целей и, возможно, некоторых характеристик проекта и среды разработки. Итоговая модель выполнения проекта должна полностью соответствовать контексту, целям и процессам разработки ПС в установленной среде.

Шаг 4. Выполнить процессы*, создать продукты, собрать данные и проанализировать их для того, чтобы обеспечить своевременную обратную связь для корректировки проекта.*

Процесс разработки должен поддерживать доступ к «носителям» накопленного опыта по всем аспектам разработки. С другой стороны, он сам должен быть поддержан различными видами анализа, выполняемого перед тем, как установить обратную связь для корректировки проекта. Чтобы поддержать этот анализ, нужно собирать данные по проекту. Процессы должны быть изначально определены как измеримые, а сбор данных - естественно в них «встроен». Так, например, классификация дефектов должна быть частью механизма управления конфигурацией, а учет потраченного времени, количества и типов обнаруженных дефектов – частью сеанса инспекции проекта. Необходима автоматизированная поддержка рутинных действий по сбору и обработке большого количества данных и информации для анализа. Нужно отметить, однако, что большинство данных не может собираться автоматически, что требует ответственного отношения исполнителей процессов к этой процедуре.

Шаг 5. Проанализировать данные*, для того чтобы оценить текущую практику разработки ПС в организации, выявить проблемы, зафиксировать связанные с ними факты (находки) и подготовить рекомендации для последующих усовершенствований.*

Собранные данные интерпретируются в контексте поставленных целей – охарактеризовать (понять), оценить, предсказать, совершенствовать, и соответствующих им вопросов (см. главу 4).

Шаг 6. Сформировать «пакет»*, содержащий сконцентрированное представление накопленного опыта.*

Опыт должен быть описан в виде обновленных и уточненных моделей и других форм структурированных знаний, полученных по завершенному и прошлым проектам, и сохранен в базе знаний для многократного использования в будущих

проектах. Это могут быть математические модели, графические представления, определения процессов, модели качества, описания алгоритмов, процедур и извлеченных уроков и др.

Схематически метод представлен на рисунке 11.4. На нем указаны два цикла обратной связи - обратная связь на уровне проекта, устанавливаемая в ходе его выполнения (в контрольных точках проекта), и обратная связь на уровне организации, устанавливаемая после того, как проект завершен. Благодаря наличию этого второго витка обратной связи, ранее накопленные знания дополняются вновь приобретенными и изменяется представление о процессе разработки и целях организации.

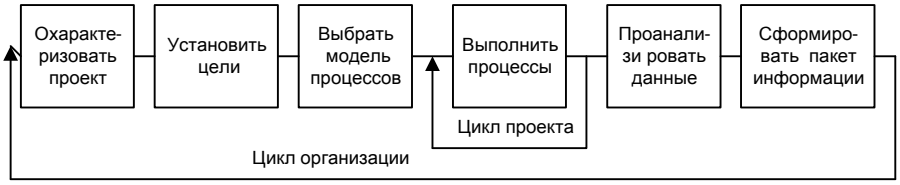


Рис. 11.4. **Схема метода QIP**

Два цикла в QIP отражают два взгляда на приобретение и использование знаний в программной инженерии:

• приобретение знаний на уровне организации. Включает шаги: *понять* состояние организации (охарактеризовать), *оценить* (установить цели, выбрать процессы, выполнить процессы, проанализировать данные) и *сформировать пакет* (упаковать вновь приобретенный опыт);

• приобретение знаний на уровне проекта. Включает шаги: выполнить *планирование* проекта (охарактеризовать, установить цели, выбрать процессы), *разработать продукт* (выполнить процессы) и *изучить опыт* (проанализировать данные).

В отличие от CMM (и других «лестниц восхождения»), использование которых обеспечивает поэтапное совершенствование организации, начиная со 2-го уровня зрелости к 5-му, применение метода QIP предполагает начать движение к совершенству с верхнего, 5-го уровня, несмотря на отсутствие в организации институциализированных процессов, соответствующих 5-му уровню зрелости. Побудительными мотивами к совершенствованию служат деловые цели и проблемы организации и проекта, а не определенная извне эталонная модель процессов.

Метод QIP не противоречит CMM и организация может использовать оценивание по CMM для того, чтобы определить уровень зрелости применяемых процессов. Однако, совершенствование процессов методом QIP, стимулируемое внутренними целями организации, дает шанс «пройти» по шкале зрелости CMM быстрее.

Для поддержки метода в SEL создана специальная организационная структура, ***Организация Фабрики Опыта*** (EFO, от Experience Factory Organization), поскольку очевидно, что улучшение процесса разработки и продукта требует непрерывной аккумуляции оцененных элементов приобретенного опыта в форме, пригодной для понимания, хранения и эффективной модификации.

Множество функций EFO разделено на две группы:

- функции *Организации проекта* по разработке программного продукта с привлечением пакетов повторно используемого опыта. Выполняются организационной структурой проекта;

- функции *Фабрики опыта*, связываемые с поддержкой процесса повторного использования всех видов знаний по программной инженерии и элементов опыта выполнения проектов. Выполняются отдельной оргструктурой.

Персонал проекта несет ответственность за действия по планированию и разработке продукта, а персонал Фабрики опыта - за изучение опыта и деятельность по передаче технологий.

Функции Организации проекта и Фабрики опыта принципиально отличны.

В Организации проекта решается *конкретная проблема*. Процессы, необходимые для ее решения, формируются таким образом, чтобы декомпозировать проблему на более мелкие составляющие, проиграть на примерах варианты решения, запланировать и реализовать различные действия по решению.

В Фабрике опыта решения *изучаются*, и опыт «упаковывается» для многократного использования. При этом выполняется *унификация* различных решений и *переопределение* проблемы, а также *обобщение* и *формализация* решений, что дает возможность делать их абстрактными, легко доступными и модифицируемыми. Кроме того, анализируется *процесс синтеза решения* и различные *экспериментальные действия*, что способствует обучению.

Модель совершенствования процессов IDEAL (от Initiating, Diagnosing, Establishing, Acting & Learning) разработана в институте SEI в 90-х годах. Описывает пятифазный жизненный цикл внедрения усовершенствований в существующие процессы организации. Фазы ЖЦ упорядочивают 14 видов деятельности, связанной с улучшением процессов, причем *необязательно* процессов разработки ПО, хотя изначально модель предназначалась для совершенствования именно софтверной деятельности на базе CMM. Особенность модели состоит в том, что она обеспечивает рамочные условия для планирования и реализации Программы совершенствования процессов, предполагающей *интеграцию* всех передовых *наработок института SEI* (технологий, методик, курсов и других услуг) в единую методологию повышения зрелости организации.

Фаза *Инициации* (Initiating) включает: определение деловых целей и их связи с выполняемой работой; распределение ресурсов и создание инфраструктуры для управления реализацией усовершенствований.

Фаза *Диагностики* (Diagnosing) включает определение текущего состояния организации и того состояния, которого желательно достичь. Это необходимо для разработки подхода к улучшению деловой практики и выработки рекомендаций.

Фаза *Учреждения* (Establishing) включает: расстановку приоритетов рекомендуемых действий, определение ограничений по их внедрению и разработку плана усовершенствования процессов.

Фаза *Внедрения* (Acting) включает разработку, проверку на пилотных проектах и утверждение *решений* по всем идентифицированным направлениям и деловым процессам (относительно инструментов, знаний, сторонней помощи и др.), а затем, повсеместное внедрение утвержденных решений.

Фаза *Изучения* (Learning) завершает цикл усовершенствований. Весь полученный опыт (при выполнении каждой фазы) пересматривается, упорядочивается и фиксируется.

Подробнее о модели IDEAL можно узнать на сайте института SEI - http://www.sei.cmu.edu/ideal/ideal.html.

11.5. Унифицированный процесс разработки RUP

11.5.1. Характеристика RUP

Унифицированный процесс разработки (RUP, Rational Unified Process) ведет свою историю от продуктов Rational Approach и Objectory Process 3.8, объединение которых произошло после слияния в 1995 г. корпораций Rational Software и Objectory Process. Теперь он поддерживается и совершенствуется Rational Software с учетом накопленного передового опыта в области программной инженерии.

RUP используется в различных прикладных областях, больших и малых проектах, и не только за рубежом, но и в Украине, что доказывает его универсальность и широкую применимость [14]. С одной стороны, RUP — это хорошо структурированное описание процесса разработки ПС, содержащее перечень работ и задач, выполняемых в ходе разработки, а также документов и моделей, создаваемых в ходе выполнения процесса. С другой стороны, RUP — это методология создания программного обеспечения, оформленная в виде размещаемой на Web-сайте (распространяемой как готовый продукт) базы знаний, снабженной поисковой системой. И, вместе с тем, и, прежде всего, RUP - это философия и практика разработки ПС [15].

Создатели RUP определили его как «итеративный, архитектурно-ориентированный, управляемый пользовательскими сценариями (прецедентами использования) процесс разработки ПО» [16].

Весь процесс разработки в RUP рассматривается в двух плоскостях. В статике процесс программной инженерии определяется через виды деятельности, процессы ЖЦ, рабочие продукты (артефакты) и роли исполнителей, а в динамике - через циклы, фазы, итерации и вехи.

Статическая составляющая включает [15]:

— описания работ и задач (в рамках работ),

— описания создаваемых рабочих продуктов (артефактов),

— рекомендации по выполнению описаний, сгруппированные применительно к таким *девяти основным* и *обеспечивающим* процессам (дисциплинам RUP):

- бизнес-моделирование (Business Modeling),
- управление требованиями (Requirements),
- анализ и проектирование (Analysis and Design),
- реализация (Implementation),
- тестирование (Test),
- внедрение (Deployment), а также
- управление конфигурациями и изменениями (Configuration and Change Management),
- управление проектом (Project Management),
- поддержка среды разработки (Environment).

Динамическую составляющую RUP представляют четыре фазы:

- начало (Inception),
- проработка (Elaboration),
- построение (Construction) и
- передача (Transition).

Действия в фазах проекта выполняются *итеративно*, причем в ходе каждой итерации сочетаются работы и задачи из различных дисциплин.

Для выполнения работ и задач в RUR определен стандартный *набор ролей*. Несколько участников проекта могут выполнять одну и ту же роль; в то же время, один участник может совмещать выполнение нескольких ролей.

Особенностью RUP является то, что в результате работы над проектом создаются и совершенствуются *модели*, всесторонне представляющие разрабатываемую ПС (а не множество бумажных документов).

Благодаря тому, что все участники проекта имеют доступ к одной и той же базе знаний и используют общий *язык моделирования продукта*, а именно Unified Modeling Language (UML), в RUP обеспечивается согласованное видение того, каким должен быть продукт и как его создавать.

Процесс разработки поддерживается множеством моделей, среди которых: Модель сценариев использования, Концептуальная модель, Аналитическая модель системы, Модель проектирования, Модель развертывания, Модель реализации (диаграмма компонентов), Модель тестирования (состоит из тестовых примеров, процедур тестирования, тестовых компонентов, не имеет отображения на UML диаграммах) и др. Их краткое описание и сопоставительный анализ назначения можно найти в Интернете на страницах сайта www.caseclub.ru [17].

Философию RUP отражают такие принципы, сформулированные П. Кроллом, как «Дух RUP», и отличающие эту методологию от других методологий итеративной разработки [18] (цитируется по [15]):

- атаковать риски как можно раньше, пока они сами не перешли в атаку;
- разрабатывать именно то, что нужно заказчику;
- главное внимание - исполняемой программе;
- приспосабливаться к изменениям с самого начала проекта;
- создавать архитектурный каркас как можно раньше;
- разрабатывать систему из компонентов;
- работать как одна команда;
- сделать качество стилем жизни.

Успех проекта и высокое качество конечного программного продукта в RUP достигается путем применения лучших практических приемов, касающихся разработки концепции ПС, планирования и управления проектом, снижения рисков и отслеживания их последствий, тщательной проверки экономического обоснования работ, управления требованиями, использования компонентной архитектуры, визуального моделирования, прототипирования, инкрементного создания и тестирования продукта, регулярного анализа результатов, управления изменениями и непрерывного контроля качества [15].

Бытует мнение, что RUP достаточно громоздкий процесс, однако он подходит как для больших, так и для маленьких коллективов разработчиков [19]. Это *конфигурируемый процесс программной инженерии*, в основе которого лежит простая и понятная архитектура процесса, обеспечивающая общность для целого семейства процессов. Поддержку конфигурирования RUP под нужды конкретных организаций обеспечивает компонент продукта *Development Kit*.

Подробную информацию о RUP и практике его использования можно найти, например, в книгах [16, 20-24].

11.5.2. Варианты адаптации RUP

Авторам известны по меньшей мере 3 варианта процессов, созданных на базе RUP: *EUP, IBM RUP, AUP*. Все они используют итеративную модель ЖЦ, принятую в RUP, с разбиением на 4 фазы (начало, проработка, построение, передача), и набор инструментов поддержки визуального моделирования, разработки и тестирования Rational Inc.

EUP (от Enterprise Unified Process**)** — унифицированный процесс предприятия — расширение RUP. К девяти основным дисциплинам RUP в EUP добавлена дисциплина «эксплуатация и поддержка (сопровождение)», а также 7 дополнительных дисциплин непосредственно для предприятия:

1. Бизнес моделирование предприятия (Enterprise Business Modeling).
2. Управление портфелем проектов (Portfolio Management).
3. Архитектура предприятия (Enterprise Architecture).
4. Стратегическое повторное использование (Strategic Reuse).
5. Управление персоналом (People Management).
6. Администрирование предприятия (Enterprise Administration).
7. Усовершенствование процесса ПО (Software Process Improvement).

Существует и второе расширение EUP с добавлением двух фаз в модель ЖЦ для охвата полного ЖЦ ПС: *производство* (Production), *изъятие* (Retirement).

IBM RUP – представляет адаптацию методологии RUP фирмы IBM. Как и в базовом процессе RUP, в IBM RUP используется итеративная модель ЖЦ с 4 фазами, а к дисциплинам RUP добавлена дисциплина «применение компонентных архитектур» (Use Component-based architectures). Другая особенность IBM RUP — постоянная проверка качества путем многократного тестирования версий.

AUP – (от Agile Unified Process) - упрощенный вариант RUP, представляющий сочетание основных концепций RUP с принципами, провозглашенными в манифеста Agile (подробнее Agile-методологии рассмотрены дальше в этой главе). Для улучшения результативности в процессе используются такие подходы как *проектирование, управляемое тестами* (или *отталкиваясь от тестов*) (test driven design (TDD)), *ускоренное моделирование* (Agile modeling), управление изменениями и переработка (рефакторинг) баз данных. В отличие от RUP, в AUP вместо 9 используется 7 дисциплин (исключены дисциплины «управление требованиями» и «анализ и проектирование»).

11.6. Методология подготовки программных решений MSF

Методология подготовки решений Microsoft Solutions Framework (MSF) представляет собой согласованный набор концепций, моделей и правил, позволяющих разрабатывать и внедрять информационные системы на основе технологий и инструментальных средств корпорации Microsoft [25].

В MSF термин “***решение***” (solution) обозначает скоординированную поставку набора элементов (включая программные средства, технические средства, документацию, обучение и сопровождение), необходимых для удовлетворения определенной деловой- (бизнес-) потребности предприятия. Хотя MSF и используется при разработке коммерческих продуктов для массового потребительского рынка, главным образом он концентрируется на поставке решений, предназначенных для *конкретного заказчика*.

Методология MSF описана в виде *пакета руководств*, включающего пять документов, так называемых «белых книг», каждый из которых охватывает определенную дисциплину или модель MSF[2].

11.6.1. Модель процессов MSF

Модель процессов MSF представлена в документе *Модель процессов MSF, версия 3.1* Модель процессов (Process Model) объединяет лучшие принципы каскадной и спиральной моделей. Она сохраняет преимущества упорядоченности каскадной модели, не теряя гибкости спиральной модели и восполняя ее недостаток - отсутствие *четких вех* выполнения работ по проекту.

В основе планирования и контроля разработки в MSF лежит управление компромиссами при определении *рамок проекта* - объемов работ, взаимосвязанных со сроками их выполнения и необходимыми ресурсами (затратами).

Наглядными средствами управления компромиссами в MSF являются *треугольник компромиссов* (рисунок 11.5 а) и *матрица компромиссов* (рисунок 11.5 б).

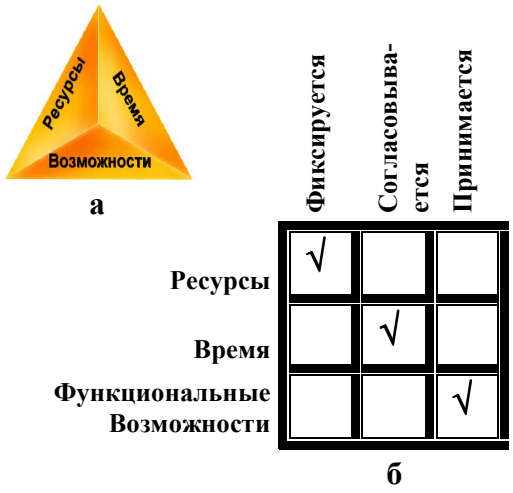


Рис. 11.5. **Средства управления компромиссами в MSF**

Матрица помогает обозначить *проектное ограничение* (одну из сторон треугольника компромиссов), воздействие на которое практически невозможно (колонка “Фиксируется”), фактор, являющийся в проекте *приоритетным* (колонка “Согласовывается”), и третий параметр, значение которого должно быть *принято* (таким, каким оно будет) в соответствии с установленными значениями первых двух величин (колонка “Принимается”). Возможны варианты:

- зафиксировав ресурсы, согласовать календарный график и принять результирующий объем функциональности решения,
- зафиксировав ресурсы, согласовать функциональность решения и принять результирующие сроки,
- зафиксировав объем функциональности решения, согласовать затрачиваемые ресурсы и принять результирующие сроки,

---

[2] Переводы этих документов на русский язык, выполненные eLine Software в 2002 году, можно найти в Интернет по адресу: www.microsoft.com/rus/msdn/msf/default.mspx

- зафиксировав объем функциональности решения, согласовать календарный график и принять результирующие затраты ресурсов,
- зафиксировав календарный график, согласовать затраты ресурсов и принять результирующую функциональность решения,
- зафиксировав сроки, согласовать объем функциональности решения и принять результирующие затраты ресурсов.

Тремя ***особенностями модели процессов*** MSF являются:

1) *Деление ЖЦ на фазы*, заканчивающиеся *главными вехами* (точками перехода от одной фазы к другой), в которых результаты работы подвергаются анализу, и которые знаменуют согласие проектной группы и заказчика продолжать работу над проектом (возможно, корректируя его рамки).

2) *Итеративный подход к разработке* (циклами) с *промежуточными вехами*, которые показывают достижение определенного прогресса в ходе проекта и расчленяют большие сегменты работы на меньшие, обозримые участки.

Методология MSF рекомендует определенный набор промежуточных вех, но на практике они могут варьироваться от проекта к проекту.

Разработка решения начинается с построения, тестирования и внедрения *базовой функциональности* (включая документацию и другие элементы решения), а затем, к этому решению добавляются все новые и новые возможности (используется стратегия версионности). Microsoft практикует *ежедневные сборки»* (билды) (daily builds) как неотъемлемую составляющую рабочего процесса. Разработка и тестирование ведутся непрерывно и одновременно, представляя собой параллельные процессы. Сборки служат проверкой совместимости всех элементов наработанного решения. Эти ежедневные сборные конструкции решения не попадают в реальную производственную среду. Выпуск решения выполняется только после того, как оно хорошо оттестировано и стабилизировано.

Для синхронизации сборок используется механизм управления конфигурациями. *Управление конфигурациями* часто путают с управлением изменениями в проекте (project change control). В действительности эти две задачи взаимосвязаны, но не идентичны. Управление конфигурациями – это протоколирование и контроль состояний элементов проекта. Управление же изменениями – это процесс рассмотрения и одобрения проектных изменений.

3) *Интегрированный подход* к созданию и внедрению решений. Одно из основных преимуществ такого подхода заключается в *улучшении взаимодействия* команд разработчиков и команд сопровождения. Зачастую команда разработчиков создает решение, не уделяя должного внимания вопросам его эксплуатации. Это приводит к низким показателям качества решения. Интегрированная модель процессов MSF обеспечивает процесс передачи ответственности от команды разработчиков к команде сопровождения сквозь ряд последовательных вех, а не как одномоментный перенос нагрузки.

В версии 3 концепция MSF охватывает весь цикл создания решений— от их обсуждения до внедрения. Используется ***5-фазовая схема цикла***: выработка концепции, планирование, разработка, стабилизация, развертывание (внедрение).

Результаты каждой из фаз становятся видимыми за пределами проектной команды (табл. 11.3, использует материал Андрея Колесова, опубликованный в журнале *BYTE/Россия.- №7.-2004*).

Таблица 11.3. **Пяти фазовая схема цикла разработки в MSF**

Фаза	Задачи	Вехи	Результат
Анализ и выработка концепции (Envision)	Определение состава команды Определение структуры проекта Определение бизнес-целей Оценка существующей ситуации Создание документа общей картины и области действия проекта Определение требований и профилей пользователей Разработка концепции решения Оценка риска Закрытие этапа	Промежуточные вехи: Костяк команды сформирован Черновой вариант концепции проекта составлен Главная веха: *Концепция решения утверждена*	• *Общее описание и рамки проекта* • *Документ оценки рисков* • *Описание структуры проекта* Определены роли и обязанности членов команды Описана иерархия отчетности и ответственности в команде, каналы взаимодействия с заказчиком Определена концепция решения, которой должна руководствоваться команда для достижения долгосрочных бизнес-целей проекта Определена область действия (рамки) проекта Проведено согласование с заказчиком и утверждение документов
Планирование (Planning)	Разработка проекта и архитектуры решения (концептуальное, логическое, физическое проектирование) Создание функциональной спецификации* Разработка детальных планов проекта Разработка календарного графика Создание среды разработки, тестирования и пилотной эксплуатации Закрытие этапа *) Утвержденные спецификации, планы и календарные графики образуют *базовую версию* проекта. Все последующие изменения *утверждаются формально*	Промежуточные вехи: Используемые технологии утверждены Базовая версия функциональной спецификации создана Базовая версия сводного плана проекта создана Базовая версия сводного календарного графика создана Среды разработки и тестирования развернуты Главная веха: *Планы проекта утверждены*	• *Функциональная спецификация* • *План управления рисками* • *Сводный план и календарный график проекта (совокупность отдельных планов)* Разработан набор сценариев использования системы пользователями различных категорий Разработана функциональная спецификация продукта Выполнена оценка стоимости работ и сроков получения результатов Разработаны взаимосвязанные планы реализации продукта, например, план внедрения, план тестирования, план эксплуатации, план мер безопасности, план обучения и др.

Фаза	Задачи	Вехи	Результат
Разработка (Developing)	Создание компонентов решения* (включая код и документацию) Разработка инфраструктуры Анализ и оценивание решения всеми заинтересованными лицами, выявление оставшихся проблем и неурегулированных вопросов, которые должны быть улажены до выпуска решения *) Выполняется параллельная разработка компонентов решения. Промежуточные сборки - это единая мера общего прогресса, наличие которой заставляет команду разработчиков синхронизировать различные составляющие на уровне решения в целом	Промежуточные вехи: Концепция подтверждена Сборка n завершена Сборка n+1 завершена … Главная веха: *Разработка завершена*	• *Исходный и исполнимый код приложений* • *Скрипты установки и конфигурирования* • *Окончательная функциональная спецификация* • *Материалы поддержки решения* • *Спецификации и сценарии тестов* Ключевые элементы решения проверены в моделируемой среде (копии реально существующей среды) Проведена верификация сформулированных требований путем демонстрации всех аспектов решения потребителям и группе сопровождения Синхронизированная параллельная разработка компонентов решения завершена
Стабилизация (Stabilizing)	Тестирование решения* Приоритезация и устранение дефектов Пилотное внедрение после стабилизации (в *часть* целевой среды) *) *Точка конвергенции* – момент, когда скорость устранения дефектов начинает превосходить скорость их обнаружения (тестирование близится к концу) *Точка достижения нуля* – момент, когда впервые все выявленные дефекты оказываются устраненными (близится стабилизация версии, которая будет зафиксирована как *версия-кандидат на выпуск*)	Промежуточные вехи: Точка конвергенции Точка достижения нуля Версии-кандидаты1…n Контрольное тестирование завершено Тестирование приемлемости для пользователей завершено Пилотное внедрение завершено Главная веха: *Готовность решения утверждена*	• *Окончательный продукт.* • *Документация выпуска (релиза)* • *Материалы поддержки решения* • *Результаты и инструментарий тестирования.* • *Исходный и исполнимый код приложений.* • *Проектная документация* • *Анализ пройденной фазы* Результаты тестирования оценены в соответствии с установленными критериями успешности Среда внедрения подготовлена Созданы нужные процедуры, скрипты и массивы данных Готовы учебные материалы Обеспечены условия для сопровождения решения Создан и протестирован план “отката”

Фаза	Задачи	Вехи	Результат
Внедрение (Deploying)	Внедрение технологии и ключевых компонентов решения (если они не были внедрены ранее) Перенос решения в среду эксплуатации (на все рабочие места) и его стабилизация Передача работы персоналу поддержки и сопровождения Получение со стороны заказчика окончательного одобрения результатов проекта Анализ выполненной работы и удовлетворенности заказчика тем, что его цели достигнуты (по завершению внедрения) Передача функций эксплуатации и сопровождения постоянному персоналу Сворачивание проекта *) *Ключевые компоненты* - компоненты инфраструктуры (контроллеры доменов, маршрутизаторы, почтовые серверы, удаленные серверы доступа, серверы баз данных и др.)	Промежуточные вехи: Ключевые элементы развернуты Внедрение на местах завершено Внедрение решения стабилизировано «*Период затишья*»[3] Главная веха: *Внедрение завершено*	• *Информационные системы эксплуатации и поддержки* • *Процедуры и процессы* • *Базы знаний, отчеты, журналы протоколов.* • *Версии проектных документов, массивы данных и программный код, разработанные во время проекта* • *Отчет о завершении проекта* • *Окончательные версии всех проектных документов* • *Показатели удовлетворенности заказчика и потребителей* • *Описание последующих шагов*

[3] Период затишья необходимый для оценки того, насколько хорошо решение работает в нормальных производственных условиях и насколько затратным будет его сопровождение (измерения количества инцидентов, времени простоя, определение эксплуатационных характеристик решения и др.)

Последовательность фаз в витке является логической в смысле зависимости последующих фаз от предыдущих. Это не означает, что фазы выполняются во времени строго одна за другой и следующая фаза может начаться только по окончании предыдущей. Фазы могут выполняться параллельно и быть частично или полностью совмещенными по времени. Фазы проекта носят итеративный характер. По достижении очередной вехи полученные материалы немедленно подвергаются проверке и оценке, результаты вызывают очередную итерацию уже проделанных работ, что не мешает начинать и продолжать работу в остальной части проекта.

Таким образом, модель процессов MSF учитывает постоянные изменения проектных требований. Она исходит из того, что разработка проектного решения должна состоять из коротких циклов, обеспечивая поступательное движение от простейших версий решения к его окончательному виду.

11.6.2. Модель проектной группы MSF

Модель проектной группы MSF представлена в документе *Модель проектной группы MSF, версия 3.1***.** Эта модель отражает подход Microsoft к организации труда исполнителей, обеспечивающий успешное выполнение проекта. В модели определяются роли исполнителей, функции, области ответственности и принципы управления, помогающие каждому из участников проекта выполнять задачи, которые ставятся перед ним по мере выполнения проекта.

Модель проектной группы в MSF – это результат осмысления недостатков пирамидальной, иерархической структуры традиционных проектных групп. Она основывается на утверждении о *равноправии ролей* в команде, где каждый *ролевой кластер* (группа специалистов с одинаковой ролью) представляет уникальную точку зрения на проект. Такая модель проектной группы MSF гарантирует присутствие всех командных ролей и их участие в процессах принятия решений от начала и до завершения проекта, что обеспечивает учетом полного спектра точек зрений при рассмотрении вопросов. Единый принцип для всех ролей в команде – нацеленность каждого на общий результат – создание качественного конечного продукта, единая начальная установка – отсутствие в нем дефектов, единое средство достижения цели – накопление опыта и обмен знаниями.

Каждый ролевой кластер имеет иерархическую структуру, что обеспечивает управление трудовыми ресурсами одной специализации. Руководители ролевых кластеров ответственны за организацию работы, координацию действий команды, в то время как ее члены имеют возможность сосредоточиться на своих индивидуальных задачах.

В MSF выделено ***шесть ролевых кластеров*** модели проектной группы:

Управление продуктом,

Управление программой,

Разработка,

Тестирование,

Удовлетворение потребителя,

Управление выпуском.

Цели деятельности, области компетенции и функции специалистов по ролям указаны в таблице 11.4, а в таблице 11.5 вкратце охарактеризованы ролевые функции специалистов в разрезе фаз проекта. В небольших проектах допускается совмещение нескольких ролей одним исполнителем (таблица 11.6).

Таблица 11.4. **Ролевые кластеры в MSF**

Роль	Цель	Компетенция	Функции
Управление продуктом	Удовлетворенные заказчики	Маркетинг Бизнес-отдача (бизнес-приоритеты) Представление интересов заказчика Планирование продукта	Выступает в роли представителя заказчика Формирует общее видение/рамки проекта Организует работу с требованиями заказчика Развивает сферы применения в бизнесе Формирует ожидания заказчика Определяет компромиссы (по треугольнику) Организует маркетинг и PR Разрабатывает, поддерживает и исполняет план коммуникаций
Управление программой	Достижение результата в рамках проектных ограничений	Управление проектом Выработка архитектуры решения Контроль производственного процесса Административные службы	Управляет процессом разработки Создает спецификацию и архитектуру Регулирует взаимоотношения и коммуникацию внутри проектной группы Следит за временным графиком проекта и готовит отчетность о его состоянии Проводит в жизнь компромиссные решения Разрабатывает, поддерживает и исполняет сводный план и календарный график проекта Организует управление рисками
Разработка	Создание продукта в соответствии со спецификацией	Технологическое консультирование Проектирование, разработка приложений и инфраструктуры	Определяет детали физического дизайна Оценивает необходимые время и ресурсы на реализацию каждого элемента дизайна Разрабатывает или контролирует разработку элементов Подготавливает продукт к внедрению Консультирует команду по технологическим вопросам
Тестирование	Дефекты выявлены и улажены	Планирование и разработка тестов. Отчетность	Обеспечивает обнаружение всех дефектов Разрабатывает стратегию и планы тестирования Осуществляет тестирование
Удовлетворение потребителя	Повышение эффективности, увеличение потребительской ценности продукта	Техническая поддержка Обучение Эргономика Графический дизайн Интернационализация	Представляет интересы потребителя в команде Организует работу с требованиями Проектирует и разрабатывает системы поддержки производительности Определяет компромиссы по качеству Определяет требования к системе помощи Разрабатывает учебные материалы и осуществляет обучение пользователей
Управление выпуском	Беспроблемное внедрение и сопровождение продукта	Инфраструктура Сопровождение Бизнес-процессы Управление выпуском готового продукта	Представляет интересы отделов поставки и обслуживания продукта Организует снабжение проектной группы Организует внедрение продукта Вырабатывает компромиссы в управляемости и удобстве сопровождения продукта Организует сопровождение и инфраструктуру поставки

Таблица 11.5. **Роли специалистов в разрезе фаз проекта**

Роли	Фаза выработки концепции	Фаза планирования	Фаза разработки	Фаза стабилизации	Фаза внедрения
Управление продуктом	Определение общих целей проекта; выявление нужд и требований заказчика; подготовка документа общего описания и рамок проекта	Мониторинг концептуального проекта; анализ деловых требований; разработка коммуникационного плана	Мониторинг ожиданий заказчика	Исполнение коммуникационного плана; планирование премьерного показа продукта	Получение отзывов и оценок заказчика; составление акта о приеме выполненной работы.
Управление программой	Мониторинг целей относительно дизайна, концепции решения, структуры проекта	Мониторинг концептуального и логического проекта; функциональной спецификации; сводного плана и сводного календарного графика проекта; бюджета	Управление функциональной спецификацией; мониторинг проекта; доработка планов	Мониторинг проекта; приоритезация дефектов	Сопоставление рамок проекта с поставленным решением; управление стабилизацией
Разработка	Прототипирование; анализ технологических возможностей; анализ осуществимости	Оценка технологий; логическое и физическое проектирование; планирование, составление графика и сметы	Разработка программного кода и инфраструктуры; документирование конфигураций.	Устранение дефектов; оптимизация программного кода.	Разрешение проблем; поддержка эскалации
Удовлетворение потребителя	Необходимые эксплуатационные характеристики решения и их влияние на его разработку.	Разработка сценариев использования, пользовательских требований, требований локализации; пользовательской документации/плана обучения/ графика тестирования удобства эксплуатации; обучение	Обучение; доработка плана обучения; тестирование удобства эксплуатации; графический дизайн.	Доработка эксплуатационных руководств; учебных материалов	Обучение; управление календарным графиком обучения

Роли	Фаза выработки концепции	Фаза планирования	Фаза разработки	Фаза стабилизации	Фаза внедрения
Тестирование	Стратегии тестирования; критерии приемлемости, их влияние на разработку решения	Оценка дизайна; разработка требований к тестированию; плана и календарного графика тестирования	Функциональное тестирование; выявление проблем; тестирование документации; доработка плана тестирования	Тестирование; сообщение об ошибках и их статусе; тестирование конфигурации	Тестирование производительности
Управление выпуском	Требования внедрения и их влияние на разработку решения; требования сопровождения	Оценка дизайна; определение эксплуатационных требований; плана и календарного графика пилотного и окончательного внедрения	Разработка контрольных вопросников для развертывания; доработка планов внедрения (включая пилотное)	Развертывание и поддержка пилотного внедрения; планирование внедрения; обучение персонала сопровождения	Управление внедрением; одобрение изменений

Таблица 11.6. **Варианты совмещения ролей специалистами**

	Управление продуктом	**Управление программной**	**Разработка**	**Тестирование**	**Удовлетворение потребителя**	**Управление выпуском**
Управление продуктом		-	-	+	+	~
Управление программной	-		-	~	~	+
Разработка	-	-		-	-	-
Тестирование	+	~	-		+	+
Удовлетворение потребителя	+	~	-	+		~
Управление выпуском	~	+	-	+	~	

Обозначения:

+ допустимо

~ не желательно

- нельзя

11.6.3. Дисциплина управления рисками

Управление рисками – это одна из основных дисциплин в MSF. В ее основе лежит подход, предложенный SEI (глава 10). Принципы управления рисками в MSF изложены в документе *Дисциплина управления рисками MSF, версия 1.1*. С самого начала учитывается, что проекты могут *изменяться* в процессе реализации, что вносит дополнительную неопределенность. MSF предлагает *упреждающий подход* к управлению рисками, их постоянную оценку и учет при принятии решений. Документ содержит рекомендации по выполнению *6-шагового процесса* управления рисками, который включает выявление и анализ рисков; планирование и реализацию стратегий по их профилактике и смягчению возможных последствий; отслеживание состояния рисков и извлечение уроков из приобретенного опыта.

11.6.4. Дисциплина управления проектами

Дисциплина управления проектами в MSF описана в документе *Дисциплина управления проектами MSF, версия 1.1*. В MSF предполагается распределение работы по управлению проектами между членами команды.

Особенность MSF - отсутствие должности менеджера проекта. Организатором работы команды является ролевой кластер «Управление программой». Типовые управленческие обязанности распределяются среди лидеров всех ролевых кластеров. Это повышает ответственность сотрудников и позволяет применить методологию MSF к широкому спектру различных по масштабам и сложности проектов. При таком подходе профессиональные менеджеры выступают в качестве консультантов и наставников команды, а не выполняют функции контроля над ней.

MSF выделяет несколько областей ответственности, навыков и деятельности по управлению проектами (таблица 11.7).

Таблица 11.7. **Области ответственности по управлению проектами в MSF**

Область управления проектом	Описание
Планирование проекта и контроль за изменениями	Интеграция и синхронизация планов проекта; организация процедур и систем управления и мониторинга проектных изменений
Управление рамками проекта	Определение и распределение объема работы (рамок проекта); управление компромиссными решениями в проекте
Управление календарным графиком проекта	Составление и ведение календарного графика исходя из оценок трудозатрат, упорядочивание задач и их соотнесение с доступными ресурсами
Управление стоимостью	Оценки стоимости исходя из оценок затрат времени; отчетность о ходе проекта; анализ хода проекта и затратных рисков; функционально-стоимостной анализ
Управление персоналом	Планирование ресурсов; формирование проектной команды; разрешение конфликтов; планирование и управление подготовкой
Управление коммуникацией	Коммуникационное планирование (между проектной группой, заказчиком, пользователями и другими заинтересованными лицами); отчетность о ходе проекта
Управление рисками	Организация процесса управления рисками в команде и содействие ему; обеспечение документооборота управления рисками

Область управления проектом	Описание
Управление снабжением	Анализ цен поставщиков услуг и/или аппаратного/программного обеспечения; подготовка тендерных документов, выбор поставщиков и субподрядчиков; составление контрактов, ведение переговоров и заключение договоров на поставку
Управление качеством	Планирование качества, определение применяемых стандартов, документирование критериев качества и процессов его измерения

В документе описаны основные принципы и избранные методики по каждой из перечисленных в таблице областей управление проектами MSF во взаимосвязи с моделью проектной группы MSF.

11.6.5. Дисциплина управления профессиональной подготовкой

Эта дисциплина посвящена управлению знаниями и профессиональными навыками, необходимыми для планирования, создания и сопровождения успешных решений. Дисциплина описана в документе *Дисциплина управления подготовкой MSF, версия 1.1*.

Степень подготовленности сотрудников для выполнения своей профессиональной роли в проекте определяется как отношение имеющегося у них уровня знаний, умений и способностей к желаемому их уровню (коэффициент KSA, от knowledge, skills and abilities). Эта мера показывает видимый (реально наблюдаемый) профессиональный потенциал специалистов в любой момент работы над созданием решения.

Подготовленность организации рассматривается как оценка общей подготовленности всех составляющих и используется при стратегическом планировании и анализе возможностей эффективного внедрения технологических инноваций. Однако, такие аспекты подготовленности организации, как совершенствование процессов и управление организационными изменениями в данной дисциплине MSF не рассматриваются.

Ключевые концепции управления подготовкой таковы:

- Учет имеющегося багажа знаний и его оценка. Управление знаниями.
- Стремление к самосовершенствованию.
- Перманентное управление подготовкой. Превращение работы над повышением квалификации в явную, планируемую деятельность.

Документ дает рекомендации по применению *превентивного подхода* к управлению знаниями на протяжении всего жизненного цикла решения, сравнивая его с реактивным подходом (таблица 11.8).

Таблица 11.8. **Сравнение превентивного и реактивного подходов к подготовке**

Превентивный подход	Реактивный подход
Позитивное отношение к планированию подготовки	Реакция на проявившуюся нехватку знаний, умений, способностей
Использование структурированного процесса подготовки	Использование ситуационных процессов или всякое их отсутствие
Прогнозирование и планирование нужд подготовки	Проведение обучения (исправление несоответствий) после проявления изъянов в подготовке
Разработка и использование системы управления знаниями	Неосведомленность о существующем потенциале знаний

На уровне организации учет текущего и желаемого состояния знаний, умений и способностей выполняется аналогично инвентаризации всех других ресурсов предприятия при планировании его архитектуры и составлении плана корпоративного развития. При этом, поскольку навыки специалистов, приобретенные в ходе реализации одного проекта, могут лечь в основу разработки последующих, важно правильно упорядочивать проекты и предусматривать своевременное достижение специалистами необходимого профессионального уровня.

На уровне проектных групп и отдельных сотрудников управление подготовкой рассматривается в рамках моделей процессов и проектной группы MSF.

Дисциплина управления подготовкой MSF включает в себя итеративный и непрерывный *процесс управления подготовкой*, помогающий достичь необходимого для создания и управления проектами и решениями уровня знаний, умений и способностей. Этот процесс состоит из четырех шагов: *определение*, *оценивание*, *корректировка* и *осмысление* знаний и опыта. В рассматриваемом документе эти шаги описаны детально.

Помимо описания методологии MSF в представленных выше документах корпорация Microsoft предоставляет множество ***шаблонов проектных документов*** для всех пяти фаз цикла создания решений. Они доступны в Интернет, например, по адресам:

- http://www.microsoft.com/downloads/details.aspx?FamilyId=9D2016AD-6F8A-47F5-84FA-BEC389DB18C1&displaylang=en или
- www.khsu.ru/trpp/Templates.zip (на английском языке).

11.6.6. Сравнение методологий RUP и MSF

Методологии RUP и MSF широко распространены в Украине и интерес к ним продолжает возрастать по мере повышения уровня зрелости организаций-разработчиков ПС. Они имеют много общих свойств и отличительных особенностей. Для того чтобы быстро сориентироваться в возможностях MSF и RUP, мы сопоставили их в таблице 11.9.

Таблица 11.9. **Сравнение методологий RUP и MSF**

Критерий сравнения	MSF	RUP
Сфера рынка	В основном, свободно распространяемые продукты, Web-сайты	Проекты на заказ. Не Web-сайты
Масштаб и сложность	Компактнее, проще в применении. Нет ряда руководств, которые, однако, могут добавляться самостоятельно. Более пригодна для малых краткосрочных проектов	Масштабнее. Множество руководств. Однако сложнее изучать, адаптировать. Более пригодна для крупных продолжительных проектов
Масштабирование	Да	Да. Сложнее
Инструментальная поддержка, включая плагины для .NET	В основном «ручная» методология[4].	Инструменты для всего ЖЦ.
Стоимость	Только курсов и руководств	Оплата за каждую лицензию

[4] С 2005 года методология поддерживается семейством Microsoft Visual Studio 2005

Критерий сравнения	MSF	RUP
Поддержка пользователей	Нет	Да
Итеративный подход	Да	Да
Количество фаз	5	4
Пофазовое сопоставление:		
1	Выработка концепции	Начало
2	Планирование	Проработка
3	Разработка	Построение
4	Стабилизация	Передача
5	Развертывание	Передача
Наличие руководств:		
- по моделированию деловых процессов	Нет	Да
- по требованиям	Кратко	Детально
- по анализу и проектированию	Кратко	Детально
- по реализации	Нет	Да
- по тестированию	Нет	Да
- по внедрению продуктов	Нет	Да
- по управлению изменениями	Нет	Да
- по управлению проектами	Кратко	Детально
- по среде разработки	Нет	Да
- по интервьюированию	Нет	Да
- по использованию UML	Нет	Да
Количество шаблонов (форм)	Не предопределено	42 предопределенных
Интеграция UML	Может поддерживаться	Встроен
Управление рисками	Детальное	Детальное
Ориентация на вехи проекта	Да	Да
Независимость от языка	Да	Да
Использование матрицы компромиссов для выбора стратегии	Да	Нет
Количество ролей в командах	6	5 основных. 31 вспомогательная
Роль менеджера проекта	Разделяется на две	Не разделяется

11.7. Процессы разработки в Agile-методологиях

11.7.1. Манифест Agile Alliance

В современных условиях рынка сфера применения классических «монолитных» методологий создания ПС (ориентированных на постоянные требования к продукту, фиксированные стоимость и сроки работ, а также продолжительный ЖЦ с развернутым «полнокровным» процессом программной инженерии) постепенно сужается, ограничиваясь преимущественно крупномасштабными государственным программами. Вкладывая значительные средства в программные проекты, заказчики не намерены длительно ожидать реальных результатов, поскольку даже постоянство требований не гарантирует их надлежащей реализации.

Поиск и оформление новых подходов к разработке ПС взяли на себя специалисты-практики и независимые эксперты в области программной инженерии, среди которых Кент Бек (Kent Beck), Мартин Фаулер (Martin Folwer), Кен Шваубер (Ken Schwaber), Джефф Сазерленд (Jeff Sutherland) и другие. В 2001 они образовали не-

коммерческую организацию **Agile Alliance**[5], пропагандирующую принципы ускоренной дробной «активной» разработки ПС (Agile Software Development), основанной на методологиях eXtreme Programming, SCRUM, Dynamic Systems Development Method (DSDM), Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming и др.[26].

Специалисты пришли к выводу, что лучшие подходы к созданию ПС нужно искать, непосредственно активно участвуя в процессе разработки и помогая другим. В принятом ими *Agile-Манифесте* говориться [27]:

«Мы придерживаемся следующих принципов:

- Наивысшим приоритетом является удовлетворенность заказчика ранними и периодическими поставками ценного для заказчика ПО.
- Приветствуйте изменения требований даже на поздних этапах разработки. Agile-процессы готовы к таким изменениям ради достижения заказчиком конкурентного преимущества.
- Выполняйте частые поставки работающего ПО. Продолжительность каждой итерации - от пары недель до пары месяцев, предпочтение отдается коротким интервалам.
- Потенциальные пользователи системы, являющиеся специалистами в предметной области, и разработчики должны работать вместе каждый день на протяжении всего проекта.
- Привлекайте для работы над проектом мотивированных людей. Создайте для них все условия, окажите поддержку во всем, что необходимо, и доверьтесь им – они выполнят работу.
- Самый действенный и эффективный способ обмена информацией как внутри команды разработчиков, так и разработчиков с внешним миром – непосредственное общение.
- Работающее ПО – главный индикатор продвижения проекта.
- Agile-процессы придерживаются равномерного темпа разработки. Работа спонсоров, разработчиков и пользователей должна все время идти в постоянном темпе.
- Постоянное стремление к техническому совершенству и хороший дизайн системы повышают agility.
- Важна простота – искусство увеличения объема работ, которых удалось избежать.
- Самые лучшие архитектуры, требования и дизайны систем создаются самоорганизующимися командами.
- Периодически команда размышляет о том, как достичь большей эффективности, после чего корректирует свой подход к разработке ПО.»

Таким образом, методологии Agile ориентированы на тесное взаимодействие команды разработчиков с пользователями, итеративную модель ЖЦ с приращениями и быструю реакцию на изменяющиеся требования, что предполагает высокий профессионализм всех участников проекта. В основе этих методологий лежит метод «суженной» разработки (LSD) уже упоминавшийся в п. 11.4.1. Хотя в целом

---

[5] Адрес специализированного сайта Agile- (Эджейл-) Альянса в Интернете: www.agilealliance.org

эти методологии придерживаются скорее «продукто-ориентированного», а не «процессного» подхода к разработке, в них используются элементы гибких ТЛ и «облегченных» процессов.

Перспективность применения данных методологий, особенно для отечественных условий разработки ПС, обусловлена тем, что во главу угла *каждой итерации* разработки ставится понятие ее «ценности для дела» (business value), как функции вложенных *затрат*, *объема функциональности* поставляемого фрагмента ПС, *времени* выполнения работы и *качества* этого фрагмента ПС. Такой подход к разработке ПС обеспечивает уменьшение зависимости не только от постоянных изменений функциональных требований заказчика, но и от проблем неритмичности общего финансирования и плана-графика разработки, а также, быстрой смены технологической среды разработки.

11.7.2. Экстремальное программирование

Самой широко известной в Украине Agile-методологией является **eXtreme Programming** (**XP**) (экстремальное программирование) [28 - 33].

Основные принципы разработки в XP и факторы ее ускорения таковы [34]:

- простота решений (реализация первого же принятого наипростейшего решения, даже если оно сопряжено с риском),
- интенсивная итерационная разработка,
- парное программирование [35],
- малочисленные команды проектов (не больше 10 человек), активное общение в команде и между командами,
- обратная связь с пользователем, который фактически вовлечен в процесс разработки,
- «кураж» и желание идти на риск.

Один из самых известных приемов XP, обеспечивающих качество ПО – ***парное (или совместное) программирование*** - процесс создания ПО двумя программистами, работающими бок о бок за одним компьютером (по схеме: один – пишет код, другой - проверяет).

