МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)

Кафедра информатики и вычислительной математики

Е. М. Лаврищева

ПРОГРАММНАЯ ИНЖЕНЕРИЯ

Тема 2. ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

Учебно-методическое пособие

МОСКВА
МФТИ
2016

УДК 681.3.06

Рецензент
Член-корреспондент РАН, доктор физико-математических наук,
профессор *И.Б. Петров*

Лаврищева, Е.М.

Программная инженерия. Тема 2. Технология программирования: учебно-методическое пособие. – М.: МФТИ, 2016. – 52 с.

Представлены школы по теории программирования (А.А. Ляпунова, Ю.И. Янова, А.П. Ершова, В.М. Глушкова, Е.Л. Ющенко, Г.Е. Цейтлина, В.Н. Редька и др.) на первых ЭВМ. Дана характеристика теории схем программ и автоматов, алгоритмического, алгебраического и синтезирующего программирования. Рассмотрены подходы к формальной спецификации программ и доказательства их правильности. Дана теория композиции и сборки модулей в сложные системы.

Предназначено для преподавания студентам 1–3 курсов, обучающихся в области информатики, программной инженерии и компьютерных наук.

Учебное издание

Лаврищева Екатерина Михайловна

ПРОГРАММНАЯ ИНЖЕНЕРИЯ

Тема 2. ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

Учебно-методическое пособие

Редактор *В.А. Котова*. Корректор *Л.В. Себова.* Компьютерная верстка *Л.В. Себова*.
Подписано в печать 15.09.2016. Формат 60 × 84 $^{1}/_{16}$. Усл. печ. л. 3,25. Уч.-изд. л. 3,0.
Тираж 50 экз. Заказ № 384.

Федеральное государственное автономное образовательное
учреждение высшего образования «Московский физико-технический институт
(государственный университет)»
141700, Московская обл., г. Долгопрудный, Институтский пер., 9
Тел. (495) 408-58-22. E-mail: rio@mipt.ru

Отдел оперативной полиграфии «Физтех-полиграф»
141700, Московская обл., г. Долгопрудный, Институтский пер., 9
Тел. (495) 408-84-30. E-mail: polygraph@mipt.ru

ВВЕДЕНИЕ

Под **технологией программирования** (ТП) понимается совокупность методов, средств и инструментов для проведения процесса программирования задач предметной области с целью получения программ их решения с заданными свойствами и функциями.

Методами могут быть: модульный, объектный, компонентный и др. Средствами являются языки программирования (ЯП), а также предметно-ориентированный язык DSL (Domain Specific Language), Венский метод VDM и др. Инструменты – это системы программирования, отладчики, редакторы и CASE-системы (UML1, UML2, Rational Rose и др.) и технологические линии (ТЛ) разработки вариантов некоторого продукта. В ТЛ отображена регламентированная последовательность действий над объектом разработки, начиная от требований, задания схемы (графа) системы из ее элементов и действий по их реализации в конкретной операционной среде. Каждое действие поддерживается системным или специальным инструментом, на вход которого подается промежуточный результат, а на выходе – код.

В ТЛ указывается среда и оборудование, на котором надо выполнять операции, и порядок выполнения операций и действий на линии. Выходной результат линии – программный продукт (ПП) и документация, а также инструкции по его применению, получению вариантов продуктов и проведению модификаций ПП в процессе эксплуатации, а также замены непригодных инструментов или функций новыми, более совершенными. ТП – это некоторое общее базовое понятие в индустрии ПП, готовые компоненты которого могут использоваться по-разному в виде отдельных вариантов, в зависимости от требований и условий применения.

Тема 2. ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

Технология программирования развивалась параллельно с зарубежной программной инженерией (Software Engineering – SE).

В нашей стране в период первых ЭВМ разработка программ и систем осуществлялась с помощью средств ТП. Первоначально это были трансляторы в языках программирования (например, Алгол-60). Позже создавались системы автоматизации производства программ (АПРОП, ЯУЗА, ПРОТВА, САТУРН и др.).

Термин SE впервые предложил П. Наур на конференции НАТО в 1968 г. SE – это, по существу, технология разработки программ, систем и ПП в соответствии с жизненным циклом (ЖЦ) программного обеспечения (ПО), представленным позднее в стандарте ISO/IEC 12207–1996, 2007. В 2001 г. комитеты ACM и IEEE выпустили стандарт для Software Engineering Body of Knowledge (SWEBOK) разработки ПО систем. SWEBOK содержит 10 областей знаний (Area Knowledges) для проведения разработки ПО, начиная от требований на ПО и вплоть до создания продукта для конкретной системы.

Технология программирования и программная инженерия активно использовались у нас в стране и за рубежом. После развала СССР SE стала более активно использоваться в нашей стране как инструмент разработки сложных систем. В результате ТП реже применяется, а в основном – технология и программная инженерия. Для SE известный в стране технолог В.В. Липаев дал трактовку областей знаний SWEBOK, стандартов ISO/IEC 12207 ЖЦ, ISO/IEC 9000 (1–4), 9126 – качество, а также сертификации разработки масштабных систем.

В данном учебно-методическом пособии содержится описание истории развития отечественной технологии программирования в начальный период появления первых ЭВМ. Будут рассматриваться фундаментальные основы ТП систем по ТЛ для производства ПП в СССР и дается анализ продуктовых линий ProductLine/ProductFamily индустрии ПП и систем.

2.1. Технология программирования больших систем

Программирование вычислительных задач начало развиваться на цифровых ЭВМ с помощью языков представления алгоритмов и программ, их преобразования в коды машин средствами программирующих программ (ППр) или систем программирования (трансляторов, интерпретаторов) с языков описания программ (Адресный, АЛГОЛ-60, Фортран, Кобол, АЛГОЛ 68 и др.). Трансляторы разрабатывались для отечественных ЭВМ (БЭСМ, М-20, М-50, Минск 32, Стрела, СМ1-3, Днепр-2 и др.) [1–4] в рамках ТП, а также создавались системы автоматизации методами синтезирующего, сборочного программирования, технологического комплекса РТК с Р-графическим языком задания программ и документов технологии В.В. Липаева по изготовлению больших программ в ВПК и др.

2.1.1. *Создание систем программирования*

Первые монографии в СССР «Элементы программирования» А.И. Китова, И.А. Криницкого (1956), «Быстродействующие электронные машины АН СССР» С.А. Лебедева (1952), «Общее описание БЭСМ и методика выполнения операций» С.А. Лебедева и В.А. Мельникова (1958) и др. освещали аспекты создания программ решения разных математических задач на ЭВМ. Их работы открыли эру алгоритмизации вычислительных задач с помощью языков, близких ЭВМ. Это: «Алгоритмы и машинное решение задач» (В.А. Трахтенгерц, 1957), «Начальные сведения о решении задач для ЭВМ» (А.А. Ляпунов и Г.А. Шестопалов, 1957), «Графсхемное программирование» (А.А. Ляпунов, А.П. Ершов), «Адресное программирование» (Е.Л. Ющенко, 1962) и др. [5–8].

Основные элементы отечественной ТП

Программирующая программа. Первая идея программирующей программы (ППр) возникла в 1953 г. у А.А. Ляпунова, который читал курс по теории представления алгоритмов с помощью операторных граф-схем

в МГУ. На языке операторных схем программа представлялась схемой, в виде управляющего графа и совокупностью спецификаций его операторов. В основе входного языка каждой из ПП лежал общий концептуальный базис, фиксирующий типы операторов и их спецификации. Программа ППр-1 сделана в 1954 г. с участием Э.З. Любимского, А.П. Ершова в МГУ под руководством М.Р. Шуры-Буры. Это первая ППр в мировой практике. ППр-1 стала прототипом для проектирования других ППр на машинах БЭСМ, Стрела, М-20 и др. В результате были созданы фундаментальные алгоритмы трансляции и положено начало системному и теоретическому программированию. Важную роль сыграли теоретические работы А.П. Ершова «Введение в теорию программирования» (1977), В.М. Глушкова, Г.Е. Цейтлина и Е.Л. Ющенко «Алгебра. Языки. Программирование» (1974), М.Р. Шуры-Буры, Э.З. Любимского «Транслятор ТА2 на языке АЛМО» (1970) и др. [8–13].

Идея ППр развивалась и в ВЦ АН УССР в период создания машины МЭСМ А.А. Лебедевым. В.М. Глушков в статье «Об одном методе автоматизации программирования» (Проблемы кибернетики. 1957. № 2. С. 181–189) и А.А. Стогний (там же, с. 190–199) в статье «О принципах построения специализированной ППр» определили библиотечный метод алгоритма решения систем дифференциальных уравнений. ПП использовались на разных машинах: УМШН, Проминь, Днепр, Днепр-2 и др. Трансляторы для этих машин были реализованы в адресном языке Е.Л. Ющенко и в специализированных языках типа Автокод [7–9].

Библиотеки стандартных программ. Е.А. Жоголев (МГУ) разработал стандартную программу, которая осуществляла статическую загрузку и связывание программ из библиотеки численных методов. Интерпретирующая система ИС-2 (М.Р. Шура-Бура) использовалась на машинах М-20, БЭСМ и др. В ней реализован метод динамического вызова библиотечных подпрограмм, их связывание, подкачка и замена. ИС-2 стала неотъемлемой частью разных ОС ЭВМ, поставляемых пользователем [15].

Операционные системы. Для БЭСМ-6 была сделана ОС в ИТМ и ВТ под руководством В.П. Иванникова, которая выполняла общие функции управления памятью, задачами и данными. ОС и долгое время выполняла эти функции. Позднее ОС получила дальнейшее развитие в ВС «Электроника ССБИС» (1985–1991). Ее цели и задачи опубликованы в ряде статей, в том числе и в итоговой статье [16]. В ней представлена ОС и ее ядро. В это ядро входят стандартные средства ОС, средства взаимодействия с IBM и Эльбрус, набор протоколов обслуживания программ и система программирования с ЯП (Фортран, С, Паскаль и др.). Базовая система программирования включает новые средства поддержки абстрактных типов данных, кластерного сборочного конвейера для загрузки модулей в ОС и в

библиотеку программ трансляторов, а также отладчик программ на ЯП и сервисные средства обслуживания пользователя.

Системы программирования

С 1960 года в стране появилось много языков программирования (ЯП). Это Адресный, Алгол-60, Фортран, Кобол, Пролог, Ада и др.

Одним из первых языков, используемых для программирования математических задач и ПП, был Адресный язык.

Адресный язык (Ющенко Е.Л., Гнеденко Б.В., Королюк В.С.) был близок к языку математики. Алгоритм в этом языке записывался набором строк операторов, в каждой из которых записывается одно или несколько действий – формул. Для задания порядка операторов используются метка и операторы безусловного перехода. Переменные обозначались буквами и соответствовали ячейкам машины. Содержимое некоторого адреса отмечалось указателем, адресом второго ранга и использовалось для многократного перехода. Этот язык также применялся для описания программ вычислительного характера и реализации ПП для машин Киев, Урал и Днепр.

Язык Алгол-60. Он возник в США как универсальный язык программирования научных и инженерных задач [11]. Этот язык в ВЦ АН СССР был принят языком реализации научных задач, и сформировалось три направления разработки трансляторов с языка Алгол (ТА): ТА1 – С.С. Лавров (ЛГУ, 1962), ТА2 – М.Р. Шура-Бура и Э.З. Любимский (ИПМ, 1965), ТА-3 (Альфа-система) в русской версии языка Алгол-60 (СО АН СССР, 1966), ТА-4 – Е.Л. Ющенко, Е.М. Лаврищева – для УВК «Днепр-2» (ИК АН СССР, 1967) [17, 18].

На основании этого языка в трансляторе ТА-1 реализована простая схема трансляции, стековый подход к реализации выражений, процедур без рекурсий. В трансляторе ТА-2 разработан оригинальный алгоритм реализации процедур, механизм управления памятью (оперативной и внешней) и метод таблично-управляемой генерации кода. В ТА-3, АЛЬФА-трансляторе [13] реализована оптимизация (выражений, циклов, процедур, памяти и др.) для повышения эффективности выходного кода, который был близок коду, созданному вручную. В ТА-3 реализованы также операции над многомерными значениями и комплексными типами данных. В трансляторе Д-АЛГАМС (ТА-4) реализован СМ-метод табличного представления БНФ и набора семантических программ их реализации (Yushchenko E.L., Lavrishcheva E.M. A method of analyzing programs based on a machine language. Cybernetics. 1972. V. 8, N 2. P. 219–223 и др.). Новым подходом к реализации системы программирования «Днепр-2» было создание общего арифметического блока для двух языков Алгол и Кобол.

Трансляторы ТА1-ТА3 и ОС для новых ЭВМ (М-20, СТРЕЛА, БЭСМ) были реализованы в машинном коде этих машин, а транслятор ТА4 – на языке, близком Адресному языку – Автокод «Днепр-2». Транслятор АКД и Д-АЛГАМС входили в состав общесистемного ПО машины «Днепр-2» [19]. Эти трансляторы были сданы в составе ОМО (ОС и систем программирования АКД, Д-АЛГАМС) УВК «Днепр-2» Госкомиссии СССР 6.12.1967. Комиссию возглавлял А.В. Дородницын. В ее состав входили Л.Н. Королев, В.М. Глушков, А.Н. Кухарчук и др. Эта машина имела счастливую судьбу. Ее купила ГДР в 1971 г., ПО машины работала там до 1991 года. На ней еще работала АСУ ТП металлургическими процессами, разработанная совместно со специалистами ГДР и Украины.

Системы программирования включали трансляторы, отладчики и редакторы программ. Э.З. Любимский (1963) предложил промежуточный язык – АЛМО для перевода теста в любом ЯП в этот язык, а затем тест АЛМО – в код ЭВМ [10]. АЛМО – это некоторая абстрактная машина, отражающая особенности класса ЭВМ в СССР. Этот язык и другие языки (Эпсилон, Сигма) стали языками-посредниками при трансляции программ с различных ЯП. Их смысл состоял в замене трансляции с m входных языков в n машинных языков, т.е. «из m в один» и «из одного в n». АЛМО реализован для основных машин того времени (М-20, БЭСМ-6, Минск-2, Урал-1) и проверен для трансляторов с Алгол-60 и Фортран [11–14].

Одной из систем программирования с ЯП была система с языка АЛГОЛ 68 (Г.С. Цейтин, А.Н. Терехов, ЛГУ, 1968–1991), переведенного на русский язык в 1979 году. Многие годы разрабатывались варианты трансляторов в ЛГУ, Минске, Новосибирске (Бета). Однако наиболее реальный транслятор был сделан под руководством А.Н. Терехова для первых ЭВМ – СМ1, СМ2, а потом для ОС ЕС и НИЦЕВТ финансировал эту работу. Она была одна из сложных и трудных разработок в стране. В работе [20] А.Н. Терехов описал наработанные многими членами РГ по АЛГОЛ 68 вопросы реализации транслятора и программ перевода языковых конструкций сначала в промежуточный язык, а потом в код машины. История развития и создания системы с языка АЛГОЛ 68 описаны в [21].

Опыт разработки программ на Алгол-60 и первые трансляторы рассматривались на первой Всесоюзной конференции (ВКП-1) по программированию в Киеве (1968), второй конференции ВКП-2 в Новосибирске (1978) и на последующих конференциях по ТП (1979, 1982–1992). В работе каждой конференции принимали участие более 1000 человек. Этим объясняется большая популярность проблематики ТП в нашей стране. По линии ГКНТ, ГКНТИ СССР в Алуште под руководством проф. Е.Л. Ющенко проводились ежегодно Всесоюзные семинары (1981–1990) по разным аспектам ТП. На конференциях и семинарах обсуждались но-

вые теоретические и прикладные аспекты проектирования, разработки, тестирования различных видов систем для больших ЭВМ, а также задачи эксплуатации и сопровождения ПО [21, 22].

ГКНТ СССР финансировал работы по разработке средств автоматизации ТП. Были специальные постановления кабинета Министров СССР, направленные на развитие средств вычислительной техники и технологии изготовления ПП для сдачи их в республиканские Фонды алгоритмов и программ и государственный Фонд СССР (1970–1991).

Системы синтеза, композиции и сборки программ

Систему ПРИЗ разработал под руководством академика ЭССР Э.Х. Тыугу. Она обеспечивает синтез программ в языке *Утопист* и программ в ЯП. В этом языке задается семантическая модель предметной области решения математических задач, представляемых пакетами прикладных программ (ППП). Метод синтеза по модели и на основе семантических программ в PL/1, Fortran, Assembler реализован путем подстановки данных в модель семантических программ. Этими процедурами управляет система ПРИЗ в ОС ЕС [23, 24].

Композиция программ. Композиционное программирование [25] служило способом объединения функций и данных в виде цепочек: «данные–функция–имя», «функции–композиция–дескрипция». Программной поддержкой этих цепочек являлась система ДЕФИПС, которая обеспечивала построение программ из функций, заданных на некотором множестве именованных данных, дескрипций и денотатов (значений). Семантика программ задавалась ординарными функциями обработки операций, интерфейсных функций, 0-арных функций и именованных данных. Операции композиции образуют подкласс стандартных композиций и декомпозиционных функций.

Сборка программ. Выпуск сложных программ В.М. Глушков определил как сборочный конвейер Форда. По его мнению, фабрика программ будет оборудована линиями программирования разных программ и линией сборки их в новую, более сложную структуру. Базисом сборки были модули как главный элемент программирования. Обмен готовыми модулями потребовал разработки *паспорта*, который содержал описание входных и выходных данных, а также операторов вызова (Call, RMI и др.) других модулей. Модули описывались в разных ЯП и имели паспорт, который определял связь (интерфейс) с другими модулями. Это описание упрощало процесс сборки разноязыковых модулей, записанных в ЯП (Алгол, Кобол, Фортран, ПЛ-1, Модула и др.). Позднее (1994) для описания интерфейса был создан язык IDL (Interface Definition language) и брокер объектных запросов (ORB) в системе CORBA в рамках объектного подхода.

Описание интерфейса через посредников stub и skeleton стал основным элементом сборки любых модулей, объектов и компонентов [26, 27].

Модульным программированием занимались в стране Е.П. Жоголев, А.П. Ершов, Е.М. Лаврищева, В.И. Орлов и др. [28, 29]. В 1980-х годах бурно развивалось направление автоматизации прикладных систем и ППП. В ИК был выполнен ряд проектов, финансируемых ГКНТ СССР. Это проекты: система автоматизации программ – АПРОП (Лаврищева Е.М.); комплекс программиста ТКП (Вельбицкий И.В.); система алгоритмических алгебр Мультипроцесист (Цейтлин Г.Е.) и др. Средства автоматизации были разработаны в ряде других научных центров СССР: система «ПРИЗ» (ЭССР, Тыугу Э.Х.), системы «Альфа», «Бета» (Новосибирск, Ершов А.П.), система АПРОП (Киев–Москва, Лаврищева Е.М.), система автоматизации математических задач (ИПМ АН СССР, Корягин Д.Н.), система модульного программирования (МГУ, Жоголев Е.А.), система послойного проектирования ПО (Ростов, Фуксман В.И.) [30] и др.

При выполнении научных проектов сформировались разные аспекты ТП к производству прикладных систем, АСУ и ППП с использованием готовых модулей, названных позднее компонентами повторного использования (КПИ). В 1975 г. вышло постановление КМ СССР «Программы – продукты производственного назначения» и другие постановления ГКНТ СССР о том, что программы общего назначения должны были сохраняться в Фондах алгоритмов и программ республиканского и Всесоюзного значения (1978–1992).

2.1.2. *Развитие технологии сборочного программирования*

Технология программирования развивалась в Украине по трем перспективным направлениям, сформулированным в статье В.М. Глушкова «Фундаментальные основы и технология программирования», ж. «Программирование», 1980 [31–38]:

- модульная система автоматизации производства программ (АПРОП) из стандартизированных программных заготовок в сложные системы [34–36];
- метод формализованных технических заданий для проектирования сложных программных комплексов ПРОЕКТ [32];
- Р-технология программирования для автоматизации проектирования систем средствами графического Р-языка для представления структур программ и данных в АСУ [39, 40].

Модульная система автоматизации программ АПРОП разрабатывалась исходя из тезиса В.М. Глушкова – конвейерная сборка разнородных модулей и интерфейсов (Лаврищева Е.М.) и в системе ПРОЕКТ (Капитонова Ю.В., Летичевский А.А.). На этой основе создавались ППП (Молчанов И.Н.); ППП математического и статистического типов (Сергиен-

ко И.В., Редько В.Н., Стукало А.С.); Р-технология (Вельбицкий И.В.), ТЕРЕМ (Мищенко Н.М.). Этими работами был внесен весомый вклад в сборочное создание ПП на ЕС ЭВМ.

В ТП сформировался новый вид – сборочное программирование для объединения разнородных модулей в более сложные структуры (АПРОП) [35, 36]. Эта технология развивалась и в отделе Глушкова для семейства трансляторов с широко используемых ЯП. Основные положения этого вида сборки базировались на идее выделения общих средств в ЯП ОС ЕС, системной реализации компонентов языковых процессоров и сборки из них трансляторов (СПТ ТЕРЕМ). В СПТ разработаны общие компоненты языковых процессов в классе языков ОС ЕС (Мищенко Н.М. О сборочном программировании языковых процессоров // Интеллектуализация программного обеспечения информационно-вычислительных систем: сб. науч. ст. Киев: Ин-т кибернетики им. В.М. Глушкова АН УССР, 1990. С. 45–52) [31, 33, 39]. СПТ включала ЯП ОС ЕС (Алгол, Кобол, ПЛ1 и др.) и построена для МВК с макроконвейерной организацией вычислений и средствами внесения изменений в программы ЯП. Более половины универсальных модулей для ЯП реализованы и вошли в ядро СПТ ТЕРЕМ, которая сдана Госкомиссии в 1989 г. [39].