Исследования положительных и отрицательных сторон парного программирования показали, что при таком подходе время разработки увеличивается на 15%, однако улучшается дизайн системы, сокращается количество дефектов, снижается риск, связанный с занятостью в проекте определенных сотрудников, растет профессионализм членов команды, улучшается взаимодействие и коммуникация. Кроме того, что не маловажно, сам процесс работы в целом доставляет гораздо больше удовольствия [36]. Кроме парного программирования, в XP пропагандируется принцип «коллективного владения кодом», то есть любой член группы может изменить не только свой код, но и код другого программиста.

Другая важная особенность методологии XP, обусловливающая качество – стратегия тестирования ПО. Тесты создаются до написания кода, причем каждый модуль снабжается автономным тестом (unit test), обеспечивая возможность регрессионного тестирования модуля. Фактически тест определяет код, а не наоборот (это вид *разработки от тестов (или, основанной на тестировании) - test-driven development*, TDD) [32]. Функциональные тесты определяет заказчик (или аналитик проекта с данной ролью). Тесты пишут сами программисты; любой программист имеет право написать тест для любого модуля. Фрагмент кода образует часть вер-

сии тогда и только тогда, когда все тесты прошли успешно, в противном случае данное изменение кода отвергается. Таким образом, большинство ошибок исправляются на стадии кодирования – либо путем просмотра кода (глава 6), либо путем динамического тестирования (глава 7).

Основные практические приемы XP подытожены в таблице 11.10.

Таблица 11.10. **Основные практические приемы XP**

Практический прием	Комментарий
Единая команда	Коллективная работа и постоянное взаимодействие. Каждый участник проекта считается членом команды с равными правами
Тесты заказчика	Постоянное участие заказчика. Если реального заказчика нет – на его роль назначается член группы (роль “Customer”)
Небольшие релизы версий	Частая интеграция и выпуск небольших, но работающих версий ПС
Циклы игрового планирования	Планирование *версий* и *итераций* (состав, сроки, разработчики). Сроки очень сжатые. Разбиение на осязаемые части (в пределах недели-двух).
Коллективное владение кодом	Любая пара программистов может в любой момент улучшить любой код (не только свой). При работе с незнакомым кодом один из членов команды выступает в роли эксперта. Побочный эффект - приобретение знаний.
Метафоры	Общее видение системы разрабатывается в форме простого текстового описания, называемого метафорой (или набором метафор). Используются единообразные обозначения, именования и др., понятные всем членам команды.
Непрерывная интеграция	Постоянно существует интегрированная версия системы. Модули интегрируются после того, как прошли все тесты.
Стандарты кодирования	Соблюдаются общепринятые стандарты кодирования, установленные в группе. Код должен быть понятен любому
Открытое рабочее пространство	Все программисты работают в одной большой комнате, окруженной маленькими
Разработка от тестов (Test-Driven Development)	На каждой итерации выполняется автономное и функциональное тестирование. При этом достигается почти 100% охват кода. Все тесты поддерживаются в актуальном состоянии и должны работать в любой итерации и версии.
Программирование парами	Программисты равноправны и работают попарно. Это гарантирует, что код проверен, и обеспечивает лучшее проектирование, тестирование и кодирование.
Простой проект (простые решения)	Сначала проектируется простая базовая архитектура системы, которая соответствует выбранной функциональности. Развитие выполняется в ходе разработки.
Рефакторинг, переработка	Постоянное перепроектирование для улучшения проекта, кода и снижения уровня сопряжений, а также как реакция на изменения требований. Сопровождается тщательным тестированием.

Поскольку в XP велика роль человеческого фактора, важное значение имеет распределение ролей в команде (таблица 11.11).

Таблица 11.11. **Распределение ролей в команде XP**

Роль	Обязанности
Заказчик (бизнес - аналитик)	Определяет требования, планирует версии, устанавливает приоритеты реализации, разрабатывает и выполняет функциональные (приемочные) тесты
Программисты	Проектируют свою часть, пишут автономные тесты, кодируют, тестируют, интегрируют. Помогают Заказчику реализовывать автоматизированные функциональные тесты
Тестировщики (не всегда)	Помогают Заказчику определять функциональные тесты. Разрабатывают планы тестирования и тестируют всю систему
Менеджеры (они же аналитики)	Помогают определять требования, осуществляют внешнее взаимодействие и управляют проектом

Процесс разработки XP состоит из 3 фаз, представленных ниже:

Анализ концепции. Вначале выполняется анализ концепции программной системы (как совокупности программных приложений), отражающей общее видение системы (называется метафорой). Цель – определить, *что* нужно делать, но не *как*. Результаты общего анализа оформляются в виде *историй* – с перечислением возможных сценариев применений системы (внешних функций), над которыми совместно работают Заказчик и программисты. Каждая история ориентирована на определенные задачи бизнеса, что позволяет ее протестировать и оценить с помощью количественных показателей (сроков реализации).

Две следующие фазы разработки выполняются циклически.

Планирование. Осуществляется на двух уровнях (версии и итерации). Касается двух основных вопросов: прогнозирование сроков завершения одной версии и итерации, и определение следующей.

1. Планирование версии. Заказчик определяет состав версии, упорядочивая функции по важности, и представляет эти функции программистам, которые оценивают их сложность (в терминах стоимости и сроков реализации). Заказчик, располагая этими оценками и зная важность характеристик, определяет примерный план выпуска версий проекта. В плане устанавливаются приоритеты реализации (хотя они могут и изменяться в процессе реализации). План регулярно пересматривается и уточняется всеми членами команды разработки.

2. Планирование итерации. Каждая версия разбивается на итерации. Заказчик представляет истории, которые должны быть реализованы в течение двух недель. Программисты дробят их на задачи (задача – это любая выполняемая работа) и оценивают стоимость их реализации. Итерации планируются на 2 недели, по истечении которых выбранная итерация должна работать. Основываясь на работах предыдущей итерации, группа решает, что можно успеть сделать в данной итерации.

Реализация. Цель каждой итерации – запустить в эксплуатацию несколько новых протестированных и готовых к работе функций (историй). Реализуются только истории, выбранные для текущей итерации.

Программисты проектируют и программируют очередную версию системы, разрабатывают тесты модулей и выполняют модульное тестирование.

По мере завершения задач разработанный код интегрируется в общую версию системы и тестируется вместе с ней. При этом либо все тесты успешно проходят, либо интеграция кода в систему оказывается невозможной. Интеграция выполняется часто (возможно и несколько раз в день) по мере готовности модулей.

Заказчик, работая параллельно с программистами, определяет один или несколько автоматизированных приемочных тестов. Группа реализует эти тесты и использует их для функционального тестирования. К концу итерации все тесты для отдельных модулей и все функциональные тесты должны работать. Эти тесты сохраняются для последующего тестирования.

После завершения итерации начинается планирование следующей (выбираются новые истории) и цикл продолжается до готовности полной версии системы. После выпуска очередной версии выбираются следующие истории и т.д.

Архитектура системы постоянно эволюционирует, а текущий проект трансформируется, при условии сохранения гарантии правильного выполнения тестов.

Процесс XP в высшей степени неформален, но требует высокого уровня самодисциплины, в противном случае он мгновенно превращается в хаотичный и неконтролируемый. Это процесс разработки с наиболее высокой степенью риска, поскольку другие методы снижения коммерческих рисков, кроме опоры на «человеческий фактор», в XP просто отсутствуют [34].

11.7.3. Методология SCRUM

Другая, менее известная в Украине, Agile-методология разработки ПС - **SCRUM** (дословно (в регби)– «схватка вокруг мяча») - гибкая методика управления проектом фирмы Advanced Development Methods, Inc., представленная впервые в 1995 году. Ныне применяется в более чем 50 организациях, включая Fuji-Xerox, Canon, Honda, NEC, Epson, Brother, 3M, Xerox and Hewlett-Packard и др.[37, 38].

В отличие от методологий, ориентированных на каскадную, спиральную или итеративную модель ЖЦ, предполагающих *фиксированный*, четко определенный предсказуемый процесс разработки с линейной последовательностью действий (возможно, в цикле) - *анализ требований, проектирование, программирование, тестирование* - SCRUM трактует процесс разработки как «черную коробку» с *контролируемыми* параметрами. Фактически наблюдается смещение парадигмы «определенный и повторяемый процесс» к парадигме - «контролируемый процесс».

SCRUM не требует «линейности» (внутри «коробки») и допускает, в зависимости от *текущих условий* выполнения проекта, переключение с одного действия на другое в любой момент разработки, что обеспечивает гибкость проекта. Таким образом, преодолеваются три основных препятствия успешной разработки:

- требования определены изначально не четко и/или не полностью,
- требования изменяются в ходе проекта,
- высокая степень непредсказуемости проекта при изменении условий среды разработки, использовании новых инструментов и методов.

Для гибкого управления проектом с такими особенностями в SCRUM применяются специальные *методы* (*приемы*) работы на стадиях проекта и *механизмы регуляции* (*рычаги регулирования*) при управлении непредсказуемым процессом.

С самого начала проекта требования к ПС преобразуются в списки функций, свойств или выполняемых работ, образуя *Портфель заказов* (Бэклог, Backlog). Требования декомпозируются до такого уровня, чтобы их можно было реализовать за один день, что улучшает прослеживаемость хода проекта. Содержимое Портфеля распределяется между командами проекта.

Команды проекта действуют как спортивные команды, где каждый член играет независимо от остальных, но всех игроков объединяет общая цель. В команде

– максимум 6 человек под руководством менеджера - Scrum-мастера, который управляет «почти хаотическим» процессом работы по методологии SCRUM.

Все запланированное время выполнения проекта разбивается на периоды продолжительностью 1 - 4 недели. В течение каждого периода, называемого *Спринтом* (рывком, броском, от sprint), команды выполняют задачи, которые определены для них содержимым Портфеля (*бэклог-задачи*).

Каждый день в ходе Спринта команда проводит *Совещание*, продолжительностью до 15 минут, на котором обсуждается состояние выполнения бэклог-задач каждого разработчика, а именно: завершенные работы (с момента предыдущего совещания), незавершенные работы (которые будут сделаны до следующего совещания), неожиданные проблемы. Результаты совещания используются для мониторинга продвижения проекта.

Процесс программной инженерии включает ***три фазы*** (таблица 11.12).

Таблица 11.12. **Фазы и действия в методологии SCRUM**

Действие	Задача	Описание
Подготовительная фаза		
Технологическая подготовка		Определение стандартов разработки, их обзор, адаптация (разработка соответствующих руководств, толкующих положения стандартов), доведение стандартов до исполнителей.
Планирование		• Разработка списка задач в Портфель заказов проекта. • Определение дат и объемов релизов (поставляемых компонентов, версий, пакетов функций) продукта. • Выбор релизов, подлежащих немедленной разработке. • Отображение множества функций и свойств релиза в группу записей Портфеля заказов, связанных с этим релизом. • Определение команды (команд), которая будет разрабатывать релиз (выполнять пакет всех работ, связанных с релизом). • Оценивание риска и механизмов его устранения. • Обзор и корректировка записей Портфеля и пакетов работ. • Утверждение или решение об изменении инструментов разработки и инфраструктуры. • Оценка стоимости релиза, включая собственно разработку, общесистемные работы, поставку и обучение. • Верификация решений и обеспечение их реализации.
Архитектурное проектирование		• Обзор записей Портфеля, связанных с релизом. • Идентификация изменений, необходимых для реализации задач в Портфеле. • Анализ ПрО в объеме, необходимом для построения, развития или обновления моделей ПрО в соответствии с новым системным контекстом и требованиями. • Уточнение архитектуры системы в свете нового контекста системы и требований. • Идентификация любых проблем или задач, связанных с разработкой или реализацией изменений. • Обзорное совещание по проектным решениям, на котором каждая команда представляет подход к решению задач из Портфеля. При необходимости, перераспределение задач по командам.

Действие	Задача	Описание
Фаза циклической параллельной итеративной разработки (Спринты)		
Спринт		Итеративный цикл разработки. Руководство контролирует достижение требований по времени разработки, качеству или функциональности продукта. По окончании всех итераций (всех спринтов) наступает завершающая фаза проекта.
	Разработка	Определение изменений в продукте, которые необходимо произвести, чтобы решить задачи и удовлетворить требования, указанные в записях Портфеля, оформив решения в виде пакетов решений. Выполнение работы, включающих анализ ПрО, проектирование, конструирование, реализацию, тестирование и документирование изменений.
	Компоновка	Оформление изменений в виде пакетов решений, создание исполняемой версии изменений и описание того, как они реализуют требования, отраженные в записях Портфеля.
	Просмотр	Обзорное совещание всех команд, на котором разработчики докладывают о проделанной работе и ее результатах, ставят и решают проблемы, добавляют новые записи в Портфель. Анализ действий, принятых по установленным факторам риска.
	Уточнение	Обобщение информации, полученной на обзорном совещании, в виде соответствующих пакетов задач (требований), с отражением различных мнений и пожеланий, а также новых свойств.
Разбор Спринта		Анализ результатов Спринта
	Просмотр ПО	Просмотр продукта руководством проекта совместно с командой разработки. Присутствовать могут также заказчики, пользователи, покупатели, специалисты по продаже и др.
	Проверка соответствия Портфелю	Рассмотрение функционирующих исполняемых подсистем (компонентов), разработку которых осуществляла команда, включая изменения, внесенные в связи с реализацией задач, указанных в виде записей в Портфеле.
	Корректировка записей в Портфеле	При необходимости, как результат просмотра, корректировка путей реализации задач, указанных в записях Портфеля.
	Добавление записей в портфель	Один из результатов проверки – внесение новых записей в Портфель и их распределение по командам, что влечет за собой содержательное изменение создаваемых релизов.
	Планирование следующих проверок	Определение времени следующей проверки, исходя из состояния разработки и сложности задач. Обычная продолжительность Спринтов – от 1 до 4 недель.
Заключительная фаза		
		Когда руководство считает, что по времени, качеству, стоимости (затратам), соответствию требованиям и другим факторам пора оформлять релиз для поставки, оно объявляет релиз «закрытым» и открывает заключительную фазу проекта. В этой фазе разработанный продукт готовится к поставке. Выполняются интеграция, системное тестирование, формирование пользовательской документации, учебных материалов, документов по передаче продукта и др.

Первая фаза (подготовительная) и последняя фаза (заключительная) включают хорошо известные фиксированные процессы, в которых все действия, входы и

выходы четко определены. Поток работ линейный с некоторыми итерациями в подготовительной фазе.

Вторая фаза (фаза Спринтов) включает совокупность эмпирических процессов, выполняемых циклически. Управление ими осуществляется только с помощью периодического анализа состояния контролируемых параметров и применения к ним управляющих воздействий для удерживания от скатывания к хаосу. Процесс разработки в Спринтерской фазе максимально гибкий, основанный на использовании существующего опыта разработчиков и накоплении новых знаний методом проб и ошибок при продвижении к завершению проекта.

Проект остается открытым для любых изменений (касающихся условий среды, сложности и качества поставляемого продукта, времени и финансирования) до наступления Заключительной фазы.

Во избежание хаоса (непредсказуемости и сложности) в SCRUM определены контролируемые объекты процесса и механизмы их контроля (таблица 11.13).

Таблица 11.13. **Контролируемые объекты процесса разработки**

Объект	**Описание**
Портфель (Backlog)	Описания функциональных требований (новых или неадекватно реализованных в существующем релизе) и задач, связанных с ошибками, дефектами, запрошенными заказчиком расширениями, конкурирующими функциями продукта, технологическим развитием и др.
Релиз/ Расширение (Release/ Enhancement)	Содержимое Портфеля, которое в определенный момент времени в совокупности представляет жизнеспособный прототип с позиций требований, периода времени, качества и др.
Пакеты продукта (Packets)	Компоненты продукта или другие объекты, которые должны быть изменены для реализации задач из Портфеля в новом релизе.
Изменения (Changes)	Изменения, которые должны быть внесены в пакет для реализации задачи из Портфеля.
Проблемы Problems	Возникшие технические проблемы, которые должны быть решены, для того чтобы можно было реализовать изменения.
Риски Risks	Непрерывно оцениваемые факторы риска (угрозы), с которыми связан успех проекта. Результаты оценивания риска могут повлиять на выбор контролируемых параметров процесса.
Решения Solutions	Комплексные решения (пакеты решений) по проблемам и рискам, зачастую приводящим к изменениям.
Вопросы Issues	Артефакты проекта и вопросы, связанные с проектом, которые не определены в терминах пакетов, изменений и проблем.

Базируясь на современной теории управления процессами, SCRUM обусловливает построение ПС надлежащего качества, в быстро изменяющихся условиях проекта и в установленные сроки.

Методология SCRUM используется, с одной стороны, в качестве «оболочки» для существующих процессов и применяемых приемов работы в организациях, и, с другой стороны, как способ совершенствования этих процессов.

Одним из существенных недостатков SCRUM является то, что интеграционные и приемочные тесты не детализируются.

Подробнее о SCRUM можно узнать на Agile Software Development Portal по адресу http://agile.csc.ncsu.edu.

11.7.4. Другие Agile-методологии: DSDM, ASD, FDD

Помимо подробно рассмотренных методологий XP и SCRUM остановимся кратко на основных принципах других методологий «ускоренной» разработки, упомянутых в начале главы.

Динамический метод разработки систем (DSDM, от Dynamic Systems Development Method) предложен в 1995 году ассоциацией поставщиков и экспертов в области разработки информационных систем в Великобритании (образовавших-Консорциум DSDM). Считается первым методом, использующим модель быстрой разработки RAD (от Rapid Application Development) с прототипированием для достижения целей построения систем в условиях ограниченных ресурсов проекта.

Основные принципы DSDM соответствуют манифесту Agile-альянса:

- Обязательное активное *привлечение пользователя*.
- Группа разработки имеет право *самостоятельно* принимать значимые для проекта решения.
- Акцент на *частом выпуске* версий.
- Удовлетворение *первоочередных* бизнес-требований пользователей.
- *Итеративная разработка с приращениями*, определяемыми в соответствии с текущими бизнес-требованиями.
- Все изменения, сделанные во время разработки, *обратимы*.
- Базовые требования к системе утверждаются *перед началом проекта*.
- Тестирование *интегрировано* в ЖЦ.
- Обязательное *взаимодействие* и сотрудничество всех ключевых участников проекта (включая заказчиков и непосредственных пользователей).

Жизненный цикл разработки систем по методу DSDM включает *три последовательные фазы* – предпроектную, проектную и постпроектную, вторая из которых разделена на 5 стадий, выполняемых циклически. Краткое схематическое описание ЖЦ при использовании DSDM представлено в таблице 11.14.

Таблица 11.14. **Структура ЖЦ при разработке систем по DSDM**

Стадия	Описание
	Предпроектная фаза
	Идентификация возможных вариантов проектов и их ресурсов.
	Проектная фаза
1. Анализ возможностей	*Две последовательные взаимодополняющие стадии*. 1. Анализ возможности применения DSDM для определенного типа проекта системы. Идентификация рисков.
2. Анализ бизнес-процессов	2. Определение деловых процессов и групп пользователей. Формирование списка требований и их приоритезация. Определение общего плана разработки. Выбор ключевых приемов разработки. Определение архитектуры системы.
3. Итерация построения функциональной модели	3.1. Визуальное функциональное моделирование. Определение возможностей функционального прототипа, являющегося результатом итерации. 3.2. Согласование графика разработки прототипа. 3.3. Разработка функционального прототипа. 3.4. Анализ корректности функционального прототипа (тестирование и/или анализ документации). Определение перечня функциональных тестов для системы.

Стадия	Описание
4. Итерация проектирования и программирования	4.1. Идентификация прототипа проектных решений. Идентификация функциональных и технических требований. Определение стратегии реализации системы. 4.2. Согласование графика реализации требований. 4.3. Создание прототипа системы, который может быть передан потенциальному пользователю для апробации и тестирования. 4.4. Анализ корректности прототипа системы (тестирование и верификация решений).
5. Реализация	5.1. Одобрение и подтверждение правильности протестированных проектных решений пользователем. Подготовка документации пользователя. 5.2. Обучение будущих конечных пользователей системы. 5.3. Развертывание протестированной системы в местах установки. 5.4. Анализ влияния системы на бизнес. Основной вопрос – достигнуты ли цели, поставленные перед открытием проекта. В зависимости от результата ответа на этот вопрос – переход на постпроектную фазу или возврат на одну из предыдущих стадий проектной фазы.
	Постпроектная фаза
	Поддержание эффективного функционирования системы. Сопровождение и развитие

Основная особенность метода DSDM – разделение разработки на *фиксированные* периоды времени с *фиксированным* финансовым обеспечением стадий и *установленным* целевым уровнем качества системы. Не фиксированным остается *объем функциональных возможностей* системы. Известно, что реализация полного набора требований часто приводит к превышению времени и бюджета проекта. В DSDM требования, которые должны быть реализованы в определенный период, выбираются из общего списка и *приоритезируются* по убыванию их значимости. Действует правило Парето: 20% требований определяют 80% реальных бизнес-потребностей. Требования из списка, которые не удалось реализовать в установленный период за выделенные средства, опускаются.

Вторая важная особенность DSDM – *прототипирование* системы и тестирование прототипов пользователями, что позволяет с самых ранних стадий проекта «подключать» пользователей к созданию системы и своевременно определять в ней узкие места.

Третья особенность DSDM – акцент на *качестве* системы. Качество достигается за счет *тестирования* каждой итерации системы. Однако методы тестирования не оговариваются, а объем тестирования определяется исходя из жестких ресурсов, что делает метод DSDM мало пригодным для критических систем, требующих очень тщательного продолжительного тестирования перед выпуском.

Ввиду итеративной природы DSDM очень важными процессами для обеспечения качества системы являются такие процессы ЖЦ, как *Управления требованиями* и *Управления конфигурацией*, выполняемые применительно к каждому функциональному фрагменту (приращению) системы.

Подробнее о методе DSDM можно узнать из электронной энциклопедии по адресу http://en.wikipedia.org/wiki/DSDM.

Адаптивная разработка ПО (ASD, от Adaptive Software Development). Методология разработана Джимом Хайсмитом (президентом Information Architects,

Inc.) специально для экстремальных проектов и базируется на теории сложных адаптивных систем [39].

В ASD статический цикл разработки ПС - *Планирование–Проектирование–Программирование* - заменяется динамическим циклом *Размышление–Взаимодействие–Обучение* (*Speculate–Collaborate–Learn*).

Этот цикл предполагает непрерывное обучение, связан с постоянными изменениями, повторными оценками, попытками предусмотреть неизвестное будущее проекта, и требует тесного взаимодействия между менеджером, разработчиками, группой качества и заказчиками.

Размышление не противопоставляется планированию, а учитывает реально существующую неопределенность при решении сложных проблем, допускает изначальную ограниченность знаний и возможность ошибок, и предполагает большую долю исследований и экспериментов, как основу *Обучения*.

Взаимодействие – заменяет традиционный стиль командного управления при принятии решений. Современные сложные приложения зачастую не «строятся заново» группой индивидуумов, а постоянно эволюционируют, требуя сбора, анализа и использования больших объемов накопленной информации, что не под силу одиночкам. Группы проектов должны постоянно взаимодействовать, коллективно повышая свою способность к совместному принятию решений. В условиях быстрых изменений ситуации в проекте принятие основных решений делегируется непосредственно командам проекта. Правильность принятых решений, в свою очередь, также зависит от *Обучения*.

Качество *Знаний*, накопленных в результате применения таких практических приемов, как ретреспективный обзор проекта, является основным показателем правильности политики *Обучения* в организации и залогом ее адаптируемости.

Базовый адаптивный цикл разработки приложений по методологии ASD показан на рисунке 11.6.

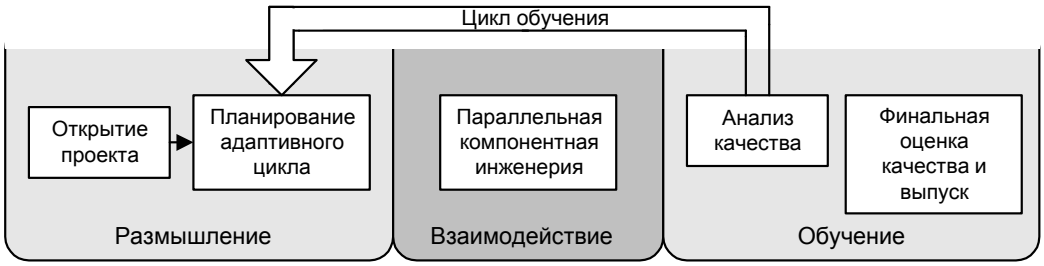


Рис. 11.6. **Адаптивный цикл разработки**

Фаза Размышления (Обдумывания) включает следующие семь шагов:

1. *Инициирование и открытие проекта*. Включает определение назначения, целей, ограничений, участников, требований; оценки масштаба, сферы действия и ключевых рисков.
2. *Определение временных рамок проекта с учетом оценок* (см. п.1).
3. *Определение оптимального числа циклов и продолжительности каждого из них*. Выбор зависит от общей продолжительности проекта и степени неопределенности в оценке ситуации. Для малых и средних проектов (размер<5000 УЕФ) цикл составляет 1-4 недели.

4. *Написание целевых утверждений для каждого цикла*. Промежуточные цели, установленные для каждого цикла, и их контроль по вехам проекта, повышают прозрачность разработки, способствуют своевременному устранению дефектов и предупреждению ошибок, и, в целом, накоплению знаний о предметной области и проекте. Верификация и тестирование – неотъемлемая часть каждого цикла разработки.

5. *Выделение основных компонентов и их распределение по циклам*. Для визуализации соотношений «компоненты-циклы» при планировании используется матрица «компоненты-циклы».

6. *Выделение технологических и вспомогательных компонентов и их распределение по циклам*. Критерии распределения таковы: продукты каждого цикла должны быть полезны пользователю, а наиболее высокие риски - устранены как можно раньше; должны учитываться естественные зависимости между компонентами, а ресурсы – распределяться рационально.

7. *Разработка списка задач*. Задачи определяются путем разбиения каждого компонента на отдельные части. В список добавляются задачи, не касающиеся компонентов непосредственно.

Фаза Взаимодействия. Предполагает параллельную разработку работоспособных компонентов. Основная задача менеджеров в этой фазе ASD – организация взаимодействия при параллельной работе распределенных команд.

Фаза Обучения. Предполагает *анализ качества* в конце каждого цикла разработки. Цели качества устанавливаются и *эволюционируют* таким образом, что всегда служат критериями завершения циклов, а также критериями пригодности применяемых практических приемов анализа, составляющих основу обучения. Цели анализа – не столько поиск дефектов, сколько *изучение* потребительских свойств будущего продукта (этот анализ выполняют заказчики), функциональных и технических характеристик продукта (технические обзоры выполняют разработчики), эффективности функционирования команд и используемых процессов, состояния проекта и необходимости перепланирования ресурсов перед началом следующего цикла. Состояние проекта определяется по состоянию каждого компонента, который был запланирован на текущий цикл разработки.

Разработка «от функциональных свойств» или ***управляемая функциональностью (FDD,*** от ***Feature Driven Development).*** Это методология разработки ПС, отталкиваясь от функциональных возможностей, постепенно «разворачиваемых» в ходе коротких итеративных циклов с приращениями. Впервые была предложена Джефом ДеЛуком для крупного банка в Сингапуре в 1997 году как концептуальное объединение многих известных *лучших* практических приемов разработки ПС. С 1999 года официально опубликована и поддерживается автором на собственном сайте - www.nebulon.com.

FDD характеризуется модельно-ориентированным (model-driven) процессом разработки, включающим *3 последовательные фазы*, в ходе которых формируется общая модель системы, и *2 итеративные фазы*, выполняемые применительно к *каждой* выделенной функции (свойству) системы. Для контроля продвижения реализации функций итеративные фазы разделены шестью *вехами* (по три вехи в каждой итеративной фазе).

Краткое схематическое описание ЖЦ ПС при использовании FDD представлено в таблице 11.15.

Таблица 11.15. **Структура ЖЦ при разработке систем по FDD**

Фаза	Описание действий
Разработка общей модели	Анализ предметной области (ПрО) и контекста применения разрабатываемой системы. Выделение направлений для детального моделирования. Разработка моделей по каждому направлению анализа и их объединение в общую модель системы. Используемые передовые практические приемы: формирование *небольших групп аналитиков со стороны пользователей (экспертов) и разработчиков*, *объектно-ориентированное моделирование ПрО*, *сквозной просмотр ПрО*, *коллегиальная проверка (анализ) моделей*.
Построение списка функций	Определение перечня функций, которыми должна обладать система. Используемый практический прием – *функциональная декомпозиция* предметной области на подобласти, связанные с отдельными видами бизнес-деятельности. Список образуют функции системы, реализующие отдельные шаги в бизнес-процессе, например, «проверка пароля пользователя» или «составление регламентного отчета». Уровень декомпозиции функций таков, что реализация каждой из них не должна требовать более 2-х недель.
Планирование реализации каждой функции	Формирование группы планирования, включающей менеджера проекта системы и лидеров-программистов. Выделение *классов* функций в списке, назначение *владельцев* классов (ответственных за проектирование и реализацию классов). Владелец класса отвечает за эффективность его реализации и концептуальную целостность. Формирование команд проектов каждой функции (т.е. укомплектование «по функциям»).
Проектирование каждой функции	Разработка *пакета* проектных решений по каждой функции (требований, альтернатив проекта, объектной модели (включающей классы, методы, атрибуты, календарный график)). Каждый лидер команды программистов выбирает небольшой набор функций из списка, которые могут быть реализованы за 2 недели. Далее совместно - лидер команды программистов и владелец соответствующего класса (кода) – разрабатывают *диаграммы последовательности разработки* для каждой функции (с указанием дат завершения автоматизации каждого бизнес-процесса); *уточняют* общую модель системы; пишут *прологи* методов и классов (определяя типы параметров, возвращаемых значений, исключительные ситуации, сообщения) и, по завершении, проводят *инспекцию* проекта.
Программная реализация каждой функции	Разработка программного кода каждой функции владельцами классов. Автономное тестирование, инспекция кода. Интеграция функции в общую конструкцию системы (формирование версии).

Для удобства контроля продвижения разработки функций системы, с каждой вехой в итеративных фазах ассоциированы уровни завершенности работы, выраженные в процентах:

Веха в итерации разработки:	*Прирост степени готовности:*
• Анализ ПрО завершен	+ 1 %
• Проектирование функции завершено	+ 40 %
• Инспекция проекта завершена	+ 3 %
• Кодирование завершено	+ 45 %
• Инспекция кода завершена	+ 10 %
• Функция «добавлена» в систему	+ 1 %

Таким образом, основные отличительные особенности FDD, способствующие повышению качества создаваемых систем, таковы:

1. *Объектно-ориентированное моделирование предметной области*. Соответствие стандартам UML. Удобство модификации (уточнения и развития) моделей.

2. *Разработка от функциональных свойств системы* (управляемая функциональностью). Снижение сложности разработки и конструктивной сложности. Прозрачность проекта для пользователей. Удобство изучения, сопровождения.

3. *Индивидуальное владение кодом*. Повышение ответственности за согласованность реализуемых функций, эффективность и концептуальную целостность класса.

4. *Группы, организованные по функциям (свойствам) системы*. Специализация разработчиков в отдельных областях бизнес-процесса. Динамические команды, коллективное принятие решений.

5. *Инспекции*. Применения передовых приемов проверки моделей, проекта, кода. Своевременное устранение ошибок и уточнение общей модели системы.

6. *Регулярные сборки версий*. Обеспечение доступности текущего состояния системы для обзора пользователями. Своевременная обратная связь от пользователей. Устранение ошибок интеграции на ранних стадиях разработки.

7. *Управление конфигурацией*. Идентификация исходного кода всех функций, которые должны быть реализованы в определенный срок. Ведение истории изменений в классах. Улучшение управления разработкой и внесения изменений.

8. *Отчетность о состоянии*. Повышение прозрачности хода разработки системы благодаря установленным вехам и количественным показателям готовности.

Дополнительную информацию о методологии FDD можно почерпнуть, например, из электронной энциклопедии *Wikipedia* по адресу в Интернет http://en.wikipedia.org/wiki/Feature_Driven_Development.

11.7.5. Сравнение Agile-методологий

Все *Agile-методологии* объединяет ряд *общих* характеристик, а именно:

- Модульность процесса.
- Итеративность с краткими циклами (от 1 до 6 недель), позволяющими выполнить быструю верификацию и реализацию, а также минимизировать риск.
- Минимизация действий в процессах (отсекаются все избыточные действия).
- Динамический жизненный цикл, адаптируемый применительно к возникающим новым рискам.
- Разработка приложений небольшими, но работающими порциями.
- Ориентация на человеческий фактор.

- Стиль работы: тесное взаимодействие и сотрудничество.
- Постоянное обучение.

В таблице 11.16 подытожены ключевые аспекты методологий и отмечены их известные недостатки [40].

Таблица 11.16. **Сравнение основных Agile методологий**

Методология	Условия применения	Ключевые аспекты	Известные недостатки
XP	- небольшие проекты; - высокая самодисциплина	- группы от 3 до 10 человек; - активное участие заказчика; - ежедневные сборки	- отсутствие цельного взгляда на ПС; - слабое применение практических приемов управления
SCRUM	ne же, что и в XP	- независимые самоорганизованные группы; - циклы выпуска версий до 30 дней	не детализированы интеграционные и приемочные тесты для версий
DSDM	- большие проекты; - деление ПС на независимые части; - предпочтение отдается при разработке ИС	- группы от 2 до 6 человек; - короткие итерации; - модель быстрой разработки (RAD)	метод доступен только для членов консорциума DSDM
ASD	поддержка распределенной разработки	- компонентный подход; - проверки с участием заказчика.	требует применения других процессов
FDD	- проекты разных типов и размеров; - возможно постепенное внедрение	- объектно-ориентированная компонентная разработка; - итерации: от 2-часов до 2 недель.	- охватывает только проектирование и реализацию; - требует применения других процессов

11.8. Бездефектная разработка ПС. Подход Cleanroom

С появлением (в конце 90-х годов) и быстрым распространением гибких и «облегченных» методологий разработки ПС, несколько в тень ушли методологии, основанные на четко регламентированном, дисциплинированном и документированном процессе разработки с разбиением на последовательные стадии и тщательным контролем состояния проекта. В литературе эти методологии получили обобщенное название «плановые», поскольку разработка и неукоснительное соблюдение документируемых планов и процедур – необходимое условие их применения.

Несмотря на критику в адрес плановых методологий за излишнюю жесткость и консерватизм, они продолжают широко использоваться в программной инженерии, особенно в проектах создания *критических систем* в разных отраслях (энергетике, авиации и др.), а также проектах, которые жестко регламентируются ведомственными стандартами и условиями договоров на разработку ПС в критических областях.

Одна из таких методологий - *Cleanroom* (от *clean room*, чистая комната).

Cleanroom – это подход к разработке ПС, основанный на идее *предупреждения* ошибок. Его характерные особенности таковы:

- *процесс разработки с приращениями*, стимулирующий непрерывное совершенствование организации;
- *технические обзоры*, предупреждающие дефекты и существенно снижающие издержки;
- *приемы проектирования и кодирования*, которые облегчают адаптацию продукта к изменяющимся требованиям;
- методы тестирования, фокусируемые на *измерении качества*;
- *команды*, создаваемые для решения определенной *проблемы*, что способствует сотрудничеству и снижению зависимости от «гуру»;
- *структурированная документация*, отражающая целостную картину разработки и помогающая членам команды осуществлять интеллектуальный контроль разработки.

Cleanroom может быть частью комплексного подхода к обеспечению качества в сочетании с ISO-9000 или TQM и применяться на всех уровнях зрелости организации. Методы Cleanroom совместимы с другими методологиями программной инженерии, включая объектно-ориентированную и клиент-серверную разработку с использованием CASE-средств и поддерживаемых ими методологий.

Подход Cleanroom был впервые использован в 80-е годы в IBM применительно к разработке критических систем диспетчеризации в авиации и обеспечения безопасности полетов, где стоимость отказов программного обеспечения измерялась в миллионах долларов и потенциально в человеческих жизнях. Д-р Х. Милс предложил разработчикам программного обеспечения перенять успешные практические приемы высокоточного производства и соединить их с передовой практикой программной инженерии. Таким образом, в Cleanroom вошел принцип предупреждения дефектов на каждом этапе работы (путем тщательного обзора рабочего продукта), сборочный подход к разработке системы начиная с наиболее важных для пользователя компонентов, формализация спецификаций и проекта *каждого* элемента сборки, согласование с пользователем всех спецификаций, тщательное документирование *любых* изменений, статистический контроль качества и командная форма работы.

Тестирование в Cleanroom *полностью отделено* от разработки - не только на уровне системы, но и на уровне модулей и компонентов, и сопровождается измерением и оценкой показателей качества (в частности, надежности). Только тестировщики, а не программисты, разрабатывающие код, могут *выполнять* его на машине. При проектировании выявляются критические структуры и функции, определяются профили использования системы и возможные режимы ее отказа, что позволяет применять дифференцированный подход к распределению ресурсов тестирования и обеспечению надежности отдельных модулей и компонентов.

Изначально высокая степень безошибочности кода достигается благодаря институциализации процедур технических обзоров проекта и инспекции соответствия кода спецификации. Причем технические обзоры носят конструктивный характер – их цель не просто обнаружить ошибки, а *коллективно* убедиться в качестве проекта. В отличие от отслеживания источников ошибок, обзоры Cleanroom локальны: просматривается каждый компонент, содержащий ограниченное количество модулей, и его качество оценивается независимо от остальных компонентов. Однако каждый обозреватель всегда видит общую архитектуру системы и старается максимально упростить проект. В результате применения такого подхода, разработанные

системы получаются компактными и легкими для сопровождения и развития. Кроме того, достигается гибкость в разработке, поскольку каждый член команды знает тонкости не только «своей» части, но и системы в целом. При коллективном обзоре снижается влияние возможных ошибок каждого, так как команда «отфильтровывает» неправильные решения.

Нужно отметить особенности процесса разработки системы. В его основе лежит *инкрементный* подход, состоящий в послойном наращивании функциональных возможностей системы и периодической поставке новых компонентов системы (инкрементов) заказчику.

Размер и содержание каждого инкремента определяются при *планировании инкрементов* на основе анализа не только приоритетов реализации функций системы, а и риска срыва проекта. Так, если требования к интерфейсу пользователя системы плохо определены, велик риск, что ее будет сложно использовать. Для контроля этого риска первыми создаются и поставляются пользователям инкременты системы, содержащие код, который реализует менее сложные функции, но обширный пользовательский интерфейс. Это дает возможность своевременно провести согласование требований к интерфейсу и внести необходимые изменения.

В отличие от высокоточного производства, где контроль качества основан на анализе соответствия спецификациям статистической выборки изделий из партии, в Cleanroom осуществляется контроль качества на основе *статистического тестирования* (глава 7) и последующего *прогнозирования надежности* программного обеспечения по моделям надежности. Статистическое тестирование выполняется в соответствии с операционным профилем, который отражает частоту выполнения пользователем различных операционных сценариев работы. Именно операционный профиль «руководит» выбором тестов, результаты которых образуют статистическую выборку для выполнения контроля. Это означает, что первыми будут обнаруживаться отказы наиболее часто используемых компонентов системы и устраняться дефекты в наиболее часто выполняемых фрагментах кода.

Эффективное применение Cleanroom требует опыта. Подход позволяет внедрять отдельные приемы постепенно, не изменяя общей методологии разработки систем, и рекомендует начать со статистического контроля качества, который поддерживается тремя приемами - инкрементной разработкой, командной формой работы и отделением тестирования от кодирования.

Подход Cleanroom естественным образом дополняет методологии совершенствования процессов разработки ПС. Он полностью разделяет принципы TQM и не противоречит CMM. Как раз CMM можно рассматривать как схему поэтапного ввода приемов Cleanroom на каждом уровне зрелости.

Одним из основных факторов успеха Cleanroom является специфицирование каждого отдельного компонента системы и его согласование с пользователями. Именно такая точка зрения на разработку идеальна для применения объектно-ориентированного или клиент-серверного подходов, хотя применение других подходов ничем не ограничивается. Даже применение CASE-методологий, «упраздняющих» программирование и бумажную технологию документирования, не отменяет квалифицированного принятия проектных решений о процессах, данных и их взаимодействии, и не заменяет коллективного оценивания качества этих решений.

Cleanroom изначально оформился как практический подход к разработке, а не явился результатом научных исследований. Поэтому, существует мало сообще-

ний о специальных контролируемых экспериментах по его применению для разных классов ПС. Однако есть многочисленные опубликованные примеры применения Cleanroom для широкого спектра отраслей и классов систем (см., например, работы [41 - 43], а также по адресу www.cleansoft.com/cleansoft_library.html).

11.9. Сравнение методологий улучшения процесса разработки

11.9.1. Сопоставление методологий разработки

В зависимости от строгости и предсказуемости предписанного процесса программной инженерии, методологии разработки образуют довольно широкий непрерывный спектр, на одном конце которого - хаотичный недисциплинированный процесс, называемый также «ковбойским» или «хакерским» [44, 45], а на другом – полностью регламентированный плановый процесс.

На рисунке 11.7 методологии упорядочены в спектре от «адаптивных» до «предсказуемых» процессов [44].

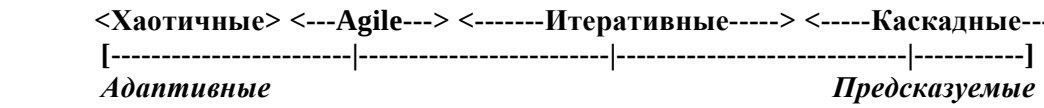


Рис. 11.7. **Спектр методологий разработки ПС**

Хаотичный процесс разработки характеризуется полным отсутствием или неупорядоченностью применяемых методов: члены группы делают то, что считают нужным, не руководствуясь определенными планами.

В Agile-методологиях процесс не хаотичен. Частые пересмотры планов разработки в этих методологиях, акцент на тесном взаимодействии и относительно небольшой объем документов вовсе не свидетельствуют о хаотичности – процесс остается дисциплинированным, хотя и не всецело плановым. Agile-методологии находятся в области адаптивных процессов [44].

Методологии разработки с адаптивными процессами ориентированы на краткосрочное планирование и быструю перестройку работы в соответствии с изменяющимися условиями. При их применении возможны проблемы, связанные с расчетами трудоемкости и стоимости проекта, определением условий договоров с заказчиком.

Методологии разработки с полностью предсказуемыми процессами (и преимущественно с каскадной моделью ЖЦ), основаны на тщательном долгосрочном планировании на весь период разработки. При их применении, как правило, возникают проблемы, связанные с изменением характеристик и задач проекта, сроков и состава работ.

Итеративные методологии, в частности, рассмотренные в этой главе RUP и MSF – представляют собой хорошо сбалансированные методологии, сочетающие в себе элементы адаптивности и предсказуемости. Длительность итераций может колебаться от нескольких недель до нескольких месяцев.

По сравнению с итеративными методологиями, в agile-методологиях предполагаются более сжатые сроки итераций и коллективное принятие решений о дальнейших действиях в проекте.

В методологиях с каскадной моделью ЖЦ прогресс разработки измеряется в терминах производимых на стадиях рабочих продуктах. Продолжительность каждой стадии колеблется от нескольких месяцев до нескольких лет.

Сравнивая разные подходы к разработке ПС, Б.Боэм упорядочил их в порядке возрастания степени *строгости планирования* и сопоставил с моделями зрелости SW-CMM и CMMI [45] (рисунок 11.8).

Рис.11.8. **Упорядочение методологий по строгости планирования**

Хотя SW-CMM, разработанная в начале 90-х годов, изначально в большей степени была ориентирована на плановые процессы, Б.Боэм предложил трактовать ее шире, как охватывающую также современные итеративные процессы. Сами авторы модели CMM также считают, что «либеральная», а не «консервативная» трактовка CMM в целом позволяет охватить основные аспекты гибких методологий [46]. В том же духе Б. Боэм предлагает интерпретировать и модель CMMI, в которую добавлены процессы управления риском, управления интегрированными группами и организационной средой, что позволяет распространить CMMI и на Agile-методологии.

Анализируя различные подходы к разработке в условиях жесткой конкуренции на рынке программных продуктов, Б.Боэм пришел к выводу, что, несмотря на устойчивую тенденцию перемещения акцентов в сторону «ускоренной» разработки, в большинстве случаев в реальных проектах необходимы сбалансированные подходы, комбинирующие «скорость» и «дисциплину».

Б.Боэм и Р.Торнер предложили метод анализа условий применения методологий и подход, основанный на управлении риском, для выработки *сбалансированной стратегии разработки*, включающей элементы разных методологий, которая может адаптироваться к целям, ограничениям и предпочтениям конкретных проектов или организаций [47, 48].

11.9.2. Определение условий применения методологий разработки

Метод анализа применимости существующих методологий к реальным проектам (Home Grounds' Analysis) основан на схеме сопоставления характеристик проектов и возможностей методологий по следующим четырем ***направлениям*** [48]:

1. *Сфера применения*: основные цели, размер и среда проекта.
2. *Управление*: отношения с заказчиком, планирование и контроль, взаимодействие.
3. *Технические аспекты проекта*: требования, разработка, тестирование;
4. *Персонал*: заказчики, разработчики, культура (сложившиеся традиции в коллективе).

В таблице 11.17 приведены ***ключевые аспекты анализа*** по всем направлениям, основные характеристики «ускоренных» процессов и предсказуемых (плановых) процессов.

Таблица 11.17. **Направления сравнения подходов**

Направление	**Ключевые аспекты**	**Основные характеристики**	
		Ускоренные процессы	**Плановые процессы**
Сфера применения проекта	Основные цели	*Быстрая демонстрация результатов работы, реакция на изменения*: - быстрая отдача от инвестиций в проект; - быстрая реакция на требования рынка программных продуктов.	*Предсказуемость, стабильность, обеспечение гарантий*: - более «прозрачное» управление проектом, инвестиции в процесс; - долгосрочные инвестиции в ПС; - необходимость разработки планов и большого объема документации, соблюдения стандартов (для критических систем).
	Размер	*Небольшие группы и проекты*	*Большие проекты и коллективы*
	Среда разработки	*Нестабильная, изменчивая*: - тесное взаимодействие разработчиков и участие заказчика.	*Стабильная, слабо изменяющаяся*: - разработка ПС по государственным заказам, аутсорсинг, распределенные проекты.
Управление	Отношения с заказчиком	*Квалифицированные заказчики (при постоянном участии в проекте), определяющие приоритеты реализуемых возможностей ПС*: - доверие заказчиков (места на рынке) поддерживается выпуском работоспособных версий ПС; - источником проблем может стать взаимодействие между заказчиком и разработчиками.	*Отношения регулируются условиями договора*: - доверие поддерживается отслеживанием рабочих продуктов в оговоренных точках (сдачи этапов работ); - источником проблем во взаимоотношениях могут быть условия договора.
	Планирование и контроль	Внутренние краткосрочные планы, контроль на качественном уровне:	Документированные долгосрочные планы (на все этапы проекта), контроль на количественном уровне
	Взаимодействие	Неформальный обмен знаниями: - тесное взаимодействие разработчиков; - открытый и доступный всем код и тесты;	Посредством обмена и изучения документов: - четкий процесс документирования, - наличие всех утвержденных рабочих продуктов (планы, требования, отчеты и т.д.),

Направление	Ключевые аспекты	Основные характеристики	
		Ускоренные процессы	**Плановые процессы**
		- улучшение процесса путем самосовершенствования (парное программирование, участие в коллективных проверках).	- улучшение процесса посредством изменения документированного процесса и доведение изменений до исполнителей.
Технические аспекты проекта	Требования	*Приоритезированные заказчиком (пользователем) неформальные требования и тесты*: - функциональные тесты составляют основную часть определения требований заказчика.	*Формализованные требования и их обязательная оценка*: - больше внимания *не* функциональным требованиям.
	Разработка	*Простой проект, короткие циклы; легко выполнять переработки и изменения*: - требования пользователя могут оказаться сложнее, чем ожидалось; - переработки и изменения также могут оказаться сложными.	*Большой подробный проект, сложные переработки*: - большая доля затрат на разработку архитектуры ПС; - возможность разработки повторно-используемых компонентов.
	Тестирование	*Функциональные требования определяются выполнимыми тестами*: - раннее и постоянное модульное и функциональное тестирование	*Документированные тест-планы и процедуры*: - позднее тестирование с акцентом на соответствие спецификациям
Персонал	Заказчики	Готовые к сотрудничеству, представительные, полномочные, знающие и квалифицированные заказчики, постоянно участвующие в проекте	Те же условия, но без постоянного участия заказчика
	Разработчики	Очень высокие требования к квалификации и владению методологией. Рекомендуемый состав группы по уровням квалификации (см. табл.11.18): не менее 30% членов группы с уровнями 2 и 3, но не 1В, с полной занятостью.	Менее жесткие требования к квалификации, учет времени на обучение. Рекомендуемый состав группы по уровням: не менее 50% членов группы с уровнем 3 (на ранних этапах проекта с полной занятостью не менее 10%), 30% на уровне 1В с полной занятостью.
	Культура	Много степеней свободы, приверженность принципам методологии	Работа регламентирована рамками политик и процедур методологии

Для оценки потребности в людских ресурсах и требований к квалификации специалистов Б.Боем применяет *шкалу оценивания*, предложенную А.Кокборном в [49] для XP (таблица 11.18).