В системе ПРОЕКТ реализован метод формализованных технических заданий для проектирования дискретных систем [32] с применением теории дискретных преобразований. Основу метода составляет формальное описание функций системы с помощью языка L2, аналогичного языку Аналитик [40] и доказательства правильности функций с помощью дискретных преобразователей и с применением алгебраических операций и средств обработки разных структур данных. К структурам данных относятся простые данные (числа, символы) и сложные данные (массивы, указатели). Вводится понятие схемы программы, содержащей множество переходов и состояний. Дается описание алгоритма проектирования системы по данному методу и системы обработки данных с применением метода Флойда. Метод прошел внедрение на практике при разработке ПО системы МАЯК [41].

Р-технология создалась под руководством И.В. Вельбицкого [39] для конструирования структур программ с помощью визуальных Р-графов и схемной их реализации. Было разработано устройство синтаксического контроля программ, которое было запатентовано и за рубежом. В нем выполнена структурная интерпретация синтаксиса и семантики ЯП с использованием Р-языка. Затем был создан ТКП для проведения анализа программ в Р-языке и комплекс РТК для машин БЭСМ-6, ЕС ЭВМ и СМ ЭВМ. РТК использовался в различных закрытых организациях СССР для обработки текстовой информации и генерации средств производства программ для АСУ, САПР и информационных систем [42, 43].

Интерфейсы в системе сборочного программирования

Концепция интерфейса в рамках системы АПРОП включала межмодульный, межъязыковый и технологический интерфейсы [35, 36, 38]. Первое определение понятия *интерфейса* и языка его описания сформулировано в проекте системы в 1976 г. [34]. Идея интерфейса намного опередила зарубежные разработки. Использовался так называемый язык MIL (Module Interface Language) для первоначальной сборки. В зарубежной литературе интерфейс появился значительно позже. В настоящее время для представления интерфейса используется программный интерфейс API (Application Programs Interface), языковый интерфейс IDL (Interface Definition Language), научный интерфейс SIDL (Scientifically IDL) для связи научных артефактов.

Межмодульный интерфейс – это модуль-посредник между двумя связываемыми программными объектами, выполняет функции передачи, приема и преобразования нерелевантности данных между ними. Язык определения интерфейсов (ЯОИ) в системе АПРОП сыграл свою роль при сборке разноязыковых модулей.

Межъязыковой интерфейс – совокупность средств и методов представления и преобразования структур и ТД ЯП с помощью алгебраических систем (типов данных и операций, т.е. функций интерфейса в библиотеке, которые обеспечивают взаимно однозначную передачу данных между разноязыковыми модулями через интерфейс (например, преобразование матрицы по строкам в Фортране в матрицу по столбцам в ПЛ/1 и обратно). Библиотека интерфейсных функций (64) в период широкого использования ЕС была разработана по договору с В.В. Липаевым (1982–1987) в рамках системы ПРОТВА и передана в 52 организации СССР по актам внедрения вместе с этой системой и самостоятельно при работе с разными языками в среде ОС ЕС (ПЛ/1, Algol, Fortran, Cobol, Assembler) [26–28].

Технологический интерфейс – совокупность методов и средств для осуществления процессов и операций ТЛ при реализации ПП, а именно нормативные, методические документы и формы (каркас ТЛ, формат документа ТЛ, язык связи процессов и др.). Данный интерфейс включал операции контроля результатов процессов, оценки заданных в требованиях показателей качества, изменения состояния продукта на процессах ЖЦ и передачи следующему процессу и др. Методика создания ТЛ (1987) проверена на шести линиях заказа АИС «Юпитер-470» и является первой работой по формализации ТЛ. Альтернатива ТЛ и интерфейса 15 лет спустя (2004) – это продуктовые линии (ProductLines) института SEI США (http://sei.cmu.edu/productlines/frame_report/). Концепция технологического интерфейса автора отмечена грамотой на Европейском конкурсе «Интерфейс СЭВ» (1987) [44].

Созданная нами концепция интерфейса модулей была автоматизирована. Роль генератора интерфейсных модулей-посредников выполняла подсистема реализации *интерфейса* в системе АПРОП (1975–1982). Она описана в монографии «Связь разноязыковых модулей в ОС ЕС» (М.: Финансы и статистика, 1982. 127 с.). Там же дано описание первого *языка описания интерфейса* (ЯОИ) и библиотеки межъязыкового интерфейса (из 64 функций), которые использовались для генерации посредников автоматизировано или вручную на основе описания модуля-посредника в языке ЯОИ. Эти средства имели большой успех среди пользователей разных ЯП ОС, они были переданы в составе ПРОТВА и самостоятельно как АПРОП в десятки организаций страны и способствовали сокращению объема работ при сборке разноязыковых объектов через интерфейсные посредники [37, 45–47].

Таким образом, было сформировано сборочное программирование, объектами которого являются любые готовые программы, модули и КПИ, а также интерфейс как способ объединения, комплексирования (по Липаеву), сборки ПС и семейств ПС из готовых объектов. Он занял свое место среди других методов. Как показали исследования [48], сборка остается главной, а также базисом других методов программирования на многочисленных современных фабриках программ и сервисов, работающих в среде операционных сред (MS.Net, Corba, Java, VSphere, Unix, Linex и др.) [49].

Система АПРОП разрабатывалась по договору с Институтом приборостроения (Москва) в составе технологии создания программ для бортовых систем ПРОТВА в рамках Министерства радиопромышленности СССР, реализованной под руководством В.В. Липаева [45, 50]. В эту систему были внедрены интерфейс (межмодульный, межъязыковый и технологический) [35, 36, 38] и библиотека интерфейсных функций преобразования нерелевантных типов данных, описываемых в модулях на разных языках и для разных платформ. Был разработан стандарт описания модулей. Он описывался в виде информационной части описания самого модуля и содержал входные и выходные параметры, передаваемые модулям друг другу. Это описание составляло интерфейс модулей. *Интерфейс стал* средством интеграции новых прикладных систем из готовых КПИ в системы с обеспечением их взаимодействия в сетевых средах [51].

А.П. Ершов считал, что «сборочное программирование эффективно, поскольку готовые запрограммированные модули позволяют быстро решить любые задачи из определенной проблемной области для ЕС ЭВМ и мини-, микро- и макро-ЭВМ» [52, 53]. Сборка стала важным технологическим решением для индустрии создания ПП как продукции производственно-технического назначения.

Метод сборки разноязыковых программ был:

- опубликован в зарубежной прессе – Lavrishcheva E.M. Modular design of large programs. Cybernetics. 1980. V. 16, N 2. P. 244–249;
- защищен Е.М. Лаврищевой в докторской диссертации «Методы, средства и инструменты сборочного программирования» (1989). Ее оппонентами стали известные специалисты СССР – Э.Х. Тыугу, А.П. Ершов (заменен на Э.З. Любимского в связи со смертью 07.12.2008) и И.В. Вельбицкий;
- описан в монографиях: «Связь разноязыковых модулей в ОС ЕС» (М.: Финансы и статистика, 1982. 127 с.) и «Сборочное программирование» (Лаврищева Е.М., Грищенко В.Н., 1991) и в книге «Технология сборочного программирования» (Липаев В.В., Позин Б.А., Штрик А.А., 1992);
- опубликованы методы и средства компонентного программирования: Grishchenko V.N., Lavrishcheva Ye.M. Methods and Tools of Component Programming. Cybernetics and Systems Analysis. 2003. V. 39, N 1. P. 33–45 и др.

2.1.3. *Теория сборочного программирования*

В сборочном программировании метод сборки или объединения КПИ (assets, reuses, services и т.п.) в *новые* программные структуры (ПС, семейство систем, продуктовые линии – Product lines и т.п.) в 80-х годах прошлого века выполнялся автоматизировано, объектами которого являются: разноязыковые программные объекты, их паспорта и библиотеки КПИ [26, 51].

Метод сборки базируется на теории спецификации и отображении типов и структур данных ЯП с помощью алгебраических систем, включающих простые и сложные типы данных (ТД), операции над ними и функции эквивалентного преобразования одних ТД в другие. Главный объект сборки – КПИ, которым, по своей сущности, соответствует конвейерная сборка (как говорил В.М. Глушков) на фабрики программ, подобно сборки автомобилей из готовых комплектующих и стыковочных деталей. В нем роль комплектующих «деталей» выполняют КПИ разной степени сложности, а роль стыковки – интерфейсы.

Процесс сборки любых изделий характеризуется не только готовыми комплектующими «деталями и узлами», а и схемой сборки, задающей связи отдельных компонентов и правилами взаимодействия между собой разных объектов ТЛ на сборочном конвейере.

Метод сборки разноязыковых модулей в новые программные структуры основан на математических формализмах определения связей (по данным и по управлению) между объектами сборки и генерации интерфейсных модулей для каждой пары объединяемых модулей.

Сущность задачи сборки пары разноязыковых модулей состоит в определении взаимно однозначного соответствия между задаваемым множеством фактических параметров $V = \{v_1, v_2, ..., v_\kappa\}$ вызывающего модуля и соответствующим множеством формальных параметров $F = \{f_1, f_2, ..., f_\kappa\}$ вызываемого модуля, а также в отображении типов данных одних параметров в другие. Если отображение не удается выполнить, то задача автоматизированной связи для данной пары модулей считается неразрешимой [52, 53].

Типы данных, которыми обменивается между собой пара модулей *Mi* и *Mj*, могут быть эквивалентными, если они имеют одинаковую семантическую структуру и обработку. Если они обмениваются неэквивалентными типами Di^j *i* Di^k от *Mi* к *Mj*, то проводится их преобразование с помощью функций, заданных такими отображениями:

$$FNi^\kappa : Ni \rightarrow N^\kappa,$$

$$FTi^\kappa : Ti \rightarrow T^\kappa,$$

$$FVi^\kappa : Vi \rightarrow V^\kappa.$$

Кроме того, между множеством Di^j и Di^k может не существовать взаимно однозначного соответствия. Например, когда нескольким элементам множества Di^j соответствует один элемент из множества Di^k и наоборот. В этом случае строится отображение нескольких типов к одному общему типу с помощью функции конструирования: $C(di^{j1}, ..., di^{jk}) = di^j$, в которой di^j является одним из элементов множества Di^j.

Отображения FNi^κ, FTi^κ, FVi^κ содержат одинаковое количество элементов. Задача отображения FNi^κ выполняется путем упорядочивания имен переменных в описании интегрированных программных компонентов. Отображение типов данных FTi^κ базируется на множестве ТД:

$$T = (X, \Omega),$$

где X – множество значений, которые могут способствовать изменению типа и элементов множества *V*;

Ω – множество операций для выполнения этих изменений.

То есть множество *T* рассматривается как алгебраическая система.

Преобразование ТД осуществляется с помощью алгебраических систем, содержащих для каждого типа множество значений и операций над ними. Каждой операции преобразования типов данных соответствует изоморфное отображение одной алгебраической системы в другую.

В общем, задача отображения A: П → Ф для множества параметров *V i F* состоит в построении *V* и *F*:

$$\Pi = \{V_1, V_2, \ldots, V_m\}, \quad \Phi = \{F_1, F_2, \ldots, F_m\},$$

$$\bigcup_{t=1}^{m} V^t = V, \quad Vt \cap Vt` = \varnothing \quad \text{при} \quad t \le t`,$$

$$\bigcup_{t=1}^{m} F^t = F, \quad Ft \cap Ft` = \varnothing \quad \text{при} \quad t = t`.$$

Формально преобразование модулей *Mi* и *Mj* с неэквивалентными типами данных в ЯП выполняется следующими этапами.

Этап 1. Построение операций преобразования типов данных $T = \{T_\alpha^t\}$ для множества языков программирования $L = \{l_\alpha\}_{\alpha = 1,n}$.

Этап 2. Построение отображения простых ТД для каждой пары взаимодействующих компонентов в $l_{\alpha 1}$ и $l_{\alpha 2}$ и применение операций селектора *S* и конструктора *C* для отображения сложных структур данных.

Формализованное преобразование типов данных осуществляется с помощью алгебраических систем для каждого типа данных T_α^t:

$$G_a^{\ t} = < X_a^{\ t}, \Omega_a^{\ t} >,$$

где *t* – тип данных; X_α^t – множество значений, которые могут принимать переменные этого типа; Ω_α^t – множество операций над этими типами данных, где *t = b, c, i, r, a, z, u, e*.

Простым и сложным типам данных современных ЯП соответствуют классы алгебраических систем:

$$\begin{aligned} \Sigma_1 &= \{G_\alpha^b, G_\alpha^c, G_\alpha^i, G_\alpha^r\}, \\ \Sigma_2 &= \{G_\alpha^a, G_\alpha^z, G_\alpha^u, G_\alpha^e\}. \end{aligned} \tag{2.1}$$

Каждый элемент класса ТД определяется на множестве значений и операций над ними: $G_\alpha^t = <X_\alpha^t, \Omega_\alpha^t>$.

Операциям преобразования каждого *t* ТД соответствует изоморфное отображение двух алгебраических систем с совместимыми ТД двух разных ЯП. В классе систем Σ_1 и Σ_2 преобразование ТД $t \to q$ для пары языков l_t и l_q обладает свойством изоморфизма:

1) G_α^t и G_β^q – изоморфны (*q* – задан на множестве, что и *t*);

2) между X_α^t и X_β^q существует изоморфизм, для которых множества Ω_α^t и Ω_β^q разные. Если $\Omega = \Omega_\alpha^t \cap \Omega_\beta^q$ не пусто, то изоморфизм существует между $G_\alpha^{t'} = <X_\alpha^t, \Omega>$ и $G_\beta^{q'} = <X_\beta^q, \Omega>$. Такое преобразование сводится к случаю 1).

Между множествами X_α^t и X_β^q может не существовать изоморфного соответствия. В этом случае необходимо построить такое отображение между X_α^t и X_β^q, чтобы оно было изоморфным. Если такое отображение

существует (в каждом конкретном случае оно может быть разным), то имеем условие случая 1) с соответствующими изменениями в определении алгебраических систем;

3) мощности алгебраических систем должны быть равны $|G_\alpha{}^t| = |G_\beta{}^q|$.

Любое отображение 1), 2) сохраняет линейный порядок, так как алгебраические системы (2.1) линейно упорядочены.

Лемма 1. *Для любого изоморфного отображения* φ *между алгебраическими системами* $G_\alpha{}^t$ *и* $G_\beta{}^q$ *выполняются равенства*

$$\varphi(X^t_{\alpha\ .\ \min}) = X^t_{\beta\ .\ \min},\ \ \varphi(X^t_{\alpha\ .\ \max}) = X^q_{\beta\ \max}.$$

Доказательство леммы тривиальное и простое. При условии, когда одно или два вышеприведенных равенства не выполняются, тогда для основных множеств алгебраических систем изменяется линейный порядок, что противоречит определению.

Формальные условия преобразования типов данных *t* = *c*, *b*, *r*, *a*, *z* определяются теоремами 1–5 [52, 53].

Теорема 1. *Пусть* φ *– отображение алгебраической системы* $G_\alpha{}^c$ *в систему* $G_\beta{}^c$. *Для того чтобы* φ *было изоморфизмом, необходимо и достаточно, чтобы* φ *изоморфно отображало* $X_\alpha{}^c$ *на* $X_\beta{}^c$ *с сохранением линейного порядка.*

Необходимость. Пусть φ – изоморфизм. Тогда при отображении сохраняются все операции множества $\Omega = \Omega_\alpha{}^c = \Omega_\beta{}^c$, в том числе и операция отношения, которая определяет линейный порядок $X_\alpha{}^c$ и $X_\beta{}^c$.

Достаточность. Пусть φ изоморфно отображает $X_\alpha{}^c$ на $X_\beta{}^c$ с сохранением линейного порядка. Операция отношения выполняется соответственно принципу упорядоченности. Операцию succ докажем с помощью леммы, согласно которой выполняется равенство $\varphi\ (X_\alpha{}^c\ .\ _{\min}) = X_\beta{}^c\ .\ _{\min}$.

Последовательно применяя операцию succ к этому равенству и учитывая линейную упорядоченность $X_\alpha{}^c$ и $X_\beta{}^c$ ($x < \mathrm{succ}(x)$), получаем, что для любого $x_\alpha{}^c \in X_\alpha{}^c$ и $x_\alpha{}^c \neq X_\alpha{}^c{}_{\min}$ из равенства $\varphi\ (X_\alpha{}^c) = x_\beta{}^c$, где $x_\beta{}^c \in X_\beta{}^c$, выполняется равенство

$$\varphi(\mathrm{succ}\ (x^c_\alpha)) = \mathrm{succ}(x^c_\beta). \qquad (2.2)$$

Операция pred доказывается аналогично с помощью $\varphi\ (X_\alpha{}^c{}_{\max}) = X_\beta{}^c{}_{\max}$.

Теорема 2. *Любой изоморфизм* φ *между алгебраическими системами* $G_\alpha{}^b$ *и* $G_\beta{}^b$ *является тождественным изоморфизмом*:

$$\begin{aligned} \varphi(X_{\alpha.\ \mathrm{false}}{}^b) &= X_{\beta.\ \mathrm{false}}{}^b, \\ \varphi(X_{\alpha.\ \mathrm{true}}{}^b) &= X_{\beta.\ \mathrm{true}}{}^b. \end{aligned} \qquad (2.3)$$

Доказательство. При отображении $G_\alpha{}^b$ и $G_\beta{}^b$ всегда справедливо $X_{\alpha.\ \mathrm{false}}{}^b < X_{\beta.\ \mathrm{true}}{}^b$. Поэтому для сохранения линейного порядка единственно возможным изоморфизмом является (2.3).

Теорема 3. *Любой изоморфизм между алгебраическими системами с соответствующими числовыми типами является тождественным автоморфизмом.*

Доказательство этой теоремы тривиальное и является следствием свойств элементов числовых множеств.

Теорема 4. *Пусть $G_\alpha{}^a$ и $G_\beta{}^a$ – алгебраические системы, которые отвечают типам данных массива* (*a*); φ_i *и* φ_v *– изоморфные отображения множеств индексов* (*i*) *и значений элементов* (*Y*) *массивов, которые сохраняют линейный порядок. Тогда изоморфизм* φ *между алгебраическими системами целиком определяется изоморфными отображениями:*

$$\varphi_i : X_\alpha^{\,a} \to X_\beta^{\,a},$$
$$\varphi_v : Y(X_\alpha^{\,a}) \to Y(X_\beta^{\,a}).$$

Изоморфизм φ между алгебраическими системами $G_\alpha{}^a$ и $G_\beta{}^a$ определяется отображениями φ_i и φ_v, которые сохраняют линейный порядок и упорядоченность элементов массива.

Теорема 5. *Пусть $G_\alpha{}^z$ и $G_\beta{}^z$ – две алгебраические системы, которые отвечают типам данных «запись» или структуре, и $x_\alpha{}^z \in X_\alpha{}^z$, $x_\beta{}^z \in X_\beta{}^z$. Тогда если между последовательностями компонентов записей $x_\alpha{}^z$ и $x_\beta{}^z$ существует взаимно однозначное соответствие, то изоморфизм* φ *между $G_\alpha{}^z$ и $G_\beta{}^z$ определяется изоморфными отображениями алгебраических систем, которым соответствуют компоненты записи или структуры.*

Преобразования между массивами и записями сводятся к преобразованию простых типов данных их элементов. Преобразования между действительными типами и другими числовыми значениями предполагают использование эмпирических случаев, так как отсутствует изоморфизм основных множеств этих алгебраических систем.

Операция конструирования *C* массива состоит в формальном приведении в порядок компонентов и определении соответствия между множеством индексов и множеством элементов массива. Аналогично эта операция определяется для записи.

Операции сборки модулей. В системе АПРОП сборка базируется на совокупности модулей, их паспортах и операторах сборки частей программ и сложных систем. Она осуществляется следующими операциями:

– Link – оператор сборки разноязыковых модулей;

– Link seg A (A2, A3, *A4) – связать модули A, A3 и A4 в виде сегмента A, где модуль A4 вызывается динамически;
– Link Prog B ((B1, B2), C = X(C1), D = (Y, D1=Y1)) – объединить модули B1, B2, а затем C и D с параметрами C1. Y, D1;
– оператор вычислить // EXEC *модуль* A_1 // PL *Trans* A_1;
– генерация интерфейсного связника mod-interface for $A_1 \cap A_2$ и др.

Оператор сборки определяет совместимость объединяемых объектов, которые содержат описание функций для согласования разных характеристик, и представлены в их паспортах.

2.1.4. *Реализация связи языков и программ в распределенных средах*

С годами проблема связи разноязыковых (по коду и среде) программ обострилась в связи с быстрым изменением архитектуры компьютеров, появлением распределенных, клиент-серверных сред и т.п. Проявилась неоднородность ЯП как в смысле представления в них типов данных, так и платформ компьютеров, на которых реализованы соответствующие системы программирования, а также в различных способах передачи параметров между объектами в разных средах – маршаллинг данных через разные виды операторов удаленного вызова. Единого подхода к решению проблемы интерфейса пока не существует. Стандарт ISO / IEC 11404-96 определил подход к решению вопросов интерфейса всех видов ЯП с помощью универсального языка LI (Language Independent), независимого от ЯП. Однако инструментальной его поддержки до настоящего времени нет. Пользователям разных ЯП приходится выбирать подходящую реализацию интерфейса из множества имеющихся в разных средах [53]. Рассмотрим особенности сред, влияющих на реализацию интерфейса.

К ним относятся:

– разные двоичные представления результатов компиляторов для одного и того же ЯП на разных архитектурах компьютеров;
– двунаправленность связей между ЯП и их зависимость от среды и платформы;
– параметры вызовов объектов отображаются в операции методов;
– связь с разными ЯП через ссылки на указатели;
– связь модулей в ЯП осуществляется через интерфейсы каждой пары из множества языков (L_1, …, L_n) среды.

Необходимыми условиями применения метода сборки является:

– наличие большого количества разнообразных КПИ и ГОР;
– паспортизация объектов сборки;
– наличие набора правил взаимосвязи объектов и линий с операциями установления связей между КПИ.

Линии продуктов. Подход к построению ТЛ сформировался при выполнении Всесоюзного проекта АИС «Юпитер» (1982–1991) автоматиза-

ции Военно-морского флота СССР. Разработан метод технологической подготовки работ (ТПР) для создания специальных ТЛ [54, 55].