Таблица 11.18. **Шкала оценивания квалификации персонала в XP**

Уровень	Характеристики
3	Способен изменить (пересмотреть) правила при возникновении неожиданной беспрецедентной ситуации
2	Способен изменить (пересмотреть) правила при возникновении неожиданной ситуации, у которой был прецедент
1A	После обучения, способен выполнять отдельные шаги метода такие как: оценка размера историй для версий, разработка шаблонов, сложные переработки (рефакторинг), сложная интеграция готовых продуктов (COTS). Со временем может достичь уровня 2.
1B	После обучения, способен выполнять отдельные процедуры метода такие как: кодирование несложных задач, простые переработки (рефакторинг), соблюдение стандартов кодирования и технологических процедур, тестирование. С опытом может достичь некоторых навыков уровня 1A.
-1	Может обладать нужной квалификацией, но не способен к взаимодействию или соблюдению принятых правил.

В основе метода анализа условий применения определенной методологии для конкретного проекта лежит оценка проекта по ***5 основным факторам*** (связанным с ключевыми аспектами выделенных направлений анализа):

1. *Размер* – количество персонала.
2. *Критичность* – степень критичности продукта (для пользователя или по безопасности).
3. *Изменчивость проекта* – частота изменения требований.
4. *Культура* – стиль руководства в организации.
5. *Персонал* – квалификация персонала.

Эти факторы, с одной стороны, являются критическими факторами успешного применения методологии, а с другой (при несоответствии реальным условиям проекта) – служат источниками риска срыва проекта.

Графическая интерпретация метода заключается в построении и анализе ***полярной диаграммы***, оси координат которой (в полярной системе координат) отображают факторы, по которым выполняется анализ.

По каждому фактору определены следующие количественные единицы измерения:

- *Размер* – количество ключевых участников проекта (не только разработчиков);
- *Критичность* – оценка величины потерь из-за отказов в ПС:

1 – нарушение удобства применения, потери времени пользователя;
2 – потери репутации фирмы-потребителя, небольшие финансовые потери;
3 –значительные финансовые потери;
4 – потеря жизни (одной);
5 – потеря жизни (массово).

- *Изменчивость* – скорость изменения требований, выраженная в % изменений в месяц;
- *Культура* – долевая оценка «хаоса» в работе - чем выше значения, тем больше свободы действий и тем больше ответственности (оценки субъективные);
- *Персонал* – соотношение количества персонала уровня 1 и персонала уровня 2 и 3 (в %).

Оценки для *agile-методологий* располагаются на графике ближе к центру полярной диаграммы (небольшие значения), а для *плановых* – ближе к периферии (большие значения). На рисунке 11.9 изображена полярная диаграмма, отражающая идеальный профиль применения Agile-методологий, а на рисунке 11.10 – идеальный профиль плановых.

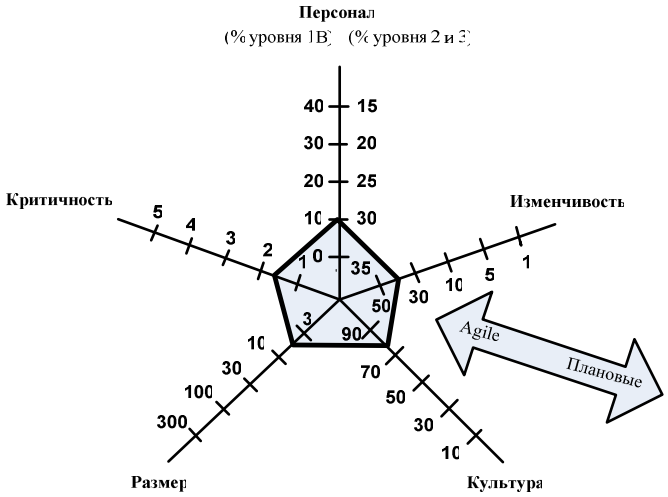


Рис. 11.9. **Идеальный профиль Agile-методологий**

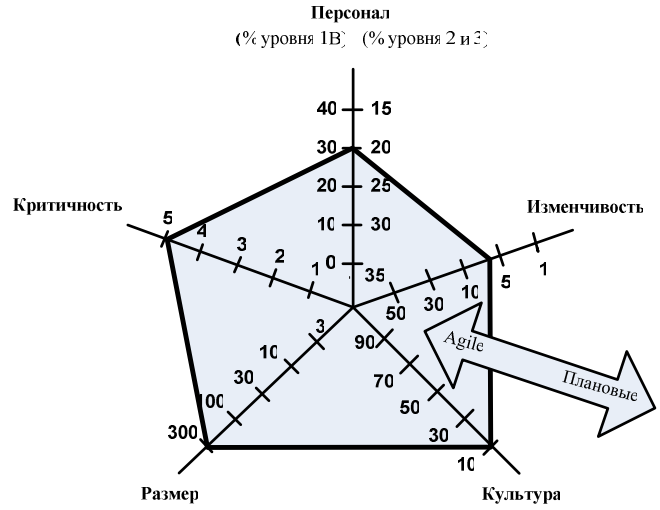


Рис. 11.10. **Идеальный профиль плановых методологий**

В таблице 11.19 подытожены *идеальные условия* применения методологий.

Можно заметить, что такие характеристики, как *Изменчивость* и *Персонал*, ассиметричны.

Так, A*gile*-методологии в основном ориентированы на высокую скорость изменений условий разработки, хотя пригодны и в случае относительной стабильности. Плановые методологии в условиях быстрых изменений практически не применимы.

Таблица 11.19. **Идеальные условия применения методологий**

Фактор	Идеальные условия применения	
	Ускоренные	**Плановые**
Размер	Небольшой проект, менее 10 человек	Большой проект (несколько лет, до 300 человек)
Критичность	Важный, но не критичный (по стоимости или безопасности) проект	Критичный по безопасности
Изменчивость	Высокая скорость изменения требований (заказчики не могут четко сформулировать потребности)	Низкая скорость изменения требований, значительные затраты на разработку подробных спецификаций требований
Культура	Небольшая компания-разработчик, четкая структура руководства, много степеней свободы	Иерархическая структура управления и контроля, работа четко регламентирована инструкциями.
Персонал	Оптимальное сочетание квалификации разработчиков в команде (уровня 2 и 3 по шкале Кокборна)	Большая доля разработчиков невысокой квалификации в команде.

Что касается персонала, ускоренные методологии требуют от разработчиков более широкого спектра знаний, чем плановые, зато последние вполне применимы и в условиях менее квалифицированного персонала.

Порядок применение метода для оценки условий проектов таков.

1. Оценить условия проекта и среду по 5 критическим факторам.
2. Отобразить оценки на полярной диаграмме.

Если все точки расположены близко к центру координат, предпочтительнее использовать ту или иную ускоренную методологию. Если же все точки расположены далеко от центра координат, можно использовать традиционную плановую методологию.

На практике, реальные условия проектов далеки от идеальных, и чаще всего профиль методологий оказывается *смешанным*. Например, на рисунке 11.11 пунктиром изображено смещение идеального agile-профиля проекта в сторону планового по фактору «Размер», а на рисунке 11.12 – смещение планового профиля к agile-профилю по фактору «Изменчивость».

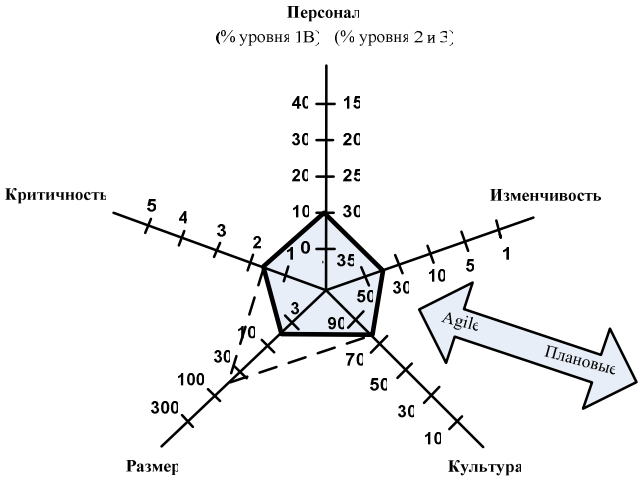


Рис. 11.11. **Смещение agile-профиля в сторону планового**

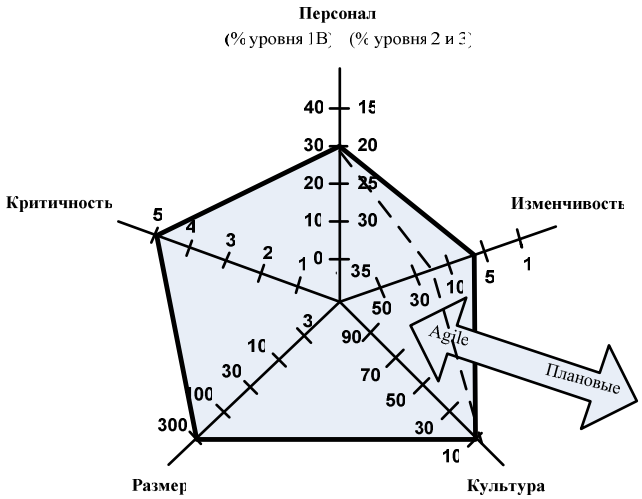


Рис. 11.12. **Смещение планового профиля в сторону agile**

Если реальные условия проекта не соответствуют идеальным, Б.Боэм и Р.Торнер предложили оценивать ***риск применения*** каждой методологии по следующей схеме:

1. Идентифицировать источники риска:

- определить риск применения *ускоренных* процессов по источникам риска:
  возможность масштабирования методологии и критичность проекта,
  неадекватное применение принципа «простой проект»,
  нехватка персонала требуемой квалификации;
- идентифицировать риск применения плановых процессов по источникам риска:
  быстрые изменения,
  необходимость быстрого получения результатов,
  не стабильные требования,
  нехватка персонала нужной квалификации.

Основные источники риска, определяемые критическими факторами, представлены в таблице 11.20.

Таблица 11.20. **Критические факторы успеха и источники риска**

Фактор	Ускоренные процессы	Плановые процессы
Размер	Тесное взаимодействие и обмен знаниями трудно обеспечить в большом коллективе.	Избыточная бюрократия и трудность настройки процесса для небольших групп и проектов.
Критичность	Не проверены для критичных проектов, свободное проектирование и слабое документирование могут стать источником риска для заказчика	Пригодны для ПС, критических по безопасности функционирования. Могут быть избыточны для ПС, не являющихся критическими.
Изменчивость проекта	В стабильной среде проекта такие характеристики, как простой проект и постоянные переработки, менее эффективны	Тщательное планирование и проектирование, применимые в стабильной среде, могут оказаться избыточными и затратными, поскольку потребуют постоянного контроля за изменениями.

Фактор	Ускоренные процессы	Плановые процессы
Культура	«Выигрыш на хаосе». Персонал имеет полномочия и счастлив нести ответственность.	«Выигрыш на порядке». Роли персонала, а также применимые процедуры четко определены.
Персонал	Достаточное количество персонала нужной квалификации (с уровнем 2 или 3) с полной занятостью. Риск использовать персонал уровня 1В.	В начале проекта (для анализа и проектирования) необходимо преобладание персонала с уровнем 2 и 3, затем могут добавляться специалисты уровня 1В.

2. Оценить рейтинг риска по 6 бальной шкале:

- минимальный,
- средний,
- серьезный, но контролируемый,
- очень серьезный, но контролируемый,
- постоянный (showstopper).

3. *Уточнить оценки риска*. Если при оценке рейтинга возникают затруднения - собрать дополнительные данные (например, с помощью прототипирования или выполнения «пилотного» проекта);

4. Принять решение:

- если преобладают риски применения плановых методологий, выбрать agile, и наоборот;
- если ни одна из оценок не преобладает – по возможности применять ускоренную разработку, в иных ситуациях – плановую.

5. *Выполнять проект (разработку ПС) по выбранной методологии*.

6. *Отслеживать прогресс, постоянно переоценивать риски и, в случае необходимости, настраивать процесс нужным образом*.

Риски *не выполнения* условий применимости методов могут возникнуть в процессе разработки, что приведет к ***изменению профиля процесса***.

Изменение agile-профиля в сторону планового. Может потребоваться, например, при следующих условиях:

1. Проект изначально попадал под условия применения agile-методологии, но оказался масштабнее, чем это ожидалось.

2. Проект изначально попадал под условия применения agile-методологии, но возникла потребность в привлечении некоторых дополнительных процессов или рабочих продуктов, свойственных плановым методологиям, как то:

- разработка высокоуровневой архитектуры для определения общей структуры ПС;
- более тщательное определение «вех» внутри итераций для улучшения управления;
- применение шаблонов проектирования (или других архитектурных подходов) для того, чтобы избежать избыточной переработки и др.

Изменение планового профиля в сторону agile. Может потребоваться, например, в случае, если организация, специализирующаяся на разработке программных продуктов определенной категории, столкнулась с проектом, в котором требования к продукту оказались быстро изменяющимися.

Изменить подход в сторону agile методологии для организации сложнее, поскольку это требует изменения таких факторов, как *культура* и *персонал*. Действи-

тельно, во-первых, возникает необходимость изменения стиля управления в сторону менее директивного, и, во-вторых, - необходимость изменения профиля персонала путем обучения, консультирования и прочее.

Анализируя разные методологии разработки и критические факторы их успешного применения, эксперты пришли к выводу, что, несмотря на важность методологий разработки и их вклад в улучшение качества ПС, все же самыми важными факторами в любых проектах остаются [50]:

- *люди* (что влечет за собой и необходимость осознания важности человеческого фактора - как разработчиков, так и заказчиков);
- *удовлетворенность заказчиков* (что обусловливает необходимость определения значимости ПС в целом и ее компонентов, приоритезацию требований и др.);
- *взаимодействие* (сотрудничество и взаимопонимание как разработчиков с заказчиками, так и самих разработчиков);
- *ожидания руководства* (не следует стремиться сделать больше, чем возможно! Необходимо быть готовым сузить область проекта).

Литература к главе 11

1. *Лапидус В.А.* Система Шухарта // материалы семинара «Всеобщее управление качеством (TQM) и статистические методы» (Киев, 20-22 февраля 2001), Украинская ассоциация качества. Межотраслевой центр качества «ПРИРОСТ». – 2001. – 29 с.
2. *Исикава К.* Японские методы управления качеством. – М.: Экономика, 1988. – 216 с.
3. *Кумэ Х.* Статистические методы повышения качества. – М.: Финансы и статистика. – 1990. – 304 с.
4. *Электронный* учебник по промышленной статистике.- M., StatSoft, Inc. – 2001. (www.statsoft.ru/home/portal/textbook_ind/default.htm)
5. *Florac W.A.*, Carleton A.D. Statistical Process Control: Analyzing a Space Shuttle Onboard Software Process // IEEE Software. - july/aug 2000. - P. 97 – 106.
6. *Weller E.F.* Practical Applications of Statistical Process Control // IEEE Software. - v.17. - n.3, may/june, 2000. - P. 48 – 55.
7. *Humphrey W.S.*, Snyder T.R., Willis R.R. Software process improvement at Hughes Aircraft // IEEE Software. - v.8. - n.4, july/aug. – 1991. - P. 11 – 23.
8. *BOOTSTRAP* - http://www.cse.dcu.ie/essiscope/sm5/approach/boot-1.html
9. *TRILLIUM* - http://www.sqi.gu.edu.au/trillium/t3_toc.html
10. *Лаврищева Е.М.* Основы технологической подготовки разработки прикладных программ СОД // АН УССР, ИК им. В.М. Глушкова, Препринт 87-5. – Киев. – 1987. – 29 с.
11. *Basili V.R.* A Quantitative Approach to Software Management and Engineering // www.cs.umd.edu/class/fall2001/cmsc735/introduction.pdf
12. *Basili V.R.*, Green S. Software Process Evolution at the SEL // IEEE Software. - v.11. - n.4, july/aug. – 1994. - P. 58 – 66
13. *Basili V.R.*, Caldiera G., Rombach H.D. Experience Factory // Encyclopedia of Software Engineering (John J. Marciniak, ed.). - v. 1. – NY.: John Wiley Sons. – 1994. - P. 469 – 476.
14. *Трофимов* Сергей. Унифицированный процесс разработки от Rational Software // «PCWeek/RE».- №4.- 2003 (а также, www.caseclub.ru/articles/rup2.html).
15. *Закис А.* RUP — знакомый незнакомец // «Открытые системы».- №4.-2004 (а также www.osp.ru/text/302/184459.html).

16. *Якобсон А.*, Буч Г., Рамбо Дж.Унифицированный процесс разработки программного обеспечения / Спб.: Питер, 2002. - 496 с.
17. *Трофимов* Сергей. Рабочие процессы RUP и диаграммы UML // http://www.caseclub.ru/articles/rup_uml.html
18. *Kroll P.* The Spirit of the RUP // www.ibm.com/developerworks/rational/library/ content/RationalEdge/dec01/TheSpiritoftheRUPDec01.pdf
19. *Rational* Unified Process http://www.interface.ru/rational/rup01_t.htm
20. *Крачтен Ф.* Введение в Rational Unified Process. — Изд. 2-е: Пер. с англ. // М.: Издательский дом "Вильямс", 2002. — 240 с
21. *Трофимов С.А.* Rational XDE для Visual Studio .NET // М.: ООО «Бином-Пресс», 2004 г. — 304 с
22. *Кролл П.*, Крачтен Ф. Rational Unified Process – это легко. Руководство по RUP для практиков / М. «КУДИЦ-Образ», 2004. - 432 с.:
23. *Бергстрем С.*, Роберг Л. Rational Unified Process – путь к успеху. Руководство по внедрению RUP / М. «КУДИЦ-Образ», 2004. - 256 с.
24. *Трофимов С.А.* CASE-технологии: практическая работа в Rational Rose. 2-е издание // М.: «Бином-пресс», 2002, 288 с.
25. *Хаф Л.* Методология разработки ПО. Часть 5. Microsoft Solutions Framework // Компьютер Пресс.-№7.-2004 (а также www.compress.ru/Archive/CP/2004/7/29/)
26. *История* манифеста Agile // agilemanifesto.org.ru/history.html.
27. *Принципы*, лежащие в основе манифеста Agile // agilemanifesto.org.ru/principles.html
28. *Martin R.C.* Extreme Programming development through dialog // IEEE Software. - july/aug 2000. - P. 12 - 13
29. *Бек К.* Экстремальное программирование» // Открытые системы. – N. 1-2. – 2000. - с. 59 – 65.
30. *Бек К.* Экстремальное программирование. Библиотека программиста. С.-Петербург: Питер. – 2001. – 224 с.
31. *Бек К.*, Фаулер М.. Экстремальное программирование: планирование. Библиотека программиста. С.-Петербург: Питер. – 2003. – 144 с.
32. *Бек К.* Экстремальное программирование: разработка через тестирование. Библиотека программиста. С.-Петербург: Питер. – 2003. – 224 с.
33. *Ауэр К.*, Миллер Р. Экстремальное программирование: постановка процесса. С первых шагов и до победного конца. Библиотека программиста. С.-Петербург: Питер, 2003. – 368 с.
34. *Хаф Л.* Методологии разработки программного обеспечения. Часть 2. Экстремальное программирование // «Компьютер пресс».- №10.- 2003 (http://www.lib.csu.ru/dl/bases/ prg/kompress/articles/2003_10_XP/index.htm)
35. *Парное* программирование // www.osp.ru/cw/2001/09/044_0.htm
36. *Коуберн А.*, Вильямс Л. Парное программирование: преимущества и недостатки // www.maxkir.com/ sd/pairprog_RUS.htm
37. *Страница* "SCRUM" в энциклопедии "Wikipedia" // en.wikipedia.org/wiki/Scrum
38. *Rising L.*, Janoff N. S. The Scrum Software Development Process for Small Teams // IEEE Software. - july/aug 2000. - P. 26 – 32.
39. *Highsmith J.* Retiring Lifecycle Dinosaurs // Software Testing & Quality Engineering. - May/June 2000 http://www.stqemagazine.com.

40. *Awad M.A.* A comparison between agile and traditional software development methodologies - Students/Files/2005/MohamedAwad/CorrectedDissertation.pdf.
41. *Tammel C.J.*, Binder L.H., Snyder C.E. The automated production control system: A case study in Cleanroom software engineering // ACM Transaction on Software Eng. And methodology. - v.1. - n.1. - jan 1992. - P. 81 – 94.
42. *Lokan C.J.* The Cleanroom process for software development // Australian Computer J.. - v.25. - n.4, November. – 1993. - P.129 - 134.
43. *Linger R.C.* Cleanroom Process Model // IEEE Software. – march 1994. - P. 50 - 58.
44. *Agile* methods information - http://www.wikimirror.com/Agile_Methods.
45. *Boehm B.* Get Ready for Agile Methods, with Care // Computer, January 2002. –P.64-69.
46. *Paulk M.* Extreme Programming from a CMM Perspective. // IEEE Software, Nov.-Dec. 2001, P. 19-26.
47. *Boehm B.*, Turner R. Balancing Agility and Discipline: Evaluating and Integrating Agile and Plan-Driven Methods. // Proceedings of the 26th International Conference on Software Engineering (ICSE'04) , P.718-719.
48. *Turner R.*, Boehm B. People Factors in Software Management: Lessons From Comparing Agile and Plan-Driven Methods //CROSSTALK The Journal of Defense Software Engineering December 2003 www//stsc.hill.af.mil.
49. *Cockburn A.*, Highsmith J. Agile Software Development: The People Factor. // Computer, Nov. 2001, P. 131-133.
50. *Comparing* Agile and Plan- Driven Processes http://www.dcs.gla.ac.uk/~ray/SEP4 Notes/SEP4-Notes5.pdf

Глава 12. ПОДХОДЫ К ОЦЕНКЕ КАЧЕСТВА ПРОГРАММНЫХ СИСТЕМ

12.1. Оценка качества программных продуктов

12.1.1. Процесс оценки программных продуктов

В основе оценивания качества программных продуктов лежат четыре базовых понятия – *модель качества*, *метод оценивания*, *метрики* и *вспомогательные инструменты*. Оценивание качества продуктов невозможно, если не известны *требования пользователя к качеству* (критерии качества) и/или не определен процесс *измерения* продуктов, процессов и ресурсов (не выполнятся сбор данных для применения внутренних и внешние метрик качества) и/или отсутствует *процесс оценки* продуктов в ходе ЖЦ.

Назначение процесса оценки продуктов – одного из процессов поддержки ЖЦ, определенных в стандарте ISO/IEC 12207, - состоит в том, чтобы «гарантировать путем систематического измерения и оценивания, что продукт удовлетворяет установленным и предполагаемым требованиям пользователей к этому продукту. В результате успешного выполнения процесса:

- будут установлены требования, касающиеся проведения оценивания;
- будут определены критерии оценки продукта;
- будут специфицированы методы выполнения оценивания, и все действия в рамках этих методов будут надлежащим образом выполняться;
- данные измерений будут собираться, а результаты применения метрик - оцениваться по отношению к критериям;
- результаты деятельности по оцениванию продукта будут доступны для всех заинтересованных сторон» [1].

Оцениваться могут промежуточные (рабочие) продукты ПС в ходе разработки, а также конечный программный продукт.

Целью оценивания качества рабочих продуктов может быть:

- принятие решения о приемке промежуточного продукта у соисполнителя;
- принятие решения о завершении какого-либо процесса и передаче продукта следующему процессу;
- предсказание, предварительная оценка качества конечного продукта [2];
- сбор информации о промежуточных продуктах с целью контроля и управления процессом разработки ПС [3].

Целью оценивания качества конечного продукта может быть:

- принятие решения о приемке продукта;
- принятие решения о сроках выпуска продукта;
- сравнение продукта с другими продуктами;
- выбор продукта из множества альтернативных продуктов;
- оценка как положительного, так и отрицательного результата использования продукта.

Выполнение процесса оценки продуктов ПС, отвечающего требованиям ISO/IEC 12207, регламентируется стандартом ISO/IEC 14598:1999 [4].

Стандарт описывает процесс оценивания в виде пошаговой процедуры, ориентированной на использование *обобщенной* модели качества, представленной в стандарте ISO/IEC 9126 (рисунок 12.1).

Рис. 12.1. **Структура процесса оценивания продуктов ПС**

Процесс оценивания устанавливает требования к *методам* измерения и оценивания *любых типов* продуктов и может интегрироваться с процессами разработки (для оценивания продуктов их *разработчиками*), процессами приобретения-поставки (для оценивания продуктов *заказчиками (потребителями)*), а также с любыми другими процессами, в которых возникает необходимость непредвзятой оценки (для оценивания продуктов *независимыми оценщиками*).

Хотя общая схема оценки продуктов (структура процесса) сохраняется, существуют особенности выполнения действий в процессе оценивания отдельными категориями исполнителей (связанные с разными целями оценивания, видами оцениваемых продуктов, критериями, метриками и др.).

Поэтому требования к оцениванию продуктов разработчиками, заказчиками (и потребителями продуктов «с полки») и независимыми оценщиками содержаться в отдельных частях стандарта, - соответственно, в части 3 [5], части 4 [6] и части 5 [7].

Наряду с дифференциацией требований к оцениванию по целям оценивания (назначению оценивания), может возникнуть необходимость выделения *уровней*

оценивания различных характеристик качества, в зависимости от уровней целостности оцениваемой ПС (и ее компонентов) и риска не достичь необходимого качества [8]. Чем выше установленный уровень целостности ПС, тем выше требования к уровню значений определенных характеристик качества и тем строже должны быть методы оценивания качества (а также выше трудоемкость оценивания).

Стандарт рекомендует определять уровень оценивания (от A – высшего до D - низшего) с учетом риска в сфере *безопасности функционирования* (угрозы жизни людей), *защиты информации* (угрозы потери информации), *экономики* (угрозы финансовых убытков), *среды функционирования* (угрозы разрушения (не восстановления) среды эксплуатации ПС).

В зависимости от уровня оценивания, установленного для каждой из характеристик качества продукта, выбирается метод (условия) измерения и оценивания. Так, например, для характеристики «функциональность» в требованиях к оцениванию может указываться одно из следующих *условий* сбора данных (в порядке возрастания строгости требований):

- сбор данных (метрик) в ходе функционального тестирования;
- сбор данных (метрик) в ходе инспекции документов разработки;
- сбор данных (метрик) в ходе автономного тестирования при условии обеспечения полноты покрытия.

Уровень целостности ПС может обусловить не только выбор уровня оценивания отдельных характеристик качества продукта, но и расстановку *приоритетов важности* тех или иных характеристик его качества и выбор внешних метрик качества (как, например, в таблице 12.1 из [8], где характеристики упорядочены по убыванию важности).

Упорядочивая характеристики (подхарактеристики) качества по важности можно также устанавливать для каждой из них *коэффициент весомости*, что позволит при ограниченных ресурсах разработки уделять больше внимание тем характеристикам, которые имеют наибольший вес.

Определение коэффициентов весомости каждой характеристики обычно проводится экспертным путем по определенной шкале. Стандарт ДСТУ 2850-94, например, рекомендует порядковую шкалу из 5 значений: 5 – «крайне важно» (высокое значение характеристики), 4 – «очень важно», 3 – «важно», 2 – «хорошо бы», 1 – «неважно» [9], а В.В. Липаев – шкалу из трех значений – «высокий», «средний», «низкий» (приоритет) [10].

Если сумма коэффициентов весомости равна 1, – эти коэффициенты называют *параметрами весомости*.

Наиболее строгая схема присваивания параметров весомости характеристикам дана в Межгосударственном стандарте ГОСТ 28195-99 [11]. Стандарт определяет *конкретные значения* параметров весомости для каждой подхарактеристики, в зависимости от *стадии* ЖЦ ПС, а также от *класса ПС* (указываются коды подклассов ПС по ОКП)[1].

---

[1] Модель качества в стандарте ГОСТ 28195 немного отличается от предлагаемой ISO/IEC 9126, а также ДСТУ 2850. Эти отличия касаются состава подхарактеристик.

Таблица 12.1. **Приоритеты характеристик с учетом уровня целостности ПС**

Уровень целостности	Характеристики качества	Наиболее важная подхарактеристика	Выбранная внешняя метрика	Возможный критерий приемки
Низкий	1. Функциональность	Точность	Число полученных точных результатов в % от ожидаемого числа	95%
	2. Удобство использования	Управляемость	Число ясных сообщений в % от общего числа просмотренных сообщений	80%
	3. Переносимость	Настраиваемость	Число модулей, подлежащих повторной компиляции, с учетом общего числа модулей, переносимых на новую платформу	< 6 модулей
	4. Эффективность	Реактивность	Промежуток времени от посылки запроса до получения ответа системы	< 5 секунд
	5. Надежность	Отказоустойчивость	Число отказов, произошедших из-за ошибки во входных данных, в % от общего числа запусков системы с вводом неправильных данных	25%
	6. Сопровождаемость	Не требуется	Не требуется	-
Высокий	1. Надежность	Готовность	Среднее время между отказами в период эксплуатации	> 6 месяцев
	2. Функциональность	Функциональная полнота	Число реализованных обязательных функциональных требований в % к общему числу специфицированных функциональных требований	100%
	3. Сопровождаемость	Модернизируемость	Число модулей, подлежащих изменению при возможных изменениях ПС	1
	4. Эффективность	Используемость ресурсов	% занятости CPU в определенный период работы системы в расчете на наихудшие операционные условия	80%
	5. Удобство использования	Понятность	Время, необходимое определенной категории пользователей, на то, чтобы понять, каким образом использовать ПС, чтобы достичь нужных результатов	< 10 минут
	6. Переносимость	Не требуется	Не требуется	-

Помимо правильного выбора уровня оценивания, успех оценивания программного продукта зависит от адекватности множества метрик и ассоциируемых с ними методов измерения и оценивания *потребностям* оценивания на выбранном уровне. Примеры взаимосвязанных метрик (внутренних, внешних и эксплуатационных) для каждой характеристики качества в *обобщенной модели* качества представлены в стандартах серии ISO/IEC 9126 [12, 13, 14]. Перечень и краткая характеристика методов, которые могут использоваться для определения значений метрик, были даны в главе 3. Практические рекомендации относительно оценивания

всех характеристик качества ПС можно найти в книге В.В.Липаева [10], а также в ГОСТ 28195-99.

12.1.2. Технологические модули оценивания

Для того чтобы грамотно применять метрики в конкретных проектах, а также разрабатывать новые метрики, понятные, согласованные и однозначно интерпретируемые исполнителями измерений и оценщиками, лучше всего руководствоваться едиными требованиями к их документированию и использованию, не допускающими разночтения. С этой целью разработчики стандарта ISO/IEC 14598 определили понятие *модуля оценивания* [15].

Стандарт ISO/IEC 14598 предлагает структурировать всю информацию, необходимую для проведения и интерпретации результатов оценивания *одной характеристики* качества[2] в виде *совокупности элементов*, каждый из которых охватывает один аспект оценивания качества и обусловливает применение одного определенного метода. Вся информация, связанная с выполнением оценивания одним методом, должна собираться и сохраняться (пакетироваться) для использования в будущем. Такой пакет информации называется модулем оценивания. Таким образом, модуль оценивания – вместилище всей информации, необходимой для выполнения оценивания определенного аспекта одной характеристики качества с применением единой методологии (включающей методы, процедуры, условия, инструменты, точность измерения и оценивания). Он обеспечивает связность метрик, методов оценивания и мер (результатов оценивания).

Модуль оценивания включает 6 обязательных частей EM0 – EM5 (EM – от Evaluation Module), а также одно необязательное приложение – EMA (от Evaluation Module Annex):

EM0 содержит общую информацию о модуле оценивания и описание используемого *метода оценивания*;

EM1 определяет *сферу применения* модуля оценивания;

EM2 указывает перечень *ссылок* на литературные источники;

EM3 содержит *определения*, используемые в модуле оценивания;

EM4 идентифицирует *входные продукты*, требуемые для проведения оценивания, и определяет собираемые данные и вычисляемые метрики;

EM5 содержит сведения о том, как нужно *интерпретировать результаты* измерений.

Приложение EMA может включать описание *подробной процедуры* применения модуля оценивания.

Требования к документированию всех частей модуля оценивания определены в части 6 стандарта ISO/IEC 14598 [15]. Там же можно найти примеры трех модулей оценивания – для *метрики* «плотность дефектов в ПО», *характеристики качества* «функциональность», а также *обобщенной характеристики* «качество при использовании».

---

[2] Модули оценивания могут разрабатываться не только для характеристик, но и для отдельных подхарактеристик или даже метрик.

12.1.3. Оценка интегрального показателя качества

В том случае, если цель оценивания качества ПС состоит в определении *общего уровня* качества ПС, а не отдельных характеристик (подхарактеристик), - вычисляется интегральный показатель качества.

Поскольку все подхарактеристики качества имеют различные метрики (методы и шкалы измерений), они не могут непосредственно сопоставляться или объединяться. Механизм их объединения состоит в том, что для всех элементов на всех уровнях модели качества принимается единая шкала оценки – от 0 до 1.

Интегральный показатель качества определяется как функция (средневзвешенное арифметическое среднее) от нескольких характеристик (подхарактеристик) качества и коэффициентов их весомости:

$$U = \sum_{i=1}^{n} (Q_i \cdot M_i)$$

где Q_i - относительное значение i-й характеристики качества, а M_i - параметр весомости. n – количество характеристик качества.

Относительное значение характеристики – это отношение ее вычисленного (фактического) значения к базовому (эталонному) значению.

Эталонное значение соответствует значению данной характеристики у *лучшего* образца ПС из числа *аналогов* (к аналогам относятся ранее созданные ПС того же класса, что и оцениваемая ПС, подобные по функциональному назначению и условиям применения, отвечающие современному уровню развития систем данного класса).

Если модель качества многоуровневая – результат оценки показателя качества, определяется результатом оценки характеристик, результат оценки характеристик – результатом оценки подхарактеристик и т.д. *снизу вверх* по уровням модели качества.

12.2. Оценивание процессов жизненного цикла

12.2.1. Эталонная модель оценивания

Оценивание процессов ЖЦ может понабиться для того, чтобы:

- осознать состояния собственных процессов организации-разработчика с целью их дальнейшего *усовершенствования*;
- установить соответствие собственных процессов организации определенным *требованиям* или классу требований (например, для проверки способности выполнить договор или оценки риска, связанного с невыполнением договора);
- определить пригодность процессов *другой* организации для конкретного договора.

Оценивание процессов ЖЦ в настоящее время регламентируется стандартом ISO/IEC 15504, принятым в 1998 году[3]. Этот стандарт – результат работы специа-

[3] Стандарт состоит из 9 частей. «Центральная» часть – 2 – содержит определение эталонной модели процессов (40 процессов). В новой редакции (которая готовится подкомитетом SC7) стандарт будет включать только 5 частей. Из стандарта изымается описание эталонной модели процессов ЖЦ, которое будет представлено только в ISO/IEC 12207.

листов по программной инженерии над проектом SPICE (Software Process Improvement and Capability dEtermination) [16].

Стандарт предлагает структурный подход к оцениванию, в основу которого положена *двумерная модель оценивания* [17]. Прежде всего, устанавливается, соответствует ли оцениваемый процесс определенным *требованиям*, зафиксированным в эталонной модели этого процесса, а затем определяется, *насколько четко* он организован, устойчив и управляем[4].

Эталоном процесса ЖЦ служит его определение в стандарте ISO/IEC 12207 (см. главу 1) [1]. Каждый процесс в эталонной модели описывается в виде формулировки *цели* (предназначения) процесса и перечня утверждений, констатирующих отличительные особенности успешного осуществляемого процесса (иначе говоря, показатели его результативности). По тому, обладает ли процесс этими особенностями, можно судить о том, что процесс действительно осуществляется, то есть выполняет действия, которые считаются нормой.

Измерение процессов в проекте (по их действиям и конкретным наработкам) образует первую меру в двумерной модели оценивания. Она используется для подтверждения того, что оцениваемый процесс существует и достигает результата. Однако факт достижения процессом конечного результата еще не свидетельствует о том, что он не пущен на «самотек».

Мера, используемая для измерения свойств (атрибутов) процесса, определяющих мощность управляющей составляющей процесса – руководящей практики - это вторая мера в двумерной модели оценивания процессов. Ее называют *мерой мощности* процесса, характеризующей степень, в которой руководящую практику можно считать совершенной.

В стандарте ISO/IEC 15504 выделены 9 атрибутов мощности процесса на 6 уровнях мощности (зрелости) процесса.

Краткая характеристика уровней мощности (зрелости) процесса:

- ***уровень 0. Незавершенный процесс.*** Происходит провал при выполнении процесса или сбой в достижении его целей - нет или очень мало рабочих продуктов или результатов процесса, свидетельствующих о том, что процесс выполняется.
- ***уровень 1. Выполняемый процесс.*** Назначение процесса в целом достигается. Сотрудники организации признают, что деятельность (соответствующую назначению процесса) нужно осуществлять. Есть общая договоренность о том, что эта деятельность осуществляется так, как нужно, и тогда, когда нужно. Рабочие продукты процесса идентифицированы, и по ним можно судить о достижении его целей. Результаты процесса могут не быть заранее строго запланированы;
- ***уровень 2. Управляемый процесс.*** Рабочие продукты производятся в соответствии с установленными процедурами. Процесс планируется и контролируется. Рабочие продукты согласованы с определенными стандартами и требованиями.

Основное отличие от уровня *выполняемого процесса* состоит в том, что ход процесса теперь *приводит* к выпуску рабочих продуктов, полностью отвечающих требованиям к качеству в пределах заданных сроков и ресурсов;

---

[4] Стандарт не предназначен для использования в какой-либо системе сертификации/регистрации мощности процессов ЖЦ в организации.

- ***уровень 3. Установившийся процесс.*** Существует базовое определение процесса, разработанное с учетом ведущих принципов и передовой практики программной инженерии и обеспечивающее достижение хороших результатов при его надлежащем использовании. Базовый процесс институциализируется (внедряется) в организации. Далее он адаптируется к условиям определенного проекта («настраивается» на конкретные рабочие продукты, сроки и т.д.). Для реализации адаптированного процесса, представленного своим определением, выделяются все необходимые ресурсы.

Основное отличие от уровня *управляемого процесса* состоит в том, что процесс на уровне установившегося процесса использует базовый процесс как такой, который действительно *способен достичь* результатов, свойственных базовому процессу, и служит *гарантией* достижения результатов;

- ***уровень 4. Предсказуемый процесс.*** Выполнение процесса на практике происходит в условиях постоянного количественного контроля достижения целей базового процесса. Продукты и ресурсы процесса детально измеряются, а результаты измерений собираются и анализируются. Это позволяет эффективно руководить процессом, предсказывать его состояние в ходе ЖЦ, а также оценивать качество продуктов в количественном исчислении.

Основное отличие от уровня *установившегося процесса* состоит в том, что адаптированный процесс выполняется постоянно и всегда можно предсказать, на каком этапе выполнения он будет находиться в определенный момент времени;

- ***уровень 5. Оптимизируемый процесс.*** Выполнение процесса оптимизируется в соответствии с текущими и будущими производственными целями организации. Существуют количественные целевые показатели экономической эффективности и производительности выполнения процесса. Установлена обратная связь в процессе, благодаря которой осуществляется постоянный мониторинг соответствия процесса целям организации и его улучшение. Оптимизируемый процесс предполагает решение задач пилотирования (апробации новых идей и технологий) и модернизации (изменения) неэффективных действий для достижения определенных целей или показателей.

Основное отличие от уровня *предсказуемого процесса* состоит в том, что не только действующий адаптированный процесс, но и базовый процесс организации динамично меняются и улучшаются с целью эффективного достижения текущих и будущих производственных целей.

Эволюция мощности любого процесса отражается в его свойствах (атрибутах).

Атрибут процесса описывает некоторый аспект общей возможности управления и совершенствования процесса.

Определенное множество атрибутов характеризует уровень мощности процесса, причем каждый последующий уровень мощности включает не только атрибуты, свойственные процессу на данном уровне, но и атрибуты процесса для всех предыдущих уровней (таблица 12.2). Совместно эти атрибуты указывают на основной прирост мощности управления процессом.

Самый первый атрибут АП 1.1 – выполнимость процесса – характеризует не столько мощность процесса, сколько его наличие, а все остальные – степень управления процессом по различным аспектам его выполнения.

Таблица 12.2. **Атрибуты мощности процесса**

Уровень мощности	Атрибут	Определение атрибута
1. Выполняемый процесс	**АП 1.1** Выполнимость процесса	Степень, в которой процесс *достигает результатов* процесса путем преобразования идентифицируемых входных рабочих продуктов в идентифицируемые выходные рабочие продукты.
2. Управляемый процесс	**АП 2.1** Управляемость выполнением	Степень, в которой выполнение процесса *направляется* на выработку рабочих продуктов, соответствующих установленным целевым показателям процесса *(параметрам управления)*.
	АП 2.2 Управляемость рабочими продуктами	Степень, в которой выполнение процесса направляется на выработку *таких* рабочих продуктов, которые должным образом документированы и верифицированы.
3. Установившийся процесс	**АП 3.1** Определенность	Степень, в которой выполнение процесса *использует определение процесса* (основанное на базовом) для достижения результатов процесса.
	АП 3.2 Используемость ресурсов	Степень, в которой *процесс использует имеющиеся в наличии ресурсы* (например, трудовые ресурсы и инфраструктуру процесса), выделенные для развертывания процесса.
4. Предсказуемый процесс	**АП 4.1** Измеримость	Степень, в которой цели и меры продукта и процесса *используются* для того, чтобы гарантировать, что выполнение процесса способствует достижению поставленных перед ним целей.
	АП 4.2 Контролируемость выполнения	Степень, к которой *процесс контролируется* (благодаря сбору, анализу и использованию мер продукта и процесса), что обеспечивает корректировку его выполнения для достижения определенных целей (относительно продукта и процесса).
5. Оптимизируемый процесс	**АП 5.1** Контролируемость модификации	Степень, в которой *изменения* в определении, управлении и выполнении процесса *находятся под постоянным контролем* в контексте достижения соответствующих производственных целей организации.
	АП 5.2 Непрерывность усовершенствования	Степень, в которой изменения в процессе идентифицируются и внедряются таким образом, что есть *гарантия непрерывного совершенствования*, способствующего достижению соответствующих производственных целей организации.

Для каждого атрибута существуют рейтинги достижения – «полностью», «существенно», «частично», «не достигнут» - в определенном *контексте* достижения конкретной поставленной цели оценивания (рисунок 12.2).

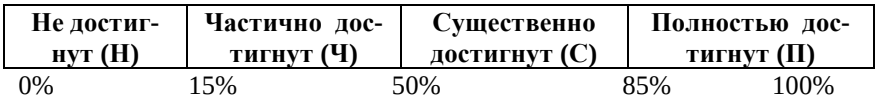


Рис. 12.2. **Эталонная шкала рейтингов атрибутов процесса**

Нужно заметить, что оценивание процессов всегда должно проводиться с учетом определенного *контекста* оценивания – ведь для одних целей оцениваемые процессы могут быть пригодны, для других – нет. Описанная здесь двумерная модель оценивания является эталонной моделью и не может быть непосредственно применена для оценивания, поскольку не существует «универсального» контекста оценивания (не ясно, по каким именно показателям проводить оценивание).

12.2.2. Совместимая модель оценивания

Перед выполнением оценивания должна быть подобрана или разработана уточненная модель оценивания, *совместимая с эталонной моделью* (рисунок 12.3).

Совместимая модель должна удовлетворять следующим требованиям.

Во-первых, она должна охватывать, по крайней мере, те процессы, которые *нужно* оценивать (один или несколько).

Во-вторых, в ней должны быть подробно описаны конкретные практические приемы, которые обеспечивают достижение *назначения процесса* и указанных в эталонной модели *результатов процесса*. Они называются *базовыми практическими приемами*.

В-третьих, должны быть четко определены *рабочие продукты*, которые существуют на входе процесса или появляются на его выходе.

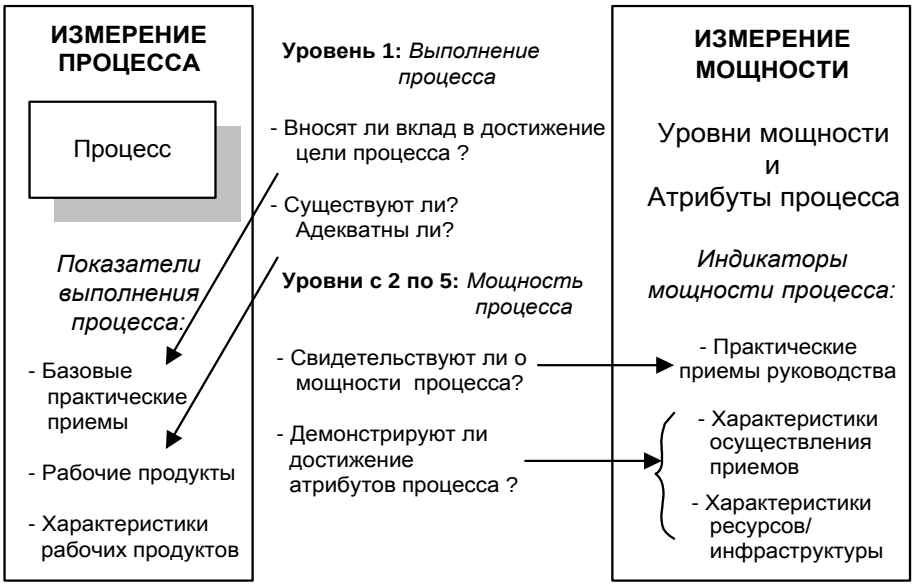


Рис. 12.3. **Совместимая модель оценивания**

В-четвертых, в описании этих рабочих продуктов должны указываться *характеристики*, по которым можно будет оценивать каждый рабочий продукт.

Кроме того, в этой же совместимой модели должны быть уточнены описания ***атрибутов процесса***, удовлетворяющие следующим требованиям.

Во-первых, должны быть указаны конкретные *практические приемы руководства* процессом.

Во-вторых, для каждого приема руководства должны указываться *характеристики его осуществления* (по которым можно судить о наличии руководства),

характеристики ресурсов и инфраструктуры (применяемые методы, инструменты), а также те процессы, с которыми может ассоциироваться данный прием руководства.

12.2.3. Требования к оцениванию

Оценивание выполняет группа оценщиков во главе с ответственным оценщиком. Группа пытается соотнести все, что делается по процессам, с тем, что определено в совместимой модели, найти объективные данные относительно осуществления базовых практических приемов и приемов руководства, и в результате выработать единое мнение о процессе.

Во входных данных для оценивания должно указываться следующее:

- личность инициатора оценивания;
- назначение оценивания;
- *сфера оценивания*, охватывающая:
  - *процессы*, подлежащие исследованию;
  - организационное подразделение, в котором протекают эти процессы;
  - *контекст (условия) применения процесса* - размер и организационная структура подразделения, предметная область, объём, критичность и сложность продуктов или услуг, показатели качества продуктов.

  Только учитывая контекст применения процесса, зафиксированный на входе оценивания, оценщик сможет правильно оценить атрибуты и выставить их рейтинги для конкретного процесса. Если, например, «процесс управления проектом» выполняется в коллективе разработчиков из 5 человек, у оценщика сформируется одно мнение относительно мощности этого процесса, а если - из 50 человек – другое;
- *ограничения при оценивании*, касающиеся:
  - доступных ключевых ресурсов (документов, которые можно читать, людей, у которых можно брать интервью);
  - максимального объёма времени, используемого при оценивании;
  - особых процессов, не подлежащих оцениванию;
  - прав собственности на результаты (продукты) оценивания и любых ограничений по их использованию;
  - особенностей управления информацией с учетом конфиденциальности и др.
- идентификация *совместимой модели*;
- *личности оценщиков*, уровень их компетентности и ответственности;
- *идентификация объектов оценивания* и многое другое.

12.2.4. Этапы процесса оценивания

В стандарте ISO/IEC 12207 организационный процесс «оценивание процесса» определен как компонент процесса «усовершенствование процесса». Его назначение «состоит в том, чтобы определить степень, в которой базовые процессы ЖЦ в организации вносят вклад в достижение ее производственных целей, и помочь организации сосредоточиться на проблеме непрерывного совершенствования процесса. В результате успешного осуществления процесса:

- появится эффективный метод оценивания процесса, предназначенный для определения способности организации и производственных процессов в ней вырабатывать продукты и услуги, отвечающие целям;
- станут понятными относительные достоинства и недостатки базовых процессов ЖЦ организации;
- будут храниться и сопровождаться тщательно подготовленные и доступные для использования учетно-отчетные документы оценивания;
- проверки базовых процессов организации будут проводиться через надлежащие промежутки времени с целью обеспечения их постоянной пригодности и эффективности в свете результатов оценивания» [1].

В соответствии с ISO/IEC 15504-3 процесс оценивания должен включать, по крайней мере, следующие этапы и виды деятельности [18].

Планирование. Для проведения оценивания разрабатывается и документируется план, определяющий требуемые входные данные, выполняемые действия при проведении оценивания, ресурсы и время, выделяемое для выполнения этих действий, состав и обязанности оценщиков, критерии для проверки выполнения требований к оцениванию, описание продуктов, оценивание которых запланировано.

Независимо от того, как будут дальше употреблены результаты оценивания, изначально устанавливается *потребность в оценивании* тех или иных процессов. Оцениваются только те процессы, которые безусловно вносят вклад в достижение определенной поставленной цели. С помощью совместимой модели оценивания устанавливается соответствие между оцениваемыми процессами и процессами, определёнными в эталонной модели процессов в стандарте ISO/IEC 12207. Для этих процессов далее выбираются атрибуты (важные аспекты руководства этими процессами), достижение которых сможет оказать существенное влияние на выполнение процессов. Указывается *целевой рейтинг* каждого атрибута (достаточный, для того чтобы достичь цели) (как, например, на рисунке 12.4). Обычно рейтинги ниже «существенного» не устанавливаются. Просто атрибуты не оцениваются.

Процесс _____		**Уровни мощности процесса**				
Атрибуты процесса		**1**	**2**	**3**	**4**	**5**
АП 1.1	Выполнимость процесса	С/П	П	П	П	П
АП 2.1	Управляемость выполнением		С/П	П	П	П
АП 2.2	Управляемость рабочими продуктами		С/П	П	П	П
АП 3.1	Определенность			С/П	П	П
АП 3.2	Используемость ресурсов			С/П	П	П
АП 4.1	Измеримость				С/П	П
АП 4.2	Контролируемость выполнения				С/П	П
АП 5.1	Контролируемость модификации					С/П
АП 5.2	Непрерывность усовершенствования					С/П
				Профиль		

Рис. 12.4. **Рейтинги уровня мощности**

Сбор данных. Каждый процесс, выбранный для оценивания, оценивается на основании объективных сведений о его атрибутах. Стратегия и методики сбора,

анализа данных и обоснования рейтингов должны быть точно идентифицированы и наглядны. Чтобы обеспечить основания для верификации рейтингов, объективные сведения, поддерживающие мнение оценщика о рейтингах атрибута процесса, протоколируются и сохраняются.

Валидация данных. Все собранные данные, необходимые для охвата сферы оценивания, проверяются и утверждаются.

Определение рейтинга процесса. На основании утвержденных данных каждому атрибуту процесса присваивается *оцененный рейтинг*. Множество рейтингов атрибута процесса протоколируется как *профиль мощности процесса* для определённого организационного подразделения (см. рисунок 12.4).

Если оцененный рейтинг совпадает с целевым - делается вывод о том, что по соответствующему аспекту руководства проблем с процессом быть не должно. Если есть расхождения – вывод о том, что существует риск не достичь цели процесса из-за нечеткости руководства процессом (рисунок 12.5).

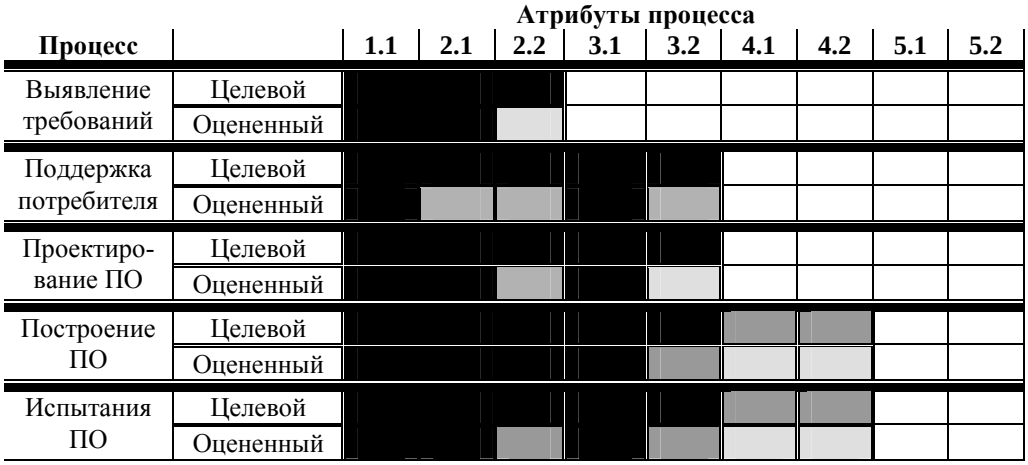


Рис. 12.5. **Целевая мощность вместе с оцененной мощностью**

Для того чтобы оценить расхождения по каждому атрибуту и по совокупности атрибутов, составляющих мощность процесса в целом, стандарт ISO/IEC 15504-3 предлагает соответствующие таблицы (таблица 12.3 и таблица 12.4).

Таблица 12.3. **Расхождение по атрибуту процесса**

Целевой рейтинг	**Оцененный рейтинг**	**Расхождение по атрибуту**
Полностью достигнут	Полностью достигнут	**Нет**
	Существенно достигнут	**Малое**
	Частично достигнут	**Большое**
	Не достигнут	**Большое**
Существенно достигнут	Полностью достигнут	**Нет**
	Существенно достигнут	**Нет**
	Частично достигнут	**Большое**
	Не достигнут	**Большое**

Таблица 12.4. **Расхождения на уровне мощности**

Количество расхождений атрибутов процесса	Расхождение на уровне мощности
Нет больших или малых расхождений	**Нет**
Только малые расхождения	**Незначительное**
Единственное, но большое расхождение на уровне мощности со 2 по 5	**Значительное**
Единственное, но большое расхождение на уровне мощности 1 или более одного большого расхождения на уровне мощности со 2 по 5	**Существенное**

Чем выше расхождение в целевой и оцененной мощности, тем больше *вероятность* того, что процесс неустойчив и может не соответствовать тем целям, для которых его хотят применить.

С другой стороны, чем на более низком уровне мощности находится процесс, и чем больше величина расхождения целевой и ожидаемой мощности, тем большими могут быть *потери*, связанные с тем, что процесс не будет эффективен в достижении тех целей, для которых он выбран (рисунок 12.6).

	Величина расхождения на уровне мощности (вероятность)			
Размещение расхождения на уровне мощности (влияние)	**Нет**	**Незначительная**	**Значительная**	**Существенная**
Оптимизируемый	Риск не обнаружен	Низкий риск	Низкий риск	Низкий риск
Предсказуемый	Риск не обнаружен	Низкий риск	Низкий риск	Средний риск
Установившийся	Риск не обнаружен	Низкий риск	Средний риск	Средний риск
Управляемый	Риск не обнаружен	Средний риск	Средний риск	Высокий риск
Выполняемый	Риск не обнаружен	Средний риск	Высокий риск	Высокий риск

Рис. 12.6. **Полный процессо-ориентированный риск**

Как видно на рисунке 12.6, если оцениваемый процесс находится на высоком уровне мощности (первая строка матрицы), небольшие отклонения в мощности не так уж страшны, риск невелик.

Составление отчёта. Отчет о результатах оценивания включает описание входных данных для оценивания, собранных объективных данных, использованного подхода к оцениванию, а также множество профилей мощности процессов (по одному для каждого оцененного процесса). По завершении процесса оценивания этот отчет направляется инициатору оценивания.

12.3. Оценивание зрелости организаций-разработчиков

12.3.1. Модели зрелости

Наряду с моделью оценивания и усовершенствования процессов разработки программных систем, предлагаемой стандартом ISO/IEC 15504, существует множество других моделей, совсем новых и относительно старых (начала 90-х годов), стандартизованных и не стандартизованных. Они разработаны в разных странах и в разной степени распространены. Степень их распространения (или предпочтения) можно определять по территориальному, ведомственному и другим признакам.

Многие сайты в Интернете предлагают свои *путеводители* по моделям. Один такой путеводитель представлен на рисунке 12.7. Это страничка «Frameworks Quagmire», разработанная фирмой Software Productivity Consortium (и представлена на сайте GDPA - http://www.tzi.de/~uniform/gdpa/director/products/jq002.htm).

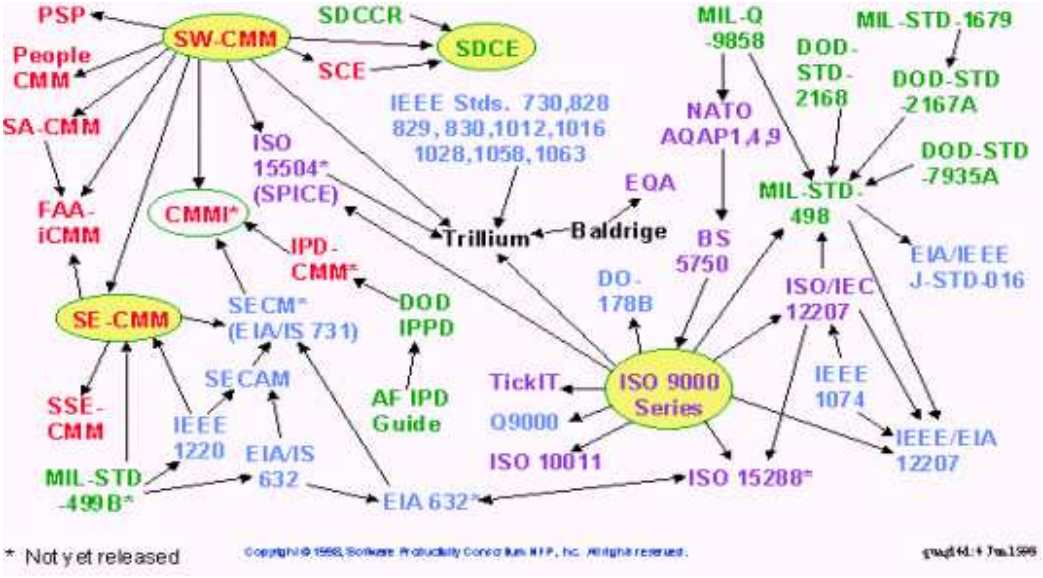


Рис. 12.7. **Путеводитель по моделям зрелости процессов разработки**

Нужно отметить, что в связи с выходом стандарта ISO/IEC 15504, который требует применения для оценивания моделей, *совместимых* с эталонной, разработчики многих моделей зрелости *продолжают* их развивать в направлении обеспечения необходимой совместимости.

Наиболее известным (и широко используемым не только в Америке, но и на других континентах) является ***семейство моделей зрелости CMMs*** (Capability Maturity Models), включающее:

- *SW-CMM* (CMM for Software)— модель зрелости процесса разработки программных продуктов,
- *SE-CMM* (System Engineering CMM) — модель зрелости процесса системной инженерии,
- *SSE-CMM* (System Security Engineering CMM) – модель зрелости процесса обеспечения безопасности системы,

- *SA-CMM* (Software Acquisition CMM) — модель зрелости процесса закупок ПО,
- *People CMM* — модель зрелости процесса управления кадрами,
- *IPD-CMM* (Integrated Product Development CMM) – модель зрелости разработки интегрированных продуктов,
- ряд других моделей, созданных при поддержке SEI и объединяющих идеи разных моделей SEI, например, модель *FAA-iCMM*, разработанная Federal Aviation Administration и охватывающая возможности SW-CMM и SE-CMM.

Однако, при таком обилии моделей, их практическое применение вызвало проблемы, связанные со следующими факторами:

- модели имели разную структуру, форматы представления информации, терминологию, способы измерения зрелости,
- происходила путаница при применении более одной модели и при переходе от одной модели к другой,
- сложно было определить интегральную оценку зрелости на базе оценок по отдельным моделям и, как следствие, сложно интерпретировать оценки и использовать их для построения программы улучшения процесса в организации,
- сложно использовать множество моделей для выбора поставщиков продукции (а это была первично основная цель разработки моделей CMM),
- модели не согласовывались с требованиями международных стандартов в области качества.

Поэтому в 2002 году SEI опубликовал новую *модель CMMI* (Capability Maturity Model Integration), объединяющую ранее выпущенные модели и учитывающую требования международных стандартов.

В этом разделе мы подробнее рассматриваем только первую (исторически) модель - модель *SW-CMM* (Capability Maturity Model for Software), предложенную У. Хамфри в 1987 году и опубликованную SEI в 1993 году [19], а также очень сжато характеризуем *Интеграцию моделей CMM* (CMMI).

Модель CMM уже была кратко описана в главе 1 (п.1.2.5). Напомним ее ***основные отличия от модели SPICE***, лежащей в основе стандарта ISO/IEC 15504.

Модель SPICE может использоваться для оценивания уровня мощности и совершенствования *любого одного или совокупности процессов ЖЦ*, выбираемых для оценивания с определенной целью. Фактически оцениваются *базовые практические приемы* процесса (характеристика их выполнения), *рабочие продукты* процесса (их наличие), *характеристики рабочих продуктов* (их наличие), а также *приемы руководства* (характеристика их выполнения) и *характеристики ресурсов и инфраструктуры* (их наличие). Результат оценивания – вектор - *профиль мощности процесса* (*уровень мощности* (от 0 до 5) и *рейтинги атрибутов мощности* на данном уровне мощности).

Модель CMM может использоваться для оценивания и совершенствования процесса *программной инженерии (в целом)* в организации (поэтому ее обычно и называют моделью *зрелости (совершенства) организации*, а не отдельных процессов ЖЦ).

12.3.2. Уровни зрелости процесса программной инженерии по CMM

CMM - это описательная модель в том смысле, что она описывает существенные (или ключевые) атрибуты, которыми должен обладать процесс в организации, находящейся на определенном уровне зрелости. В то же время CMM - нормативная модель, поскольку указывает конкретные практические приемы, которые должны применяться. CMM обеспечивает достаточный уровень абстракции и не накладывает ограничений на способы реализации процесса в организации.

В любом контексте применения CMM, должна существовать разумная интерпретация практических приемов. CMM нельзя считать предписывающей моделью, поскольку она дает ответ на вопрос, какими свойствами должен обладать процесс в организации, имеющей тот или иной уровень зрелости, но не говорит о том, какими средствами обеспечить улучшение процесса и достижение соответствующего уровня.

Модель CMM (здесь и далее имеется в виду SW-CMM) выделяет и дает строгое описание *18 ключевых направлений (областей, участков) процесса KPA (Key Process Areas)* программной инженерии (схожих по назначению с поддерживающими и организационными процессами ЖЦ в ISO/IEC 15504), которые *«распределены» по уровням* зрелости (от 2 до 5).

Для того чтобы организация достигла определенного уровня зрелости, она должна внедрить (институциализировать) соответствующее множество KPA и предоставить экспертам (имеющим права оценивания и владеющим методами экспертного оценивания) документальное подтверждение внедрения KPA в процесс программной инженерии. Результат оценивания – сертификат уровня зрелости и/или рекомендации по дальнейшему совершенствованию процесса.

Пять уровней зрелости CMM, ассоциированных с KPA, представлены на рисунке 12.8. Надпись на стрелке указывает уровень достигнутой мощности процесса, который официально утверждается организацией на каждой ступени модели зрелости. Названия уровней зрелости отражают сущность изменений в основном процессе программной инженерии.

Уровень зрелости определяет проблемы, которые *преобладают* на этом уровне. Например, на уровне 1 основная проблема касается управления, а остальные проблемы скрыты из-за сложности планирования и управления программными проектами.

Каждый уровень образует *фундамент* для эффективной реализации процессов на последующих уровнях. Пропуск уровней противоестественен.

Организации могут с успехом использовать (внедрять) направления процесса, описанные на вышележащих уровнях, находясь при этом на более низком уровне. Однако, направления, не отнесенные к, но применяемые на нижележащих уровнях, не могут в полной мере раскрыть свой потенциал, пока не будет создан соответствующий фундамент на нижних уровнях CMM.

Таким образом, CMM идентифицирует уровни, через которые организация *должна эволюционировать* для утверждения культуры программной инженерии.

Рис. 12.8. **Уровни зрелости в модели CMM**

Организации, находящиеся на 1 уровне и пытающиеся создать фиксированный процесс (уровень 3), не создав перед тем повторяемый процесс (уровень 2), обычно не достигают успеха, поскольку менеджеры проекта больше всего озабочены проблемами сроков и стоимости проекта. Это основная причина, по которой нужно сначала усовершенствовать процесс управления, а затем процессы собственно инженерии.

Может показаться, что определить и реализовать процесс инженерии легче, чем процесс управления (особенно с точки зрения разработчика), но без дисциплины управления процесс инженерии неминуемо скатится к проблемам сроков и стоимости. Способность организации осуществлять деятельность по направлениям, ассоциированным с высшими уровнями зрелости, не дает ей права «переступать» через уровни зрелости.

KPA сгруппированы по уровням зрелости таким образом, что каждое KPA всегда относится *только к одному* уровню CMM. Хотя в организации, находящейся на определенном уровне зрелости, могут выполняться процедуры в рамках направлений, относящихся к другим уровням зрелости, - заключение о том, какой

уровень зрелости занимает организация, делается только по KPA, *соответствующим* данному уровню.

Пути достижения целей KPA могут быть различны в разных проектах из-за различий в проблемных областях или средах. Несмотря на это все *цели направления* должны быть достигнуты для того, чтобы организация преуспела по данному направлению. Когда цели направления достигаются *безусловно* по всем проектам в организации, можно считать, что это направление имеет статус официально утвержденного, а соответствующие процедуры процесса *внедрены* во все проекты организации.

Приставка «ключевой» говорит о том, что существуют и другие направления процесса, которые хотя и влияют на его результативность, но не являются ключевыми для достижения уровня зрелости. Ключевые направления определены SEI по результатам многолетнего опыта программной инженерии и управления и пятилетнего опыта аналитического и экспертного оценивания процесса программной инженерии.

Каждый уровень зрелости, за исключением первого, может быть декомпозирован на составные части (рисунок 12.9).

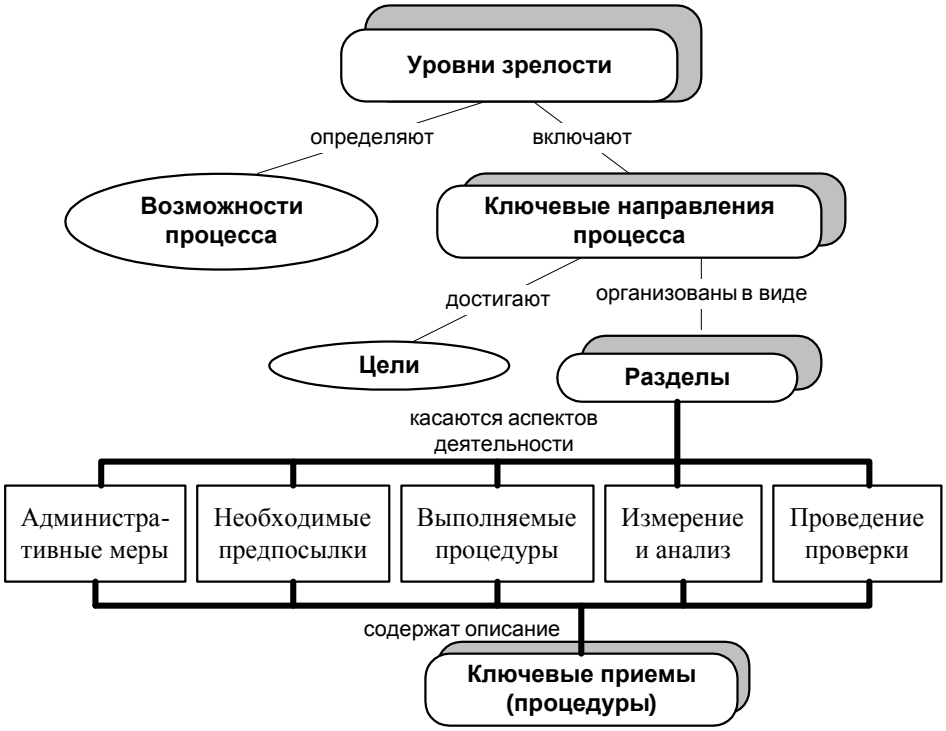


Рис. 12.9. **Структура уровня зрелости в модели CMM**

Каждое направление представлено пятью *разделами*, а каждый раздел определяет перечень рекомендуемых *практических действий* (приемов, процедур). При выполнении *всех* рекомендуемых действий достигаются *цели*, которые являются приоритетными для соответствующего ключевого направления процесса (считаются важными для расширения возможностей процесса).

Раздел KPA касается одного аспекта проблем, связанных с выполнением соответствующего участка процесса. В CMM выделены следующие пять разделов KPA [20]:

- *административные меры*. В этом разделе описаны действия, которые должна предпринять организация, чтобы обеспечить пуск процесса и его стабильность. Административные меры обычно касаются формирования политики и обеспечения финансовой поддержки;
- *необходимые предпосылки*. В этом разделе описаны условия, которые должны быть созданы в рамках организации или проекта для обеспечения готовности процесса (необходимые ресурсы, организационные структуры и система обучения);
- *выполняемые процедуры*. В этом разделе описаны правила и процедуры, которые необходимо соблюдать для успешной реализации соответствующего участка процесса. К числу таких процедур обычно относят разработку планов и процедур, выполнение технологических операций, а также действия по проверке (трассировке) и необходимой корректировке;
- *измерение и анализ*. В разделе описаны требования к проведению измерений в ходе процесса и анализа полученных результатов измерений, а также приведены примеры обычно собираемых данных (показателей), необходимых для определения состояния и эффективности процесса. Ключевые процедуры этого раздела описывают основные приемы измерений, необходимых для определения состояния работ по ключевым процедурам, представленным в разделе «выполняемые процедуры». Примеры предлагаемых метрик приводятся в качестве дополнительной информации, поскольку в разных средах проектов могут требоваться разные метрики и подходы к измерению;
- *проведение проверки*. В разделе описаны меры, предпринимаемые для проверки соответствия выполняемых действий требованиям существующего процесса. К методам проверки обычно относят обзоры и аудиторские проверки (ревизии) в ходе управления и обеспечения гарантии качества ПС. В этот раздел входят ключевые процедуры, касающиеся контроля со стороны руководства организации и руководства проекта, а также каких либо действий по проверке надлежащего выполнения ключевых процедур со стороны группы качества или других групп.

Только те практические действия, которые приводятся в разделе «выполняемые процедуры», непосредственно ассоциируются с представлением о практических возможностях процесса. Действия же, перечисляемые во всех остальных разделах, в целом образуют основу для их проведения в жизнь (внедрения).

Описание каждой п*роцедуры (или приема)* содержится в одном предложении текста. Описанные таким образом процедуры называют также основополагающими (ключевыми) процедурами самого верхнего уровня, составляющими фундамент политики и практики по соответствующему ключевому направлению. Процедуры предписывают «что» должно быть сделано для достижения целей, и не касаются того, «как» это должно быть сделано. Описание каждой ключевой процедуры раздела (успешность выполнения которой оценивается) может сопровождаться перечнем вспомогательных процедур, анализ которых поможет определить, выполнена ли ключевая процедура, а также дополнительной информацией, включающей примеры и ссылки на другие KPA.

Предписанные CMM ключевые процедуры процесса не предъявляют каких-либо требований к модели ЖЦ ПС, организационной структуре, распределению обязанностей и ответственностей, подходам к управлению и разработке ПС. Они акцентируют внимание на описании *существенных элементов* эффективного процесса.

12.3.3. Методы оценивания зрелости по CMM

CMM предлагает *критерии*, позволяющие оценить зрелость организаций-разработчиков. Эти критерии могут использоваться организациями-разработчиками для улучшения процессов разработки и сопровождения ПС, а также государственными и коммерческими организациями-заказчиками для оценки рисков заключения договоров на разработку программных проектов с определенными организациями-исполнителями.

На базе CMM SEI разработал 2 метода оценивания зрелости процесса:

- ***метод SPA*** *(*от *Software Process Assessment) - оценивание текущего состояния процесса*. Используется для обследования процесса программной инженерии в организации, определения его текущего состояния, выявления существующих проблем, выбора высокоприоритетных целей улучшения процесса разработки, выработки соответствующей стратегии улучшения и получения поддержки со стороны руководства [21];
- ***метод SCE*** *(*от *Software Capability Evaluation) - оценка способностей организации-разработчика*. Используется для идентификации риска заказчика, связанного с определенным проектом или контрактом с организацией-исполнителем на разработку высококачественного ПО в соответствии с установленными сроками и бюджетом. Может использоваться при определении потенциальных организаций-исполнителей программных проектов или для управления эффективностью процесса в организациях-исполнителях, располагающих определенными ресурсами разработки [22].

Методы SPA и SCE отличаются мотивацией, целями, структурой результирующих данных и способами интерпретации результатов. А это, в свою очередь, определяет применяемые процедуры оценивания, условия проведения обследования, динамику интервьюирования, спектр задаваемых вопросов, характер и объем собираемой информации, а также принципы подготовки специалистов для групп оценивания.

Обследование методом SPA с целью улучшения процесса в организации выполняется регулярно (с периодичностью 18 - 36 месяцев) в условиях открытости и сотрудничества с руководством и коллективом разработчиков.

Оценивание методом SCE выполняется в условиях, приближенных к условиям проведения ревизий. Рекомендации экспертов помогают выбрать наиболее надежных исполнителей проектов.

Основные шаги выполнения оценок по CMM методами SPA и SCE.

Шаг 1. Выбор группы экспертов, обученных основам CMM и специфике методов оценивания текущего состояния или потенциальных возможностей организации. Члены группы должны быть профессионалами в программной инженерии и менеджменте.

Шаг 2. Получение от оцениваемой организации ответов на вопросы *контрольного вопросника*, который будет использоваться при проведении оценивания (приложение 7) [23].

Шаг 3. Анализ ответов и идентификация тех участков процесса, которые требуют *дальнейшего* обследования. Эти участки соответствуют KPA в модели CMM.

Шаг 4. Посещение организации. Его цель - произвести *интервьюирование разработчиков и обзоры документации* и сопоставить полученные результаты с результатами анализа по вопроснику. Руководящими материалами в этом процессе служат описание KPA и практических приемов CMM. В своей работе группа использует методы проведения экспертизы, что дает ей возможность оценить, в какой мере KPA удовлетворяют целям процесса по каждому направлению. В том случае, если обнаруживаются расхождения между ключевыми процедурами CMM и действующей практикой в организации, - группа должна документировать обоснование своих решений по каждому направлению.

Шаг 5. По завершении работы в организации группа формирует *перечень «находок»* (обнаруженных отклонений), которые идентифицируют сильные и слабые стороны процесса в организации. Если целью работы группы является оценивание текущего состояния и возможностей улучшения процесса в организации - она дает руководству организации соответствующие рекомендации, если же цель - оценка способности организации выполнять контракты на разработку, - «находки» используются для анализа риска, проводимого соответствующей инстанцией.

Шаг 6. Группа готовит *отчет*, в котором в разрезе KPA показывает, по каким направлениям и в какой степени организация достигает или не достигает целей KPA. Цели могут считаться достигнуты и в том случае, когда отмечены отдельные недочеты, но они не касаются основных решений, по которым оценивается достижимость целей.

Для получения достоверной информации о ходе выполнения плана мероприятий по улучшению процесса в промежутках времени *между* обследованиями по методу SPA, институтом SEI был предложен ***метод IP*** (от Interim Profile) - метод *быстрой промежуточной оценки* состояния процесса по контрольному вопроснику с минимальным привлечением дополнительной информации со стороны исполнителей проектов [24]. Условием применения этого метода является предварительная оценка по методу SPA и наличие официально утвержденного плана мероприятий по улучшению процесса в организации.

12.3.4. Иерархия оценок зрелости процесса по модели CMM

В общем случае, оцениванию подлежат (в приведенной последовательности):

- ключевые процедуры (если их оценка предусмотрена в плане работ по SPA или SCE);
- разделы (если их оценка предусмотрена в плане работ по SPA или SCE);
- цели ключевого направления (всегда);
- ключевые направления уровня (всегда);
- уровень зрелости (если целью оценивания является определение уровня зрелости).

Цель определенного KPA считается достигнутой (оценка «удовлетворительно»), если в результате обследования процесса обнаруживается, что все ключевые

процедуры по всем разделам направления процесса определены, реализованы практически и внедрены *во все проекты* организации. Оценка «не удовлетворительно» присваивается в том случае, если отмечены существенные недостатки в реализации и внедрении оцениваемых элементов СММ. Каждый метод оценивания может предлагать расширенную шкалу ранжирования, учитывающую частичность реализации целей КРА.

Ключевое направление процесса получает оценку «удовлетворительно», если эта же оценка присвоена *всем* целям, достижение которых предусмотрено по данному направлению. Если хотя бы одна из целей КРА не достигается (с оценкой «удовлетворительно») - КРА получает оценку «не удовлетворительно».

Определенный *уровень зрелости* считается достигнутым, если *все* ключевые направления процесса, с которыми связывается данный уровень зрелости в модели СММ, а также *все* ключевые направления нижележащих уровней получили оценку «удовлетворительно».

Таким образом, *обязательным условием аттестации* организации-разработчика на соответствующий уровень зрелости является *достижение всех целей* по *всем направлениям* данного и всех нижележащих уровней, указанных в модели СММ, для *всех проектов* организации (текущих и будущих) *на все время* существования организации.

Организациям-разработчикам, совершенствование процессов в которых будет осуществляться в направлении достижения второго уровня зрелости по модели СММ, целесообразно:

- детально изучить цели и процедуры КРА второго уровня [20] (его описание доступно на сайте SEI);
- получить административную и финансовую поддержку;
- создать соответствующие организационные структуры и другие элементы процесса, рекомендуемые СММ (см. главу 1, п.1.1.4 о требованиях к базовому процессу организации);
- подготовить нормативно-методическую и учебную базу. Перечень необходимых (для достижения уровня 2) международных и отечественных стандартов, которые могут использоваться в качестве ориентиров при выполнении работ по ключевым направлениям, представлен в приложении 3;
- организовать процесс обучения специалистов программных проектов;
- составить глобальный план работ по совершенствованию процесса организации, рассчитанный на 6 - 8 лет;
- обеспечить надлежащее управление работами.

12.3.5. Выбор организаций-исполнителей программных проектов

Предлагаемая ниже процедура оценивания зрелости организаций-разработчиков не является адаптацией ни одного из перечисленных выше методов (SPA, SCE, IP). Цель ее разработки авторами книги - предоставить организации-заказчику *приемлемый механизм* выбора организаций-исполнителей программных проектов, концептуально согласующийся с СММ и адекватный уровню отечественной программной инженерии.

Процедура ориентирована на ранжирование зрелости организации-исполнителя по шкале от 0 до 2, где рейтинг 2 соответствует второму уровню зрелости по модели CMM.

Процедура основана на использовании фрагмента оригинала контрольного вопросника SEI в части, касающейся уровня 2 CMM (приложение 7), и включает следующую последовательность шагов:

Шаг 1. Организация-заказчик составляет *проект паспорта* программного (системного) продукта, подлежащего разработке, по форме, представленной на рисунке 12.10;

Паспорт программного продукта

_____
(название)

разработанного _____
(название организации)

Класс системы	(например, АСУ ТП, АИС и др.)
Прикладная область	(например, бухгалтерский учет)
Масштабность: • **продолжительность** • **количество исполнителей** • **объем (размер) продукта** • **степень повторного использования**	 (в месяцах) (количество человек, принимающих участие в разработке) (объем ПО в строках исходного кода или УЕФ) (___% исходного кода, ___% модифицированного кода, ___% повторно используемого кода) Примечание (например, большое количество COTS - большие затраты на разработку)
Долевое участие в работе	(например, головной исполнитель, все виды работ и др.)
Организационный подход	(например, временный трудовой коллектив, интегрированная группа и др.)
Языки и среды программирования	используемые языки (среды) программирования
Заказчик	наименование организации-заказчика
Применяемые стандарты	(группа применяемых отечественных и международных стандартов)
Наличие соисполнителей	(да/нет, количество организаций-соисполнителей)
Новизна	(например, взамен действующей системы)
Платформа функционирования	характеристика аппаратной, программной и телекоммуникационной среды
Другие требования	

Ответственный исполнитель проекта: _____ Подпись _____

Телефон _____

Дата _____

Рис. 12.10. **Структура паспорта программного продукта**

Шаг 2. Организация-заказчик рассылает претендентам на роль исполнителей *форму паспорта* и *контрольный вопросник*;

Шаг 3. Организация-претендент, ознакомившись с проектом паспорта заказываемого продукта, подбирает несколько (но не менее трех) завершенных или находящихся в стадии завершения проектов, разработанных в данной организации и «схожих» с проектом, предлагаемым к разработке;

Шаг 4. Разработчик проекта *заполняет паспорт* разработанного (разрабатываемого) продукта по форме паспорта и *отвечает на все вопросы* контрольного вопросника (приложение 7). Ответы на вопросы по каждому направлению проставляются посредством отметки (знак «+» при ручном заполнении формы или число «1» при машинном заполнении) в соответствующих колонках интервальной шкалы;

Шаг 5. Организация-претендент отсылает заполненные паспорта и контрольные вопросники организации-заказчику;

Шаг 6. Эксперт организации-заказчика обрабатывает все паспорта и контрольные вопросники организации-претендента и определяет уровень зрелости организации.

Обработка контрольных вопросников для получения оценок включает выполнение следующих действий:

1) каждой оценке присваивается эквивалентный числовой коэффициент (таблица 12.5).

Таблица 12.5. **Коэффициенты для оценки ответов на вопросы**

Оценка частоты выполнения процедур	**Коэффициент**
Почти всегда	1
Часто	0.75
Иногда	0.5
Редко	0.25
Никогда	0

2) обрабатывается *один вопросник* для *одного проекта*: подсчитывается количество ответов по каждой оценке одного направления процесса (количество отметок «+» или «1» в столбце). Это количество ответов умножается на соответствующий коэффициент и вычисляется их сумма. Затем эта сумма делится на количество вопросов, касающихся данного направления, и умножается на 100% (для получения оценки достижимости целей направления в процентах).

Ниже приведен пример заполнения опросного листа по направлению «Управление требованиями» и оценка уровня достижимости целей по данному направлению. Соответствующий опросный лист содержит 6 вопросов. Пример заполнения опросного листа приведен в таблице 12.6. Вычисленная оценка КРА по ответам на вопросы по данному направлению составляет

$(2\times1 + 1\times 0.75 + 1\times0.5 + 2\times0) / 6 = 0.54$

или в процентном отношении - $0.54\times100\% = 54\%$

Процедура повторяется по всем шести направлениям, представленным в вопроснике.

Таблица 12.6. **Пример заполнения опросного листа**

Управление требованиями	Почти всегда	Часто	Иногда	Редко	Никогда	Не исполь-зуется
1. Используются ли *системные требования*, *делегированные ПО*, в качестве основы для выполнения разработки и управления процессом разработки?	+					
2 Выполняется ли корректировка *планов ПО*, рабочих продуктов и действий при изменении системных требований, делегированных ПО?	+					
3. Руководствуется ли проект принятой в организации *политикой* в части управления системными требованиями, делегированными ПО?		+				
4. Прошли ли лица, которым поручено управление делегированными требованиями, обучение приемам управления требованиями?			+			
5. Проводятся ли измерения с целью определения адекватности действий, выполняемых по управлению делегированными требованиями (например, есть ли учет общего числа предложенных изменений в требованиях, числа принятых предложений по изменениям, числа произведенных корректировок в *базовой версии* и пр.)?					+	
6. Подвергаются ли действия по управлению требованиями в проекте *ревизиям* с целью *обеспечения гарантии качества ПО*?					+	

3) подобным образом обрабатываются ответы на вопросы *по всем проектам*;

4) по завершении обработки опросных листов оценки по каждому направлению для всех проектов *усредняются*.

Усредненная оценка направления по всем проектам вычисляется как медиана частных оценок. Например, если в результате обработки вопросов по первому направлению для пяти проектов были получены такие оценки:

54 58 75 79 80

то медианой ряда будет значение 75 и это будет средняя оценка данного направления по представленным проектам.

5) полученные суммарные оценки проектов в процентах по каждому направлению заносятся в *итоговый отчет* по форме, представленной в таблице 12.7.

6) для *расчета уровня зрелости Lзр* организации применяется формула:

$$Lзр = 2/6 \cdot \sum_{i=1}^{6} (КРА\%_i / 100),$$

где $КРА\%_i$ - полученные суммарные оценки i-го проекта в процентах.

Таблица 12.7. **Оценка уровня зрелости по КРА**

КРА	Почти всегда >90% случаев	Часто 60-90% случаев	Почти поровну 40-60% случаев	От случая к случаю 10-40%	Крайне редко < 10%
Управление требованиями					
Планирование проектов					
Мониторинг проектов					
Управление соисполнителями					
Обеспечение гарантии качества					
Управление конфигурацией					

12.3.6. Интеграция моделей CMM (CMMIntegration)

Модель CMMI выпущена в двух вариантах — стадийное представление (как в SW-CMM) и непрерывное представление (как в SPICE) (рисунок 12.11).

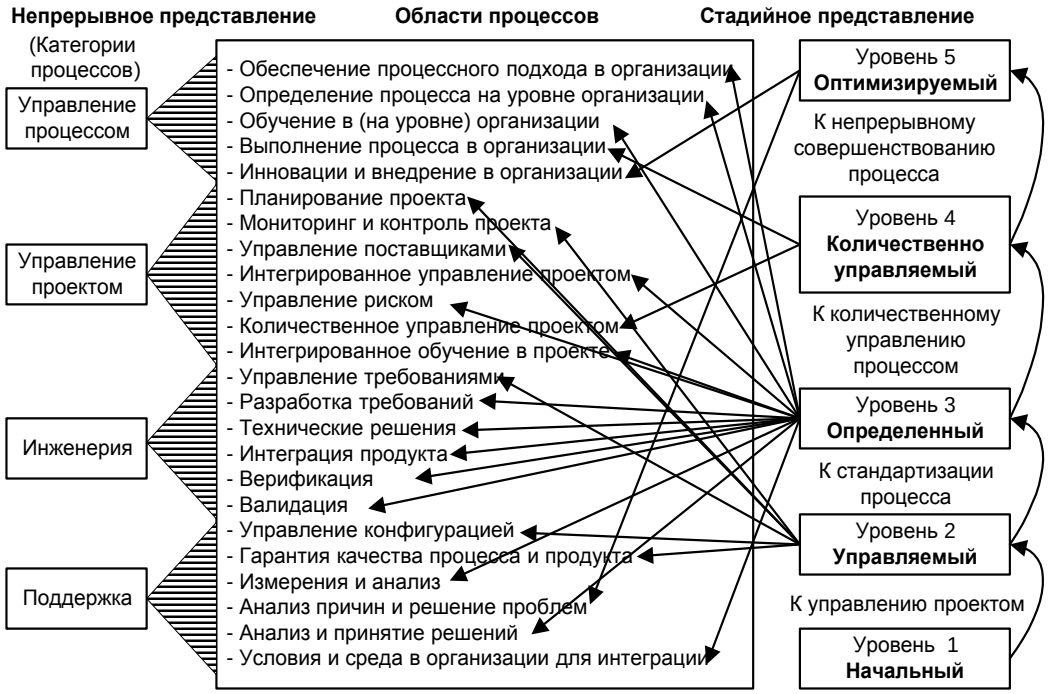


Рис. 12.11. **Модель CMMI**

В основе стадийного представления лежит концепция зрелости базового процесса программной инженерии в организации в целом (5 уровней зрелости), а в основе непрерывного - концепция мощности процессов из четырех категорий (6 уровней мощности). В концепции мощности процессов рассматривается комплекс действий (базовых практических приемов), связанных с одной областью процессов, а в концепции зрелости - комплекс процессов в масштабах всей организации.

Модель CMMI не предписывает, какие именно процессы должны быть учреждены в организации или проекте, но устанавливает минимальные критерии, необходимые для планирования и применения процессов, выбранных организацией в качестве основы для улучшений.

Модель CMMI описана специалистами из SEI в [25, 26], а также достаточно детально представлена в [27, 28].

12.4. Сертификация систем менеджмента качества

12.4.1.Стандарты для построения и проверки систем менеджмента качества

Под *системой качества* любого предприятия (организации) понимается совокупность организационной структуры, ответственности, методик, процессов и ресурсов, необходимых для улучшения качества производимой продукции и предоставляемых услуг [29].

Побудительными причинами создания системы качества могут послужить:

- *факторы государственного регулирования* (необходимость лицензирования, подтверждения производителем современного научно-технического уровня);
- *факторы гражданско-правовых отношений* (необходимость получения госзаказа, субподрядов и соответствия условиям тендеров);
- *факторы рыночного регулирования* (требования сертификации системы качества, необходимость обеспечения стабильности поставок, подтверждения устойчивости организации (прежде всего в части рисков, связанных с качеством и безопасностью продукции));
- *желание совершенствовать модель бизнеса* (устранить «скрытое» производство (переделки продуктов), предупредить поступление претензий и жалоб потребителей, реструктурировать управление организацией (улучшить управляемость, изменить роль высшего руководства, вовлечь персонал в дело обеспечения качества всех процессов и др.);
- *необходимость продемонстрировать финансовую привлекательность и надежность* (получить кредиты, создать совместные предприятия и др.).

Сфера действия системы качества определяется ее *моделью*. В 2004 году прекратили действие стандарты серии ISO 9000 версии 1994 года, которые рекомендовали и поддерживали выбор и разработку трех различных моделей системы качества и, соответственно, трех схем их сертификации. Действующая версия этой серии стандартов (2000 года) основана не на бизнес–функциях (элементах качества), а на бизнес-процессах предприятия, то есть *всецело поощряет применение процессо-ориентированного подхода*.

В редакции 2000 года серия ISO 9000 регламентирует построение, внедрение и использование эффективной *системы менеджмента качества* (СМК) в организациях любого типа и размера, не имеющих ранее сертифицированных систем качества[5].

---

[5] Это же касается и стандартов ДСТУ, гармонизированных с соответствующими стандартами ISO.

Нужно отметить, что ISO не только реструктурировала серию стандартов 9000, но и пересмотрела общую часть их названия. Теперь это стандарты не для «систем качества» (quality systems), а для «систем менеджмента качества» (quality management systems). Это еще раз подчеркивает важность гибкого *управления* качеством с ориентацией на «удовлетворение потребностей потребителя относительно качества продукции», а не просто *обеспечения гарантий* «соответствия продукции установленным требованиям к качеству» (см. отличие определений процессов «обеспечение гарантии качества» и «управление качеством» в начале главы 5).

Серия ISO 9000 включает следующие стандарты:

- ***ISO 9000:2000*** (или ДСТУ ISO 9000:2001) «СМК. Основные положения и словарь» – определяет основную терминологию в области качества и обеспечивает ввод в действие системы менеджмента качества. Полный текст документа доступен в Интернет, например, по адресу [30].
- ***ISO 9001:2000*** (ДСТУ ISO 9001:2001) «СМК. Требования» – устанавливает детальные требования к системе менеджмента качества, демонстрация соответствие которым подтверждает способность организации обеспечить надлежащее качество продукции [31].
- ***ISO 9004:2000*** (ДСТУ ISO 9004:2001) «СМК. Руководящие указания по улучшению деятельности» – обеспечивает руководство по внедрению широко развитой СМК с целью постоянного совершенствования деловой деятельности [32].

Стандарты ISO 9001 и ISO 9004 разработаны как *согласованная пара* дополняющих друг друга стандартов на *системы менеджмента качества*.

Стандарт ISO 9001:2000 устанавливает *требования* к СМК, которые предназначены для внутреннего применения организациями *в целях сертификации* или *заключения договоров (контрактов)*. Он формулирует минимальный набор условий, которым должна удовлетворять СМК, обеспечивающая гарантии выпуска продукции, отвечающей установленным требованиям и чаяниям потребителей. Предусматривает 4 группы процессов связанных с СМК:

- процессы управленческой деятельности высшего руководства,
- процессы обеспечения ресурсами,
- процессы жизненного цикла продукции,
- процессы измерения, анализа и улучшения.

Как видно из этого перечня, кроме процессов, непосредственно касающихся выпуска продукции, в стандарте явно выделены процессы измерения, проверки (анализа, аудита) и совершенствования, как продукции, так и самой системы менеджмента качества, а также процессы управления ресурсами, включая трудовые ресурсы. Отмечается также усиление роли высшего руководства организации в развитии и улучшении системы менеджмента качества.

Стандарт ISO 9004:2000 предоставляет руководство по *внедрению* и применению СМК для *совершенствования работы* организации в целом. Может использоваться как методическое пособие по построению систем менеджмента качества, поскольку содержит рекомендуемую структуру СМК, характеристики ее основных функциональных элементов, определенные требования к организационной структуре, составу и содержанию данных, которые должны или могут применяться в СМК. В стандарте также рассматриваются экономические аспекты качества, различные виды расходов и статьи затрат на качество, даются указания по проведе-

нию внутренних проверок (аудитов) СМК и, кроме того, содержатся руководства по самооцениванию СМК.