ТЛ комплектуется из процессов, которые соответствуют некоторой ПрО, стандартных инструментов и модулей реализации специфики функций ПрО системными средствами и нормативно-методическим обеспечением. Набор процессов ТЛ создается с учетом требований межведомственного стандарта ГОСТ 3918–1999. Процессы поддерживаются отобранными методами, средствами и инструментами, которые преобразуют состояния промежуточных элементов процессов и их выполнения в заданной среде.

Для сборки программных элементов подбираются готовые ресурсы, КПИ, средства и инструменты порождения и реализации отдельных функций. На проекте АИС «Юпитер» было разработано 5 ТЛ и комплект документов стандарта предприятия для разработки программ для ВМФ. Каждый исполнитель, сборщик модулей, КПИ выполняет по ТЛ процессы и операции для получения некоторого вида продукта. Исполнители новых программ: менеджер, разработчик, верификатор, валидатор, тестировщик, оценщик качества и др. Они осуществляли сборку систем из КПИ.

Перспективы развития ТП по Ершову (1986)

Академик А.П. Ершов выступил на Всесоюзной конференции «Технология программирования» в 1986 году и сказал следующее [52, 53].

«Следует различать технологию программирования как технологическую теорию, как конкретный способ организации и осуществления создания, распространения и сопровождения программного продукта и как процедуру индивидуальной деятельности профессионала, разрабатывающего или принимающего участие в разработке ПП».

В своем докладе он дает новую трактовку ТП. «Технологии и методологии – это всегда наука, в то время как метод входит в технологию составной частью. Говоря о технологии профессионального, производственного программирования, следует отметить принципиальную важность отчуждаемости и тиражирования программного продукта. Сколько-нибудь технология начинается тогда, когда она охватывает ЖЦ ПП.

ТП – это совокупность методологических положений, организационно-административных установлений, инструментально-технических средств, их информационного и программного обеспечения, регламентирующая и поддерживающая производственную деятельность людей, вовлеченных в процесс создания, распространения и сопровождения ПП.

Законченная ТП должна:

- охватывать весь жизненный цикл ПП;
- способствовать применению методологии, повышающей уровень достоверности, надежности и доказательности программирования;

на современные технические средства в виде автоматизированных рабочих мест, объединенных в локальную сеть;
- обеспечивать организационную управляемость и контролируемость производственных процессов;
- обеспечивать устойчивость ПП по отношению к смене технических средств;
- обеспечивать развиваемость ПП в связи с изменением условий функционирования целевой системы, использующий этот продукт и других условий».

Пример ТП на мировом уровне приведен для фирмы IBM (1980, № 4 на примере CHATTL и 1984, № 9).

Ориентиром для развития ТП в советских условиях является статья А.П. Ершова в журнале «Программирование», 1986, № 3. В ней он говорит о трех поколениях интегральной промышленной ТП и ее перспективах.

«**Первое направление** (*организационное программирование*) 1975–1985 гг.

Язык программирования не формализован. Переход от прототипа к программной версии не формализован…

Языки программирования – ФОРТРАН, КОБОЛ, ПЛ/1, Ассемблер.

База знаний отсутствует…

Развитие продукта – версионное.

Второе направление (*сборочное программирование*) 1985–1995 гг.

Язык спецификации регламентирован.

Переход от протопипа к промышленной версии регламентирован. Язык разработки – формализованный язык высокого уровня со средствами модуляризации.

Язык программирования объединен с языком разработки, допуская комплексирование с ассемблерными модулями…».

Третье направление (*доказательное программирование*) 1995–2005 гг.

Язык разработки формализован и содержит систему формальных преобразований, необходимых для доказательства программ…

Язык программирования объединен с языком разработки…

База данных проекта машинизирована.

Применение ЭВМ в проекте – полное.

Развитие продукта – эволюционное – *адаптивное*…».

В заключение он сказал – «Было бы полезно выработать норматив по технологии *второго поколения*, который, не затрагивая конкретного методологического или языкового наполнения, унифицировал бы:
- общую этапность разработки ПП;
- нормативы производительности и надежности;
- организационно-документационную структуру;

– вычислительную инструментальную среду;
– ***межмодульный интерфейс***, поддерживающий принцип *сборочного программирования…*».

В завершение рассмотрения сборки разнородных программ приведена конкретная реализация процесса сборки в системе .NET.

2.1.5. *Сборка программ в современных средах*

Среда MS.NET. Платформа .NET состоит из следующих компонентов [56]:

– ОС Microsoft (Windows 2007/XP/ME/) как базовый уровень;
– серверы MS.Net (.Net Enterprise Servers) уменьшают сложность разработки сложных ПС (например, Application Server, Exchange Server, SQL Server);
– сервисы .Net Building Block Services – это готовые «строительные блоки» для сложных ПС;
– интегрированная среда Visual Studio.NET (VS.NET) – верхний уровень MS.NET, которая обеспечивает создание сложных ПС.

Подсистема MS.NET Framework обеспечивает построение и выполнение любых приложений. В ее состав входят: общеязыковая среда CLR и библиотека классов FCL (Framework Class Library), состоящая из наборов классов для работы со строками, числовыми данными и параллельного вычисления. Эта подсистема включает средства управления памятью, ТД, межъязыковым взаимодействием, развертыванием (deploy-ment) приложений в ЯП.

Любой компонент на ЯП трансформируется к обобщенной спецификации типов CTS (Common Type System), которая содержит все ТД ЯП, определяет их взаимосвязи и сохраняет их отображение. Компилятор программ в ЯП создает файл на языке CIL, который ассемблируется (assembly) – собирается в (Portable Executable) код. CIL-код используется для перевода на промежуточный язык и машинный (native).

Система типов в .NET представлена на рис. 2.1.

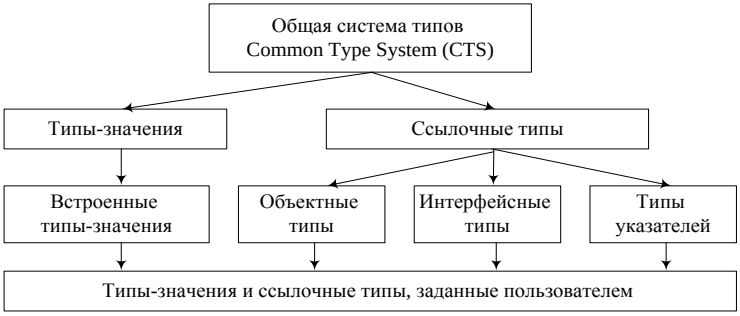

Рис. 2.1. Структура системы типов MS.NET

В этой системе выделены две группы типов: *типов-значений* (value type) *и типов-ссылок* (reference type), а также существует механизм отображения типов CTS в типы конкретных ЯП и наоборот.

Типы-значения – это статические типы в CTS, их значения могут занимать память от 8 до 128 байтов. Они не принимают участия в наследовании и копируются при присвоении им значений.

Типы-ссылки, или ссылочные типы, используют указатели на объекты, которые они типизируют, а также механизмы хранения и освобождения памяти. Ссылочные типы включают: объектные типы (object type), интерфейсные типы (interface type) и типы-указатели (pointer type).

CTS включает ТД, которые поддерживаются средой выполнения, определяет одни ТД, которые могут взаимодействовать с другими, и они представляются в форме метаданных .NET. Для того чтобы собрать программу в ЯП .NET, необходимо использовать принятые правила в CLS.

В CTS используются другие библиотеки: CLR (Common Language Runtime), CLS (Common Language Specification), CIIL. Роль CLR заключается в том, чтобы выявлять и загружать ТД в системе .NET и осуществлять управление ими.

Все ЯП оперируют с целочисленными данными или данными с плавающей точкой с форматом и единственной длиной, а представление строк тоже будет единственным для всех ЯП. За счет этой системы типов достигается более простая интеграция компонентов и кодов, написанных на разных ЯП. В отличие от COM-технологии [57], основанной на наборе стандартных типов, представленных в бинарном виде, CLR позволяет выполнять интеграцию любых кодов. Сервисы в CLR предоставлены библиотекой классов (более 1000) и моделью ASP.NET.

Двоичные, или бинарные, файлы представляются в платформонезависимом «промежуточном языке» Microsoft Intermediate Language (MIIL или IL), который задает промежуточный уровень для любого языка .NET. MIIL конвертирует данные в код CPU во время выполнения (just-in-time-JIT). Если программа в языке Effil, то она конвертируется в IL.

Программы, расположенные на разных типах компьютерах сети, передают друг другу данные через интерфейсные протоколы. Переданные данные преобразуются к формату данных новой машины и среды.

Компонентная модель реализует *компонентный подход* к проектированию приложений путем сборки объектов на основе интерфейсов (или фрагментов программ), представляющих собой независимые компоненты. Для инсталляции программ создаются инсталляционные комплекты в форме *сборок*. Каждый тип сборки имеет уникальный идентификатор – номер версии сборки, а каждый программный проект формируется в виде

сборки как самодостаточный компонент для развертывания, тиражирования и повторного использования.

Между сборками и пространством имен существует следующее соотношение. Сборка может включать несколько пространств имен, и в то же время пространство имен может занимать несколько сборок. Сборка может иметь в своем составе как один, так и несколько файлов, которые объединяются в манифест сборки, содержит метаданные о компонентах сборки, идентификатор автора и версии, сведения о типах и зависимости. При этом метаданные описывают все типы, представленные в сборке.

В результате компиляции кода в среде вычислений .NET создается сборка или так называемый *модуль*. При этом сборка существует в форме выполняемого файла (с расширением EXE) или файла динамической библиотеки (с расширением DLL). В состав сборки входит и манифест.

Между именами *простых* типов в C# и именами FCL-*типов* существует взаимно однозначное соответствие. В ней установлено соответствие типов данных языков C# и FCL. Запись в графе «Отвечает FCL-типу» указывает пространство имен, которое содержит объявление соответствующего типа. Данная таблица используется при компиляции и сборке компонентов, которые описаны в приведенных ЯП.

Взаимодействие объектов в ONC SUN и OSF DSE [56]**.** Системы обеспечения взаимодействия объектов основаны на операторе удаленного вызова RPC, задаваемого языками высокого и низкого уровня в виде описания интерфейса взаимодействующих объектов. Интерфейс – это посредник stub, операторы которого (тип протокола, размер буфера данных и др.) обеспечивают передачу данных по сети.

Формальные средства интеграции в этих системах такие:

- оператор передачи сообщений (через RPC-вызов);
- сетевые сообщения между компонентами для передачи данных;
- средства преобразования типов данных с ЯП высокого уровня к типам данных ЯП низкого уровня, а также кодирование и декодирование данных, подобно операциям put и get.

Преобразования данных в основном связаны с различиями в архитектуре машин и в транслированных кодах различных компиляторов, выполняются путем отображения релевантных типов данных к двоичному коду компонента и устранения неадекватного перевода программ в ЯП разными компиляторами в выходной или промежуточный код распределенной среды. В случае сложных структур данных (например, деревья, сеть) проводится их линеаризация.