Для разработчиков ПС интерес представляет также международный стандарт ***ISO 90003:2004*** (нет гармонизированного стандарта ДСТУ) *"Software engineering -- Guidelines for the application of ISO 9001:2000 to computer software"* (Программная инженерия -- Руководящие указания по применению ISO 9001:2000 к компьютерному ПО). Этот стандарт разъясняет порядок применения ISO 9001 при приобретении, поставке, разработке, эксплуатации и сопровождении программного обеспечения и связанных с ним услуг, не вступая в противоречие с требованиями ISO 9001. По сравнению с отмененным стандартом ISO 9000-3 он в большей степени приспособлен к специфике отрасли, в частности, ссылается на модели ЖЦ ПС и детально рассматривает вопросы, характерные для разработки ПО. Однако стандарт ISO 90003:2004 – это стандарт обеспечения качества и не может быть использован для оценки уровня зрелости и предсказания результата программного проекта. Положения стандарта не определяют критерии оценивания СМК при ее сертификации.

Проверка систем менеджмента качества регулируется международным стандартом ***ISO 19011:2002*** (ДСТУ ISO 19011:2003) «Руководство по аудитам СМК и (или) систем экологического менеджмента», который обеспечивает руководящие указания по управлению и проведению внутреннего и внешнего аудитов СМК. Стандарт содержит, в частности, требования к процедурам проверки, квалификации аудиторов СМК, а также управлению программой проверок [33].

Достаточно полный обзор и рекомендации по применению стандартов ISO для создания систем менеджмента качества программной продукции можно найти в книге В.В. Липаева «Обеспечение качества программных средств. Методы и стандарты» [34]. Сопоставительный анализ моделей CMM, CMMI, SPICE, ISO 9000 и др. проведен, например, в [35].

12.4.2. Сертификация программных продуктов и систем менеджмента качества

Сертификация – это эффективный механизм обеспечения гарантии соответствия *продуктов*, *технологий*, используемых для их создания, а также *системы менеджмента качества* требованиям нормативно-методической базы качества продукции. Основные понятия в области сертификации продукции изложены в стандарте ДСТУ 2464 [36].

Сертификация выполняется аккредитованными независимыми *Органами сертификации*, которые призваны подтверждать, что продукция, выпускаемая предприятием (организацией), отвечает обязательным требованиям нормативных документов, а все технические административные и человеческие факторы, влияющие на ее качество, находятся под постоянным контролем. Результаты сертификации оформляются *сертификатом*.

Сертификация системы менеджмента качества проводится в соответствии с ДСТУ 3419-96 [37], заключается в сравнение фактической информации о системе менеджмента качества предприятия (организации) с моделью СМК, определенной в стандартах ISO, и может выполняться по *инициативе Заявителя* или по *решению Органа сертификации*.

Цели сертификации могут быть разными:

- получение независимой оценки состояния СМК для обеспечения *внутренней уверенности* в способности предприятия стабильно обеспечивать выпуск продукции, отвечающей установленным требованиям;
- получение сертификата соответствия как средства *повышения доверия* к предприятию со стороны потребителей и расширения рынков сбыта продукции (услуг);
- получение сертификата соответствия *по требованию конкретного потребителя* для повышения его уверенности в способности предприятия стабильно выполнять условия договора.

Продолжительность подготовки к сертификации составляет в среднем один-два года и зависит от множества факторов, основными из которых являются:

- уровень управления качеством на предприятии к моменту начала подготовки к сертификации (традиции в области управления качеством, уровень подготовки персонала, уровень взаимопонимания в отношениях с партнерами и т.д.);
- отношение руководства организации к вопросу управления качеством;
- наличие аттестованных специалистов по сертификации систем менеджмента качества.

Время на разработку и внедрение системы менеджмента качества можно сократить, если подготовить несколько экспертов по сертификации СМК из числа руководящих работников организации, а также группу экспертов по внутренним проверкам СМК.

Примерная последовательность шагов *процесса сертификации СМК* указана ниже петитом.

1. Заявитель подает в Орган сертификации обращение о намерении сертифицировать систему менеджмента качества.
2. Орган сертификации регистрирует заявку и уведомляет заявителя о ее принятии и условиях начала работы.
3. Заявитель оплачивает регистрационный взнос.
4. Орган сертификации регистрирует копию платежного поручения об оплате регистрационного взноса, уведомляет Заявителя о произведенной оплате и дает «добро» на начало работ.
5. Орган сертификации направляет Заявителю следующие документы: *Форма декларации-заявки*, *Комплект исходных форм документов*, *Перечень документов, необходимых для предварительной оценки СМК*.
6. Заявитель предоставляет следующие документы: *Декларация-заявка*, *Политика в области качества*, *Руководство по качеству*, *Анкета-вопросник* (заполненная форма), *Исходные данные для предварительной оценки производства* (заполненная форма), *Структурная схема организации*, *Структурная схема службы качества*, *Перечень документов СМК*.
7. Дополнительно Орган сертификации может запросить: *Стандарт предприятия «Управление документацией»*, Стандарт предприятия «*Внутренние проверки СМК*».
8. Орган сертификации проводит анализ комплектности представленной документации.
9. Орган сертификации уведомляет Заявителя о принятии заказа на сертификацию или об отказе в принятии заказа на сертификацию (в последнем случае – возврат к шагу 1).
10. Орган сертификации оформляет договор на проведение предварительной оценки СМК.
11. Стороны подписывают договор и Заявитель производит оплату услуг Органа сертификации.
12. Орган сертификации назначает главного эксперта и формирует комиссию.

13. Комиссия выполняет анализ системы качества Заявителя по представленным исходным документам и материалам.

14. Комиссия составляет заключение по результатам предварительной оценки. В случае отрицательного заключения – возврат к шагу 1).

15. Орган сертификации оформляет договор на проведение второго этапа сертификации.

16. Стороны подписывают договор и Заявитель производит оплату услуг.

17. Комиссия разрабатывает программу проверки.

18. Комиссия проводит проверку.

19. Комиссия составляет заключение по результатам проверки.

20. Комиссия принимает решение рекомендовать систему менеджмента качества к сертификации. Если заключение отрицательное – возврат к шагу 1.

21. Орган сертификации принимает решение о регистрации сертификата в Реестре Регистра и выдаче лицензии на применение знака соответствия.

22. Орган сертификации пересылает заявителю: *Сертификат системы менеджмента качества*, *Лицензии на применение знака соответствия*.

23. Орган сертификации оформляет договор на проведение инспекционного контроля.

24. Стороны подписывают договор и Заявитель производит оплату услуг.

25. Орган сертификации ежегодного проводит инспекционный контроль соответствия сертифицированной системы менеджмента качества. Если результаты отрицательны – возврат к шагу 1.

26. Орган сертификации подтверждает действие сертификата и знака соответствия. Он может также приостановить его действие, либо вообще аннулировать сертификат.

Сертификация может быть *обязательной* или *добровольной*. Обязательная сертификация применима по отношению к системам менеджмента качества, предназначенным для поддержки разработки критических ПС, и ее необходимость определяется заказчиком (потребителем) ПС. В остальных случаях рекомендуется необязательная (добровольная) сертификация.

В области обязательной сертификации систем менеджмента качества, продукции и услуг в Украине работает государственная Система сертификации продукции УкрСЕПРО, объединяющая 149 Органов сертификации, а также 811 испытательных лабораторий.

В области необязательной сертификации продукции работает Межотраслевой центр качества «ПРИРОСТ» (prirost.udc.com.ua), аккредитованный в "МО СовАсК" с 1994 г., а в УкрСЕПРО - с 2000 г. Это головная организация по вопросам качества и сертификации Украинской ассоциации качества, единственная в Украине избранная членом Европейского фонда управления качеством (EFQM).

Одной из организаций, предоставляющих услуги ***сертификации программных средств***, является Украинский научный центр государственной регистрации и сертификации информационных технологий «Софт-Рейтинг» (www.softrating.com.ua/ukr/about.htm). Это базовая организация рабочей группы РГ6 «Випробування, сертифікація та управління якістю програмних засобів ЕОМ» подкомитета ПК7 «Інженерія програмних засобів. Автоматична ідентифікація та методи роботи з даними. Управління даними та обмін» Украинского технического комитета по стандартизации ТК20 «Інформаційні технології». По данным сайта «Софт-Рейтинг» - это первая в Украине независимая и технически компетентная организация – Орган сертификации (УКРСЕРТСОФТ), аккредитованный Госпотребстандартом Украины. Он проводит добровольную сертификацию программной продукции в системе

сертификации УкрСЕПРО, а также предоставляет услуги по созданию систем менеджмента качества и испытательных лабораторий. Сертификат соответствия выдается УКРСЕРТСОФТ на период 1, 2 или 3 года в зависимости от выбранной схемы сертификации. Схема учитывает серийность программного средства (единичное или серийное) и спектр необходимых услуг (обследование производства, аттестация производства, сертификация системы менеджмента качества, испытания, технический надзор) (http://www.softrating.com.ua/ukr/Rules.htm).

Кроме указанных *Органов сертификации* услуги сертификации СМК в организациях-разработчиках ПС Украины оказывают, например, *Бюро Международной Сертификации – SIC* (www.sic.com.ua/org.php), *Бюро Веритас Украина* (http://www.bureauveritas.com.ua) и другие Органы сертификации, осуществляющие деятельность на территории Украины (http://www.sic.com.ua/org.php).

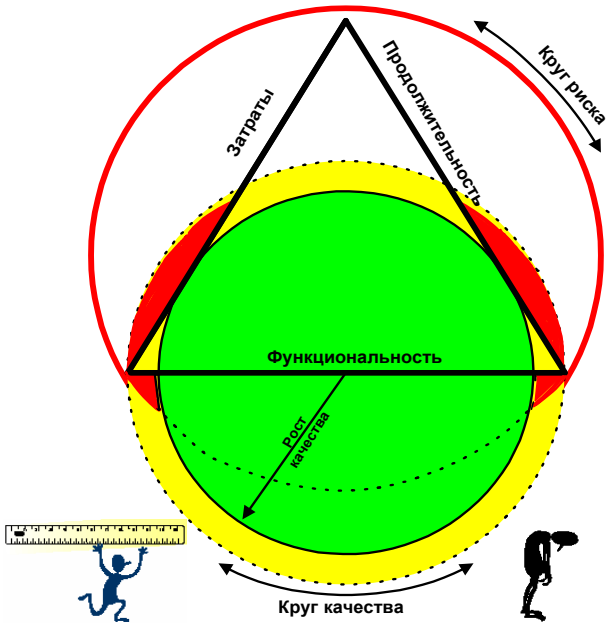
Литература к главе 12

1. *ISO/IEC 12207:1995* / Amd.1:2002 Information technology – Software life cycle processes
2. *Коваль Г.И.* Подход к прогнозированию надежности ПО при управлении проектом // «Проблемы программирования». –2002. - № 1 – 2. – С. 282 – 290.
3. *Подход* к управлению качеством программных систем обработки данных / Лаврищева Е.М., Коваль Г.И., Коротун Т.М. // Кибернетика и системный анализ. – 2006.- .-№5.- с.174-185.
4. *ISO/IEC 14598-1*:1999. Information technologies - Software product evaluation - Part 1: General overview.(или ДСТУ ISO/IEC 14598-1-2004. Інформаційні технології. Оцінювання програмного продукту. Частина 1. Загальний огляд).
5. *ISO/IEC 14598-3*:2000. Software Engineering - Product evaluation - Part 3: Process for developers.
6. *ISO/IEC 14598-4*:1999. Software Engineering - Product evaluation - Part 4: Process for acquirers.
7. *ISO/IEC 14598-5*:1998. Information Technologies - Software product evaluation - Part 5: Process for evaluators.
8. *ISO/IEC 15026*:1998. Information technologies - System and software integrity levels.
9. *ДСТУ 2850*-94. Програмні засоби ЕОМ. Показники і методи оцінювання якості.
10. *Липаев В.В.* Выбор и оценивание характеристик качества программных средств. Методы и стандарты. Серия «Информационные технологии». М.: СИНТЕГ, 2001. – 228 с.
11. *ГОСТ 28195*-99. Оценка качества программных средств. Общие положения.
12. *DTR 9126-3:2001.* Software Engineering - Product Quality - Part 3:Internal Metrics // ISO/IEC JTC1/SC7 N2416, Software & System Eng. Secretariat, Canada. – 2001. – 66 p.
13. *DTR 9126-2:2001.* Software Engineering - Product Quality - Part 2:External Metrics // ISO/IEC JTC1/SC7 N2419, Software & System Eng. Secretariat, Canada. – 2001. – 111 p.
14. *DTR 9126-4:2001.* Software Engineering – Software Product Quality - Part 4: Quality In Use Metrics // ISO/IEC JTC1/SC7 N2430, Soft. & Syst. Eng. Secretariat, Canada. – 2001. – 70 p.
15. *ISO/IEC 14598-6*:1998. Information Technology - Software product evaluation - Part 6: Documentation of evaluation modules.
16. *Вронский К.* Стандарты ISO и управление качеством проектов IT - http://www.interface.ru/fset.asp?Url=/qad/mfg/isoit.htm

17. *ISO/IEC* TR 15504-1:1998. Information technologies - Software process assessment - Part 1: Concepts and introductory guide.
18. *ISO/IEC* TR 15504-3:1998. Information technologies - Software process assessment - Part 3: Performing an assessment.
19. *Paulk M.C.* et al. Capability Maturity Model for Software. Version 1.1. Technical Report CMU/SEI-93-TR-024. - Software Eng.Inst., Pittsburgh, PA 15213, 1993. – 82 p.
20. *Paulk M.C.* et al. Key Practices of the Capability Maturity Model. Version 1.1. Technical Report CMU/SEI-93-TR-025. - Software Eng.Inst., Pittsburgh, PA 15213, 1993. – 479 p.
21. *Masters S.*, Bothwell C. CMM Appraisal Framework. Version 1.0. Technical Report CMU/SEI-95-TR-001. - Software Eng. Inst., Pittsburgh, PA 15213, 1995. – 76 p.
22. *Bumes P.*, Phillips M. Software Capability Evaluation (SCE). Version 3.0. Method Description. Technical Report CMU/SEI-96-TR-002. - Software Eng. Inst., Pittsburgh, PA 15213, 1996. – 192 p.
23. *Zubrow D.* et al. Maturity Questionnaire. Technical Report CMU/SEI-94-SR-007. - Software Eng. Inst., Pittsburgh, PA 15213, 1994. – 57 p.
24. *Whitney R.* et al. Interim Profile: Development and Trial of Method to Rapidly Measure Software Engineering maturity Status. Technical Report CMU/SEI-94-TR-004. - Software Eng. Inst., Pittsburgh, PA 15213, 1994. – 44 p.
25. *CMMI*^{SM} for Software Engineering, Version 1.1, Staged Representation (CMMI-SW, V1.1, Staged) - http://www.sei.cmu.edu/publications/documents/02.reports/02tr029.html
26. *CMMI*^{SM} for Software Engineering, Version 1.1, Continuous Representation (CMMI-SW, V1.1, Continuous) - http://www.sei.cmu.edu/publications/documents/02.reports/02tr028.html
27. *Мильман К.*, Мильман С. CMMI – шаг в будущее // Открытые системы.- 2005.-№5-6 - http://old.osp.ru/os/2005/05-06/085.htm
28. *Липаев В.В.* Модели зрелости программной инженерии - CMMI. Содержание и применение // Jet Info Online.-2006.-N6 - http://www.jetinfo.ru/2006/6/2006.6.pdf
29. *ДСТУ 2844-94*. Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення.
30. *ДСТУ ISO 9000:2001* - http://www.zstu.zaporizhzhe.ua/base/i2/iff/k3/ukr/welding/guide/iso/iso9000.htm
31. *ДСТУ ISO 9001:2001* - http://www.iso.staratel.com/ISO9000/DOC/DSTUISO9001/DSTUISO9001.doc
32. *ДСТУ ISO 9004:2001* - http://www.iso.staratel.com/ISO9000/DOC/DSTUISO9004/DSTUISO9004.htm
33. *ДСТУ ISO 19011:2003* - http://www.nsu.ru/smk/files/iso_19011/iso_19011.doc
34. *Липаев В.В.* Обеспечение качества программных средств. Методы и стандарты. Серия «Информационные технологии». М.: СИНТЕГ, 2001. – 380 с.
35. *Колдовский В*. Разработка ПО: стандарты качества // Компьютерное обозрение.-2005 (9 сентября).- http://itc.ua/article.phtml?ID=21715&IDw=33&pid=52
36. *ДСТУ 2462-94*. Сертифікація. Основні поняття терміни та визначення.
37. *ДСТУ 3419-96*. Система сертифікації УкрСЕПРО. Сертифікація систем якості. Порядок проведення -http://www.mdoffice.com.ua/pls/MDOffice/MDODOC.FindHelp?p_file=49&p_page=394&context=

ЗАКЛЮЧЕНИЕ

В инженерии качества не так уж все сложно и скучно (а может быть и грустно). Главное в ней – это *здравый смысл*, *наблюдательность*, *усидчивость* и *немного фантазии*. Вот, например, рисунок в стиле авангард, обобщающий наш взгляд на проблему разработки высококачественных ПС в условиях ограниченных ресурсов проекта, а за ним – краткое изложение содержания этой книги в переложении на *человеческий* язык менеджера проекта.



Пределы роста качества ПС при ее разработке
в условиях ограниченных ресурсов проекта с учетом рисков

- Проект – это монолитный сплав *функций* будущей ПС, *затрат* и *сроков* (классический треугольник).
- Проект находится под постоянными угрозами срыва и «окружен» *риском* (большой круг риска). Нельзя допускать превышения затрат и сроков (две стороны треугольника). Нет смысла увеличивать функциональность ПС (система не должна делать того, для чего она не предназначена (фиксированный максимальный размер основания треугольника)).
- Объем функциональных возможностей будущей ПС можно «прикинуть» в самом начале проекта, применив для этого методы измерения в концепции FSM.
- Хорошо бы достичь абсолютного качества ПС и «обеспечить» лучшими характеристиками каждую «*функциональную точку*» в основании треугольника. Получился бы круг качества максимального размера (окружность пунктиром). Жаль, этого нельзя добиться – такой круг качества частично «выйдет» в области риска по факторам затрат и продолжительности (черные сегменты).
- Можно бы вообще не заботиться о качестве («стянуть» круг качества в точку и забыть). Однако остается не прикрытым сегмент риска функциональной

пригодности ПС (нижний сегмент круга риска). Это риск разработчика (заказчик откажется от уже «почти» готовой ПС). И это риск заказчика (риск отказа ПС при эксплуатации).

- Выход – планомерно повышать качество (от центра круга качества в направлении к его пунктирной границе), постепенно отвоевывая площадь у нижнего сегмента круга риска. Для этого – построить все необходимые процессы и начать их применять.
- Вечный вопрос – где остановиться? Нужно расширять круг качества, пока он не коснется двух сторон треугольника. Можно считать, что оставшиеся не прикрытыми кусочки нижнего сегмента риска, - это риск отказа ПС, но не по ее вине.

Если в ряды разработчиков вашего проекта случайно «затесался» научный сотрудник, да еще и математик – отдайте рисунок ему. А сами займитесь институциализацией процессов ЖЦ.

Опять скучно? Вспомните слова Лапидуса – «стукните» по любому сотруднику проекта. Пока он вытянет всю цепочку проблем, у Вас будет время применить целе-ориентированный и процессо-ориентированный подходы. Ибо это – ***элемент обязательной программы для менеджеров***, без выполнения которой не будет ясен *внутренний смысл* происходящих в проекте процессов (это - как обучение программированию *в DOS* современных программистов).

В 2003 журнал IEEE Software опубликовал дискуссионную статью Р.Аустина и Л.Девина[1] о проблемах формальной и не формальной разработки ПО.

Примечательно то, что один из авторов статьи (Л.Девин) драматург, работающий в театральном обществе, а другой (Р.Аустин) – экономист и менеджер, преподающий в Гарвардской школе бизнеса.

В статье отмечается, что с недавних пор можно наблюдать войну методологий. Борются «формалисты» (formalists) – сторонники промышленного подхода, ставящие во главу угла слово «процесс» и «следование планам», и «эйжилисты» (agilists), ключевые понятия для которых - «взаимодействие личностей» и «реагирование на изменения».

Авторы статьи сравнивают работу над проектом с театральной постановкой, когда все участники методом проб и ошибок *выстраивают роли* в спектакле, совершенствуя их от одной репетиции к другой, чтобы потом играть спектакль долгие годы. Конечно, отступления от текста роли возможны – заученная роль, формально соответствующая требованиям, это еще не показатель хорошей игры. Однако четко знать роль «по шагам» все же *нужно*.

Надеемся, что эту точку зрения специалистов вы разделите (вместе с нами).

Всяческих успехов.

---

[1] Rob Austin, Lee Devin “Beyond Requirements: Software Making as Art”, IEEE Software, Jan/febr, 2003

Приложение 1. МОДЕЛИ ЖИЗНЕННОГО ЦИКЛА

В приложении приведен краткий обзор основных моделей жизненного цикла программных систем (ПС).

Жизненный цикл ПС определяется как «…весь период существования системы от начала разработки до завершения ее использования»[1]

ЖЦ делится на *упорядоченные стадии*, основные из которых:

- Определение потребностей.
- Анализ требований и формирование концепции.
- *Разработка* (*далее рассматривается подробнее).*
- Производство.
- Внедрение/продажа.
- Эксплуатация.
- Сопровождение и поддержка.
- Изъятие из эксплуатации.

Внутри каждой из этих стадий происходит дальнейшая детализация выполняемых действий по более мелким стадиям (на более низком уровне).

Модели ЖЦ описывают *взаимосвязи* стадий.

Рассмотренные в данном приложении модели касаются стадий ЖЦ, связанных с *процессом разработки ПС*, основные из которых:

- Анализ требований.
- Проектирование (предварительное и детальное).
- Реализация.
- Тестирование.

Наиболее известными *типами* моделей ЖЦ в настоящее время являются: последовательные и итерационные. На практике эти модели могут комбинироваться, образуя смешанные модели ЖЦ.

Каждая из моделей имеет свои достоинства и недостатки, которые должны исследоваться при выборе конкретной модели для проекта.

1.1. Назначение моделей разработки

Модели ЖЦ могут использоваться для:

- организации, планирования, распределения ресурсов (трудозатрат и времени) и управления проектом разработки;
- организации взаимодействия с заказчиками и определения состава документов (рабочих продуктов), разрабатываемых на каждой стадии;
- анализа и/или оценивания распределения ресурсов и затрат на протяжении ЖЦ;
- наглядного описания или в качестве основы для проведения финансовых расчетов с заказчиками;
- проведения эмпирических исследований с целью определения влияния моделей на эффективность разработки и общее качество программного продукта.

В стандарте ISO/IEC 12207 определена структура процессов, но не указаны способы их взаимодействия в рамках разных моделей. Рекомендации по возмож-

[1] ДСТУ 2941-94. Разработка систем. Термины и определения.

ному отображению процессов ЖЦ на основные модели разработки приведены в Руководстве по применению стандарта[2].

1.2. Модели последовательного выполнения стадий

1.2.1. Каскадная модель

Каскадная (Waterflall) или стандартная модель - наиболее известная модель разработки, по умолчанию предполагаемая стандартами (в частности, ГОСТ 34). Эта модель характеризуется набором стадий, выполняемых последовательно (рисунок 1.1). Каждая стадия должна быть завершена до перехода к следующей, а создаваемые на ней рабочие продукты после их верификации и валидации должны быть «заморожены» и переданы на следующую стадию в качестве эталона. Пользователь видит работающий программный продукт в самом конце разработки. Наиболее жесткое ограничение этой модели - необходимость «замораживания» требований. При этом, чтобы минимизировать риск увеличения стоимости, допускаются только небольшие изменения.

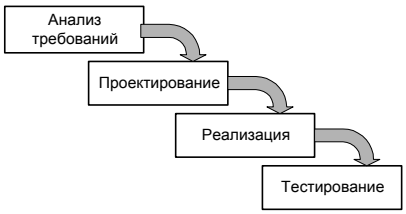


Рис. 1.1. **Каскадная модель**

С точки зрения качества ПС, в этой модели стоимость исправления дефектов на стадии тестирования наибольшая (по сравнению с другими моделями), поскольку тестирование выполняется в самом конце разработки. Из-за недостатка времени на переделки и тестирование существует значительный риск выпуска ПС с серьезными дефектами.

1.2.2. Каскадная модель с обратной связью

Эта модель расширяет стандартную модель включением в нее *циклов обратной связи* для возврата на предыдущую стадию при изменении требований, проекта (конструкции) и по результатам инспекций или действий по V&V (рисунок 1.2).

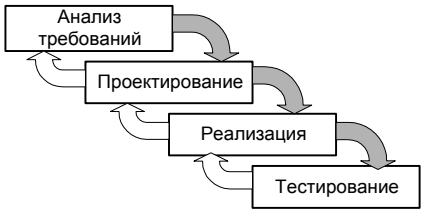


Рис. 1.2. **Каскадная модель с обратной связью**

Процессы V&V, выполняемые после завершения каждой стадии разработки, играют в этой модели важнейшую роль.

---

[2] Guide for ISO/IEC 12207 (Software Life Cycle Processes) // ISO/IEC JTC1/SC7 N1656 1997-02-01.

В таблице 1.1. подытожены характеристики каскадных моделей и преимущества, обеспечиваемые моделью с обратной связью.

Таблица 1.1. **Характеристики каскадной модели**

Характеристики	Преимущества
Последовательное упорядочение стадий	Применение формальных проверок позволяет своевременно выявить дефекты
Формальные проверки по завершении каждой стадии (инспекции, технические обзоры)	Четкие критерии начала и завершения стадий
Наличие документированных требований и проекта	Четкие требования и цели проекта

1.2.3. V-образная модель

V-образная (V-shape) модель расширяет каскадную модель включением в нее действий по *раннему планированию тестирования*. Структура и описание этой модели приведены в главе 7. Характеристики и преимущества модели перечислены в таблице 1.2.

Таблица 1.2. **Характеристики V-образной модели**

Характеристики	Преимущества
Проверка и оценка тестопригодности требований на ранних стадиях разработки (посредством анализа, выполняемого при планировании тестирования)	Обеспечивает обратную связь с пользователем на ранних стадиях ЖЦ
Наличие документированных тестовых требований	Улучшает планирование и распределение затрат на тестирование
	Четкие документированные цели тестирования

1.2.4. Каскадная модель с прототипированием (пилообразная модель)

Модель является модификацией V-образной модели с включением в нее прототипирования для моделирования требований и проекта (рисунок 1.3).

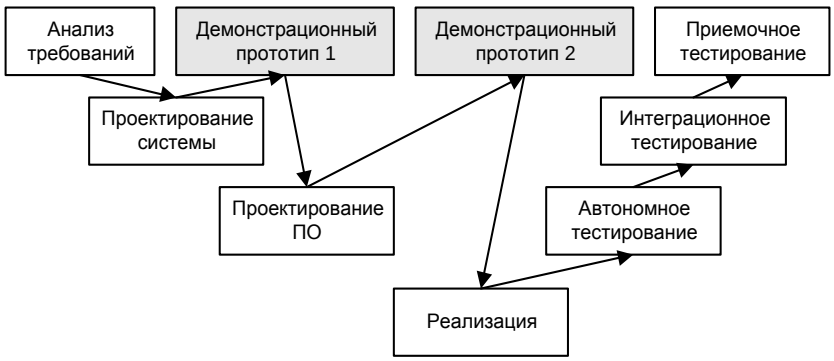


Рис. 1.3. **Последовательная модель с прототипированием**

Прототипы служат для демонстрации и после разработки проекта выбрасываются, а реализация проекта может выполняться в другой среде. Дополнительные преимущества этой модели, перечислены в таблице 1.3.

Таблица 1.3. **Характеристики модели с прототипированием**

Характеристики	Преимущества
Для анализа и моделирования проектных решений применяется прототипирование	Устраняет проблемы, связанные с неполнотой и нечеткостью требований

В таблице 1.4 перечислены основные риски, связанные с применением последовательных моделей и условия, при которых их лучше использовать.

Таблица 1.4. **Риски и условия применения последовательных моделей**

Риски, связанные с выбором модели	Когда лучше применять
Требования не полностью понятны	Требования понятны и не будут существенно изменяться
Система слишком большая, чтобы быть реализованной сразу	Разрабатываемая система имеет небольшой размер и сложность
Быстрые изменения в технологии	Все возможности должны быть реализованы сразу
Частое изменение требований	Новая система разрабатывается взамен старой и нужно полностью заменить старую систему
Пользователь не может использовать промежуточные продукты	

1.3. Итерационные модели

Итерационные модели в целом можно разделить на два класса: *модели с приращениями* (Incremental) и *эволюционные* (Evolutionary). В соответствии со всеми этими моделям программный продукт разрабатывается *итерациями*, и каждая итерация заканчивается выпуском работоспособной версии программного продукта. Основное отличие между моделями - в подходах к определению требований.

1.3.1. Итерационные модели с приращениями

По моделям с приращениями (Incremental) программный продукт разрабатывается итерациями, с добавлением на каждой итерации функциональных возможностей. При этом вначале определяются *все* требования к ПС, а возможно и разрабатывается предварительный проект. Дальнейшая разработка ПС разбивается на итерации. В каждой итерации разработка выполняется последовательно и завершается выпуском работоспособной версии программного продукта. В первой итерации реализуется набор основных требований, обеспечивающих базовую функциональность. Остальные итерации реализуются в порядке критичности требований для конечного пользователя. При появлении в середине итерации нового набора требований они откладываются до реализации следующей версии. В реальной жизни это допущение модели может нарушаться и допускается пересмотр требований.

В разных моделях этой группы итерации могут выполняться последовательно или с перекрытием (новая итерация начинается до завершения предыдущей итерации или когда первая стадия предыдущей итерации завершена примерно на 80%).

На рисунке 1.4 приведена структура модели с перекрытием итераций, а в таблице 1.5 - характеристики и преимущества моделей с приращениями.

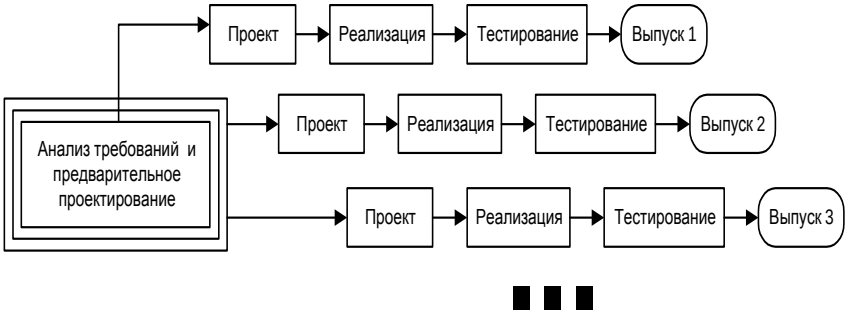


Рис. 1.4. **Итерационная модель с перекрытием итераций**

Итерационные модели с приращениями широко применяются для разработки коммерческих программных продуктов, которые развиваются в течение длительного периода времени или для которых внешние требования изменяются слабо.

Таблица 1.5. **Характеристики итерационных моделей с приращениями**

Характеристики	Преимущества
Анализ и проектирование выполняются для всей системы	Критические функции реализуются в первую очередь
Базовые функциональные требования реализуются первыми	Критические функции тестируются более тщательно
Остальные требования реализуются в последующих версиях	Наименее критические задачи реализуются в последнюю очередь, что минимизирует последствия отказов из-за дефектов
Промежуточные версии пригодны для использования	Завершение первой версии окончательно утверждает требования и проект
	Раннее планирование и выполнение тестирования
	Раннее выявление дефектов пользователями

В таблице 1.6 перечислены основные риски, связанные с применением моделей, и условия, при которых их лучше использовать.

Таблица 1.6. **Риски и условия применения моделей с приращениями**

Риски, связанные с выбором модели	Когда лучше применять
Требования не полностью понятны	Требуется быстрая реализация основных возможностей
Требования не стабильны	Если проект системы можно естественным образом разделить на независимые части
Все возможности должны быть реализованы сразу	
Быстрые изменения в технологии	

1.3.2. Эволюционные модели

В отличие от моделей с приращениями, эволюционные модели применяются в тех случаях, когда *все* требования не могут быть определены сразу или известно, что они могут измениться. Разработка проекта по этим моделям также выполняется итерациями, но каждая итерация охватывает все стадии разработки, от анализа вы-

бранного набора требований до выпуска версии. На каждой итерации выполняется прототипирование требований и проекта.

К наиболее известным эволюционным моделям в настоящее время относятся *спиральная модель* и *модель эволюционного прототипирования*.

Спиральная модель (Spiral) - разработана Б.Боэмом. Отражает управляемый риском процесс эволюции проекта от анализа до готовности продукта (рисунок 1.5).

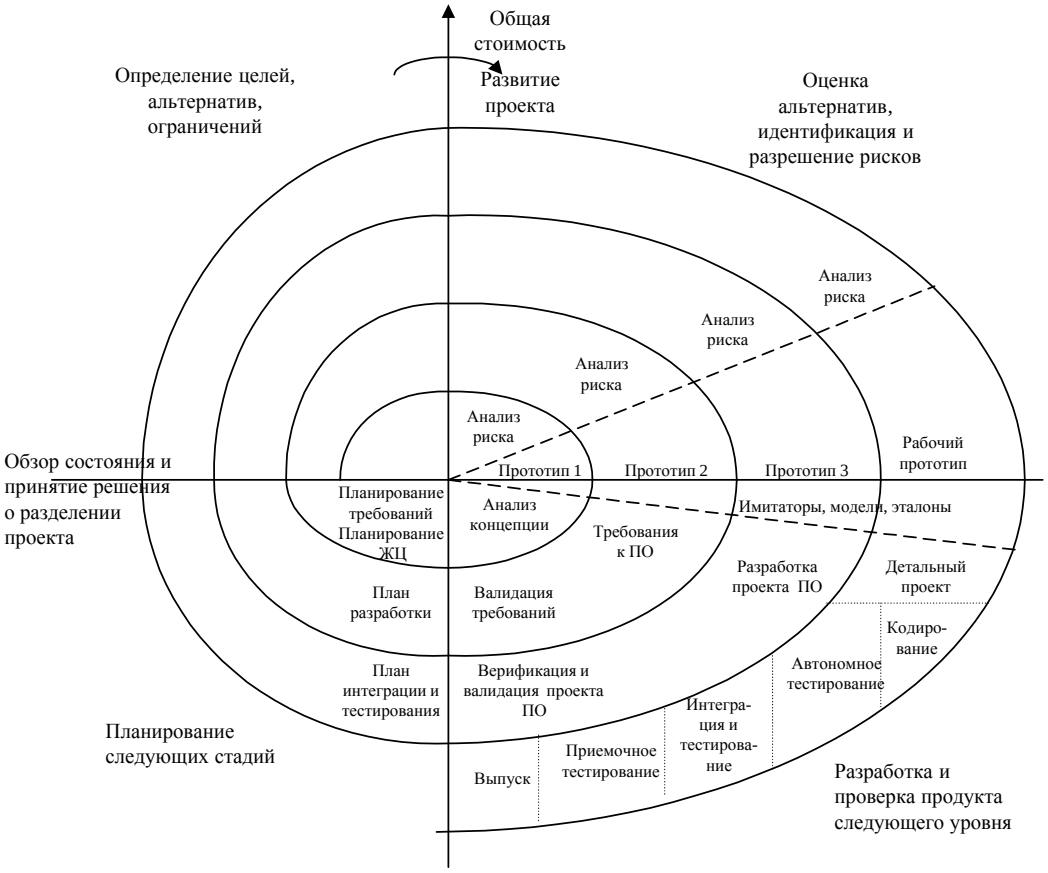


Рис. 1.5. **Спиральная модель ЖЦ**

На каждом витке спирали (стадии) выполняются следующие действия:

1. Определяются цели стадии. Рассматриваются альтернативные решения для достижения этих целей.
2. Проводится оценивание этих решений. Идентифицируются риски завершения стадии и выполняется их анализ. Принимаются решения о продолжении или завершении стадии.
3. Разрабатываются рабочие продукты стадии и план для следующей стадии.
4. Последний виток спирали может иметь структуру каскадной модели.

Виды рассматриваемых рисков - риски, касающиеся технических аспектов разработки, финансовые риски (соотношение эффективость/затраты и ресурсы), а также риски эксплуатации.

Спиральная модель применяется для сложных проектов или в тех случаях, когда проблемы проекта недостаточно понятны.

Характеристики и преимущества спиральной модели перечислены в таблице 1.7, а в таблице 1.8 - основные риски, связанные с применением модели, и условия, при которых ее лучше применять.

Таблице 1.7. **Характеристики спиральной модели**

Характеристики	Преимущества
Первый прототип моделирует концепцию. Результатом является план требований. Перед переходом к разработке следующего прототипа выполняется анализ риска.	Неопределенности в требованиях пользователя, требованиях к ПО и проекте моделируются до их реализации в коде
Второй прототип моделирует требования к ПО. Результатом является план разработки. Перед переходом к разработке следующего прототипа выполняется анализ риска.	Минимизируются ошибки, связанные с отсутствующими, недостаточно подробными или противоречивыми требованиями
Третий прототип моделирует проект. В результате создается интегрированный и протестированный прототип. Перед переходом к следующей стадии выполняется анализ риска.	Прототип сохраняется как физическая модель потребностей пользователя
Последний прототип (рабочий) используется как основа для детального проектирования, кодирования и тестирования.	Промежуточные версии пригодны для использования

Таблица 1.8. **Риски и условия применения спиральной модели**

Риски, связанные с выбором модели	Когда лучше применять
Все возможности должны быть реализованы сразу	Проект крупный, сложный и требования не могут быть определены сразу
Проект нельзя естественным образом разделить на независимые части	Новая технология и требуется ее освоение
	Проект можно естественным образом разделить на независимые части
	Пользователи не могут четко сформулировать требования
	Требуется ранняя демонстрация возможностей

Дальнейшим развитием этой модели является *Win-Win Spiral Model*, которая основана на привлечении к разработке разных категорий участников проекта и определении условий успеха (выигрыша) системы или подсистемы, которые обсуждаются на каждой итерации. Возможные альтернативные решения должны оцениваться по отношению к целям и ограничениям проекта.

Модель эволюционного прототипирования. Эта модель основана на применении *эволюционного прототипирования* в рамках всего ЖЦ разработки (а не только для моделирования требований). В литературе она часто называется моделью *быстрой разработки приложений* (RAD от Rapid Application Development). Моделирование включает следующие шаги:

1. Анализ применимости модели.
   Изучение возможности применения модели для проекта.
2. Обследование заказчика.
   Изучение потребностей пользователя и разработка плана создания прототипа.
3. Итерация разработки функционального прототипа.
   Создание и согласование прототипа интерфейса пользователя, определение нефункциональных требований и стратегии реализации системы.
4. Итерация проектирования и построения.
   Построение протестированной системы, удовлетворяющей всем функциональным и нефункциональным требованиям.
   На шагах 3 и 4 разработчики выполняют определение прототипов, согласование сроков разработки, построение и проверку прототипов. Эти шаги выполняются итеративно и включают три итерации: начальное ознакомление, уточнение и согласование.
5. Реализация.
   Установка системы в среде заказчика, разработка документации и обучение.

Эта модель применяется для разработки не критических бизнес-приложений, для которых наиболее важными являются функциональные возможности. Ее применение предполагает тесное взаимодействие разработчика и пользователя. Разработка приложений обычно выполняется в среде мощных CASE-средств.

Структура модели представлена на рисунке 1.6, а таблице 1.9 перечислены основные характеристики и преимущества модели. В таблице 1.10 перечислены основные риски, связанные с применением модели, и условия, при которых ее лучше применять.

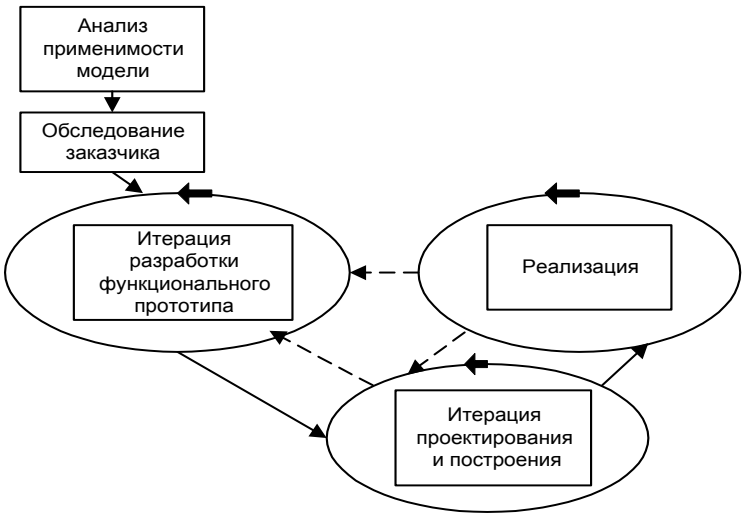


Рис. 1.6. **Модель эволюционного прототипирования**

Таблица 1.9. **Характеристики модели**

Характеристики	Преимущества
Гибкость. Возможность быстро реагировать на изменения и расширения требований	Раннее выявление дефектов в интерфейсе
Приоритеты функциональных характеристик перед техническими (качества)	Быстрая демонстрация функциональных возможностей

Таблица 1.10. **Риски и условия применения модели**

Риски, связанные с выбором модели	Когда лучше применять
От разработчика требуется хорошее владение CASE- методами и инструментами	Пользователи не могут четко сформулировать требования
Разрабатываемое приложение должно быть не критичным	Требуется ранняя демонстрация возможностей
Требуется наличие мощных CASE-средств	

Приложение 2. ПРИМЕРЫ МЕТРИК В ЭТАЛОННОЙ МОДЕЛИ КАЧЕСТВА

Метрики функциональности. В таблице 2.1 представлены внутренние, а в таблице 2.2 - внешние метрики качества для характеристики «функциональность».