Связь объектов в DCOM и CORBA [58]**.** Объектная модель DCOM устанавливает связь объектов и документов, а архитектура OMA системы CORBA – взаимосвязь объектов, выполняется брокером ORB через stub-

клиента и stub/skeleton сервера. Объекты описываются в современных ЯП, в том числе С, С++. Формализмы сборки в системе CORBA:

- механизмы передачи запросов через stub и skeleton;
- обмен данными через сеть и их преобразование в случае отличий в архитектуре и платформе компьютеров среды;
- системы преобразования типов данных для каждой пары ЯП (С ↔ Смолток, Смолток ↔ Ада, Ада ↔ Кобол, Кобол ↔ Java, Кобол ↔ Ада и др.), которые строятся аналогично.

Формализмы преобразования в системе DCOM такие:

- механизмы передачи данных (типа RPC-вызов);
- сетевой обмен данными;
- система передачи данных и преобразования нерелевантных типов данных (С ↔ ++), кодирование и декодирование этих данных.

Процедуры преобразования данных для сред ONC, DCE и CORBA реализованы на языке С++. Интерфейс описывается в языке IDL. Спецификации ТД в языке IDL системы CORBA такие: *in* – входные, *out* – выходные, *inout* – результат. К передаваемым ТД относятся – simple (short, long, unsigned short, unsigned long, float, double, boolean, char, octet, enum). При изменении типа поля у структуры также изменяется тип «fixed» или «variable». Это приводит к переписыванию этих полей во многих местах программы. Определение типа переменной длины – рекурсивное, оно влечет за собой изменение «fixed» или «variable» для составных типов. Параметры OUT и RESULT для любых типов переписывается в программы.

Для любого сложного типа данных *T* вводится специальный тип указателей на данные этого типа – *T_var*. Схема работы с параметрами на стороне клиента в интерфейсном посреднике одинакова для всех таких типов. Все параметры для объекта сервера передаются через *T_var*. Данные копируются перед заполнением их в *T_var*.

Например, заполнение типа String_var var: CORBA::String_var var = «some string», где копирование строк выполняется с помощью функции string_dup: CORBA::String_var var = CORBA::string_dup («some string»). Для всех других ТД функция string_dup отображения не предусмотрена. Заполненный по умолчанию *T_var* не может использоваться для доступа к данным типа *T*, поскольку в нем сохраняется нулевая ссылка.

Принципы взаимодействия объектов в среде CORBA. Основной принцип взаимодействия объектов в этой среде – это запрос от клиента для выполнения метода через интерфейс. При взаимодействии ЯП производится отображение ТД в типы данных клиентских и серверных стабов (stub) средствами брокера ORB.

Для всех ЯП системы CORBA (С++, JAVA, Smalltalk, Visual C++, COBOL, Ada-95) предусмотрен общий механизм связи (stub, sceleton) и

параметров для методов объектов через брокер ORB (рис. 2.2). Если в ОМ CORBA входит ОМ COM, то в ней ТД определяются статически, и конструирование сложных ТД осуществляется для массивов и записей.

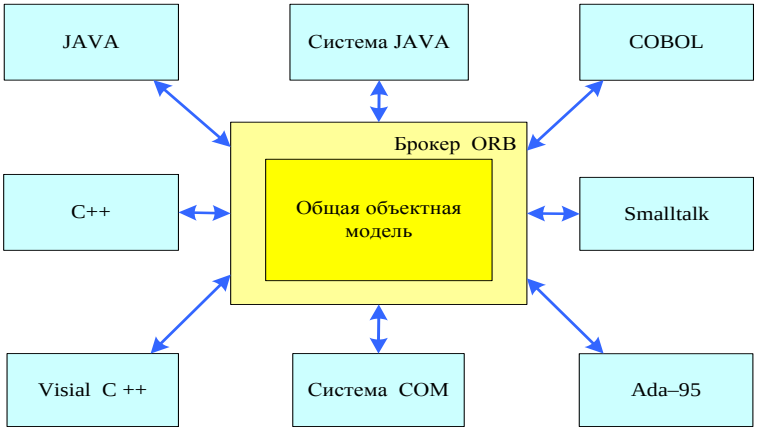

Рис. 2.2. Интегрированная среда системы CORBA

Методы объектов используются в двоичном виде и допускается совместимость машинного кода объекта из одной среды разработки к коду другой среды, а также совместимость разных ЯП за счет отделения интерфейсов объектов от реализаций через stub, sceleton в IDL.

В случае вхождения в состав модели CORBA объектной модели JAVA/RMI вызов удаленного метода объекта осуществляется ссылками на объекты, задаваемые указателями на адреса памяти. Интерфейс как объектный тип реализуется классами и предоставляет удаленный доступ к нему сервера. Компилятор JAVA создает байт-код, который переносит с одной платформы на другую CORBA.

Средства преобразования типов данных в среде JAVA. Язык JAVA и операторы вызова RMI позволяют проектировать распределенные приложения и обеспечивать их взаимодействие. Виртуальная машина работает с byte-кодами компонентов в других ЯП и, таким образом, обеспечивает взаимодействие компонентов в ЯП Java и C++ [59].

Формализмы сборки в системе Java:

– оператор вызова методов RMI;

– сетевой обмен данными между удаленными компонентами;

– интерпретация битовых кодов объектов C++ в среде Java.

Преобразование сложных данных объектов осуществляется с помощью функций отображения типов, описанных в IDL. Трансформация дан-

ных типа struct, например, включает преобразование всех ее полей в порядке, указанном в спецификации функции в языках IDL и C++. Эту функцию выполняет компилятор IDL, порождая файлы отображения в конструкции C++, набор вспомогательных процедур, необходимых для обращения к брокеру ORB.

Функции преобразования базовых типов учитывают информацию о границах выравнивания и размере данных в классе CDR. Данные типа array преобразуются функциями и процедурами для простых ТД.

2.2. Технология ООП

ООП – парадигма, основанная на представлении предметной области в виде системы абстрактных *объектов* и их реализаций [18].

В парадигме ООП различают – программирование, основанное на классах (class-based programming) и программирование, основанное на прототипах (prototype-based programming) [47].

Программирование «на классах». В этом прграммировании определена новая структура данных – *класс,* в котором кроме данных содержатся функции их обработки. Объект является ***экземпляром***. Класс определяет структуру и функциональность (*поведение*), одинаковую для всех экземпляров данного класса. Один класс отличается от других именем и набором интерфейсов, которые задаются набором сообщений. Экземпляр является носителем данных и обладает *состоянием.* Новый экземпляр создается конструктором класса и имеет структуру и поведение, заданные его классом.

Все объекты могут обмениваться между собой *сообщениями*. Объект имеет ассоциативный *контейнер*, позволяющий получить по сообщению нужный метод для обработки. Если метод не найден, сообщение перенаправляется *объекту-предку*. 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 класса);

абстракция (взаимодействие в терминах сообщений/событий);

полиморфизм (замена одного объекта другим).

Многие современные языки созданы на классах (Smalltalk, C++, Java, Python, PHP, Object Pascal (Delphi), VB.NET, Xbase++ и др.). Парадигма ООП на классах используется при проектировании сложных систем и моделировании в языке UML и средствах его поддержки.

На этих принципах построена система UML (Unified Modelling Language), 1994. Главное ее назначение – построение структуры предметной области с помощью 11 диаграмм и их преобразование к некоторому ЯП.

2.3. Технология программирования нового времени

Развитие возможностей задания сценариев обеспечило появление сценарной парадигмы [60]. Она предполагает разбиение задачи на отдельные части, каждая из которых решается специализированными программными средствами, сценарий выступает в роли «диспетчера», ответственного за организацию их взаимодействия. Эта концепция возникла как естественное развитие языка LISP. Первые сценарные (скриптовые) языки служили средством управления командной оболочкой ОС – командный файл на языке ОС содержал сценарий управления последовательностью действий по организации взаимодействия элементов системы.

Сценарные языки связаны также с развитием Internet-технологий и используются для создания динамических, интерактивных web-станиц, содержание которых модифицируется в зависимости от действий пользователя и состояния других страниц и данных. Сценарий задается теми возможностями, которыми обладает ОС, графическая среда и 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, Python, PHP, ASP.

2.3.1. *Сервисно-ориентированное программирование (СОП)*

СОП (service-oriented architecture) – развитие компонентного подхода к разработке ПО, 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 [49].

Программные системы, разработанные в соответствии с этой парадигмой, реализуются как набор web-сервисов, интегрированных с помощью стандартных языков Интернет SOAP, WSDL и др.

Web-сервисы – это новая перспективная архитектура, обеспечивающая распределенность на уровне Интернет [49]. 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 владель-

цами. Благодаря web-сервисам функции любой программы в сети могут стать доступными через Интернет, а результаты обращения к ним – с помощью PHP, ASP, JSP-скриптов, JavaBeans, COM-объектов или др. 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 действие, а затем отправляет ответ.

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

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 задаются в SOAP-конверты (SOAP envelopes). Сообщения содержат либо запрос на осуществление действия либо ответ. 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) для обеспечения общего каталога web-сервисов, предоставляя всем компаниям возможность «опубликовать» свой web-сервис. Этот каталог работает как телефонная книга всех web-сервисов.

При реальном использовании SOAP, языка XML и системы UDDI решаются вопросы надежности приложений из сервисов.

2.3.2. *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 [49]. Этот элемент локализован и может быть сопоставлен с другими. Однако *нефункциональные* свойства системы, например реакция на ошибки, обеспечение доступа к памяти (динамический заказ-освобождение), синхронизация параллельно действующих объектов и др.,

обычно «рассеиваются» по всем элементам системы, «пересекая» структуру системы. Для реализации свойств, отражающих тот или иной нефункциональный *аспект* системы, предложены *аспектные языки* для описания компонентов и аспектов. Цель АОП состоит в разделении компонентов и аспектов. Реализация приложений из них обеспечивается следующим:

ЯП для описания компонентов,

аспектным языком для описания аспектов,

компоновщиком (weaver) аспектов для интеграции приложения. Компоновка (вплетение аспектов) в систему может производиться либо во время выполнения (runtime weaving) или во время компиляции (compile time weaving). В АОП принято рассматривать следующие сущности:

точка присоединения, определяющая место в программе,

срез – набор точек присоединения по заданному правилу,

фрагмент вставки – набор инструкций ЯП для интеграции всех точек заданного среза.

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

Представление – правило изменения структуры класса.

Парадигма АОП призвана прийти на смену объектно-ориентированному и компонентному программированию и способствовать существенному повышению качества программ, особенно их сопровождения. Она поддерживает *мультипарадигмовую* концепцию программирования, сущность которой состоит в том, что разные аспекты проектируемой системы реализуются разными парадигмами программирования.

В настоящее время существует несколько реализаций АОП, наиболее известная из которых – разработка центра Xerox PARC – *AspectJ* (www.aspectj.org) в языке Java. Новый релиз AspectJ 1.1 встраивается в такие системы, 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. АОП продолжают развиваться в компании *Intentional Software* Грегором Кишалесом и Чарльзом Саймони.

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

2.3.3. *Агентное программирование* (*АП*)

Основу АП составляет интеллектуальный агент – сущность, способная формулировать цели, обучаться, планировать свои действия и принимать решения в динамически изменяемых случаях [49]. Появление концепции агентов обусловлено уровнем информационных технологий и назревшей объективной необходимостью обработки больших объемов информации, распределенной в информационных сетях. Простым примером задачи, которая может эффективно решаться с помощью агентов, является поиск нужных сведений в Интернет, анализ и отсеивание лишней информации.