Таблица 2.1. **Внутренние метрики характеристики «функциональность»**

Название	Описание
Функциональная пригодность	
Функциональная адекватность	**X = 1 – A/B** **A** – число функций, определение (описание) которых не отвечает поставленным задачам ПС и могут повлечь проблемы при реализации **B** – число проверенных функций **Источники данных:** Спецификация требований, проект, код, отчет о проверке. Виды проверки описаны в главе 5. **Интерпретация значений:** 0<=X<=1 (ближе к 1 – лучше)
Полнота функциональной реализации	**X = 1 – A/B** **A** – число обнаруженных пропущенных функций **B** – число функций, описанных в спецификации требований **Источники данных и интерпретация значений:** *здесь и далее - те же Примечание.* Спецификация требований должна быть всегда актуальной
Корректность функциональной реализации	**X = 1 – A/B** **A** – число обнаруженных некорректно реализованных или пропущенных функций **B** – число функций, описанных в спецификации требований
Стабильность функциональной спецификации	**X = 1 – A/B** **A** – число функций, которые изменялись в ходе ЖЦ разработки **B** – число функций, описанных в спецификации требований
Точность	
Тщательность реализации-функций	**X = A/B** **A** – число функций, для которых указанные в спецификации требования к точности были полностью реализованы **B** – число функций, для которых требования к точности были установлены в спецификации требований
Точность вычисления данных	**X = A/B** **A** – число элементов данных, для которых при реализации обеспечивается специфицированный уровень точности вычислений **B** – число элементов данных, для которых в спецификациях установлен уровень точности вычислений
Способность к взаимодействию	
Способность к обмену данными (основанному на формате)	**X = A/B** **A** – число форматов интерфейсных данных, реализация которых корректна по отношению к специфицированным требованиям **B** – число форматов данных (предназначенных для обменов), установленных в спецификациях требований

Название	Описание
Согласованность интерфейса (протокол)	**X = A/B** **A** – число протоколов интерфейса, реализующих формат, установленный в спецификации требований **B** – число протоколов интерфейса, которые должны быть реализованы в соответствии со спецификацией требований
Защищенность	
Наблюдаемость доступа	**X = A/B** **A** – число видов доступа, обеспечивающих вход в систему способом, предусмотренным в спецификации требований **B** – число видов доступа, предусмотренных в спецификации требований
Контролируемость доступа	**X = A/B** **A** – число требований к контролю доступа и выполнения легальных операций, реализованных корректно по отношению к спецификации требований **B** – число требований к контролю доступа, установленных в спецификации требований
Степень предупреждения повреждений	**X = A/B** **A** – число обработчиков для предупреждения повреждения данных, реализованных корректно по отношению к спецификации требований **B** – число способов выполнения работы/доступа, идентифицированных в требованиях как такие, которые могут привести к повреждению/разрушению данных *Примечание*: с учетом необходимых уровней защиты
Шифрование данных	**X = A/B** **A** – число предусмотренных обработчиков шифрования/дешифрования элементов данных, реализованных корректно по отношению к спецификации требований **B** – число элементов данных, по отношению к которым должно быть применено шифрование/дешифрование, как того требует спецификация *Примечание*: шифрование – для данных в открытых базах данных, данных в общедоступных коммуникациях
Соответствие нормам и правилам, касающимся обеспечения функциональности ПС	
Функциональное соответствие	**X = A/B** **A** – число корректно реализованных элементов, к которым предъявляются требования функционального соответствия действующим нормам и правилам (стандартам, соглашениям), выявленных при проверке **B** – общее число элементов, по отношению к которым установлены нормы и правила функционального соответствия
Соответствие стандартам интерфейса	**X = A/B** **A** – число корректно реализованных интерфейсов по отношению к принятым нормам и правилам для системы **B** – общее число интерфейсов, требующих согласования с принятыми для системы нормами и правилами

Таблица 2.2. **Внешние метрики характеристики «функциональность»**

Название	Описание
Функциональная пригодность	
Функциональная адекватность	**X = 1 – A/B** **A** – число функций, реализация которых не отвечает задачам ПС **B** – число оцениваемых функций ПС **Источники данных:** Спецификация требований, отчет об оценивании **Интерпретация значений:** 0<=X<=1 (ближе к 1 – лучше)
Полнота функциональной реализации	**X = 1 – A/B** **A** – число пропущенных функций, обнаруженных при оценивании ПС **B** – число функций, описанных в спецификации требований *Примечание.* Спецификация требований должна быть всегда актуальной. Каждая функция из спецификации подвергается функциональному тестированию (см. главу 6) **Источники данных и интерпретация значений:** *здесь и далее - те же*
Корректность функциональной реализации	**X = 1 – A/B** **A** – число некорректно реализованных и пропущенных функций, обнаруженных при оценивании ПС **B** – число функций, описанных в спецификации требований
Стабильность функциональной спецификации	**X = 1 – A/B** **A** – число функций, которые изменялись с момента ввода ПС в действие **B** – число функций, описанных в спецификации требований
Точность	
Ожидаемая точность реализации функций	**X = A/T** **A** – число неприемлемых отклонений результатов выполнения функций от *ожидаемых*, зафиксированное пользователями ПС **T** – время тестирования (использования) ПС *Примечание:* Выполняется путем прогона тестов и сравнения ожидаемых и реально полученных результатов **Источники данных:** Спецификация требований, руководство пользователя, мнение пользователя, отчет о тестировании. **Интерпретация значений:** 0<=X (ближе к 0 – лучше)
Тщательность реализации функций	**X = A/T** **A** – зафиксированное пользователями число случаев неправильного выполнения функций по отношению к спецификации требований **T** – время использования ПС **Источники данных:** Спецификация требований, отчеты о проблемах **Интерпретация значений:** 0<=X (ближе к 0 – лучше)
Точность вычисления данных	**X = A/T** **A** – зафиксированное пользователями число результатов вычислений, не отвечающих требованиям уровня точности **T** – время использования ПС **Источники данных и интерпретация значений:** те же
Способность к взаимодействию	
Способность к обмену данными	**X = A/B** **A** – число форматов интерфейсных данных, успешно участвующих в обменах с другим ПО или системами при тестировании

Название	Описание
(основанному на формате данных)	**B** – общее число форматов данных, которые должны участвовать в обменах *Примечание:* выполняется путем прогона тестов для проверки транзакций данных **Источники данных:** Спецификация требований, руководство пользователя, отчет о тестировании. **Интерпретация значений:** 0<=X<=1 (ближе к 1 – лучше)
Способность к обмену данными (основанному на успешных попытках пользователя)	**X = 1 - A/B** **A** – зафиксированное пользователем число случаев отказа функций при попытках обмена данными с другим ПО или системами **B** – число попыток пользователя выполнить функции обмена данными **Y = A/T** **T** – период времени использования ПС **Источники данных:** те же **Интерпретация значений:** 0<=X<=1 (ближе к 1 – лучше), 0<=Y (ближе к 0 – лучше)
Защищенность	
Наблюдаемость доступа	**X = A/B** **A** – число «доступов пользователя к системе или данным», зафиксированных в базе данных истории доступов. **B** – число «доступов пользователя к системе или данным», произведенных в ходе оценивания *Примечание:* выполняется путем прогона тестов, имитирующих атаки на ПС, внесение вирусов и др. **Источники данных:** тестовые спецификации, отчет о тестировании **Интерпретация значений:** 0<=X<=1 (ближе к 1 – лучше)
Контролируемость доступа	**X = A/B** **A** – число обнаруженных различных типов нелегальных операций **B** – число типов нелегальных операций, требования к контролю которых установлены в спецификации **Источники данных:** тестовые спецификации, отчет о тестировании, отчет о проблемах при эксплуатации **Интерпретация значений:** 0<=X<=1 (ближе к 1 – лучше)
Частота повреждения данных	а) **X = 1 - A/N** **A** – число случаев критического повреждения данных (данные нельзя восстановить, вторичные последствия утраты данных и др.) **N** – число тестовых ситуаций, в которых предпринимались попытки повредить данные б) **Y = 1 - B/N** **B** – число случаев не критического повреждения данных в) **Z = A/T** или **B/T** **T** – период времени функционирования (в течение периода тестирования) **Источники данных:** те же **Интерпретация значений:** 0<=X<=1 (ближе к 1 – лучше), 0<=X<=1 (ближе к 1 – лучше), 0<=Z (ближе к 0 – лучше)

Название	Описание
Соответствие нормам и правилам, касающимся обеспечения функциональности ПС	
Функциональное соответствие	**X = 1 - A/B** **A** – число элементов функциональности, которые не отвечают требованиям действующих норм и правил (стандартов, соглашений), выявленных при функциональном тестировании **B** – общее число элементов, по отношению к которым установлены нормы и правила функционального соответствия **Источники данных:** описание продукта (руководство пользователя), применимые стандарты, директивы и др., тестовые спецификации, отчеты о тестировании **Интерпретация значений:** 0<=X<=1 (ближе к 1 – лучше)
Соответствие стандартам интерфейса	**X = A/B** **A** – число корректно реализованных интерфейсов **B** – общее число интерфейсов, требующих согласования с принятыми для системы нормами и правилами **Источники данных и интерпретация значений:** те же

Метрики завершенности. В таблице 2.3 дано описание внутренних и внешних метрик подхарактеристики надежности «*завершенность*»

Таблица 2.3. **Метрики подхарактеристики *завершенность***

Название метрики	Измеряемые величины	Обозначение	Формула и данные	Интерпретация
Внутренние метрики «завершенности»				
Интенсивность выявление дефектов	Количество найденных дефектов	x_1	$F = \frac{x_1}{M(x)}$	$F \geq 0$ - высокое качество проверки $x_1=0$ *не* гарантирует отсутствия дефектов
	Ожидаемое количество дефектов	$M(x)$		
Интенсивность устранение дефектов	Количество откорректированных дефектов в рабочем продукте (проект, код)	x_2	$F = x2$	$F \geq 0$ - чем более F, тем меньше осталось дефектов
	Часть (%) откорректированных дефектов в количестве выявленных		$F = \frac{x_2}{x_1}$	$0 \leq F \leq 1$ - чем ближе до 1, тем лучше
Точность проверок	Количество категорий дефектов, запланированных к проверке, в плане проверки	$x3$	$F = \frac{x_3}{N(x)}$	$F \geq 0$ - чем более F, тем выше точность проверки
	Количество категорий дефектов, которые нужно охватить проверкой для обеспечения надлежащего объема проверки	$N(x)$		

Название метрики	**Измеряемые величины**	**Обозначение**	**Формула и данные**	**Интерпретация**
Внешние метрики «завершенности»				
Оценка плотности скрытых дефектов	Общее ожидаемое количество дефектов в ПП	*y1*	$F = \frac{\lvert y_1 - y_2 \rvert}{V}$	$F \geq 0$ - в зависимости от стадии проверки. На более поздних стадиях – чем меньше, тем лучше
	Общее количество реально выявленных дефектов в ПП	*y2*		
	Размер (объем) ПП	$\underline{V}$		
Плотность отказов (относительно тестовых ситуаций)	Количество событий отказов (за период)	*y3*	$F = \frac{y_3}{y_4}$	$F \geq 0$ - в зависимости от стадии тестирования. На более поздних стадиях – чем меньше, тем лучше
	Количество выполненных тестов	*y4*		
Плотность дефектов	Общее количество реально выявленных дефектов в ПП	*y2*	$F = \frac{y_2}{V}$	$F \geq 0$ - в зависимости от стадии тестирования. На более поздних стадиях – чем меньше, тем лучше
	Размер ПП	$\underline{V}$		
Интенсивность устранение дефектов	Количество откорректированных дефектов в ПП при тестировании	*y6*	$F = \frac{y_6}{y_2}$	$0 \leq F \leq 1$ - чем ближе до 1, тем лучше, меньше дефектов осталось
	Общее количество реально выявленных дефектов в ПП	*y2*	$F = \frac{y_6}{y_1}$	$F \geq 0$ - чем ближе до 1, тем лучше, меньше дефектов осталось
	Общее ожидаемое количество дефектов в ПП	*y1*		
Среднее время между отказами	Период операции	*T1*	$F_1 = \frac{T_1}{A}$ $F_1 = \frac{T_2}{A}$	*F1, F2 > 0* - чем дольше, тем лучше, более длинный период между отказами
	Сумма по периодам безотказной работы	*T2*		

Приложение 3. СТАНДАРТЫ В ОБЛАСТИ ИНЖЕНЕРИИ КАЧЕСТВА

В таблице 3.1 перечислены украинские (национальные) стандарты в области качества ПС (вкючая стандарты, гармонизированные с международными). Электронный каталог стандартов ДСТУ можно найти по адресу в Интернете[1]:

http://www.csm.kiev.ua/new/main/CatalogGost/main.aspx

Таблица 3.1. **Основные национальные стандарты в области качества**

Обозначение	Наименование
Системы качества	
ДСТУ ISO 9000-2001	Системи управління якістю. Основні положення та словник
ДСТУ ISO 9001-2001	Системи управління якістю. Вимоги
ДСТУ ISO 9004-2001	Системи управління якістю. Настанови щодо поліпшення діяльності
ДСТУ ISO 10011-1-97	Настанови щодо перевірки систем якості.Частина 1. Перевірка (ISO 10011-1:1990)
ДСТУ ISO 10011-2-97	Настанови щодо перевірки систем якості.Частина 2. Кваліфікаційні вимоги до аудиторів з систем якості (ISO 10011-2:1991)
ДСТУ ISO 10011-3-97	Настанови щодо перевірки систем якості.Частина 3. Управління програмами перевірок (ISO 10011-3:1991)
ДСТУ ISO/TR 10013-2003	Настанови з розробляння документації системи управління якістю (ISO/TR 10013:2001, IDT)
ДСТУ ISO 19011-2003	Настанови щодо здійснення аудитів систем управління якістю і (або) екологічного управління (ISO 19011:2002, IDT)
Информационные технологии	
ДСТУ 3918-1999	Інформаційні технології. Процеси життєвого циклу програмного забезпечення (ISO/IEC 12207-1995)
ДСТУ 3919-1999	Інформаційні технології. Основні напрямки оцінювання та відбору CASE-інструментів (ISO/IEC 14102-1995)
ДСТУ 4302-2004	Інформаційні технології. Настанови щодо документування комп'ютерних програм (ISO/IEC 6592:2000, MOD[2])
ДСТУ ISO/IEC 12119-2003	Інформаційні технології. Пакети програм. Тестування і вимоги до якості (ISO/IEC 12119:1994, IDT)
ДСТУ ISO/IEC TR 12182-2004	Інформаційні технології. Класифікація програмних засобів (ISO/IEC TR 12182:1998, IDT)
ДСТУ ISO/IEC 14598-1-2004	Інформаційні технології. Оцінювання програмного продукту. Частина 1. Загальний огляд (ISO/IEC 14598-1:1999, IDT)
ДСТУ ISO/IEC 14764-2002	Інформаційні технології. Супроводження програмного забезпечення (ISO/IEC 14764:1999, IDT)
ДСТУ ISO/IEC TR 15504-1-2002	Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 1. Концепції та вступна настанова (ISO/IEC TR 15504-1:1998, IDT)

[1] Для уточнения ссылки искать «Каталог нормативних документів»

[2] Степень гармонизации стандарта: MOD- модифицирован, IDT- идентичен

Обозначение	Наименование
ДСТУ ISO/IEC TR 15504-2-2002	Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 2. Еталонна модель процесів та потужності процесу (ISO/IEC TR 15504-2:1998, IDT)
ДСТУ ISO/IEC TR 15504-3-2002	Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 3. Виконання оцінювання (ISO/IEC TR 15504-3:1998, IDT)
ДСТУ ISO/IEC TR 15504-4-2002	Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 4. Настанови з виконання оцінювання (ISO/IEC TR 15504-4:1998, IDT)
ДСТУ ISO/IEC TR 15504-5-2002	Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 5. Модель оцінювання та настанови щодо показників (ISO/IEC TR 15504-5:1999, IDT)
ДСТУ ISO/IEC TR 15504-6-2003	Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 6. Настанови з визначання компетентності оцінювачів (ISO/IEC TR 15504-6:1998, IDT)
ДСТУ ISO/IEC TR 15504-7-2003	Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 7. Настанови з удосконалювання процесу (ISO/IEC TR 15504-7:1998, IDT)
ДСТУ ISO/IEC TR 15504-8-2003	Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 8. Настанови з визначання потужності процесу постачальника (ISO/IEC TR 15504-8:1998, IDT)
ДСТУ ISO/IEC TR 15504-9-2003	Інформаційні технології. Оцінювання процесів життєвого циклу програмних засобів. Частина 9. Словник термінів (ISO/IEC TR 15504-9:1998, IDT)
Программные средства	
ДСТУ 2844-94	Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення
ДСТУ 2850-94	Програмні засоби ЕОМ. Показники і методи оцінювання якості
ДСТУ 2851-94	Програмні засоби ЕОМ. Документування результатів випробувань
ДСТУ 2853-94	Програмні засоби ЕОМ. Підготовлення і проведення випробувань
ДСТУ ГОСТ 31078-2004	Захист інформації. Випробування програмних засобів на наявність комп'ютерних вірусів. Типова настанова (ГОСТ 31078-2002, IDT)

В таблице 3.2 перечислены действующие международные стандарты ISO/IEC и проекты стандартов в области управления качеством и программной инженерии.

Таблица 3.2. **Международные стандарты и проекты стандартов ISO/IEC в области инженерии качества**

Обозначение	Наименование (англ.)	Наименование (рус.)
ISO 9000:2000	Quality management systems - Fundamentals and vocabulary	Системы управления качеством. Основные положения и словарь
ISO 9001:2000	Quality management systems - Requirements	Системы управления кочеством. Требования

Обозначение	Наименование (англ.)	Наименование (рус.)
ISO 9004:2000	Quality management systems – Guidelines for performance improvements	Системы управления качеством. Руководящие указания по улучшению деятельности
ISO 10005:1995	Quality management - Guidelines for quality plans	Управление качеством. Руководящие указания по планам качества
ISO 10006:1997	Quality management - Guidelines to quality in project management	Управление качеством. Руководящие указания по обеспечению качества при управлении проектом
ISO 10007:1995	Quality management - Guidelines for configuration management	Управление качеством. Руководящие указания по управлению конфигурацией
ISO 10011-1:1990	Guidelines for auditing quality systems - Part 1: Auditing	Руководящие указания по аудиту систем качества. Часть 1. аудиторская проверка
ISO 10011-2:1991	Guidelines for auditing quality systems - Part 2: Qualification criteria for quality systems auditors	Руководящие указания по аудиту систем качества. Часть 2. Квалификационные требования к аудиторам систем качества
ISO 10011-3:1991	Guidelines for auditing quality systems - Part 3: Management of audit programmes	Руководящие указания по аудиту систем качества. Часть 3. Управление программами аудита
ISO 10013:1995	Guidelines for developing quality manuals	Руководящие указания по разработке руководств по качеству
ISO/TR 10014:1998	Guidelines for managing the economics of quality	Руководящие указания по управлению экономическими аспектами качества
ISO 10015:1999	Quality management – Guidelines for training	Управление качеством. Руководящие указания по подготовке
ISO/TR 10017:1999	Guidelines on statistical techniques for ISO 9001:1994	Руководящие указания по статистическим методам
ISO/IEC 90003:2004	Software engineering (SE^3) - Guidelines for application of ISO 9001:2000 to computer software	Программная инженерия (ПЕ)- Руководящие указания по применению ISO 9001:2000 к компьютерному ПО
ISO/IEC 2382-7 :2000	Information Technology (IT) - Vocabulary - Part 7: Computer programming	Информационные технологии (ИТ) - Словарь – Часть 7: Программирование для компьютера
ISO/IEC 2382-20 :2000	IT - Vocabulary - Part 20: System development	ИТ – Словарь – Часть 20: Разработка системы
ISO/IEC 6592:2000	IT - Guidelines for the documentation of computer-based application systems	ИТ – руководящие указания по документированию компьютерных прикладных систем

[3] Далее используются сокращения – SE- Software Engineering, ПЕ- Программная Инженерия, IT- Information Technology, ИТ- Информационные технологии

Обозначение	Наименование (англ.)	Наименование (рус.)
ISO/IEC 9126-1: 2001	SE - Product quality - Part 1: Quality model	ПЕ – Качество продукта – Часть 1: Модель качества
ISO/IEC TR 9126-2: 2003	SE - Product quality - Part 2: External metrics	ПЕ – Качество продукта – Часть 2: Внешние метрики
ISO/IEC TR 9126-3: 2003	SE - Product quality - Part 3: Internal metrics	ПЕ – Качество продукта – Часть 3: Внутренние метрики
ISO/IEC TR 9126-4: 2004	SE - Product quality - Part 4: Quality in use metrics	ПЕ – Качество продукта – Часть 4: Качество при использовании
ISO/IEC TR 9294:2005	IT - Guidelines for the management of software documentation	ИТ – Руководящие указания по управлению документацией ПО
ISO/IEC 12119 :1994	IT – Software packages – Quality requirements and testing	ИТ – Пакеты программ – Требования к качеству и тестирование
ISO/IEC TR 12182 :1998	IT - Categorization of software	ИТ – Классификация ПО
ISO/IEC 12207:1995 /Amd 1:2002 /Amd 2:2004	IT – Software life cycle processes	ИТ – Процессы ЖЦ ПО
ISO/IEC 14102:1995	IT – Guideline for the evaluation and selection CASE tools	ИТ- Руководство по оценке и выбору CASE-инструментов
ISO/IEC 14143-1 :1998	IT - Software measurement - Functional size measurement - Part 1: Definition of concepts	ИТ – Измерение ПО – Измерение объема (размера) функциональных возможностей – Часть 1: Определение понятий
ISO/IEC 14143-2 :2002	IT - Software measurement - Functional size measurement - Part 2: Conformity evaluation of software size measurement methods to ISO/IEC 14143-1:1998	ИТ – Измерение ПО – Измерение объема функциональных возможностей – Часть 2: Оценивание соответствия методов измерения размера ПО требованиям ISO/IEC 14143-1
ISO/IEC 14143-3 :2003	IT - Software measurement - Functional size measurement - Part 3: Verification of functional size measurement methods	ИТ – Измерение ПО – Измерение объема функциональных возможностей – Часть 3: Верификация методов измерения размера ПО
ISO/IEC 14143-4 :2005	IT - Software measurement - Functional size measurement - Part 4: Reference model	ИТ – Измерение ПО – Измерение объема функциональных возможностей – Часть 4: Эталонная модель
ISO/IEC 14143-5 :2004	IT - Software measurement - Functional size measurement - Part 5: Determination of functional domains for use with functional size measurement	ИТ – Измерение ПО – Измерение объема функциональных возможностей – Часть 5: Определение функциональных доменов, в которых могут использоваться методы измерения функционального размера

Обозначение	Наименование (англ.)	Наименование (рус.)
ISO/IEC 14471:1999	IT - SE - Guidelines for the adoption of CASE tools	ИТ – ПЕ – Руководящие указания по адаптации CASE-инструментов
ISO/IEC 14598-1 :1999	IT - Software product evaluation - Part 1: General overview	ИТ – Оценка программного продукта - Часть 1: Общий обзор
ISO/IEC 14598-2 :2000	SE - Product evaluation - Part 2: Planning and management	ПЕ – Оценка продукта – Часть 2: Планирование и управление
ISO/IEC 14598-3:2000	SE - Product evaluation - Part 3: Process for developers	ПЕ – Оценка продукта – Часть 3: Процесс для разработчиков
ISO/IEC 14598-4 :1999	SE - Product evaluation - Part 4: Process for acquirers	ПЕ – Оценка продукта – Часть 4: Процесс для заказчиков (потребителей)
ISO/IEC 14598-5:1998	IT - Software product evaluation - Part 5: Process for evaluators	ПЕ – Оценка программного продукта – Часть 5: Процесс для оценщиков
ISO/IEC 14598-6:2001	SE - Product evaluation - Part 6: Documentation of evaluation modules	ПЕ – Оценка продукта – Часть 6: Документация модулей оценивания
ISO/IEC 14756:1999	IT – Measurement and rating of performance of computer-based software systems	ИТ – Измерение и ранжирование производительности компьютерных ПС
ISO/IEC 14764:1999	IT – Software maintenance	ИТ – Сопровождение ПО
ISO/IEC 15026:1998	IT - System and software integrity levels	ИТ – Уровни целостности системы и ПО
ISO/IEC 15271:1998	IT - Guide for ISO/IEC 12207	ИТ – Руководство по ISO/IEC 12207
ISO/IEC 15288:2002	Systems engineering -- System life cycle processes	Системная инженерия - процессы жизненного цикла систем
ISO/IEC 15504-1:2004	IT -Process assessment - Part 1: Concepts and vocabulary	ИТ – Оценивание процесса – Часть 1: Концепции и словарь
ISO/IEC 15504-2:2003 / Cor 1:2004	IT - Process assessment - Part 2: Performing an assessment	ИТ – Оценивание процесса – Часть 2: Выполнение оценивания
ISO/IEC 15504-3:2004	IT - Process assessment - Part 3: Guidance on performing an assessments	ИТ – Оценивание процесса – Часть 3: Руководство по выполнению оцениваний
ISO/IEC 15504-4:2004	IT - Process assessment - Part 4: Guidance on use for process improvement and process capability determination	ИТ – Оценивание процесса – Часть 4: Руководство по использованию для совершенствования процесса и определения мощности процесса
ISO/IEC 15504-5:2006	IT - Process assessment - Part 5: An exemplar Process Assessment Model	ИТ – Оценивание процесса – Часть 5: Пример Модели оценивания процесса
ISO/IEC TR 15846: 1998	IT - Software life cycle processes - Configuration Management	ИТ – Процессы ЖЦ ПО – Управление конфигурацией

Обозначение	Наименование (англ.)	Наименование (рус.)
ISO/IEC 15910:1999	IT – Software user documentation process	ИТ – Процесс подготовки документации пользователя ПО
ISO/IEC 15939:2002	SE – Software Measurement Process	ПЕ – Процесс измерения ПО
ISO/IEC 16085:2004	IT - Software life cycle processes - Risk management	ИТ – Процессы ЖЦ ПО - Управление риском
ISO/IEC TR 16326 :1999	SE - Guide for the application of ISO/IEC 12207 to project management	ПЕ – Руководство по применению ISO/IEC 12207 при управлении проектом
ISO/IEC 18019:2004	Software and system engineering -- Guidelines for the design and preparation of user documentation for application software	Инженерия систем и ПО - Руководящие указания по разработке и подготовке пользовательской документации для прикладного ПО
ISO/IEC TR 19759 :2005	SE - Guide to the Software Engineering Body of Knowledge (SWEBOK)	ПЕ – Руководство к Ядру знаний по программной инженерии (SWEBOK)
ISO/IEC TR 19760 :2003	SE - A guide for the application of ISO/IEC 15288 (System life cycle processes)	Системна яинженерия - Руководство по применению ISO/IEC 15288 (Процессы ЖЦ систем)
ISO/IEC 19761:2003	SE - COSMIC-FFP - A functional size measurement (FSM)method	ПЕ - COSMIC-FFP – Метод измерения функционального размера (ИФР) ПО
ISO/IEC 20926:2003	SE - IFPUG 4.1 Unadjusted FSM method - Counting practices manual	ПЕ - IFPUG 4.1 Метод ИФР без учета сложности требований. Руководство по расчету
ISO/IEC 20968:2002	SE - Mk II Function Point Ana-lysis (FPA) - Counting Practices Manual	ПЕ – Метод Mk II FPA – Руководство по расчету
ISO/IEC 23026:2006	SE - Recommended Practice for the Internet - Web Site Engineering, Web Site. Management, and Web Site Life Cycle	ПЕ - Рекомендуемая практика для сети Интернет. Разработка. web-сайта, администрирование web-сайта и жизненный цикл web-сайта
ISO/IEC 24570:2005	SE - NESMA FSM- method version 2.1 - Definitions and counting guidelines for the application of FPA	ПЕ – NESMA Метод ИФР, версия 2.1 – Определения и руководство по расчету с применением FPA
ISO/IEC 25000:2005	SE - Software product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE	ПЕ – Требования к качеству ПО и его оценивание (SQuaRE). Руководство
ISO/IEC 25051:2006	SE - Software product Quality Requirements and Evaluation (SQuaRE) - Requirements for quality of COTS software product and instructions for testing	ПЕ – Требования к качеству ПО и его оценивание (SQuaRE). Требования к качеству коммерческого ПО и инструкции по тестированию

Обозначение	Наименование (англ.)	Наименование (рус.)
ISO/IEC 25062:2006	SE - Software product Quality Requirements and Evaluation (SQuaRE) - Common Industry Format (CIF) for usability test reports	ПЕ – Требования к качеству ПО и его оценивание (SQuaRE). Общий Промышленный Формат (CIF) для отчетов по тестированию удобства применения

В таблице 3.3 указаны стандарты Института инженеров по электротехнике и радиоэлектронике (IEEE).

Таблица 3.3. **Стандарты IEEE**

Обозначение	Наименование (англ.)	Наименование (рус.)
IEEE Std. 610.12 :1990	IEEE Standard Glossary of Software Engineering Terminology	Глоссарий терминологии по программной инженерии
IEEE Std. 1062:1998	IEEE Recommended Practice for Software Acquisition	Рекомендованные приемы приобретения ПО
IEEE Std. 1220:1998	IEEE Standard for the Application and Management of the System Engineering Process	Применение и управление процесса системной инженерии
IEEE Std. 1228:1994	IEEE Standard for Software Safety Plans	Планы по обеспечению безопасности ПО
IEEE Std. 1233:1998	IEEE Guide for Developing System Requirements Specifications	Руководство по разработке спецификаций требований к системе
IEEE/EIA Std. 12207.0 :1996	Software life cycle processes	Процессы ЖЦ ПО
IEEE/EIA Std. 12207.1 :1997	Software life cycle processes - Life cycle data	Процессы ЖЦ ПО – Данные ЖЦ ПО
IEEE/EIA Std. 12207.2 :1997	Software life cycle processes - Implementation consideration	Процессы ЖЦ ПО – Вопросы реализации
IEEE Std. 730:1998	IEEE Standard for Software Quality Assurance Plans	Планы обеспечения гарантии качества ПО
IEEE Std. 730.1:1995	IEEE Guide for Software Quality Assurance Planning	Руководство по планированию обеспечения гарантии качества ПО
IEEE Std. 828:1998	IEEE Standard for Software Configuration Management Plans	Планы управления конфигурацией ПО
IEEE Std. 829:1998	IEEE Standard for Software Test Documentation	Документация тестирования ПО
IEEE Std. 830:1998	IEEE Recommended Practice for Software Requirements Specification	Рекомендованные приемы спецификации требований к ПО
ANSI/IEEE Std. 1008 :1987	IEEE Standard for Software Unit Testing	Автономное тестирование ПО
IEEE Std. 1012:1998	IEEE Standard for Software Verification and Validation	Верификация и валидация ПО

Обозначение	Наименование (англ.)	Наименование (рус.)
IEEE Std. 1012a :1998	Supplement to IEEE Standard for Software Verification and Validation: Content Map to IEEE/EIA 12207.1-1997	Дополнение к стандарту IEEE по верификации и валидации: отображение содержания на IEEE/EIA 12207.1-1997
IEEE Std. 1016:1998	IEEE Recommended Practice for Software Design Description	Рекомендованные приемы описания проекта ПО
IEEE Std. 1028:1997	IEEE Standard for Software Reviews	Обзоры ПО
ANSI/IEEE Std. 1042 :1987	IEEE Guide to Software Configuration Management	Руководство по управлению конфигурацией
IEEE Std. 1044:1993	IEEE Standard Classification for Software Anomalies	Классификация аномалий ПО
IEEE Std. 1044-1 :1995	IEEE Guide to Classification for Software Anomalies	Руководство по классификации аномалий ПО
IEEE Std. 1045:1992	IEEE Standard for Software Productivity Metrics	Метрики производительности ПО
IEEE Std. 1058:1998	IEEE Standard for Software Project Management Plans	Планы управления проектом ПО
IEEE Std. 1059:1993	IEEE Guide for Software Verification and Validation Plans	Руководство по планам верификации и валидации ПО
IEEE Std. 1061:1998	IEEE Standard for a Software Quality Metrics Methodology	Методология измерения качества ПО
IEEE Std. 1063:1987	IEEE Standard for Software User Documentation	Документация пользователя ПО
IEEE Std. 1074:1997	Standard for Developing Software Life Cycle Processes	Разработка процессов ЖЦ ПО
IEEE Std. 1219:1998	IEEE Standard for Software Maintenance	Сопровождение ПО

В таблице 3.4 представлены некоторые ведомственные стандарты и руководства в области программной инженерии.

Таблица 3.4. **Другие стандарты и руководства**

Обозначение	Наименование (англ.)	Наименование (рус.)
DEF STAN 0055 :1997	Requirements for Safety Related Software in Defence Equipment	Стандарт Министерства Обороны Великобритании. Требования к ПО обеспечения безопасности военного оборудования (www.dstan.mod.uk)
MIL Std. 498 :1994	Software Development and Documentation	Стандарт МО США. Разработка и документирование ПО (www.pogner.demon. co.uk/mil_498)
NASA Std. 2100 :1991	NASA Software Documentation Standard	Документирование ПО
NASA Std. 2201 :1993	NASA Software Assurance Standard	Обеспечение гарантии ПО
NASA Std. 2202 :1993	NASA Software Formal Inspections Standard	Формальные инспекции ПО

Обозначение	Наименование (англ.)	Наименование (рус.)
NASA Std 8719.13A :1997	NASA Software Safety Standard	Стандарт по Безопасности функционирования ПО
NASA CM Gdbk :1995	NASA Software Configuration Management Guidebook	Руководство по управлению конфигурацией ПО
NASA GB A201 :1889	NASA Software Assurance Guidebook	Руководство по обеспечению гарантии ПО
SMAP GB A301 :1990	NASA Software Quality Assurance Guidebook	Руководство по обеспечению гарантии качества ПО
NASA GB A302 :1993	NASA Software Formal Inspections Guidebook	Руководство по формальным инспекциям ПО
NASA GB 001 :1994	NASA Software Measurement Guidebook	Руководство по измерению ПО
NASA GB 001 :1995	NASA Software Process Improvement Guidebook	Руководство по совершенствованию процесса разработки ПО
NASA GB 001 :1996	NASA Software Management Guidebook	Руководство по управлению разработкой ПО
NASA GB 1740.13 :1996	NASA Guidebook for Safety Critical Software – Analysis and Development	Руководство по анализу и разработке ПО критических систем
NASA-GB 1740.13:2002	NASA Software Safety Guidebook	Руководство по обеспечению безопасности функционирования ПО

Приложение 4. СРЕДСТВА ПОДДЕРЖКИ РАЗРАБОТКИ ВЫСОКОКАЧЕСТВЕННЫХ ПРОГРАММНЫХ СИСТЕМ

1. Поддержка качества в Oracle Designer'2000

Продукт Oracle Designer'2000 предоставляет средства инструментальной поддержки практически всех вспомогательных процессов ЖЦ. Они сосредоточены в основном в таких утилитах Designer, как *Навигатор объектов репозитория* (RON, Repository Object Navigator), *Построитель матричных диаграмм* (или матричный диаграммер) (MD, Matrix Diagrammer), *Составитель отчетов* по содержимому репозитория (Repository Reports). Эти утилиты обеспечивают навигацию по рабочим продуктам ПС - *объектам репозитория*, просмотр и изменение их свойств.

Матричный диаграммер. Это удобный инструмент верификации рабочих продуктов, позволяющий просматривать и обслуживать связи между любыми двумя элементами в репозитории. Диаграммер представляет информацию в форме таблицы, строки и столбцы которой указывают элементы репозитория, а перекрестные значения - *способ использования* одного элемента другим. Полезными при проведении верификации могут оказаться матрицы, перечисленные в таблице 4.1.

Таблица 4.1. **Виды матриц Матричного диаграммера**

Строка + столбец матрицы	Содержание
Business Functions + Entities	Связь Деловых функций с Сущностями
Business Functions + Attributes (трехмерная матрица)	Связь Деловых функций с Атрибутами
Business Units + Business Functions	Связь Организационных единиц с Деловыми функциями
Business Units + Modules или PL/SQL Definitions	Связь Организационных единиц с Модулями или Определениями PL/SQL
Modules + Module Components	Связь Модулей с Компонентами модулей
Module Components + Table Definitions	Связь Компонентов модулей с Определениями таблиц
Modules или PL/SQL Definitions + Business Functions	Связь Модулей или Определений PL/SQL с Деловыми функциями
Table Definitions + Entities	Связь Определений таблиц с Сущностями
Business Units to Entities	Связь Деловых единиц с Сущностями
Business Functions + Key Performance Indicators	Связь Деловых функций с Ключевыми показателями эффективности
Nodes to Locations	Связь Узлов с Местами расположения
Modules или PL/SQL Definitions + Nodes	Связь Модулей или Определений PL/SQL с Узлами
Assumptions + Modules или PL/SQL Definitions	Связь Предположений с Модулями или Определениями PL/SQL
Assumptions + Relation Definitions	Связь Предположений с Определениями отношений
Modules + Columns (3-мерная матрица)	Связь модулей со Столбцами таблиц
Requirements + Modules	Отображение требований на Модули
Requirements + Tables	Отражение Требований в Таблицах базы данных

Кроме того, можно создавать различные комбинации матриц используя такие моделируемые при разработке элементы, как: *Business Units* (Организационные единицы), *Objectives* (Цели), *Problems* (Проблемы), *Locations* (Размещения), *Critical Success Factors* (Критические факторы успеха), *Key Performance Indicators* (Ключевые показатели эффективности) и др.

Если при разработке используется метод RAD или метод включения в проект существующих наработок (Design Capture), важными становятся матрицы: *Module Components to Table Definitions и Modules to Module Components*.

Располагая информацией о видах объектов в репозитории и матриц, отражающих их взаимосвязь, можно построить *вопросники* для проведения инспекций на всех стадиях ЖЦ. В них могут быть включены, например, такие вопросы:

- для каждой ли функции существует механизм ввода данных в объекты, с которыми она работает?

- для каждого ли объекта существуют характеристики создания, изменения и удаления? и др.

Для целей верификации рабочих продуктов ПС, с помощью утилиты API могут быть также созданы собственные средства автоматизированной проверки ассоциаций в репозитории [1].

Однако построение *отчета* об определенном наборе элементов репозитория возможно только с использованием утилиты *Repository Reports Runtime*.

Утилита Repository Reports Runtime. Содержит около *100 стандартных отчетов*, предназначенных для отображения объектов базы данных. Обеспечивает изменение параметров отчетов, что позволяет тщательнее описать получаемый результат. Отчеты могут отображаться на экран, а также в файлы формата .PDF или .HTML. Пользователю предоставляется три способа просмотра списка отчетов – по группам, по иерархии (по типам элементов) и по имени отчета. Группы отображают отчеты по стадиям ЖЦ или функциональным областям (рисунок 4.1).

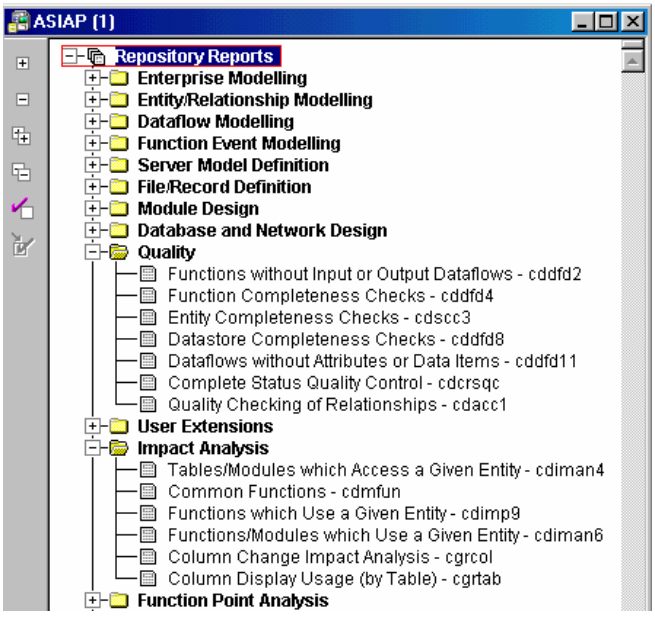


Рис. 4.1. **Группы отчетов Repository Reports**

Порядок следования отчетов в группе примерно соответствует последовательности создания этих объектов в традиционной модели ЖЦ. Пользователь может также создавать собственные отчеты средствами *Report Builder* в Developer'2000 и добавлять их в списки отчетов утилиты *Repository reports*. Краткое описание групп отчетов представлено в таблице 4.2.

Таблица 4.2. **Группы отчетов Repository Reports**

Наименование	Содержание
Enterprise Modeling (Моделирование производственного процесса)	Включает отчеты, относящиеся к моделированию деловых процессов – о потоках данных в системе, об источниках информации, о триггерах событий, об организационных единицах, об основных действиях, критических факторах успеха, ключевых показателях эффективности, целевых показателях и проблемах.
Entity/ Relationship Modeling (Моделирование Сущностей/ Связей)	Включает отчеты о сущностях, атрибутах и связях в определенной прикладной системе, а также о пользователях БД. Могут быть построены отчеты в табличной форме, показывающие взаимосвязь функций и сущностей или функций и атрибутов сущностей.
Dataflow Modeling (Моделирование данных)	Включает отчеты о потоках и хранилищах данных в определенной прикладной системе - об источниках поступления данных в хранилище, функциях, атрибутах и элементах данных, о направлении потоков данных.
Function Event Modeling (Моделирование функций)	Включает отчеты о деловых функциях – о функциональной иерархии, о деталях описания каждой функции (общих функций, главной функции, др.). Могут быть построены табличные отображения: функций на сущности, функций на атрибуты сущностей.
Server Model Definition (Определение модели сервера)	Включает отчеты, касающиеся моделирования сервера (проекта базы данных) - таблиц, взглядов, снапшотов, кластеров, столбцов, ограничений, определений PL/SQL, последовательностей, синонимов и триггеров базы данных.
File/Record Definition (Определение Файла/Записи)	Включает отчеты, касающиеся определений файлов, записей и полей в определенной прикладной системе.
Module Design (Проект модуля)	Включает отчеты о модулях в целом; о модулях типа экран, меню, отчет, пакет, процедура и функция; и о структурах данных и пользовательских предпочтениях.
Database and Network Design (Проект базы данных и сети)	Включает отчеты, содержащие перечень определений для баз данных, пользователей баз данных, объектов баз данных, ролей и определений хранилищ, а также размеры баз данных и индексов.
Quality (Качество)	Включает отчеты, предназначенные для контроля качества. В них указываются, например, перечни потоков данных, не имеющих определений атрибутов или элементов данных, или отметки о полноте определений сущностей или деловых функций.
User Extensions (Пользовательские расширения)	Включает отчеты об элементах или текстовой информации, определенной с использованием средств расширения возможностей пользователя, предоставляемых утилитой Repository Administration Utility.

Impact Analysis (Анализ воздействий)	Включает отчеты, отражающие влияние изменений в одних элементах на другие элементы. Перечисляются, например, функции, использующие определенную сущность, или модули PL/SQL, использующие определенную таблицу
Function Point Analysis (Анализ размера и функциональной сложности)	Включает отчеты, содержащие информацию, используемую при выполнении анализа размера и сложности разработки ПС с применением современных методологий анализа показателей функциональности (FPA) - FPA Mk-I и FPA Mk-II (Великобритания)
Global (Общие отчеты на уровне прикладных систем)	Включает отчеты, касающиеся прикладных систем или обеспечивающие другую информацию на уровне приложений

Возможности Oracle Designer для отслеживания проблем. Для отслеживания проблем, обнаруживаемых при системном тестировании ПС независимой группой тестирования, можно использовать возможности узла Problems, находящегося в группе Enterprise Modeling утилиты Repository Object Navigator (рисунок 4.2).

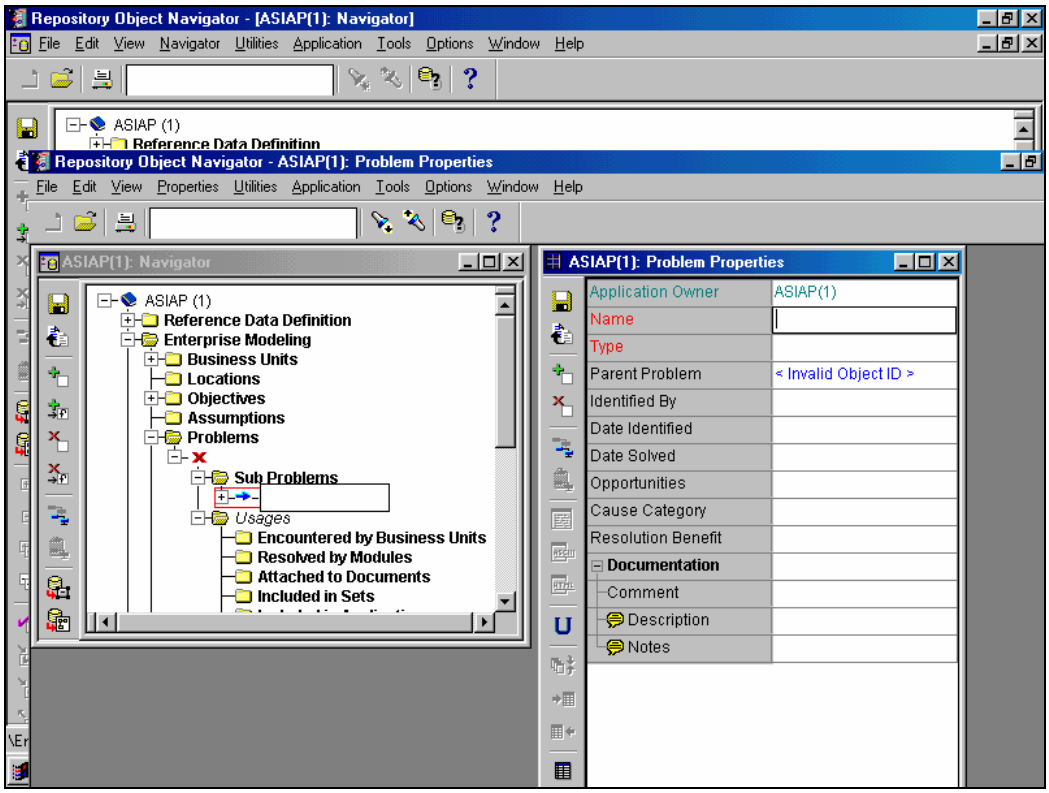


Рис. 4.2. **Узел Problems для указания характеристик проблемы**

В окне *Problem Properties* узла *Problems* указываются следующие элементы, составляющие описание проблемы:

Application Owner – имя прикладной системы, для которой выполняется отслеживание проблемы,

Name – наименование проблемы,

Type - тип определяемой проблемы (серьезность – High, Medium, Low),

Parent Problem – наименование родительской проблемы, если проблемы образуют иерархию,

Identified By – лицо или подразделение, которое обнаружило проблему,

Date Identified - дата, когда проблема была впервые идентифицирована,

Date Solved - дата, когда проблема была решена,

Opportunities - любые средства, способствующие решению проблемы,

Cause Category - категория проблемы (например, Internal (внутренняя), External (внешняя)),

Resolution Benefit - польза от устранения проблемы.

Наряду с описанием характеристики каждой проблемы указывается следующая информация для ее документирования:

Comment - комментарий или краткое описание проблемы,

Description - многострочный текст с описанием ситуации, в которой возникла проблема при тестировании продукта,

Notes - многострочный текст, описывающий возможное местонахождение проблемы.

После описания проблемы ее нужно связать с одним или несколькими модулями, в которых она возникает, через узел *Usages: Resolved by Modules* (Способы использования: Устраняются модулями) для конкретного экземпляра проблемы. Такая связь позволяет создавать отчеты и матричные диаграммы, ассоциирующие описание проблем с модулями.

Систему отслеживания проблем можно распространить на другие элементы репозитория (например, таблицы, взгляды), что даст возможность отслеживать проблемы не только в модулях, но и в этих элементах.

Для отслеживания проблем можно также использовать Matrix Diagrammer, создавая матрицу (Проблемы + Модули). Это позволит получить общую картину проблем в привязке к модулям.

Продукты корпорации Oracle для поддержки принятия решений. Принятие решений в области программной инженерии, основанных на глубоком анализе исторических данных измерений программных проектов, накопленных в хранилищах данных – надеемся, дело недалекого будущего. А пока, стоит присмотреться к продуктам, предлагаемым корпорацией Oracle для поддержки принятия решений в других сферах деятельности:

- *Oracle reports.* Предназначен для построения стандартных и нерегламентированных отчетов. Отчеты могут иметь достаточно сложную структуру, содержать результаты нескольких запросов, автоматически формировать итоги и подитоги, а также включать в себя разнообразную графическую и ссылочную информацию;
- *Oracle Discoverer.* Предназначен для получения произвольных отчетов, формирования нерегламентированных запросов и анализа данных;
- *Oracle Express.* Семейство OLAP-продуктов, предназначенных для решения аналитических задач высокого уровня, связанных со сложными расчетами, прогнозированием, моделированием сценариев «что-если» и экстраполяцией;
- *Oracle Darwin Data Mining Suite.* Инструментальная среда, предназначенная для анализа данных методами, относящимися к технологии data mining. Под-

держивает все этапы технологии, включая постановку задачи, подготовку данных, автоматическое построение моделей, анализ и тестирование результатов, а также использование моделей в реальных приложениях. Обеспечивает построение пяти различных типов моделей - нейронные сети, классификационные и регрессионные деревья решений, ассоциации (в частности, ближайшие k-окрестности), байесовские сети и кластеризацию. Уточненные и проверенные модели можно включать в существующие приложения путем генерации их описаний на языках C, C++, Java, а также разрабатывать новые специализированные приложения с помощью входящей в состав Darwin среды разработки (SDK);

- *Oracle Data Miner*. Инструментальная среда, интегрированная с СУБД Oracle9.x. Заменит Oracle Data Mining Suite, упрощая, автоматизируя и расширяя применение функциональности data mining.

Дополнительную информацию об этих и других продуктах Oracle для поддержки принятия решений можно получить в компании Interface Ltd. (по адресам в Интернете: www.interface.ru или www.olap.ru).

2. Поддержка качества в Microsoft Visual Studio 2005

В компании Microsoft существует одна из групп, основной задачей которой является документирование и публикация *лучших наработок компании*. Эта группа называется Microsoft Patterns and Practices (Модели и практики Microsoft), а ее внутреннее название — PAG (Prescriptive Architecture and Guidance — предписания по архитектуре и управлению проектами). В разрабатываемой этой группой документации можно найти практические советы из любой области: данные, безопасность, конфигурирование, внедрение и т.д. Документацию можно бесплатно загрузить с сайта Microsoft, обратившись по адресу http://www.microsoft.com/patterns.

Командная разработка ПС в среде Visual Studio 2005 Team System. Одна из последних наработок PAG - программный инструмент *Visual Studio® 2005 Team System*, который призван помочь преодолеть неэффективное взаимодействие членов команды проекта, сложности удаленного сотрудничества, плохо интегрированные друг с другом инструментальные средства, отсутствие единой, заранее определенной, стратегии и недостаток актуальной информации о состоянии проекта.

По словам Р. Хандхаузена, «задумав Visual Studio 2005 Team System, Microsoft совершила нечто уникальное: она «отступила назад» и исследовала способы успешной, а также и неудачной разработки программного обеспечения, после чего создала продукт, заметно повышающий предсказуемость успеха проекта. От начала и до конца вашей работы Team System будет координировать ее и управлять ею, направляя к единственной цели. Выпустив Team System, Microsoft кардинально изменила наши представления о Visual Studio. Никогда эта система уже не будет просто хорошей средой разработки — теперь это мощное средство, объединяющее всю команду вокруг единого решения на протяжении всего времени его воплощения в жизнь» [2].

Разработчики Visual Studio 2005 Team System ставили перед собой следующие фундаментальные цели:

- повышение предсказуемости успеха проекта;

- рост производительности труда команды разработчиков за счет понижения сложности процесса проектирования и реализации современных сервисно-ориентированных решений;

- обеспечение сотрудничества команды путем интеграции коммуникационных и прочих средств, используемых ее членами;

- расширение возможностей командной работы за счет обеспечения удаленным пользователям доступа к надежному, защищенному и масштабируемому окружению;

- предоставление команде разработчиков гибкого и расширяемого коммуникационного решения с настраиваемыми сервисами.

Последняя цель предполагает обеспечение возможности для сторонних компаний создавать надстройки для Team System.

Team System сама по себе не есть воплощением какой-либо методологии в программном продукте. Это просто инструмент, обеспечивающий управление разработкой и упорядочение доступа к элементам проекта.

Team System входит в состав комплектов MSF версии 4.0, выпускаемой в двух вариантах, представляющих два подхода к разработке программного обеспечения:

MSF for Agile Software Development (MSF для гибкой разработки ПО) и MSF for CMMI Process Improvement (MSF для усовершенствования процессов CMMI). Предназначена для команд, привыкших к быстрой работе в постоянно изменяющихся условиях и в тесном контакте с заказчиком. Эта методология используется в Team System по умолчанию;

MSF for CMMI Process Improvement (MSF для усовершенствования процессов CMMI). Ориентирована на большие и сложные проекты с многоуровневой отчетностью, когда долгосрочное планирование и эффективное взаимодействие внутри команды разработчиков важнее быстрого исполнения и постоянной двусторонней связи с заказчиками. В Team System реализована часть CMMI, применимая к программной инженерии. Она требует заполнения значительно большего числа документов о состоянии и более детальной отчетности, чем MSF Agile, но при таком более формализованном подходе снижаются риски больших проектов и закладываются основы для получения в будущем сертификатов различных уровней CMMI либо ISO 9000/9001.

Элементы MSF for Agile Software Development, поддерживаемые в Team System. В Team System поддерживаются следующие элементы MSF for Agile Software Development:

роли,
рабочие элементы,
задачи и
рабочие потоки.

Определены *роли* (role) бизнес-аналитика, руководителя проекта, программного архитектора, разработчика, тестировщика и руководителя релиза (выпуска).