Парадигма АП 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 искусственного интеллекта и компьютерных наук. Под агентом, как правило, понимают *компьютерную программу*, которой присущи следующие свойства:

- *автономность* – способность работать без вмешательства с контролем своих действий и состояний;
- *социальная активность* – способность сотрудничать с другими 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 полезную информацию из базы данных.

Интеллектуальность агента повышается путем:

- *убеждения*, основанного на знании текущего состоянии окружения, и изменений, 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 и неосуществимые желания;
- *целей*, рассматриваемых как непротиворечивые желания 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 специализированы и не обладают всеми вышеописанными свойствами.

Существует множество подходов к классификации агентов:

– 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 постусловия активации модулей и изменений во внешнем окружении;
– гибридная архитектура, в которой одна из подсистем – рассудительная, другая – реагирующая (действует по плану и на события);
– по *функциональному назначению*:
– интерфейсные агенты, взаимодействующие с пользователями;
– задачные агенты, привлекаемые к решению определенных задач;
– информационные агенты, непосредственно работающие с источниками данных;
– мобильные (распределенные) агенты, полностью автономные, способные перемещаться от сервера к серверу в поисках информации и нести в себе информацию о своем состоянии и др.

Наиболее известные агентные архитектуры – PRS, JAM, TOURINMACHINE, COSY, INTERRAP. Разработанные агенты могут образовывать *мультиагентную систему* 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, собственные и кооперативные, а также условия поведения в определенных ситуациях и локального планирования действий.

В основе АП лежат языки:
– описания моделей (ментальной, социальной);
– спецификации информационных, временных, мотивационных и функциональных действий агента;
– интерпретации спецификаций агента;
– конвертирования программ к агентам.

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 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.

К средствам *коммуникации* агентов относятся транспортный протокол TCP/IP, FIPA и стандарт языка передачи сообщений – ACL (Agent Communication Language).

Коммерческий язык программирования агентов – 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), предоставляющие средства для организации и спецификации мультиагентной системы.

2.3.4. *Технология программирования по прототипу*

В этом стиле программирования понятие класса отсутствует, а повторное использование (наследование) производится путем клонирования существующего экземпляра объекта – прототипа [49].

В отличие от стиля программирования, основанного на классах и на отношениях между ними, стиль прототипного программирования заостряет внимание на поведении небольшого количества «образцов», которые далее используются как «базовые» 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: 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 авто-

матическое изменение всех его потомков. Другой недостаток состоит в том, что простейшие реализации этой модели приводят к увеличенному (по сравнению с моделью делегирования) расходу памяти.

Примером языка прототипа является Self. Он положен в основу таких ЯП, как JavaScript, Squeak, Cecil, NewtonScript, Io, MOO, REBOL, Kevo и др.

2.3.5. *Agile-технологии*

В области программной инженерии специалисты Кент Бек (Kent Beck), Мартин Фаулер (Martin Folwer), Кен Шваубер (Ken Schwaber), Джефф Сазерленд (Jeff Sutherland) и другие образовали некоммерческую организацию **Agile Alliance**, пропагандирующую Agile Software Development, методологию eXtreme Programming, SCRUM, Dynamic Systems Development Method (DSDM), Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming и др.

Методология Agile ориентирована на тесное взаимодействие команды разработчиков с пользователями, итеративную модель ЖЦ с приращениями и быструю реакцию на изменяющиеся требования и «продуктоориентированный подход к разработке ПО. 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. Самой широко известной Agile-методологией является **eXtreme Programming** (**XP**) (экстремальное программирование).

В XP реализуется принцип «коллективного владения кодом». В нем любой член группы может изменить не только свой код, но и код другого программиста, и каждый модуль снабжается автономным тестом (unit test), обеспечивая возможность регрессионного тестирования модулей. Тесты пишут программисты, и они имеют право написания теста для любого модуля.

Таким образом, большинство ошибок исправляются на стадии кодирования либо при просмотре кода и при динамическом тестировании.

SCRUM – гибкая методика управления проектом фирмы Advanced Development Methods, Inc., широко используется в сотнях организаций (Fuji-Xerox, Canon, Honda, NEC, Epson, Brother, 3M, Xerox and Hewlett-Packard и др.). Эта методика задает итеративную модель ЖЦ с четко определенным процессом разработки, включающим *требования, проектирование, программирование, тестирование* (http://agile.csc.ncsu.edu).

2.3.6. *Другие Agile-технологии: DSDM, ASD, FDD*

DSDM (*Dynamic Systems Development Method*) – это метод разработки информационных систем консорциума DSDM. Он использует модель

быстрой разработки RAD (от Rapid Application Development) с прототипированием для достижения целей построения систем в условиях ограниченных ресурсов проекта.

ASD (*Adaptive Software Development*) – методология разработана Д. Хайсмитом (президентом Information Architects, Inc.) специально для экстремальных проектов и базируется на теории сложных адаптивных систем. В ней статический цикл разработки ПС включает *Планирование–Проектирование–Программирование*, а динамический цикл *Размышление–Взаимодействие–Обучение.*

FDD (*Feature Driven Development*) – методика разработки ПС, основанная на функциональных возможностях, предложена Д. де Луком для крупного банка (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 для *каждой* выделенной функции системы.

Таким образом, технология программирования прошла долгий путь развития и сыграла важную роль в истории программирования у нас и за рубежом. Появились новые зарубежные теории (UML, OWL, WSDL и др.), Model Variability на линиях производства программ, в Грид-системах и системных средах фабрики программ AppFab. Особого развития получил научный сервис в Интернете, выступающий в роли КПИ при композировании из них ПС. Самое главное, стали использоваться методы Data Minning для анализа данных в хранилищах данных Big Data и извлечения знаний из готовых систем.

2.4. Типы данных: фундаментальные (FDT) и общие (GDT)

Наиболее используемыми средствами представления данных при решении различных вычислительных задач являются FDT (Fundamental Data Type) и GDT (ISO/IEC 11404 – General Data Types) 1997, независимые от ЯП. При решении современных E-science задач сформировались новые неструктурированные данные, которые поступают с разных датчиков при исследовании земли и атмосферы (проект Grid). Они входят в класс больших данных BigData и требуют нестандартных приемов для их анализа и обработки.

Тип данных – это фундаментальное понятие в теории программирования, определяющее множество значений и операций, которые можно применять к этим значениям и способам их хранения.

Данные, с которыми оперируют программы в современных ЯП, относятся к простым, структурным и сложным типам данных FTD. Аксиоматика FTD разработана в 70-х годах прошлого столетия специалистами –

С. Дейкстра, Н. Вирт, В. Турский, П. Наур, Н. Агафонов, А. Замулин и др. [61–65]. В этой аксиоматике ТД (массив, файл, таблица и др.) рассматриваются как агрегатная структура, которая создается из простых типов данных. Операции над значениями типа задаются аксиомами и функциями, устанавливающими взаимно однозначное соответствие переданных данных. При их несоответствии производится преобразование к соответствующим ТД другого ЯП. Система аксиом определяет структуру множества значений типа, принадлежность отдельных элементов, их свойства и связь с другими ТД.

Математический тип – это множество всех значений, принадлежащих типу, и предикатная функция, определяющая принадлежность объекта к данному типу. Тип в математике «целое число» не имеет ограничений, а в программировании ограничения связаны с диапазоном значений и объемом занимаемой ими памяти. Выход за границу целого не приводит к исключительной ситуации.

По способу ТД разделяются на языки:

1) с полиморфным типом данных (например, Basic – тип данных вариант, Prolog, Lisp – списки). Переменные принимают значение любого типа и возвращают значения того типа. Выполнение $a + b$ трактуется как сложение чисел или как конкатенация строк, если они строкового значения, и недопустимые, если типы не совместимы. Это называется динамической типизацией, а в ООП и в теории чисел – полиморфным типом.
2) с неявным определением строковых типов [65];
3) с типом ЯП.

Все ТД практически совместимы. Их применяют в любых выражениях, а транслятор совершает необходимое преобразование. В языке Ада типы строго типизированы и каждая операция требует задания типов. В Приложении приведены термины ТП и SE.

2.4.1. *Фундаментальные ТД*

Для анализа основных ТД используется теория структурной организации данных. Она базируется на формализованном подходе к определению типов, основанном на аксиоматизации каждого типа и правилах выполнения операций над объектами. Система аксиом определяет структуру множества значений типа, принадлежность ему отдельных элементов, их основные свойства, связь с другими ТД.

Для каждой операции, выполняемой над переменными рассматриваемого типа, определяются типы операндов и результата. Теория структурной организации данных ориентирована на ее применение в ЯП. Во множестве ЯП основные положения данной теории воплощены в Паскале, Модула-2, Ада и др.

В FDT существуют четыре предопределенных ТД: целый (INTEGER); вещественный (REAL); булевой (BOOLEAN); символьный (CHARACTER). Они характеризуются тем, что на уровне архитектуры большинства ЭВМ имеются средства для их обработки и практически для всех ЯП. Остальные являются производными и образуются с помощью средств конструирования новых ТД.

Все ТД делятся на *простые* и *структурные*. К простым относятся перечислимые и числовые, к структурным – массивы, записи, множества, списки, последовательности и т.д. Аксиоматика фундаментальных ТД подробно рассмотрена в [52].

2.4.2. *Общие типы данных GDT*

Общие ТД GDT (General Data Types) представлены в стандарте ISO / IEC 11404–2007, которые являются:

- независимыми от языка (Independed Language) ТД и используются для формального описания концептуальных ТД;
- поддерживают полуструктурированные и неструктурированные совокупности данных, где типы данных являются неопределенными заранее ТД или перспективными;
- расширяемыми пользователем ТД.

Данный стандарт GDT устанавливает номенклатуру и распределенную семантику для набора ТД, которые чаще всего используются в ЯП и в интерфейсах программных систем.

Понятие «независимый от языков» ТД означает, что специфицированные ТД составляют классы типов данных, реальные представители которых используются в ЯП и других объектах.

Формальный синтаксис GDT

Стандарт GDT ISO/IEC 11404 определяет ТД, которые частично совпадают с ТД FDT, и включают [63–65]:

- примитивные ТД (real, integer, char, boolean) и другие ТД,
- сложные ТД (массив, запись, последовательность, портфель, ...),
- сгенерированные ТД – это данные, которые получаются после генерации, т.е. генератор ТД создает новый ТД.

Примитивные ТД GDT

Рациональный (rational) –математический ТД, который отвечает рациональным (действительным) числам.

Масштабированный (scaled) – семейство ТД, пространством значений которого является подмножество рациональных чисел, и каждый отдельный ТД имеет фиксированный знаменатель и предусматривает аппроксимацию его значений.

Комплексный (complex) – семейство ТД, каждый из которых задает числовой математический ТД в структуре комплексных чисел.

Пустой (void) – ТД, который задает объект с необходимыми синтаксическими и семантическими описаниями.

Основные понятия GDT

Пространство значений в GDT – это совокупность (коллекция) значений ТД, которая определяется одним из следующих способов:

1) перечислением;
2) аксиоматичным определением;
3) подмножеством уже определенного пространства значений, которое имеет тот же набор свойств;
4) комбинацией любых значений некоторого пространства значений с помощью специальной процедуры конструирования новых значений.