Рабочими элементами (work item) являются: сценарии, требования к качеству, риски, задачи и ошибки.

Все рабочие элементы могут быть связаны с *артефактами* (artifacts), такими как документы, электронные таблицы, проектные планы, исходный код и другие материальные результаты действий.

Рабочие элементы создаются по завершении тех или иных действий. Кроме того, они могут служить предпосылками к совершению действий.

Действиями (activity) называют работы, выполняемые совместно с одной целью. При осуществлении действий могут использоваться либо производиться продукты труда. Действия можно отслеживать с применением рабочих элементов.

Объединяясь в группы, действия образуют *рабочие потоки* (work stream) — действия, состоящие из других действий. Рабочие потоки являются строительными блоками процессов; их можно назначать одной или нескольким ролям.

Издания Visual Studio 2005 Team System. Team System — это не просто особая версия Visual Studio, а целая серия версий ролевой ориентации — в каждой из них реализована определенная роль. Данный продукт не предназначен для профессионалов одиночек или независимых консультантов, он ценен в первую очередь тем, что может использоваться для коллективной работы, например, командами, в которых имеются роли руководителя проекта, архитектора, разработчика и тестировщика.

Team System оптимизирована для работы команд, насчитывающих от 5 до 500 членов.

Visual Studio 2005 Team Edition for Software Architects. Это издание предназначено для архитекторов инфраструктуры и приложений. К их числу относятся лица, называемые проектировщиками распределенных приложений (distributed application designer) и проектировщиками архитектуры, ориентированной на сервисы (service oriented architecture designer, SOA designer).

Работа архитекторов, в ее внешнем проявлении, заключается в составлении диаграмм, представляющих логические центры данных, приложения, прикладные системы и схемы развертывания готовых продуктов. При этом архитектор пользуется такими базовыми операциями, как перетаскивание и связывание, давно и широко применяемыми в Visual Studio. Диаграммы обладают определенной интеллектуальностью и содержат метаданные, которые можно проверять на соответствие стандартным и пользовательским ограничениям, а затем одним щелчком мыши превращать в программный код. Microsoft хранит всю заложенную в диаграмме информацию в особых файлах SDM (System Definition Model — модель определения системы), реализуя стратегию DSI (Dynamic Systems Initiative — инициатива в области динамических систем).

Visual Studio 2005 Team Edition for Software Developers. Данное издание предназначено для разработчиков и программистов, и является наиболее востребованным из всех изданий Team System. В дополнение к базовым функциям Visual Studio 2005 разработчики получают статический анализатор кода (подобный FxCop), средство модульного тестирования (подобное NUnit) и средство анализа покрытия кода и профилирования системы. Некоторые из перечисленных компонентов входят также в состав Visual Studio 2005 Team Edition for Software Testers.

Visual Studio 2005 Team Edition for Software Testers. Это издание предназначено как для разработчиков, так и для тестировщиков программного обеспечения. Оно содержит все, что необходимо для тщательного и всестороннего тестирования

продукта, в том числе для анализа покрытия кода, тестирования качества продукта и его нагрузочного тестирования. Обеспечивает функции Web Testing (функция веб тестирования, похожая на Application Center Test), Unit Testing (модульное тестирование), Code Coverage (анализ покрытия кода) и средства управления тестированием, позволяющие составлять тесты, выполнять их и централизованно отслеживать данный процесс. Позволяет также подключать любые тесты, созданные вручную. Некоторые из перечисленных функций предусмотрены и в Team Edition for Developers.

Visual Studio 2005 Team Foundation Server. Это издание Visual Studio содержит базы данных и веб сервисы, позволяющие членам команды сотрудничать, совместно используя рабочие элементы, исходный код, сборки и другие артефакты. В состав Team Foundation Server входит независимый клиент Team Explorer — облегченный вариант Visual Studio 2005. Он является альтернативой Excel, Project и различным выпускам Visual Studio, в части создания рабочих элементов и управления ими. Предназначен для «временно заинтересованного лица» — члена команды, проверяющего документацию, управляющего графикой для веб проекта и т.п.

Visual Studio 2005 Team Suite. Продукт предназначен для членов команды, выполняющих в ней более одной роли, или же для консультантов, способных выполнять любые роли. Microsoft включила в его состав все три ролевых издания Visual Studio 2005 Team System — архитектора, разработчика и тестировщика.

Издания Visual Studio 2005. Team System входит не во все издания Visual Studio 2005. Для новичков, любителей, студентов и профессиональных разработчиков предлагаются издания Visual Studio классов Express, Standard и Professional:

семейство Visual Studio 2005 Express Edition;
Visual Studio 2005 Standard Edition;
Visual Studio 2005 Professional Edition;
Visual Studio 2005 Team Edition for Software Architects;
Visual Studio 2005 Team Edition for Software Developers;
Visual Studio 2005 Team Edition for Software Testers;
Visual Studio 2005 Team Foundation Server;
Visual Studio 2005 Team Suite.

Все издания, которые включают Team System, основаны на *Visual Studio 2005 Professional Edition*.

Интеграция Team System с другими продуктами Microsoft. Team System состоит из множества компонентов. Клиентские издания данного продукта представляют собой просто надстройки для Visual Studio 2005 Professional Edition. Что касается Team Foundation Server, то это целая сервисная архитектура.

Ниже перечислены некоторые другие продукты Microsoft и указано, как они интегрируются с Team System.

Microsoft SQL Server™ 2005. Репозитарий для всех рабочих элементов, исходного кода и данных сборок, включая все артефакты Team System. Входящими в состав SQL Server подсистемами Analysis Services (сервисы анализа) и Reporting Services (сервисы отчетности), поддерживаются различные отчеты Team System, а также портал этой системы.

Microsoft Windows® SharePoint® Portal Services (но не SharePoint Portal Server). Программное обеспечение портала проекта Team System.

Visual Studio 2005. Основная рабочая среда для всех ролей.

Microsoft Project 2003 (но не Project Server). Альтернативное средство для руководителей проекта.

Microsoft Excel 2003. Альтернативное средство для руководителей проекта.

Microsoft Internet Explorer. Используется для взаимодействия с порталом проекта Team System, а также для просмотра отчетов.

Microsoft планирует также выпустить надстройку ***MSSCCI*** **(**Microsoft Source Code Control Interface — интерфейс Microsoft для управления исходным кодом). Она обеспечит базовую интеграцию функций управления исходным кодом нового компонента Team Foundation Version Control и предыдущих версий Visual Studio. До этого времени, равно как и в случае применения ряда программных пакетов и платформ, для расширения функций управления исходным кодом и интеграции с другими средами разработки можно пользоваться любыми API и утилитами с интерфейсом командной строки.

В дополнение к перечисленным средствам Microsoft и ее партнеры анонсировали выпуск множества интегрированных инструментов и сервисов Team System. Microsoft планирует включить в состав продукта миграционные утилиты для Microsoft Visual SourceSafe, а ее партнеры обещают поддержку дополнительных средств моделирования, сбора требований и тестирования.

3. Поддержка качества в семействе продуктов IBM Rational

Методология фирмы *IBM Rational*[1] ориентирована на решение следующих основных задач:

- поддержка основных процессов создания программного обеспечения – моделирования, разработки и тестирования (*IBM Professional Bundle, IBM Rational Suite*);

- организация совместной работы проектной команды (*IBM Team Unified Process*);

- управление проектами и портфелями (*IBM Portfolio Manager*).

Согласно идеологии фирмы, высокий уровень качества разрабатываемых программных продуктов может быть обеспечен применением специальной *методологии* построения программного обеспечения, а также инструментов поддержки всех стадий разработки на разных операционных платформах**.** Как и в семействе продуктов Visual Studio 2005, в IBM Rational предоставляются разные варианты комплектации наборов инструментов.

Вот некоторые примеры из всего разнообразия средств IBM Rational (по материалам сайта http://rational.aplana.ru/tools)

Rational Unified Process (RUP). Это методология создания программного обеспечения, основанная на базе знаний, размещаемой на Web и снабженной поисковой системой. RUP обеспечивает строгий подход к распределению обязанностей (выполняемых работ) и ответственности внутри проекта. Способствует повышению производительности коллективной разработки за счет использования *образцов*, воплотивших в себе все лучшее из накопленного опыта по созданию ПС, - руко-

[1] После приобретения в 2003 году фирмы Rational концерном IBM в название всех ее продуктов было добавлено соответствующее название.

водств, шаблонов и рекомендаций, касающихся различных аспектов применения инструментальных средств. В качестве языка моделирования объектов в общей базе знаний используется Unified Modeling Language (UML). Методология RUP кратко описана в главе 11.

IBM Professional Bundle - пакет, объединяющий *новую* линейку продуктов ***Atlantic*** для моделирования, разработки и тестирования ПО, создаваемого на платформе J2EE[2]**.**

В состав пакета входит весь комплекс программных средств, необходимых предприятию для разработки, конструирования и тестирования приложений, ориентированных на J2EE/порталы/службы, на платформах Windows и Linux, а также для тестирования приложений .NET. Интегрируется со средствами организации совместной работы *IBM Rational Team Unifying Platform*. В состав пакета входят следующие средства:

IBM Rational Software Architect - инструмент проектирования и разработки приложений на основе моделей на языке UML. Позволяет выполнить детальное проектирование ПС и обеспечить создание качественной архитектуры. Инструменты моделирования интегрированы со средствами разработки.

IBM Rational Performance Tester – инструмент для создания, выполнения и анализа тестов производительности, предназначенный для проверки надежности и масштабируемости Web-приложений до их развертывания. Объединяет простое в использовании средство для записи тестов с масштабируемым механизмом выполнения, поддерживающим планирование, создание отчетов в реальном времени и автоматическое варьирование данных.

IBM Rational Functional Tester – интегрированный набор инструментов для поддержки выполнения функционального и регрессионного тестирования широкого спектра приложений, включая клиент-серверные приложения, Java, web-приложения, web-сервисы, Microsoft Visual Studio .NET. Поддерживает управление версиями при одновременной разработке и тестировании распределенных территориально групп. Включает, также, копию ClearCase LT.

IBM Rational Suite – пакет, объединяющий инструменты поддержки организации совместной работы по моделированию и тестированию ПС для разных платформ разработки. В его состав входят следующие инструменты:

IBM Rational Rose — инструмент визуального моделирования ПС с использованием широкого круга инструментальных средств и платформ. Расширяет возможности моделирования ПС, выходящих за рамки платформы *J2EE* и инструментальных средств моделирования в составе IBM Rational Professional Bundle.

IBM Rational XDE Developer и *IBM Rational XDE DeveloperPlus.* Эти инструменты позволяют выполнять разработку информационных систем с использованием Round-Trip подхода, предполагающего автоматизированную генерацию каркаса кода на базе визуальной модели и обратное проектирование – построение модели на основе существующего кода.

IBM Rational XDE DeveloperPlus - дополнительно включает продукты Rational Purify, Quantify и PureCoverage. Также в данный инструмент входит Rational Visual

[2] J2EE – Java2 Enterprise Edition

Trace, который позволяет создать диаграмму последовательности на основе выполнения приложения. Этот продукт позволяет отображать созданные объекты приложения и последовательность вызовов их методов.

Средства *IBM Rational XDE Developer* и *IBM Rational XDE DeveloperPlus* поставляются для конкретных платформ разработки приложений.

IBM Rational XDE Tester – інструмент для функционального и регрессионного тестирования Java- и Web-приложений для сред разработки Eclipse IDE, IBM WSAD и Rational XDE.

IBM Rational Robot - инструмент поддержки создания, записи и выполнения скриптов автоматизированного функционального и регрессионного тестирования Java- и Web-приложений, а также предоставляющий полную поддержку тестирования элементов управления Visual Studio .NET для приложений Oracle Forms, Borland Delphi, Sybase PowerBuilder.

IBM Rational PurifyPlus for Windows - выявление ошибок, связанных с обращением к динамической памяти.

Rational PureCoverage. Это динамический анализатор покрытия, помогающий выявить участки кода, пропущенные при тестировании. Определяет размер программы в строках кода (SLOC). Входит в состав набора инструментов тестирования Rational PurifyPlus.

IBM Rational Quantify. Предназначен для выявления узких мест в коде, оказывающих влияние на производительность ПС.

IBM Rational Team Unifying Platform *(Объединяющая платформа)* **–** набор инструментов для организации *совместной работы команды проекта*. Использование средств коллективной разработки IBM Rational помогает синхронизировать работу разнопрофильных специалистов, вовремя предупреждая всех участников проекта об изменениях.

В основе работы инструментария лежит подход *Unifying Change Management* (UCM), позволяющий организовать индивидуальные рабочие пространства для каждого участника проекта на базе единого репозитория. В зависимости от специфики деятельности конкретного участника проекта он использует необходимые инструменты. Интеграция указанных средств позволяет объединить деятельность всей команды в единый и четкий процесс построения эффективной ПС.

В состав набора входят следующие средства совместного управления проектами и создания коллективной инфраструктуры:

IBM Rational RequisitePro - средство управления требованиями, позволяющее коллективу справляться с постоянно меняющимися требованиями;

IBM Rational ClearQuest – средство отслеживания всех типов запросов на изменения для любого проекта, позволяющее управлять запросами на внесение изменений и отслеживание проблем (от их возникновения до закрытия) на всех стадиях разработки. Изменения архивируются в специально создаваемой для этого базе данных. В качестве СУБД может использоваться MS SQL, MS Access, Sybase SQL Anywhere или Oracle. Обеспечивает возможность добавлять собственные SQL-запросы к существующей базе данных. Предоставляет стандартные и настраиваемые пользователем графические и табличные отчеты о состоянии проекта и истории внесения изменений. Имеет средства автоматического предупреждения и оповещения по электронной почте.

Интеграция *ClearQuest* с *Rational RequisitePro* позволяет связать требования к разрабатываемой ИС с конкретными запросами на изменение. Таким образом, если появились новые требования или изменились существующие, то легко определить, какие запросы на улучшение или обнаруженные ошибки в системе послужили источником для этого.

Интеграция *ClearQuest* с *Rational ClearCase* предоставляет возможность наладить контроль не только за запросами на изменения, но и за любыми изменениями в артефактах проекта. В этом случае никто не сможет изменить ни один артефакт до тех пор, пока для него не определена соответствующая задача. Таким образом, значительно повышается контроль за ходом процесса разработки.

Интеграция ClearQuest со средствами тестирования позволяет вносить описания обнаруженных ошибок прямо по его результатам. Описание ошибки можно взять либо прямо из журнала тестирования, сформированного в Rational TestManager, либо из отчета по результатам анализа, выполненного в реальном времени с помощью Rational Purify, Rational Quantify, Rational PureCoverage.

Интеграция ClearQuest с Rational SoDA позволяет автоматически формировать отчеты в необходимом виде по любым запросам на изменения.

Интеграция ClearQuest с Microsoft Project позволяет быстро сформировать список задач для участников проекта, детализировать этот список до конкретных поручений и реально отслеживать, не выходит ли проект за определенные в плане проекта сроки, ресурсы и бюджет.

IBM Rational ClearCase — инструмент управления конфигурацией, версиями, служащий в качестве общего репозитория для всех объектов разработки ИС. Все, что создается и меняется в процессе разработки, может быть поставлено на версионный и конфигурационный контроль. Таким образом, в значительной мере повышается надежность и качество работы проектной команды.

IBM Rational TestManager – инструмент для поддержки планирования и управления процессом тестирования, а также классификации и отслеживания дефектов, обнаруженных на любой стадии разработки. Позволяет связывать тесты с требованиями, планами тестирования, проверять полноту охвата требований набором тестов и пр. Поддерживает стандартные и настраиваемые пользователем отчеты о качестве и результатах тестирования. Продукт интегрирован с остальными продуктами Rational Suite, но может использоваться самостоятельно для поддержки процесса «Решение проблем» и управления тестированием.

IBM Rational Project Console — средство мониторинга ключевых показателей проекта, которое упрощает контроль за статусом проекта и генерирование объективных метрик проекта с целью улучшения его предсказуемости.

IBM Rational SoDA — средство документирования, автоматизирует создание и сопровождение проектной документации и отчетов.

Помимо перечисленных примеров наборов инструментов, специально для поддержки контроля качества ПС, IBM Rational предлагает инструмент *TestFactory* для формирования набора тестовых скриптов, предназначенных для функционального тестирования ПС. Новаторство инструмента заключается в автоматическом создании «эффективных» сценариев тестирования, которые позволяют проверить все части приложения за минимальное число шагов. Инструмент позволяет построить карту приложения, на основе которой с помощью Rational Robot можно автоматически генерировать тестовые скрипты. После этого TestFactory использует мет-

рики покрытия кода для определения того, какая часть приложения была протестирована. Поскольку тестовые скрипты генерируются автоматически, то тестировщики могут протестировать каждую сборку, проверяя ее на надежность, независимо от стадии разработки.

Дополнительную информацию об этих и других продуктах Rational можно получить в компании *Interface Ltd.* (www.interface.ru) и на сайтах rational.aplana.ru/tools/testing-tools.asp и www-306.ibm.com/software/rational/).

4. Отслеживание дефектов с помощью инструмента Borland StarTeam

Наряду с рассмотренными выше инструментами известных фирм, в Украине также широко используются программные продукты фирмы **Borland.** Среди них для контроля за процессом разработки и отслеживания динамики устранения дефектов может использоваться инструмент управления конфигурацией и изменениями *Borland StarTeam.*

Ниже перечислены основные возможности *Borland StarTeam* по обеспечению качества **(**по материалам сайта www.borland.com/starteam/):

• *Хранение и отслеживание важной информации с помощью центрального репозитория*. Центральный репозиторий Borland StarTeam обеспечивает безопасность, доступность и просмотр всех информационных ресурсов, необходимых для разработки проекта, включая файлы, запросы на изменение, задачи, требования, темы и другие определяемые пользователем объекты.

• Интеграция с другими инструментами поддержки жизненного цикла приложения. Поскольку StarTeam предназначен для прямой интеграции с существующими средствами разработки, его можно интегрировать с различными технологиями и инструментами на протяжении всего ЖЦ приложения. StarTeam полностью поддерживает стандарт Microsoft Source Code Control (SCC) API и, следовательно, может быть интегрирован с инструментами на основе этого стандарта. Кроме того, открытая архитектура StarTeam позволяет включать в любые приложения файлы, запросы на изменения, задания, темы и требования. Независимо от того, используются ли Borland JBuilder, Borland CaliberRM, Mercury Interactive TestDirector, Microsoft Project, соответственно, в качестве средств разработки, управления требованиями, управления тестированием и управления проектом или нет, StarTeam может связать любой содержащийся в его репозитории информационный объект с соответствующими позициями, на которые этот объект оказывает влияние.

• Обеспечение высокой производительности удаленно работающих групп. Репозиторий StarTeam предоставляет всем сотрудникам рабочих групп доступ к самым последним версиям информационных объектов, обеспечивая заданный уровень безопасности и выполнение необходимых процессов управления изменениями. Поскольку web-архитектура StarTeam поддерживает связь по протоколу TCP/IP со всей областью доступа сервера, пользователь может использовать различные клиентские интерфейсы с любого рабочего места с любой из следующих платформ: Windows, UNIX, Linux, Mac OS X.

• Гибкая настройка. Функциональные возможности StarTeam могут быть настроены и расширены в зависимости от потребностей конкретной организации. Комплект разработчика программного обеспечения StarTeam SDK, содержащий открытые программные интерфейсы Java и COM, поддерживает разработку спе-

циализированных решений, включая настраиваемые клиентские приложения и интеграции.

5. Инструменты тестирования Web-приложений

В таблице 5.1 для справки перечислены некоторые примеры программных продуктов, позволяющих выполнять тестирование Web-приложений (по материалам сайта http://rational.aplana.ru/tools/testing-tools.asp).

Таблица 5.1. **Некоторые инструменты тестирования Web-приложений**

Название	Краткая характеристика	Разработчик
IBM Rational Web Developer	Инструмент для быстрой разработки, тестирования и развертывания *web*-приложений *web-служб* и приложений *Java* в среде *IDE*	*IBM Rational* http://rational.aplana.ru/products
IBM Rational XDE Tester.	Инструмент для функционального и регрессионного тестирования Java- и Web-приложений.	*IBM Rational* http://rational.aplana.ru/products
e-TEST suite	Набор интегрированных инструментов для тестирования Web-приложений: e-Manager Enterprise – управление процессом тестирования; e-Tester – автоматизированное функциональное и регрессионное тестирование; e-Load – тестирование производительности.	*Empirix* http://www.empirix.com
Microsoft Web Application Stress Tool	Тестирование производительности Web-сервера для среды Windows.	*Microsoft* www.javausergroup.at/events/was.pdf
PureLoad Web	Инструмент тестирования Web-серверов для платформ Windows, Sun Solaris, Red Hat Linux.	*Minq Software* www.minq.se/ products/ pure-load/ doc/ html/ web/ users-guide/ index.html
LoadRunner	Анализ производительности крупных распределенных ИС. Поддерживает тестирование Web-серверов, баз данных, компонентов COM, JavaBeans, виртуальных машин Java и т.д	*Mercury Interactive* http://www.mercury.com/us/products
QALoad	Инструмент тестирования и анализа Web-серверов, FTP-серверов, серверов баз данных, а также распределенных систем, построенных на основе CORBA	*Compuware* www.compuware.com/products/qacenter/qaload/detail.htm

Литература

1. *Дорси П.* Oracle Designer Настольная книга пользователя. Издание второе. –М.: ЛОРИ. – 1999. – 592 с.

2. *Хандхаузен Р.* Знакомство с Microsoft Visual Studio 2005 Team System. - Санкт-Петербург: Изд. «Питер». – 2006 (http://www.piter.com/chapt.phtml?id=978546901218)

Приложение 5. ПРИМЕРЫ КОНТРОЛЬНЫХ ВОПРОСНИКОВ ДЛЯ ПРОВЕДЕНИЯ ИНСПЕКЦИИ

Вопросы для инспекции описания требований

1. Ясность

1.1. Действительно ли требования написаны на «не техническом» понятном языке?
1.2. Нет ли требований, которые могли бы иметь более одной интерпретации?
1.3. Действительно ли каждая характеристика конечного продукта описана с применением унифицированной терминологии?
1.4. Существует ли глоссарий, в котором разъясняется смысл каждого термина?
1.5. Могут ли быть требования осмыслены и реализованы независимой группой разработчиков?

2. Полнота

2.1. Присутствует ли оглавление (содержание)?
2.2. Помечены ли все рисунки, таблицы и диаграммы?
2.3. Имеют ли все рисунки, таблицы и диаграммы правильные перекрестные ссылки?
2.4. Все ли термины определены?
2.5. Все ли термины упомянуты в индексе?
2.6. Все ли единицы измерения определены?
2.7. Специфицированы ли области, где нет полной информации из-за того, что разработка не начата?
2.8. Есть ли в определении требований указания на упущенную информацию?
2.9. Должны ли какие-либо требования быть специфицированы подробнее?
2.10. Должны ли какие-либо требования быть специфицированы менее подробно?
2.11. Все ли требования определены?
2.12. Все ли требования, касающиеся функциональности, включены?
2.13. Есть ли какие-либо требования, которые Вы считаете сложными?
2.14. Все ли требования, касающиеся производительности, включены?
2.15. Все ли требования, касающиеся ограничений проекта, определены?
2.16. Все ли требования, касающиеся атрибутов, включены?
2.17. Все ли требования, касающиеся внешних интерфейсов, включены?
2.18. Все ли требования, касающиеся баз данных, включены?
2.19. Все ли требования, касающиеся ПО, включены?
2.20. Все ли требования, касающиеся коммуникаций, включены?
2.21. Все ли требования, касающиеся аппаратного обеспечения, включены?
2.22. Все ли требования, касающиеся входов, включены?
2.23. Все ли требования, касающиеся выходов, включены?
2.24. Все ли требования, касающиеся отчетов, включены?
2.25. Все ли требования, касающиеся безопасности, включены?
2.26. Все ли требования, касающиеся сопровождаемости, включены?
2.27. Все ли требования, касающиеся инсталляции, включены?
2.28. Все ли требования, касающиеся критичности, включены?
2.30. Все ли требования, касающиеся ограничений производительности, включены?
2.31. Специфицированы ли возможные изменения в требованиях?
2.32. Для каждого ли требования специфицирована вероятность его изменения?

3. Согласованность

3.1. Есть ли какие-либо требования, описывающие тот же объект, что и другие требования, конфликтуя с ними в отношении терминологии?
3.2. Есть ли какие-либо требования, описывающие тот же объект, что и другие, конфликтуя с ними в отношении характеристик?

3.3. Есть ли какие-либо требования, которые описывают два и более действия, логически конфликтующих?
3.4. Есть ли какие-либо требования, которые описывают два и более действия, терминологически конфликтующих?

4. Трассируемость

4.1. Все ли требования трассируемы к определенным нуждам пользователей?
4.2. Все ли требования трассируемы к определенным исходным документам или другим источникам информации (например, устным договоренностям)?
4.3. Все ли требования допускают трассировку к ним определенных проектных документов?
4.4. Все ли требования допускают трассировку к ним определенных модулей ПО?
4.5. Верифицируемость
4.6. Все ли требования трассируемы к определенным нуждам пользователей?
4.7. Включены ли какие-либо требования, которые невозможно реализовать?
4.8. Для каждого ли требования существует процесс, который может быть выполнен вручную или автоматизировано с целью проверки требования?
4.9. Есть ли какие-либо требования, которые будут представлены в пригодных для верификации терминах позже?

5. Модифицируемость

5.1. Все ли требования трассируемы к определенным нуждам пользователей?
5.2. Действительно ли документ требований четко и структурировано изложен?
5.3. Отвечает ли форма и содержание документа требований принятым стандартам?
5.4. Есть ли дублирование в требованиях?

6. Общее содержание

6.1. Каждое ли требование адекватно проблеме и ее решению?
6.2. Не являются ли какие-либо требования в действительности проектными деталями?
6.3. Не являются ли какие-либо требования в действительности деталями верификации?
6.4. Не являются ли какие-либо требования в действительности деталями управления проектом?
6.5. Есть ли в документе раздел введения?
6.6. Есть ли в документе раздел основного описания требований?
6.7. Есть ли в документе раздел с описанием сферы действия?
6.8. Есть ли в документе раздел с определениями, акронимами, аббревиатурой?
6.9. Есть ли в документе раздел, в котором представлены специальные требования?
6.10. Есть ли в документе раздел с описанием перспектив применения продукта?
6.11. Есть ли там раздел с описанием функций продукта?
6.12. Есть ли там раздел с описанием категорий пользователей продукта?
6.13. Есть ли там раздел с описанием общих ограничений?
6.14. Есть ли там раздел с описанием предположений и зависимостей?
6.15. Есть ли там раздел с указанием специальных требований?
6.16. Присутствуют ли в документе все необходимые приложения?
6.17. Присутствуют ли в документе все необходимые рисунки?
6.18. Присутствуют ли в документе все необходимые таблицы?
6.19. Присутствуют ли в документе все необходимые диаграммы?

7. Специальные требования

Входы

1. Все ли источники ввода специфицированы?
2. Все ли требования к точности входных данных специфицированы?
3. Все ли области значений для входов специфицированы?
4. Указана ли частота поступления данных от всех источников?

5. Все ли форматы входных данных специфицированы?

Выходы

1. Все ли пункты назначения выходных данных специфицированы?
2. Все ли требования к точности выходных данных специфицированы?
3. Все ли области значений для выходов специфицированы?
4. Указана ли частота выдачи данных?
5. Все ли форматы выходных данных специфицированы?

Отчеты

1. Все ли форматы отчетов специфицированы?
2. Все ли вычисления /формулы, используемые в отчетах, специфицированы?
3. Все ли требования по фильтрации данных в отчетах специфицированы?
4. Все ли требования по сортировке данных в отчетах специфицированы?
5. Все ли требования по подведению итогов в отчетах специфицированы?
6. Все ли требования в форматированию отчетов специфицированы?

Функции

1. Все ли функции ПО специфицированы?
2. Все ли входы для каждой функции специфицированы?
3. Все ли аспекты обработки специфицированы для каждой функции?
4. Все ли выходы специфицированы для каждой функции?
5. Все ли требования по производительности для каждой функции специфицированы?
6. Все ли проектные ограничения специфицированы для каждой функции?
7. Все ли атрибуты специфицированы для каждой функции?
8. Все ли требования по безопасности специфицированы для каждой функции?
9. Все ли требования по сопровождаемости специфицированы для каждой функции?
10. Все ли требования к базам данных специфицированы для каждой функции?
11. Все ли операционные требования специфицированы для каждой функции?
12. Все ли инсталляционные требования специфицированы для каждой функции?

Внешние интерфейсы

1. Все ли пользовательские интерфейсы специфицированы?
2. Все ли интерфейсы пакетной обработки специфицированы?
3. Все ли интерфейсы с оборудованием специфицированы?
4. Все ли программные интерфейсы специфицированы?
5. Все ли коммуникационные интерфейсы специфицированы?
6. Все ли проектные ограничения специфицированы?
7. Все ли требования по безопасности интерфейсов специфицированы?
8. Все ли требования по сопровождаемости интерфейсов специфицированы?
9. Все ли сценарии человеко-машинного взаимодействия специфицированы при описании пользовательских интерфейсов?

Внутренние интерфейсы

1. Идентифицированы ли все внутренние интерфейсы?
2. Все ли характеристики внутренних интерфейсов специфицированы?

Затраты времени

1. Все ли ожидаемые периоды процессорной обработки специфицированы?
2. Все ли интенсивности передачи данных специфицированы?
3. Все ли интенсивности трафиков в системе специфицированы?

Надежность

1. Специфицированы ли последствия отказа ПО для каждого требования?
2. Специфицирована ли информация, которая должна защищаться при отказе?
3. Специфицированы ли стратегии обнаружения дефектов?

4. Специфицированы ли стратегии корректировки дефектов?

Компромиссы

1. Специфицированы ли приемлемые компромиссы для конкурирующих атрибутов?

Техническое обеспечение

1. Специфицирована ли минимальная оперативная память?
2. Специфицирована ли минимальная дисковая память?
3. Специфицирована ли максимальная оперативная память?
4. Специфицирована ли максимальная дисковая память?

Программное обеспечение

1. Специфицированы ли программные среды и операционные системы?
2. Специфицированы ли все требуемые программные утилиты?
3. Специфицированы ли все приобретаемые продукты ПО, которые должны использоваться в системе?

Коммуникационные среды

1. Специфицирована ли сеть, с которой будет использоваться система?
2. Специфицированы ли требуемые сетевые протоколы?
3. Специфицирована ли производительность сети?
4. Специфицирована ли требуемая (ожидаемая) пропускная способность сети?
5. Специфицировано ли оценочное количество сетевых соединений?
6. Специфицированы ли минимальные требования к производительности сети?
7. Специфицированы ли максимальные требования к производительности сети?
8. Специфицированы ли оптимальные требования к производительности сети?

Вопросы для инспекции спецификации проекта

1. Функциональность

1.1. Определена ли используемая терминология? Соответствует ли она общепринятой терминологии в данной области и совпадает ли она с терминологией спецификации в техническом задании (ТЗ)? Обозначает ли один и тот же термин одно понятие?

1.2. Представлена ли информационно-функциональная схема проекта отражающая взаимосвязь задач и потоков данных в ПО? Адекватна ли она концептуальной модели проблемной области?

1.3. Представлена ли таблица или схема соответствия функций, указанных в ТЗ, и задач, которые реализуют эти функции?

1.4. Покрывают ли представленные постановки задач все функции, указанные в ТЗ?

1.5. Все ли функции ТЗ (задачи) представлены описаниями своих входов, выходов и алгоритмов обработки (решения)?

1.6. Представлено ли общее описание алгоритма функционирования ПО? Адекватно ли оно модели пользователя ПС (отражающей технологию использования ПО)?

1.7. Достаточно ли обоснован выбор математических методов и алгоритмов? Адекватны ли математические методы и алгоритмы модели предметной области?

1.8. Достаточно ли полно определены модели данных по отношению к требованиям к информационным структурам в ТЗ?

1.9. Обоснован ли выбор методов организации и доступа к данным? Адекватны ли методы организации и доступа к данным выбранным моделям данных?

1.10. Описаны ли допущения и ограничения, связанные с выбранным математическим аппаратом? Адекватны ли они применяемым алгоритмам, описаниям структур данных, а также требованиям к функциональным характеристикам ПО и техническим характеристикам среды функционирования ПО?

1.11. Обоснован ли выбор применяемых языков программирования, общесистемного ПО (СУБД, сетей и пр.) и технических средств в контексте принятых в проекте алгоритмов функционирования, модели данных, математического аппарата? Адекватен ли он соответствующим требованиям ТЗ? Согласованы ли решения по данному вопросу?

1.12. Указаны ли используемые при разработке эскизного проекта (ЭП) международные, государственные, отраслевые и другие стандарты и нормативно- технические документы? Соблюдены ли их требования по форме и содержанию?

2. Совместимость

2.1. Все ли функции будут реализованы непосредственно в данном проекте? Предполагается ли использование заимствованных (унифицированных) проектных решений?

2.2. Достаточно ли полно определены "внутренние" интерфейсы для использования заимствованных компонентов ПО?

2.3. Согласованы ли внутренние интерфейсы по сопрягаемым алгоритмам, передаваемым (принимаемым, промежуточным) структурам данных?

2.4. Предполагается ли интеграция разрабатываемого ПО с другими? Достаточно ли полно описаны внешние интерфейсы?

2.5. Обеспечивается ли совместимость выбранных языков программирования, общесистемных программных и технических средств?

2.6. Предполагает ли проект разработку собственных средств межъязыкового интерфейса для обеспечения совместимости языков программирования?

2.7. Обеспечивается ли информационная совместимость в проекте? Все ли информационные структуры совместимы?

3. Производительность (эффективность)

3.1. Регламентируется ли приемлемый уровень эффективности (по скорости обработки, использованию оперативной памяти и др.) имеющимися действующими стандартами или требованиями заказчика?

3.2. Обоснована ли эффективность предлагаемых проектных решений? Указаны ли ожидаемые технико-экономические показатели?

3.3. Учитывают ли проектные решения требования стандартов по эффективности и соответствующие данные аналогов ПО?

3.4. Учитывают ли методы организации и доступа к данным особенности модели данных, объемы и интенсивность обработки данных?

3.5. Обеспечивают ли применяемые математические методы и алгоритмы, а также языки программирования требуемую точность вычислений? Учтены ли вычислительные возможности технических средств?

3.6. Обеспечивают ли алгоритмы требуемую скорость обработки данных?

3.7. Предполагается ли использование оптимизирующих компиляторов, сопроцессоров, расширенной памяти и других специализированных методов и средств? Адекватно ли принятое решение возможностям предполагаемой среды функционирования ПО?

4. Надежность

4.1. Обеспечивают ли принятая организация данных, методы доступа к данным и алгоритмы защиту данных?

4.2. Обеспечивается ли разграничение доступа к возможностям ПО (при наличии требований в ТЗ)?

4.3. Предполагается ли разработка специальных средств повышения устойчивости к отказам программных и технических средств? Как будет их объем и стоимость сказываться на эффективности ПО?

4.4. Предусматривает ли проект разработку средств контроля и обнаружения ошибок пользователя, ошибок ввода и передачи данных, контроля хода вычислений и соответствующих средств диагностики?

4.5. Предусматривает ли проект разработку средств восстановления вычислительного процесса после устранения отказов программных и технических средств (например, контрольные точки, ведение страховой копии)?

4.6. Адекватны ли проектные решения по восстановлению требованиям в ТЗ к среднему времени восстановления (при их наличии)?

4.7. Разработаны ли планы и процедуры тестирования (испытаний)? Адекватны ли они требованиям к проведению испытаний?

4.8. Обеспечивают ли методы проектирования минимизацию сложности структуры проекта (иерархичность, структурность, понятность)?

5. Удобство применения

5.1. Учитывает ли проект в целом эргономические требования (при их наличии) к ПО, связанные с удобством применения и освоения?

5.2. Предполагается ли режим подсказки (помощи, контекстные справки) для пользователя?

5.3. Представлена ли входная и выходная информация в терминах профессиональной лексики пользователя?

5.4. Предусмотрена ли разработка средств, облегчающих освоение ПО, а также примеров, иллюстрирующих возможности ПО, учебных материалов и курсов?

5.5. Соответствует ли проект интерфейса конечного пользователя эргономическим требованиям (выбор цветовой палитры, удобное размещение информации на экране и т.п.)?

5.6. Соответствует ли проект синтаксиса входного языка и сообщений профессиональному уровню пользователя?

6. Сопровождаемость

6.1. Предусматривается ли возможность выбора (изменения) варианта организации данных?

6.2. Обеспечивает ли функциональное деление проекта минимизацию связей между основными программными функциями?

6.3. Достаточно ли прослеживаемы все проектные решения?

6.4. Учитывает ли проект возможность последующего развития (Открытость архитектуры, возможности технических средств)?

6.5. Предполагается ли разработка средств администратора в ПО (при наличии в ТЗ требований к специальному сопровождению программного обеспечения, баз данных и т.п.)?

6.6. Обеспечивает ли выбранный подход к проектированию удобство модификации (структурность, независимость и т.д.)?

6.7. Предусматривает ли проект разработку сервисных средств сопровождения ПО, средств трассировки, диагностики, сбора статистики, дампирования и анализа?

7. Мобильность (переносимость)

7.1. Достаточно ли универсальны представленные алгоритмы с позиций их переносимости?

7.2. Используются ли стандартные соглашения по организации доступа к данным в проекте, передаваемым и принимаемым данным? Предполагается ли применение нестандартных методов организации и доступа к данным? Насколько они обоснованы?

7.3. Отвечают ли выбранные языки программирования, операционная система требованиям мобильности?

7.4. Учитывает ли структура проекта требования переносимости (независимость верхних уровней иерархии от среды реализации)?

Вопросы для инспекции программного кода

1. Простота понимания и соответствие стандартам кодирования

1.1. Понятны ли спецификации элемента?
1.2. Понятен ли код элемента? Есть ли проблемы в понимании?
1.3. Откомментирован ли код? Помогают ли комментарии понять процедуры и функции элемента?
1.4. Могут ли быть трудности при модификации элемента?
1.5. Установлены ли в проекте стандарты кодирования?
1.6. Соответствует ли код установленным стандартам кодирования?
1.7. Разработаны ли тесты для проверки соблюдения стандартов?

2. Корректность обращения к данным

2.1. Используются ли переменные с не установленными значениями?
2.2. Не выходит ли значение каждого из индексов за границы, определенные для соответствующего измерения при всех обращениях к массиву?
2.3. Принимает ли каждый индекс целые значения при всех обращениях к массиву? Не целые индексы не обязательно являются ошибкой, но представляют практическую опасность.
2.4. Для всех ли обращений с помощью указателей или переменных-ссылок память, к которой производится обращение, распределена (есть ли "подвешенные" обращения)?
2.5. Корректны ли атрибуты при всех псевдонимах? Если одна и та же область памяти имеет несколько псевдонимов (имен) с разными атрибутами, то имеют ли значения данных в этой области корректные атрибуты при обращении по одному из этих псевдонимов?
2.6. Соответствуют ли атрибуты записи и структуры?
2.7. Отличаются ли типы или атрибуты переменных величин от тех, которые предполагались компилятором? Это может произойти в том случае, когда программа считывает записи из памяти и обращается к ним как к структурам, но физическое представление записей отлично от описания структуры.
2.8. Вычислимы ли адреса битовых строк? Передаются ли битовые строки в качестве аргументов?
2.9. Если к структуре данных обращаются из нескольких процедур или подпрограмм, то определена ли эта структура одинаково в каждой процедуре?
2.10. Не превышены ли границы строки при индексации в ней?
2.11. Существуют ли какие-нибудь другие ошибки в операциях с индексацией или при обращении к массиву по индексу?

3. Корректность описания данных

3.1. Все ли переменные описаны явно?
3.2. Если начальные значения присваиваются переменным в операторах описания, то правильно ли инициализируются эти значения?
3.3. Правильно ли для каждой переменной определены длина, тип и класс памяти?
3.4. Согласуется ли инициализация переменной с ее типом памяти?
3.5. Есть ли переменные со сходными именами?

4. Корректность вычислений

4.1. Есть ли вычисления, использующие переменные недопустимых типов?
4.2. Есть ли вычисления, использующие данные разного типа (смешанные вычисления)? Возможно ли усечение дробной части?
4.3. Существуют ли вычисления, использующие переменные одного типа, но разной длины?
4.4. Не меньше ли длина результата, чем длина вычисляемого значения?

4.5. Возможно ли переполнение или потеря промежуточного результата во время вычисления?
4.6. Возможно ли, чтобы делитель в операторе был равен нулю?
4.7. Возможет ли выход значения переменной за пределы ее диапазона ?
4.8. Правильно ли используется целочисленная арифметика, особенно деление.

5. Корректность сравнения данных

5.1. Сравниваются ли величины несовместимых типов (например, строка символов с адресом)?
5.2. Сравниваются ли величины различных типов?
5.3. Корректны ли отношения сравнения?
5.4. Корректны ли булевские выражения?
5.5. Объединяются ли сравнения и булевские выражения?
5.6. Сравниваются ли дробные величины, представленные в двоичной форме?
5.7. Понятен ли порядок следования операторов?

6. Корректность передачи управления

6.1. Может ли значение индекса в переключателе превысить число переходов?
6.2. Будет ли завершен каждый цикл?
6.3. Будет ли завершена программа?
6.4. Возможно ли, что из-за входных условий цикл никогда не сможет выполняться?
6.5. Корректны ли возможные погружения в цикл?
6.6. Есть ли ошибки отклонения числа итераций от нормы?
6.7. Соответствуют ли друг другу операторы DO и END в группах DO-END?

7. Корректность интерфейса между модулями

7.1. Есть ли несоответствие количества, порядка, типов и размеров параметров в вызывающем и вызываемом модулях (подпрограммах)?
7.2. Если модуль имеет несколько точек входа, передается ли параметр всегда вне зависимости от точки входа?
7.3. Не изменяет ли подпрограмма параметр, который должен использоваться только как входная величина?
7.4. Все ли глобальные переменные используются правильно?
7.5. Передаются ли в качестве аргументов константы?

8. Корректность ввода-вывода

8.1. Нет ли ошибок в описаниях атрибутов файлов и операторах ввода-вывода?
8.2. Соответствует ли размер буфера размеру записи?
8.3. Открыты ли файлы перед их использованием?
8.4. Обнаруживаются ли признаки конца файла?
8.5. Обнаруживаются ли ошибки ввода-вывода?
8.6. Существуют ли текстовые ошибки в выходной информации?

Приложение 6. ВОПРОСЫ ДЛЯ ОЦЕНКИ РИСКА ПРОЕКТА

Вопросы для анализа и оценки риска проекта объединены в группы в соответствии с трехуровневой таксономией “класс-элемент-атрибут” (таблица 6.1).

Каждая из представленных ниже таблиц содержит перечень вопросов, касающихся атрибутов *одного* из элементов таксономии. Вопросы имеют *сквозную* нумерацию. Форма задания вопросов выбрана с учетом удобства автоматизации обработки вопросника.

Вопросник содержит *несколько* вопросов к одному атрибуту таксономии, способных прояснить угрозы качеству, стоимости и срокам разработки продукта, которые связаны с указанным атрибутом.

Каждому вопросу приписан максимальный *вес*, отражающий важность данного вопроса для снижения общего риска по соответствующему атрибуту. Чем выше вес, тем существеннее вопрос с точки зрения обнаружения риска.

Возможные варианты ответа на вопрос таковы: *“Да”, “Нет”, “Частично”, “Не знаю”, “Не применим”*.

Ответ “*Да*” свидетельствует об *отсутствии* риска, суть которого сформулирована в вопросе.

Ответ “*Нет*” или “*Не знаю*” свидетельствует о *наличии* риска.

Ответ “*Частично*” также свидетельствует о наличии риска, но дает возможность эксперту, оценивающему риск, указать вес вопроса, отличный от предлагаемого в таблице максимального веса.

Ответ “*Не применим*” соответствует неправомерности задания вопроса для данного проекта.

Класс А. Технические аспекты разработки

Таблица 6.1. **Вопросы для оценки рисков по элементу «Требования»**

Класс:		**Технические аспекты разработки**	
Элемент:		**Требования**	
Атрибут	**Код**	**Вопрос**	**Вес**
а) Стабильность (цель - выяснить, изменяются ли требования в процессе реализации)	1.1.1	*Остаются* ли стабильными требования в ходе разработки?	10
	1.1.2	*Не могут* ли измениться внешние интерфейсы?	10
б) Полнота (цель - выяснить, есть ли пропущенные или не полностью описанные требования)	1.2.1	*Нет* ли требований, уточнение которых отложено для последующего определения?	10
	1.2.2	*Отражены* ли в спецификации все известные или предполагаемые требования?	10
	1.2.3	*Можно* ли установить любое пропущенное требование к системе?	10
	1.2.4	*Зафиксированы* ли документально все требования и пожелания заказчика к системе?	10
	1.2.5	Если да, то *можно* ли отразить эти требования в системе?	10

	1.2.6	*Полностью* ли определены внешние интерфейсы?	10
в) Однозначность (цель - выяснить, нуждаются ли требования в разъяснении или интерпретации)	1.3.1	*Понятны* ли требования, изложенные в документе?	10
	1.3.2	*Можно* ли устранить все неоднозначности в требованиях?	10
г) Достоверность (цель - выяснить, отвечают ли требования к системе ожиданиям заказчика)	1.4.1	*Включены* ли в требования все пожелания пользователя?	10
	1.4.2	*Есть* ли план действий по проверке охвата требованиями пожеланий пользователя?	10
	1.4.3	*Одинаково* ли понимание требований разработчиком и заказчиком?	10
	1.4.4.	Если да, то *можно* ли это обосновать (например, с помощью прототипирования, анализа, моделирования и др.)?	10
	1.4.5	*Есть* ли согласованный план валидации требований?	5
д) Реализуемость (цель - выяснить, показывает ли анализ требований возможность их реализации)	1.5.1	*Все* ли требования технически реализуемы?	10
	1.5.2	*Можно* ли выявить и упорядочить по приоритетам трудности технической реализации требований?	10
	1.5.3	*Проводится* ли исследование возможности реализации требований?	10
	1.5.4	*Есть* ли гарантии правильности результатов исследования?	6
е) Новизна (цель - выяснить, есть ли у разработчика опыт создания подобных систем)	1.6.1	*Удалось* ли избежать “беспрецедентных” требований (в таких областях как новые технологии, методы, языки или аппаратное обеспечение)?	8
	1.6.2	*Имеют* ли разработчики и заказчики необходимый спектр знаний в этих областях?	10
	1.6.3	*Есть* ли план получения необходимых знаний в этих областях?	10
ж) Масштабность (цель - выяснить, есть ли опыт разработки систем такого уровня объема и сложности)	1.7.1	*Приемлемы* ли объем и сложность системы для разработки?	8
	1.7.2	*Есть* ли опыт разработки подобных крупномасштабных систем?	8
	1.7.3	*Могут* ли быть обеспечены в организации условия разработки, адекватные масштабу разрабатываемой системы (достаточно ли велика сама организация-разработчик)?	10

Таблица 6.2. **Вопросы для оценки риска по элементу «Проект»**

Класс:	**Технические аспекты разработки**		
Элемент:	**Проект**		
Атрибут	**Код**	**Вопрос**	**Вес**
а) Функциональность (цель - выяснить, есть ли потенциальные проблемы с удовлетворением функциональных требований)	2.1.1	*Соответствуют* ли алгоритмы или проектные решения предъявленным функциональным требованиям?	10
	2.1.2	*Существует* ли утвержденная процедура демонстрации возможности реализации алгоритмов или проектных решений (например, прототипирование, анализ, моделирование)?	10
б) Сложность (цель - выяснить, трудно ли будет спроектировать и/ или реализовать проект)	2.2.1	*Все* ли проектные решения основаны на достоверных и устоявшихся предпосылках?	10
	2.2.2	*Все* ли требования или функции целесообразно реализовывать?	10
	2.2.3	*Есть* ли проектные решения по всем требованиям?	8
в) Интерфейсы (цель - выяснить, хорошо ли определяются и контролируются все внутренние интерфейсы)	2.3.1	*Хорошо* ли определены внутренние интерфейсы (программно-аппаратные, программные)?	8
	2.3.2	*Оформлено* ли определение внутренних интерфейсов в виде процесса?	10
	2.3.3	*Обеспечивает* ли этот процесс контроль за изменениями внутренних интерфейсов?	10
	2.3.4	*Ведется* ли разработка требуемого аппаратного обеспечения параллельно с разработкой ПО?	10
	2.3.5	*Окончательны* ли спецификации аппаратного обеспечения?	10
	2.3.6	*Определены* ли все интерфейсы с ПО?	10
	2.3.7	*Будут* ли модели проектирования ПО таковыми, что смогут применяться для тестирования ПО?	10
г) Производительность (цель - выяснить, установлены ли жесткие ограничения по времени обработки или пропускной способности каналов)	2.4.1	*Решены* ли проблемы производительности (в таких областях риска как пропускная способность каналов, распределение асинхронных событий в масштабе реального времени, доступ к БД и обработка запросов)?	10
	2.4.2	*Проведен* ли анализ характеристик производительности?	10
	2.4.3	*Есть* ли уверенность в его результатах?	10
	2.4.4	*Существует* ли модель отслеживания производительности от начала проекта до реализации продукта?	10
д) Тестопригодность	2.5.1	*Легко* ли будет тестировать ПО?	8

(цель - выяснить, в какой мере будет сложно тестировать продукт)	2.5.2	*Включены* ли в проект средства (свойства), облегчающие тестирование?	10
	2.5.3	*Привлекаются* ли тестировщики к процессу анализа требований?	10
е) Аппаратные ограничения (цель - выяснить, есть ли ограничения на целевое аппаратное обеспечение)	2.6.1	*Обеспечивает* ли аппаратура реализацию требований к системе (к архитектуре, объему памяти, пропускной способности каналов, времени обработки запросов, производительности БД, функциональности, надежности, пригодности)?	10
ж) Приобретаемое ПО (цель - выяснить, есть ли проблемы с повторно используемым или приобретенным ПО)		**Если применяется повторно используемое или модифицированное ПО**	
	2.7.1	*Не применяется* ли при разработке системы повторно используемое или модифицируемое ПО?	5
	2.7.2	*Разрешены* ли все проблемы с поставкой, документацией, эффективностью функционирования или настройкой такого ПО?	10
		Если применяется коммерческое ПО	
	2.7.3	*Решены* ли все проблемы, связанные с использованием приобретенного коммерческого ПО (нехватка документации для определения интерфейсов, размера или производительности; низкая эффективность; большой объем требуемой оперативной памяти, сложность взаимодействия с прикладным ПО, наличие ошибок, отсутствие сопровождения разработчика и т.п.)?	10
	2.7.4	*Обеспечивается* ли обновление версий приобретенного ПО и их проверка?	10
	2.7.5	*Приобретено* ли коммерческое ПО *у официальных* представителей (дилеров)?	10

Таблица 6.3. **Вопросы для оценки риска по элементу «Кодирование и автономное тестирование»**

Класс:	**Технические аспекты разработки**		
Элемент:	**Кодирование и автономное тестирование**		
Атрибут	**Код**	**Вопрос**	**Вес**
а) Реализуемость (цель - выяснить, легко ли реализовать проект)	3.1.1	*Достаточно* ли полно и однозначно определены элементы проекта для реализации?	15
	3.1.2	*Легко* ли можно реализовать алгоритмы и проектные решения?	5
б) Автономное тестирование	3.2.1	*Будет* ли автономное тестирование начинаться *только после* верификации (проверки) исходного кода по отношению к проекту?	10
	3.2.2	*Отделено* ли автономное тестирование от отладки (как самостоятельное действие)?	10

(цель - выяснить, адекватно ли установлены уровни и время автономного тестирования)	3.2.3	*Достаточно* ли времени для выполнения всех автономных тестов?	10
	3.2.4	*Сохранится* ли запланированный объем автономного тестирования при недостатке времени на его полное выполнение?	10
в) Кодирование/ реализация (цель - выяснить, есть ли проблемы, связанные с кодированием/реализацией)	3.3.1	*Достаточно* ли подробны проектные спецификации для написания кода?	10
	3.3.2	*Окончательны* ли проектные решения и спецификации перед началом кодирования?	10
	3.3.3	*Адекватны* ли системные ограничения (по времени, оперативной и внешней памяти) сложности реализации?	10
	3.3.4	*Удачно* ли выбран язык реализации ПО для данной системы?	10
	3.3.5	*Один* ли язык используется для реализации ПО?	8
	3.3.6	Если нет, *совместимы* ли их межъязыковые интерфейсы?	10
	3.3.7	Разрабатывается ли ПО *на той же* модели компьютера, на которой предполагается его эксплуатация?	10
	3.3.8	Если нет, то *поддерживают* ли оба компьютера один и тот же компилятор?	10
	3.3.9	Если в рамках системы разрабатывается аппаратное обеспечение, то *адекватны* ли спецификации аппаратуры для реализации ПО?	10
	3.3.10	*Окончательны* ли спецификации аппаратуры перед началом реализации?	10

Таблица 6.4. **Вопросы для оценки риска по элементу «Интеграция и интеграционное тестирование»**

Класс:	**Технические аспекты разработки**		
Элемент:	**Интеграция и интеграционное тестирование**		
Атрибут	**Код**	**Вопрос**	**Вес**
а) Среда (цель - выяснить, адекватна ли среда для интеграции и тестирования)	4.1.1	*Соответствуют* ли аппаратные средства задачам интеграции и тестирования?	10
	4.1.2	*Реалистичны* ли разработанные сценарии и тестовые данные для демонстрации выполнимости любых требований к системе (например, специфицированных потоков данных, обработки запросов в масштабе реального времени, управления асинхронными процессами, многопользовательского режима взаимодействия)?	10
	4.1.3	*Можно* ли проверить производительность системы в среде разработки?	8
	4.1.4	*Применяются* ли аппаратные и программные средства поддержки тестирования?	10

	4.1.5	Если да, то *достаточны* ли они для полного тестирования?	10
б) Интеграция продукта (цель - выяснить, есть ли плохо определенные интерфейсы и неадекватные средства и достаточно ли времени на интеграцию)	4.2.1	*Доступна* ли, при необходимости, целевая аппаратно-программная среда?	10
	4.2.2	*Согласованы* ли критерии приемки по всем требованиям?	10
	4.2.3	Если да, то *строго* ли они установлены?	10
	4.2.4	*Хорошо* ли определены, описаны и стандартизованы внешние интерфейсы?	10
	4.2.5	*Можно* ли проверить все требования?	10
	4.2.6	*Хорошо* ли специфицирована интеграция продукта?	10
	4.2.7	*Достаточно* ли времени выделено на интеграцию и тестирование?	10
	4.2.8	Если применяются готовые программные продукты, то *будут* ли данные поставщика учитываться при верификации требований, связанных с этими продуктами?	10
	4.2.9	Если да, то *оговорено* ли это в контракте с поставщиком?	10
в) Интеграция системы (цель - выяснить, хорошо ли координируются и управляются работы по интеграции системы, хорошо ли определены интерфейсы и системные средства)	4.3.1	*Достаточно* ли четко специфицирована интеграция системы?	10
	4.3.2	*Достаточно* ли времени выделено для интеграции и тестирования системы?	10
	4.3.3	*Все* ли уровни соисполнителей представлены в группе, выполняющей системную интеграцию?	10
	4.3.4	*Не встраивается* ли продукт в существующую систему?	5
	4.3.5	Если встраивается, то *выделен* ли период времени для встраивания?	10
	4.3.6	Если не встраивается, то *можно* ли гарантировать, что интегрированная система будет работать корректно?	10
	4.3.7	*Не придется* ли выполнять интеграцию у заказчика (пользователя)?	5

Таблица 6.5. **Вопросы для оценки риска по элементу «Нефункциональные характеристики»**

Класс:		**Технические аспекты разработки**	
Элемент:		**Нефункциональные характеристики**	
Атрибут	**Код**	**Вопрос**	**Вес**
а) Удобство сопровождения	5.1.1	*Учтены* ли в системной архитектуре, проектных решениях или программном коде требования к удобству сопровождения?	10

(цель - выяснить, будет ли реализованная система сложной для понимания и сопровождения)	5.1.2	*Вовлекается* ли персонал сопровождения системы в процесс проектирования системы?	10
	5.1.3	*Адекватна* ли системная документация потребностям сопровождения системы внешней или сторонней организацией?	10
б) Надежность (цель - выяснить, трудно ли обеспечить требования к надежности и готовности системы)	5.2.1	*Учтены* ли требования к надежности системы при разработке ПО?	10
	5.2.2	*Учтены* ли требования к готовности (восстанавливаемости) системы при разработке ПО?	10
	5.2.3	*Удовлетворительно* ли время восстановления работоспособности системы при сбоях?	10
в) Защищенность (цель - выяснить, можно ли обеспечить достижимость требований к защите системы)	5.3.1	*Не предъявляются* ли требования к защите системы при разработке ПО?	10
	5.3.2	Если предъявляются, то *можно* ли обеспечить реализацию этих требований?	10
	5.3.3	*Удовлетворены* ли необходимые требования к защищенности?	10
г) Безопасность (цель - выяснить, не являются ли требования к безопасности жестче обычных для данного класса систем)	5.4.1	*Не являются* ли требования к уровню безопасности более строгими, чем это принято для данного класса систем?	10
	5.4.2	*Был* ли требуемый уровень безопасности уже реализован ранее?	10
д) Человеческие факторы (цель - выяснить, могут ли возникнуть трудности при эксплуатации системы из-за плохо определенного интерфейса с конечным пользователем)	5.5.1	*Удовлетворяет* ли интерфейс конечного пользователя установленным требованиям?	10
	5.5.2	*Применяется* ли прототипирование ПО с “выбрасыванием” прототипа?	5
	5.5.3	*Делается* ли эволюционная разработка?	5
	5.5.4	*Есть* ли опыт разработки по применяемой модели прототипирования?	10
	5.5.5	*Поставляются* ли заказчику промежуточные версии?	5
	5.5.6	*Согласованы* ли поставки с процедурами контроля изменений версий?	10
е) Спецификации (цель - выяснить, соответствует ли документация потребностям проектирования, реализации и тестирования системы)	5.6.1	*Адекватны* ли спецификации требований проекту системы?	10
	5.6.2	*Адекватны* ли спецификации аппаратуры проекту и реализации ПО?	10
	5.6.3	*Хорошо* ли описаны спецификации внешних интерфейсов?	10
	5.6.4	*Адекватны* ли описания тестов потребностям полного тестирования системы?	10
	5.6.5	*Адекватны* ли спецификации проекта потребностям реализации системы?	10

Класс B. Среда и технология разработки

Таблица 6.6. **Вопросы для оценки риска по элементу «Процесс разработки»**

Класс:		**Среда и технология разработки**	
Элемент:		**Процесс разработки**	
Атрибут	**Код**	**Вопрос**	**Вес**
а) Формализованность (цель - выяснить, трудно ли будет понять и поддерживать процесс реализации)	6.1.1	*Выполняется* ли разработка системы в рамках единой модели (спиральной, каскадной, пошаговой)?	10
	6.1.2	Если нет, то *просто* ли выполнять координацию между разными моделями?	10
	6.1.3	*Есть* ли четкие и контролируемые планы по всем действиям разработки?	10
	6.1.4	*Хорошо* ли эти планы специфицируют процесс разработки?	10
	6.1.5	*Знакомы* ли разработчики с этими планами?	10
б) Укомплектованность (цель - выяснить, соответствуют ли элементы процесса модели разработки (ЖЦ) ?)	6.2.1	*Соответствует* ли процесс разработки типу продукта?	10
	6.2.2	*Поддерживается* ли процесс разработки совместимым набором процедур, методов и инструментов?	10
в) Контролируемость процесса (цель - выяснить, выполняется ли контроль процесса с помощью метрик?)	6.3.1	*Все* ли разработчики действуют в рамках установленного процесса разработки?	10
	6.3.2	*Можно* ли измерить, насколько процесс разработки обеспечивает достижение целей производительности и качества?	10
	6.3.3	*Хорошо* ли координируются работы между соисполнителями?	10
г) Опыт применения (цель - выяснить, смогут ли разработчики применять процесс)	6.4.1	*Имеют* ли разработчики опыт работы в рамках установленного технологического процесса разработки?	10
д) Контролируемость продукта (цель - выяснить, существуют ли механизмы контроля изменений продукта)	6.5.1	*Существует* ли механизм трассировки требований с целью их контроля начиная от исходных спецификаций и до спецификаций тестов?	10
	6.5.2	*Применяется* ли механизм трассировки для оценивания влияния изменений требований на разработку?	10
	6.5.3	*Формализован* ли процесс контроля изменений?	10
	6.5.4	Если да, то *отслеживаются* ли изменения всех рабочих продуктов (или релизов) разработки (требований, проекта, кода и документации)?	10

	6.5.5	*Проводится* ли сквозная трассировка изменений, внесенных на любом уровне - от уровня системы и до уровня тестов?	10
	6.5.6	*Проводится* ли надлежащий анализ влияния изменений при добавлении в систему новых требований?	10
	6.5.7	*Есть* ли возможность проследить интерфейсы?	10
	6.5.8	*Обновляются* ли планы и процедуры тестирования в рамках процесса изменений?	10

Таблица 6.7. **Вопросы для оценки риска по элементу «Система поддержки разработки»**

Класс:	**Среда и технология разработки**		
Элемент:	**Система поддержки разработки**		
Атрибут	**Код**	**Вопрос**	**Вес**
а) Мощность (цель - выяснить, достаточна ли для работы мощность рабочих станций, объем внешней и оперативной памяти)	7.1.1	*Достаточно* ли количество и мощность рабочих станций для всех разработчиков?	10
	7.1.2	*Достаточно* ли мощностей для перекрывающихся стадий разработки, таких как кодирование, интеграция и тестирование?	10
б) Укомплектованность (цель - выяснить, насколько система поддержки разработки охватывает все стадии, процедуры и функции)	7.2.1	*Обеспечивает* ли система поддержки разработки автоматизацию всех аспектов разработки (анализ требований, анализ производительности, проектирование, кодирование, тестирование, документирование, управление конфигурацией, трассировку требований)?	10
в) Удобство применения (цель - выяснить, насколько легко применять систему поддержки разработки)	7.3.1	*Считают* ли пользователи систему поддержки разработки удобной и легкой в применении?	10
	7.3.2	*Есть* ли хорошая документация по системе разработки?	10
г) Опыт применения (цель - выяснить, достаточно ли опыта применения)	7.3.3	*Есть* ли у разработчиков опыт работы с инструментами и методами системы поддержки разработки?	10
д) Надежность (цель - выяснить, надежна ли система в работе)	7.4.1	*Считается* ли система поддержки разработки надежной (отказоустойчивой, восстанавливаемой и др.)?	10
е) Сопровождаемость	7.5.1	*Проходят* ли сотрудники обучение применению инструментов разработки?	10

(цель - выяснить,есть ли регулярное обслуживание системы со стороны продавца, консультанта)	7.5.2	*Доступны* ли квалифицированные специалисты по системе?	10
	7.5.3	*Быстро* ли поставщики системы реагируют на возникающие проблемы?	10
ж) Поставка (цель - выяснить, предъявляются ли определенные требования к поставке)	7.6.1	*Устанавливается* ли система поддержки разработки у заказчика?	10
	7.6.2	*Выделены* ли соответствующие ресурсы (средства, сроки, люди) для проверки и выполнения установки этой системы?	10

Таблица 6.8. **Вопросы для оценки риска по элементу "Процесс управления"**

Класс:		**Среда и технология разработки**	
Элемент:		**Процесс управления**	
Атрибут	**Код**	**Вопрос**	**Вес**
а) Планирование (цель - выяснить, пригоден ли план управления разработкой)	8.1.1	*Выполняется* ли управление проектом в соответствии с планом?	10
	8.1.2	*Вносятся* ли своевременно изменения в план при его срывах?	10
	8.1.3	*Привлекаются* ли исполнители всех уровней к планированию своих работ?	10
	8.1.4	*Есть* ли альтернативные планы для известных рисков?	10
	8.1.5	*Есть* ли установленные критерии для определения момента ввода альтернативных планов в действие?	10
	8.1.6	*Адекватно* ли отражаются в плане вопросы долгосрочного перспективного планирования?	10
б) Организация проекта (цель - выяснить, ясны ли персоналу должностные обязанности и распределение ответственности в проекте)	8.2.1	*Эффективна* ли организация работ?	10
	8.2.2	*Понимают* ли сотрудники собственные обязанности и обязанности других?	10
	8.2.3	*Знают* ли сотрудники, кто несет ответственность за каждый участок работы?	10
в) Опыт управления (цель - выяснить, имеет ли руководитель проекта опыт в разработке ПО, управлении разработкой, предметной области, утвержденном процессе разработки и др.)	8.3.1	Есть ли в проекте опытные менеджеры в таких областях, как управление разработкой ПО и крупномасштабных проектов, технологический процесс разработки, предметная область проекта и др.?	10

г) Организация взаимодействия (цель - выяснить, хорошо ли налажено взаимодействие всех участников проекта и заинтересованных лиц)	8.4.1	*Нет* ли проблем взаимодействия на всех уровнях управления разработкой?	10
	8.4.2	*Своевременно* ли фиксируются и устраняются разногласия с заказчиком?	10
	8.4.3	*Привлекаются* ли разработчики к участию в совещаниях с заказчиком?	10
	8.4.4	*Дает* ли управление работами уверенность в том, что все замечания заказчика воплощаются в решения, касающиеся функциональных и эксплуатационных характеристик продукта?	10
	8.4.5	*Объективная* ли информация о состоянии проекта доводится до заказчика и руководства организации?	10

Таблица 6.9. **Вопросы для оценки риска по элементу «Методы управления»**

Класс:		**Среда и технология разработки**	
Элемент:		**Методы управления**	
Атрибут	**Код**	**Вопрос**	**Вес**
а) Мониторинг (цель - выяснить, определены ли метрики управления проектом и отслеживается ли продвижение разработки)	9.1.1	*Составляются* ли периодические структурированные отчеты о состоянии разработки?	10
	9.1.2	Если да, *получают* ли составители отчетов отклики на них?	10
	9.1.3	*Направляется* ли соответствующая информация о состоянии проекта на нужные организационные уровни?	10
	9.1.4	*Отслеживается* ли продвижение разработки по отношению к плану?	10
	9.1.5	*Есть* ли у руководства проекта объективное представление о состоянии проекта?	10
б) Управление персоналом (цель - выяснить, достаточно ли обучены сотрудники и используются ли они в соответствии с квалификацией)	9.2.1	*Проходят* ли сотрудники необходимое обучение?	10
	9.2.2	*Входит* ли это в планы проекта?	10
	9.2.3	*Принимается* ли во внимание при назначении исполнителей их опыт работы по соответствующему профилю?	8
	9.2.4	*Легко* ли управлять данным коллективом?	10
	9.2.5	Знают ли разработчики всех уровней о своем статусе в проекте согласно утвержденному плану?	10
	9.2.6	*Советуются* ли руководители с исполнителями перед принятием решений, касающихся выполняемой ими работы?	8
	9.2.7	*Осознают* ли сотрудники важность соблюдения плана?	10
	9.2.8.	*Привлекают* ли руководители разработчиков для участия в совещаниях с заказчиком?	10

в) Обеспечение качества (цель - выяснить, есть ли надлежащие ресурсы и процедуры для обеспечения качества продукта)	9.3.1.	*Подобран* ли компетентный персонал для выполнения функций обеспечения качества ПО в рамках проекта?	10
	9.3.2	*Есть* ли определенный механизм проверки качества?	10
	9.3.3	*На всех* ли уровнях и стадиях разработки выполняются процедуры обеспечения качества?	10
	9.3.4	*Привыкли* ли люди к выполнению процедур обеспечения качества?	10
г) Управление конфигурацией (цель - выяснить, есть ли процедуры внесения изменений или контроля версий продукта, в частности, при территориально распределенной разработке)	9.4.1	*Есть* ли адекватная система управления конфигурацией?	10
	9.4.2	*Подобран* ли компетентный персонал для выполнения функции управления конфигурацией?	10
	9.4.3	*Не нужна* ли координация с какой-либо установленной на месте системой?	10
	9.4.4	Если координация нужна, то *выполняется* ли управление конфигурацией этой установленной системы?	10
	9.4.5	*Отражает* ли система управления конфигурацией изменения, происходящие по месту установки?	10
	9.4.6	*Не распределена* ли разработка по нескольким соисполнителям?	10
	9.4.7	Если да, то *рассчитана* ли система управления конфигурацией на поддержку распределенной разработки?	10

Таблица 6.10. **Вопросы для оценки риска по элементу «Рабочая обстановка»**

Класс:	**Среда и технология разработки**		
Элемент:	**Рабочая обстановка**		
Атрибут	**Код**	**Вопрос**	**Вес**
а) Качество работы (цель – выяснить, ориентированы ли исполнители на качественное выполнение работы)	10.1.1	*Осведомлены* ли сотрудники всех уровней о процедурах обеспечения качества?	10
	10.1.2	*Не является* ли график разработки помехой обеспечению качества?	10
б) Кооперация (цель – выяснить, нет ли преград к сотрудничеству, не нужны ли административные меры для погашения конфликтов)	10.2.1	*Есть* ли сотрудничество в рамках совместно выполняемых функций?	10
	10.2.2	*Эффективно* ли взаимодействуют сотрудники на пути достижение общих целей?	10
	10.2.3	*Редко* ли нужно вмешательство руководства для обеспечения совместной работы исполнителей?	10
в) Коммуникация (цель – выяснить, достаточно ли	10.3.1	*Хорошо* ли осуществляются взаимосвязи между всеми участниками проекта?	10
	10.3.2	*Предрасположены* ли менеджеры к обмену информацией с исполнителями?	10

информированы сотрудники об общих целях и задачах, нет ли преград на пути распространения информации о рисках)	10.3.3	Если да, то *может* ли исполнитель свободно попросить менеджеров о помощи?	10
	10.3.4	*Могут* ли члены проекта самостоятельно выявлять риски, не заручаясь поддержкой менеджеров?	10
	10.3.5	*Оповещаются* ли своевременно члены проекта о событиях, которые могут касаться выполняемой ими работы?	8
г) Моральный климат (цель – выяснить, способствует ли рабочая атмосфера в коллективе эффективной работе)	10.4.1	*Хороший* ли моральный климат в коллективе?	10
	10.4.2	*Быстро* ли выясняются и устраняются любые причины, ухудшающие моральный климат в коллективе?	
	10.4.3	*Есть* ли возможности удержать в коллективе нужных специалистов?	

Класс С. Внешние ограничения проекта

Таблица 6.11. **Вопросы для оценки риска по элементу «»Ресурсы»**

Класс:		**Внешние ограничения проекта**	
Элемент:		**Ресурсы**	
Атрибут	**Код**	**Вопрос**	**Вес**
а) **Сроки разработки** (цель - выяснить, являются ли сроки нереальными и непостоянными)	11.1.1	*Стабильны* ли сроки, отведенные для разработки?	8
	11.1.2	*Реальны* ли сроки?	10
	11.1.3	*Основан* ли метод определения сроков на исторических данных?	10
	11.1.4	*Хорошо* ли этот метод зарекомендовал себя?	10
	11.1.5	*Адекватно* ли планирование продолжительности разработки внутренним и внешним факторам?	10
б) Штат проекта (цель - выяснить, укомплектован ли проект необходимым штатом исполнителей)	11.2.1	*Все* ли необходимые специальности и должности для выполнения работ по проекту предусмотрены в штатном расписании проекта?	8
	11.2.2	*Достаточно* ли исполнителей требуемой квалификации на всех уровнях разработки (укомплектован ли штат)?	10
	11.2.3	*Стабилен* ли коллектив разработчиков?	10
	11.2.4	*Возможно* ли, при необходимости, привлечение специалиста требуемой квалификации?	10
	11.2.5	*Имеют* ли разработчики опыт разработки систем данного типа?	10
	11.2.6	*Есть* ли дублирование ведущих специалистов, на которых опирается проект?	8
	11.2.7	*Есть* ли резерв свободных специалистов на всех уровнях разработки?	10

в) Финансирование (цель - выяснить, достаточно ли и стабильно ли финансирование)	11.3.1	*Стабильно* ли финансирование?	10
	11.3.2	*Основана* ли стоимость проекта на реалистических оценках?	10
	11.3.3	*Основан* ли метод оценки на исторических данных?	10
	11.3.4	*Хорошо* ли этот метод зарекомендовал себя?	10
	11.3.5	*Не происходит* ли потеря функций и свойств разрабатываемого продукта в связи с оптимизацией проекта по критерию стоимости?	10
	11.3.6	*Выделено* ли финансирование на все виды работ (например, на анализ и разработку концепции, оценку качества, обучение эксплуатации и сопровождению и т.д.)?	10
	11.3.7	*Корректируется* ли стоимость при изменении требований?	10
	11.3.8	Если да, то *является* ли это частью процесса контроля за изменениями?	10
г) Средства разработки (цель - выяснить, достаточно ли средств для разработки и поставки продукта)	11.4.1	*Адекватны* ли средства поддержки разработки потребностям проекта?	10
	11.4.2	*Адекватна* ли среда интеграции?	10

Таблица 6.12. **Вопросы для оценки риска по элементу «Условия договора»**

Класс:	**Внешние ограничения проекта**		
Элемент:	**Условия договора**		
Атрибут	**Код**	**Вопрос**	**Вес**
а) Тип договора (цель - выяснить, может ли риск быть связан с типом заключаемого договора)	12.1.1	*Подходит* ли тип договора по всем аспектам разработки проекта (не обременителен ли договор)?	8
	12.1.2	*Устраивают* ли установленные в договоре требования к подготавливаемой документации (по составу и объему)?	10
б) Ограничения договора (цель - выяснить, накладывает ли договор ограничения на методы и средства)	12.2.1	*Есть* ли необходимая объективная информация для выполнения разработки (по методам, средствам поддержки разработки, по разрабатываемому и приобретаемому ПО и др.)?	10
в) Договорные зависимости (цель - выяснить, зависит ли разработка проекта от внешних факторов)	12.3.1	*Не зависит* ли проект от внешних факторов, которые могут повлиять на характеристики разрабатываемой системы, сроки или стоимость (зависимость от поставщиков, соисполнителей, заказчика и т.д.)?	10

Таблица 6.13. **Вопросы для оценки риска по элементу «Интерфейсы проекта»**

а) Заказчик (цель - выяснить, есть ли проблемы, связанные с заказчиком, такие как затянутое согласование решений, плохое взаимодействие, неадекватное обследование предметной области)	13.1.1	*Своевременно* ли заказчик утверждает все решения (по документации, ревизиям ПО, формальным ревизиям)?	10
	13.1.2	Приступаете ли вы к выполнению работ *только после* согласования решений с заказчиком?	10
	13.1.3	*Разбирается* ли заказчик в технических аспектах системы?	10
	13.1.4	*Разбирается* ли заказчик в ПО?	10
	13.1.5	*Избегает* ли заказчик вмешательства в процесс разработки или столкновений с исполнителями?	10
	13.1.6	*Своевременно* ли руководство проекта и заказчик достигают взаимоприемлемых решений (по устанавливаемым требованиям, тестовым критериям, срокам)?	10
	13.1.7	*Эффективен* ли механизм взаимодействия с заказчиком?	10
	13.1.8	Пожелания *всех* ли категорий заказчиков учитываются при достижении соглашений?	10
	13.1.9	Если да, то *определен* ли этот процесс формально?	10
	13.1.10	*Объективную* ли информацию предоставляет руководство разработчика заказчику?	10
Если проект выполняется ассоциированными смежниками:			
б) Смежники (цель - выяснить, не возникают ли проблемы в проекте из-за сбоев во взаимодействии смежников)	13.2.1	Проводится ли изменение внешних интерфейсов *только после* надлежащего уведомления, координации или выполнения формальных процедур изменения?	10
	13.2.2	*Есть* ли адекватный план взаимодействия смежников?	10
	13.2.3	Если да, то *соблюдается* ли он всеми смежниками?	10
	13.2.4	*Своевременно* ли вы получаете от смежников информацию о сроке и предмете (объекте) установления очередного интерфейса?	
	13.2.5	*Пунктуальны* ли смежники?	
Если головной исполнитель проекта имеет соисполнителей (субподрядчиков):			
в) Соисполнители (цель - выяснить, есть ли риск головного исполнителя при работе с соисполнителями)	13.3.1	Если есть соисполнители, то *устранены* ли все неясности в определении их задач?	10
	13.3.2	*Используют* ли соисполнители *те же* процедуры управления и отчетности, что и головной исполнитель?	8
	13.3.3	*Не осуществляется* ли административное и техническое управление соисполнителем сторонней (не головной) организацией?	10

	13.3.4	*Есть* ли возможность не полагаться на экспертизу соисполнителя в любых областях?	10
	13.3.5	*Возможно* ли извлечение и передача знаний соисполнителя головному исполнителю?	10
	13.3.6	*Своевременно* ли вы получаете от соисполнителей информацию о сроках и данных по запланированным интерфейсам?	10
Если проект выполняется в рамках субдоговора с головным исполнителем:			
г) Головной исполнитель (цель - выяснить, есть ли риск проекту, выполняемому соисполнителем, со стороны головного исполнителя)	13.4.1	*Хорошо* ли вы понимаете свою задачу, сформулированную головным исполнителем?	10
	13.4.2	*Осуществляется* ли административное и техническое руководство *одним* исполнителем?	10
	13.4.3	*Независимы* ли вы от экспертов головного исполнителя в любых областях?	10
	13.4.4	*Своевременно* ли вы получаете от головного исполнителя информацию о сроках и данных по запланированным интерфейсам?	10
д) Высшее руководство (цель - выяснить, достаточна ли поддержка проекту со стороны высшего руководства организации)	13.5.1	*Сообщает* ли руководство проектом о своих проблемах высшему руководству?	10
	13.5.2	Если да, то *имеет* ли это эффект?	10
	13.5.3	*Обеспечивает* ли высшее руководство своевременную поддержку в решении ваших проблем?	10
	13.5.4	*Позволяет* ли управление со стороны высшего руководства (на макроуровне) избежать управления на микроуровне?	10
	13.5.5	*Предоставляет* ли ваше руководство высшему руководству объективную информацию о состоянии проекта?	10
е) Продавцы (цель - выяснить, надежны ли поставщики)	13.6.1	*Можно* ли положиться на продавцов, ответственных за поставку критических компонентов системы (в плане сроков, стоимости и технических характеристик)?	10
ж) Политические принципы (цель - выяснить, нет ли угроз проекту со стороны полит. разногласий заинтересованных сторон)	13.7.1	*Свободен* ли проект и принимаемые технические решения от влияния политических принципов руководства организаций-участников проекта?	8

Приложение 7. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ОЦЕНКИ ТЕХНОЛОГИЧЕСКОЙ ЗРЕЛОСТИ ОРГАНИЗАЦИИ

Представленный в данном приложении контрольный вопросник содержит вопросы из вопросника SEI, касающиеся ключевых направлений технологического процесса разработки программного обеспечения (ПО), отнесенных к уровню 2 модели зрелости CMM.

По каждому ключевому направлению процесса (**KPA**, от Key Process Area) в вопросник включены не все, а только избранные вопросы по шести-восьми ключевым процедурам с тем, чтобы время, потраченное на ответы, не превысило 1 часа. Перечень вопросов предваряется кратким описание данного KPA. Ответ на каждый вопрос может быть прокомментирован.

В помощь респондентам, не владеющим терминологией CMM, в конце вопросника приводится определение используемых терминов, а сами термины в тексте выделяются курсивом.

7.1. Инструкция по заполнению опросных листов

1. Ответ на вопрос дается путем внесения отметки (“+” при ручном заполнении или “1” при машинном заполнении) в соответствующую графу формы: **“Почти всегда”**, **“Часто”**, **“Иногда”**, **“Редко”**, **“Никогда”** и **“Не используется”**. Допускается только одна отметка по соответствующей ключевой процедуре. Ответы должны быть даны на все вопросы. Для краткости в формах используется сокращенное обозначение отметок.

2. Отметка **“Почти всегда” (ПВ)** проставляется в том случае, если упомянутая в вопросе процедура внедрена, обеспечена организационной, нормативно-методической и технической поддержкой и практически всегда выполняется (за исключением чрезвычайных и фиксируемых случаев).

3. Отметка **“Часто” (Ч)** проставляется в том случае, если упомянутая в вопросе процедура внедрена (опыт использования менее года), находится в стадии внедрения (адаптируется к условиям разработки конкретно проекта) и декларируемые в ней действия, как правило, выполняются надлежащим образом.

4. Отметка **“Иногда” (И)** проставляется в том случае, если упомянутые в вопросе действия еще не оформлены в виде процедуры и не обеспечены всеми видами поддержки, но примерно в половине случаев выполняются и было бы желательно их официальное внедрение в технологический процесс.

5. Отметка **“Редко” (Р)** проставляется в том случае, если упомянутые в вопросе действия изредка выполняются (при благоприятном стечении обстоятельств).

6. Отметка **“Никогда” (Н)** проставляется в том случае, если упомянутые в вопросе действия никогда не выполняются. Эта отметка может также проставляться в том случае, если респондент не понимает сути вопроса или вообще не знаком с элементами KPA, к которым относится вопрос (например, если при ответах на вопросы по KPA “Управление конфигурацией ПО” респондент не имеет четкого представления о смысле управления конфигурацией).

7. Отметка **“Не используется” (НИ)** проставляется в том случае, если респондент знает о состоянии дел в организации и проекте, понимает вопрос, но счи-

тает, что постановка вопроса не применима к данному проекту. На уровне 2 СММ такая ситуация возможна только для КРА “Управление работой соисполнителя” (поскольку в условиях отсутствия соисполнителей проекта лишены смысла все вопросы, относящиеся к КРА “Управление работой соисполнителя”). Эта отметка не участвует в рейтинговом оценивании и используется только для корректировки расчетных формул.

Поле “Комментарий” используется для обоснования ответа на вопрос (например, ссылки на действующие нормативно-методические и организационно-распорядительные документы, подтверждающие выполнение соответствующей ключевой процедуры).

7.2. Управление требованиями

Цель данного КРА состоит в формировании согласованного взгляда со стороны заказчика и исполнителя проекта на пользовательские (технические и не технические) требования, которые должны быть реализованы в проекте ПО. Достигнутые соглашения образуют базис для оценивания, планирования, выполнения и контроля действий в ходе всего жизненного цикла проекта. При любом изменении системных требований в части, касающейся проекта ПО, выполняется корректировка соответствующих планов, *рабочих продуктов* и действий с целью обеспечения их согласованности с обновленными требованиями.

Управление требованиями	**ПВ**	**Ч**	**И**	**Р**	**Н**	**НИ**
1. Используются ли *системные требования*, *делегированные ПО*, в качестве основы для выполнения разработки и управления процессом разработки? **Комментарий:**						
2. Выполняется ли корректировка *планов ПО*, рабочих продуктов и действий при изменении системных требований, делегированных ПО? **Комментарий:**						
3. Руководствуется ли проект принятой в организации *политикой* в части управления системными требованиями, делегированными ПО? **Комментарий:**						
4. Прошли ли лица, которым поручено управление делегированными требованиями, обучение приемам управления требованиями? **Комментарий:**						
5. Проводятся ли измерения с целью определения адекватности действий, выполняемых по управлению делегированными требованиями (например, есть ли учет общего числа предложенных изменений в требованиях, числа принятых предложений по изменениям, числа произведенных корректировок в *базовой версии* и пр.)? **Комментарий:**						
6. Подвергаются ли действия по управлению требованиями в проекте *ревизиям* с целью *обеспечения качества ПО*? **Комментарий:**						

7.3. Планирование проекта ПО

Цель данного KPA состоит в формировании обоснованных планов выполнения действий по разработке ПО и управления проектом ПО. Планирование проекта ПО предполагает определение оценок выполняемой работы, достижение необходимых *соглашений* и создание плана выполнения работы.

Планирование проекта ПО	ПВ	Ч	И	Р	Н	НИ
1. Документируются ли оценки (например, размера, стоимости и сроков) с целью использования при планировании и контроле выполнения проекта ПО? **Комментарий:**						
2. Документируют ли *планы ПО* действия, которые должны быть выполнены, и принятые соглашения по проекту ПО? **Комментарий:**						
3. Поддерживают ли все участники (группы и отдельные лица) принятые соглашения по проекту ПО? **Комментарий:**						
4. Руководствуется ли проект принятой в организации *политикой* в части планирования проекта ПО? **Комментарий:**						
5. Имеются ли адекватные ресурсы для планирования проекта ПО (например, реальные денежные средства и опытные специалисты)? **Комментарий:**						
6. Проводятся ли измерения с целью определения правильности планирования проекта ПО (например, согласуются ли между собой сроки завершения работ и др.)? **Комментарий:**						
7. Производит ли менеджер проекта как *периодическую ревизию*, так и *событийную ревизию* деятельности по планированию проекта? **Комментарий:**						

7.4. Мониторинг проекта ПО

Цель данного KPA состоит в обеспечении надлежащего наблюдения за развитием проекта с целью принятия руководством адекватных корректирующих воздействий при значительных отклонениях в выполнении проекта от *планов ПО*. Корректирующие воздействия могут заключаться в пересмотре плана разработки ПО для учета реальной ситуации, перепланировании оставшейся части работ или принятии мер по улучшению характеристик проекта. Мониторинг проекта предполагает наблюдение за деятельностью и контроль результатов выполнения работ по отношению к предварительным оценкам, *соглашениям* и планам и настройку этих планов исходя из реальной ситуации и полученных результатов.

Мониторинг проекта ПО	ПВ	Ч	И	Р	Н	НИ
1. Совпадают ли реальные результаты реализации проекта (например, по срокам, размеру и стоимости) с оценками, приведенными в *планах* ПО? **Комментарий:**						
2. Предпринимаются ли корректирующие воздействия в случае значительного отклонения реальных результатов от планов ПО проекта? **Комментарий:**						
3. Согласовываются ли изменения в принятых соглашениях со всеми заинтересованными группами и лицами? **Комментарий:**						
4. Руководствуется ли проект принятой в организации *политикой* в части контроля и управления действиями по разработке ПО? **Комментарий:**						
5. Есть ли в проекте лицо, несущее определенную ответственность за трассировку (отслеживание) *рабочих продуктов* и действий (например, трудоемкости, сроков и бюджета)? **Комментарий:**						
6. Проводятся ли измерения с целью определения реального состояния дел по мониторингу (например, общих затрат усилий на выполнение мониторинга)? **Комментарий:**						
7. Производит ли менеджер проекта *периодическую ревизию* деятельности по мониторингу проекта (например, эффективности реализации проекта, открытых вопросов, риска)? **Комментарий:**						

7.5. Управление работой соисполнителя

Цель данного КРА состоит в подборе квалифицированных соисполнителей разработки ПО и эффективном управлении договорами на исполнение работ соисполнителями. Управление деятельностью соисполнителей предполагает выбор соисполнителя, принятие *соглашений* с соисполнителем и контроль за деятельностью и результатами работы соисполнителя. Процедуры управления касаются только той части ПО соисполнителя, которое разрабатывается в рамках договора с головным исполнителем, а также системных компонентов, содержащих аппаратно-программные и другие средства, разрабатываемые соисполнителем в рамках договора.

Управление соисполнителями	ПВ	Ч	И	Р	Н	НИ
1. Используется ли *документированная процедура* (инструкция) для выбора соисполнителей исходя из их способности выполнять работу? **Комментарий:**						
2. Действительно ли изменения в договор вносятся только по соглашению обеих сторон - головного исполнителя и соисполнителя? **Комментарий:**						
3. Проводятся ли технические ревизии и согласования характеристик разрабатываемого соисполнителем ПО (например, с целью доведения до соисполнителя требований заказчика и конечных пользователей или с целью проверки правильности интерпретации и реализации технических требований)? **Комментарий:**						
4. Контролируется ли соответствие результатов выполнения работ соисполнителями принятым соглашениям? **Комментарий:**						
5. Руководствуется ли проект принятой в организации *политикой* в части управления исполнением работ соисполнителями? **Комментарий:**						
6. Прошли ли лица, которым поручено управление соисполнением работ по договорам, обучение приемам управления соисполнителями? **Комментарий:**						
7. Проводятся ли измерения с целью определения реального состояния дел по управлению работами соисполнителей (например, соблюдение план-графика выпуска *релизов* и общих затрат усилий на управление деятельностью соисполнителей)? **Комментарий:**						
8. Производит ли менеджер проекта как *периодическую ревизию*, так и *событийную ревизию* деятельности соисполнителей по договору? **Комментарий:**						

7.6. Обеспечение гарантии качества ПО

Цель данного KPA состоит в обеспечении руководства организации достоверной информацией о процессе реализации проекта ПО и о создаваемом продукте ПО. *Обеспечение гарантии качества ПО* (**SQA**, от Software Quality Assurance) предполагает проведение *ревизий* и *аудиторских проверок* продуктов ПО и выполняемых действий с целью подтверждения их соответствия применяемым *докумен-*

тированным процедурам и стандартам, а также информирование руководства проекта ПО о результатах этих ревизий и аудиторских проверок.

Обеспечение гарантии качества ПО	ПВ	Ч	И	Р	Н	НИ
1. Является ли *обеспечение гарантии качества ПО (SQA)* плановой деятельностью? **Комментарий:**						
2. Обеспечивает ли SQA объективную проверку соответствия программных продуктов и действий применяемым стандартам, процедурам и требованиям? **Комментарий:**						
3. Доводятся ли результаты ревизий и аудиторских проверок в рамках SQA до заинтересованных групп и лиц (например, до непосредственных исполнителей и руководителей работ)? **Комментарий:**						
4. Выносятся ли на уровень высшего руководства проблемы несогласованности, которые не могут быть разрешены на уровне проекта (например, отклонения от применяемых в организации стандартов)? **Комментарий:**						
5. Руководствуется ли проект принятой в организации *политикой* в части обеспечения качества ПО? **Комментарий:**						
6. Имеются ли адекватные ресурсы для проведения работы по обеспечению качества ПО (например, реальные денежные средства или ответственное лицо, принимающее меры относительно элементов проекта, по которым в ходе проверок отмечены нарушения) ? **Комментарий:**						
7. Проводятся ли измерения с целью определения реального состояния дел по обеспечения качества ПО в плане соблюдения сроков и стоимости проверок (например, учет завершенных работ, затрат усилий и потраченных денежных средств)? **Комментарий:**						
8. Производит ли высшее руководство организации *периодическую ревизию* деятельности по обеспечению качества ПО? **Комментарий:**						

7.7. Управление конфигурацией ПО

Цель данного КРА состоит в обеспечении целостности *релизов ПО* в ходе жизненного цикла ПО проекта. Управление конфигурацией ПО (**SCM**, от Software Configuration Management) предполагает определение конфигурации ПО (напри-

мер, избранных *рабочих продуктов ПО* и их описаний) в заданные моменты времени, систематический контроль изменений в конфигурации и поддержание целостности и трассируемости (прослеживаемости) конфигурации в ходе жизненного цикла ПО. К релизам, включаемым в контур управления конфигурацией, относятся продукты ПО, поставляемые заказчику, и отдельные элементы, отождествляемые с (или требуемые для) создания этих продуктов ПО.

Управление конфигурацией ПО	ПВ	Ч	И	Р	Н	НИ
1. Является ли управление конфигурацией ПО (SCM) плановой деятельностью в проекте? **Комментарий:**						
2. Обладает ли проект идентифицируемыми, контролируемыми и доступными рабочими продуктами ПО благодаря использованию управления конфигурацией? **Комментарий:**						
3. Следует ли проект *документированной процедуре* с целью контроля изменений в *элементах конфигурации*? **Комментарий:**						
4. Предоставляются ли заинтересованным группам и лицам стандартные отчеты о *базовой версии продукта*? **Комментарий:**						
5. Руководствуется ли проект принятой в организации *политикой* в части управления конфигурацией ПО? **Комментарий:**						
6. Прошел ли персонал проекта, ответственный за управление конфигурацией, обучение приемам SCM? **Комментарий:**						
7. Проводятся ли измерения с целью определения состояния управления конфигурацией ПО (например, учет затрат усилий и потраченных денежных средств на деятельность по SCM)? **Комментарий:**						
8. Выполняются ли периодические *аудиторские проверки* соответствия базовых версий продукта документации, связанной с этими базовыми версиями (например, группой SCM)? **Комментарий:**						

7.8. Определение используемых терминов

Системные требования, делегированные ПО. Подмножество системных требований, которые должны быть реализованы в виде программных компонентов системы. Делегированные требования являются основным исходным материалом для плана разработки ПО. В ходе анализа этих требований формируются требования к ПО, оформляемые соответствующим документом.

Планы ПО. Собрание формальных и неформальных планов, описывающих, каким образом будут выполняться действия по разработке и/или сопровождению ПО (примеры: план разработки ПО, план обеспечения качества ПО, план управления конфигурацией ПО, план тестирования ПО, план управления риском, план улучшения процесса разработки ПО).

Рабочие продукты ПО. Результаты деятельности или решения задач в ходе определения, сопровождения или использования технологического процесса разработки ПО, включая описания, планы, процедуры, компьютерные программы и связанную с ними документацию, которые предназначаются (или не предназначаются) к поставке заказчику или пользователю. Рабочие продукты составляют исходную информацию для следующего шага технологического процесса или архивную информацию по проекту ПО для использования будущими проектами (например, планы, оценки, данные о реальной трудоемкости, документы требований и др.).

Политика организации. Руководящий принцип, обычно утверждаемый высшим руководством и используемый организацией или проектом для принятия решений.

Базовая версия продукта. Элементы программного продукта (проектные решения, код и др.), которые по согласованию сторон проходят формальную ревизию в определенные моменты жизненного цикла ПО и в дальнейшем служат основой (базисом) для продолжения разработки.

Ревизия. Проверки, которые выполняются по концу решения задачи менеджерами, заказчиком, конечными пользователями или другими заинтересованными лицами, и по результатам которых делается заключение об одобрении состояния продукта или необходимые замечания.

Обеспечение гарантии качества ПО (**SQA** - от Software Quality Assurance). Множество действий, предусмотренных с целью оценки качества процесса разработки и/или сопровождения рабочих продуктов и обеспечения гарантий их соответствия установленным техническим требованиям.

Соглашение. Договоренность, которая будет добровольно выполняться всеми участниками.

Периодическая ревизия/деятельность. Ревизия, которая проводится регулярно через определенные интервалы времени и не связана с завершением каких-либо важных событий.

Событийная ревизия/деятельность. Ревизия или другая деятельность, которая выполняется в связи с наступлением события в проекте (например, формальная ревизия или завершение стадии жизненного цикла).

Документированная процедура (Инструкция**).** Письменное описание порядка действий, предпринимаемых для выполнения определенной задачи.

Релизы. Полное множество или любой из отдельных элементов множества компьютерных программ, процедур, данных и соответствующей документации, предназначенных для передачи заказчику или конечному пользователю. Все рели-

зы являются в то же время рабочими продуктами, но не все рабочие продукты представляют собой релизы.

Аудиторская проверка. Независимая проверка рабочего продукта или множества рабочих продуктов с целью оценки соответствия стандартам, договорным соглашениям и другим критериям.

Элемент конфигурации. Агрегация аппаратного, программного или обоих видов обеспечения, на которую распространяется управление конфигурацией, и которая трактуется как единая сущность в процессе управления конфигурацией.