Каждое отдельное значение принадлежит только одному ТД, хотя оно может принадлежать и нескольким подтипам этого ТД.

Равенство. Для каждого пространства значений существует понятие равенства (equality), задаваемого следующими аксиомами.

Аксиома 2.1. *Для любых двух значений* (*a*, *b*) *из пространства значений выполняется условие равенства b, специфицированное как a = b, или не равняется b, специфицированное как a ≠ b.*

Аксиома 2.2. *Не существует пары таких значений* (*a*, *b*) *из пространства значений, для которых одновременно выполняются условия a = b и a ≠ b.*

Аксиома 2.3. *Для каждого значения a из пространства значений выполняется условие a = a.*

Аксиома 2.4. *Для любых двух элементов значений* (*a*, *b*) *из пространства значений a = b тогда и только тогда, когда b = a.*

Аксиома 2.5. *Если для произвольных трех элементов значений* (*a*, *b*, *c*) *из пространства значений выполняются условия a = b и b = c, то выполняется условие a = c.*

Для каждого ТД операция равенства *Equal* определяется как свойство равенства значений. Для любых значений *a* и *b* из пространства значений *Equal* (*a*, *b*) есть *true*, если *a = b*, и *false* – в противном случае.

Порядок. Пространство значений упорядочено, если для него установлено отношение порядка (order), которое задается знаком «меньше или равно» (≤) и удовлетворяет правилам:

1) для каждой пары значений (*a*, *b*) из пространства значений выполняется условие $a \le b$ или $b \le a$, или оба этих условий;
2) для любых двух значений (*a*, *b*), если $a \le b$ и $b \le a$, то $a = b$;
3) для любых трех значений (*a*, *b*, *c*), если $a \le b$ и $b \le c$, то $a \le c$.

Запись $a < b$ используется для нотации: $a \le b$.

ТД упорядочен, если отношение порядка определяется на его пространстве значений. Тогда операция InOrder определяется для произвольных двух значений a и b из пространства значений InOrder, (a, b) есть true, если $b \le a$, и false – в противном случае.

Ограниченность. ТД ограничен сверху, если он упорядочен и существует такое значение U из его пространства значений, при котором для всех значений s этого пространства выполняется условие $s \le U$. Значение U образует верхнюю границу пространства значений. Аналогично, ТД ограничен снизу, если он упорядоченный и существует такое значение L из его пространства значений, что для всех s этого пространства выполняется условие $L \le s$. Значение L образует нижнюю границу пространства значений. ТД называется ограничением, если его пространство значений имеет верхнюю и нижнюю границу.

Кардинальность. Пространство значений основывается на математической концепции кардинальности (cardinality): может быть конечным или бесконечным. ТД должен иметь кардинальность своего пространства значений. Моделью предусмотрены три категории ТД, пространство значений которых:

1) конечное;
2) очное (exact) и бесконечное;
3) приближенное и имеет конечную или бесконечную модель, пространство значений которой может быть бесконечным.

Точный и приближенный. Если каждое значение в пространстве значений ТД можно отличить от другого значения в пространстве этой модели, то ТД считается точным (exact).

Математические ТД, которые имеют значения и не имеют определенного представления, называются приближенными (approximate). Пусть M – математический ТД, а C – соответствующий вычисляемый ТД, P – преобразователь пространства значений M в пространство значений C. Тогда для каждого значения v' с C существует соответствующее значение ТД v с M и такое действительное значение h, что $P(x) = v'$ для всех x с M и $|v - x| < h$.

Таким образом, v' – это приближение C для всех значений M и находится в h-области значения v''. Кроме того, для одного значения v' в C существует более чем одно такое значение в M, что $P(y) = v'$. Таким образом, C не является точной моделью M. Делаем вывод, что C не является точной моделью M.

ТД, значение которого не имеет этого свойства, называется нечисловым (non numeric).

Пространство значений базируется на математической концепции или свойстве кардинальности (cardinality), то есть оно может быть конечным или бесконечным. ТД должен иметь кардинальность (мощность) своего

пространства значений за категориями ТД, пространство значений которых может быть: конечным, точным (exact), бесконечным и приближенным.

Сгенерированные типы данных

Сгенерированные ТД (generated datatypes) – это ТД, полученные в результате применения генератора типов данных.

ТД, с которыми работает генератор, называются параметрическими или компонентными. Сгенерированный ТД семантически зависит от параметрических ТД, но имеет собственные характеристические операции. Важной характеристикой всех генераторов ТД является то, что генератор может применяться до многих разных параметрических ТД. Генераторы указателя и процедуры генерируют ТД, значения которых атомарные, тогда как генератор Выбора и агрегатных типов данных генерирует ТД, значения которых позволяют производить их декомпозицию.

Генератор ТД (datatype generator) – это концептуальная операция над одним или несколькими ТД, которая создает новый ТД. Генератор ТД оперирует с ТД, а не с его значениями и представляет собой:

1) набор критериев для характеристик ТД, над которыми будут выполнены операции;
2) процедуры конструирования, которые допускают набор ТД с данным критерием для создания нового пространства значений этих ТД;
3) набор характеристических операций, которые применяются в конечном пространстве значений, определяет новый тип данных.

Агрегатный ТД (aggregate datatype) – это сгенерированный ТД, каждое значение которого получено из значений параметрических ТД. Параметрические ТД агрегатного ТД или его генератор включают в себя имена компонентов ТД. Генератор агрегатного ТД генерирует ТД с помощью алгоритмической процедуры агрегатного ТД.

В отличие от других сгенерированных ТД агрегатный ТД обеспечивает доступ к компонентам значений через характеристические операции. Агрегатные значения разных типов различаются между собой свойствами, которые характеризуют отношение между компонентами ТД и отношение между каждым компонентом и агрегатным значением.

2.4.3. *Генератор сложных типов данных GDT*

Тип данных **множество** (set) задает ТД, пространство значений которого составляет набор всех поднаборов пространства значений. Операции соответствуют математическому множеству *set.* Тип данных – портфель (bag) задает сам тип и значения, которые составляют коллекции образцов значений данного типа. Многочисленные образцы того же значения могут задаваться в этой коллекции, в котором порядок несуществен.

Стандарт включает генераторы ТД сложных типов данных: выбор (choice), указатель (pointer), процедура (procedure), запись (record), набор (set), портфель (bag), последовательность (sequence), массив (array), таблица (table) и т.п.

Выбор (choice) генерирует ТД. Каждое значение образуется из любого набора альтернативных типов данных с признаком (tag).

Указатель (pointer) генерирует ТД, каждое значение которого устанавливает средства ссылки на значение другого типа данных, специфицированного типом данных *element-type*.

Процедура (procedure) генерирует ТД, каждое значение которого является значением параметров других типов данных. Такой ТД включает в себя набор всех операций над значениями конкретной коллекции типов данных, концептуально атомарных.

Запись (record) генерирует ТД, значения которого составляют совокупность значений компонентов, и каждая совокупность имеет значение специфицированного идентификатором поля field-identifier.

Набор (set) генерирует ТД из пространства значений из поднаборов пространства значений типа данных как элемент с операциями, которые свойственны множеству *set*.

Портфель (bag) генерирует ТД, значения которого составляют коллекции образцов значений типа данных элемент. Многочисленные образцы того же значения могут подаваться в этой коллекции, порядок несущественный.

Последовательность (sequence) генерирует ТД, значениями которого являются упорядоченные последовательности значений типов данных из значений, несвойственных этому типу данных; одно и тоже значение может встречаться многократно в этой последовательности.

Массив (array) генерирует ТД, значения которого ассоциируются с произведением пространств одного или нескольких конечных типов данных, которые называются индексными ТД. Пространство значений этого типа данных такой, что каждому значению из пространства индексного типа данных соответствует только одно значение элемента.

Таблица (table) генерирует ТД, значения которого составляют коллекции значений из пространства одного или нескольких типов данных поле, такое что каждое значение из пространства задает ассоциации между значениями его полей.

Объявленный ТД (defined) – это ТД, определенный путем объявления типа type-declaration с идентификатором для объявленного типа и ссылкой на ТД или генератор типов данных. Последний определяется как Actual-type-parameters, если он соответствует номеру и типу объявления type-declaration. Если formal-parameter-type составляет type-specifier, то actual-type-parameters будет value-expression, определяя значение ТД. Если

formal-parameter-type есть «type», то actual-type-parameter этого type-specifier и имеет свойства этого параметрического типа данных в объявлении генератора.

Type-declaration идентифицирует type-identifier в type-reference с одним ТД, семейством типов данных или генератором типов данных. Если идентификатор типа type-identifier задает семью типов данных, то type-reference ссылается на тот член семьи. Пространство их значений определяется посредством type-definition после замены каждого значения actual-type-parameters для всех входов formal-parametric-value. Если type-identifier задает генератор типов данных, то type-reference означает ТД, который получается путем применения генератора типов данных к реальным параметрическим типам данных.

Характеристические операции. Эти операции создают значение любого типа с помощью генератора ТД, задавая пространство значений параметрических ТД. Такие операции необходимы для выделения ТД по их названиям и генерации агрегатных ТД как композиции следующих операций:

1) с нулевой арностью для генерации значений этого ТД;
2) с унарной операцией (арности 1), которая превращают значение этого ТД в новое значение этого же ТД или в значение boolean;
3) с арностью 2, которую преобразуют пары значений этого ТД в значение этого же ТД или в значение boolean;
4) с *n*-арностью, преобразующей упорядоченные *n*-элементные группы значений, каждая из которых относится к ТД и может быть параметрическим типом или агрегатным.

Практически не существует уникальной коллекции характеристических операций для заданного ТД. Одна коллекция операций для ТД (или генератора типов) достаточна для выделения этого ТД среди других из пространства значений той же мощности.

Таким образом, существует посимвольная замена, которая преобразует все пространство значений одного ТД (domain) в подмножество значений пространства другого ТД (диапазон, range**)** так, чтобы значение отношений и характеристических операций домена сохранялись бы в соответствующих значениях отношений и характеристических операциях диапазона ТД.

Вопросы и задания

1. Дайте определение технологии программирования.
2. В чем суть технологии программирования?
3. Определите основы модульного программирования.

4. Определите пути развития технологии.
5. Определите смысл систем программирования.
6. Что такое синтез, сборка, композиция?
7. Дайте определение парадигмы программирования.
8. Определите парадигму сборочного программирования.
9. Определите основу объектного программирования.
10. Какие новые парадигмы программирования Вы знаете?
11. Почему возникло сервисное программирование?
12. Как появилось аспектное программирование?
13. В связи с чем возникли агенты в программировании?
14. Что такое типы данных в технологии программирования.
15. Фундаментальные и общие ТД.

Литература к теме 2

1. Быстродействующая вычислительная машина М-20 / под ред. И.С. Брука. – М.: 1957. – 228 с.
2. Математическое обеспечение машины БЭСМ. – М.: ИТМиВТМ, ВЦ АН СССР, 1967.
3. *Глушков В.М., Ющенко Е.Л.* Вычислительная машина «Киев». – Киев: ГИТЛ, 1962. – 183 с.
4. Управляющая вычислительная система «Днепр-2» / А.Г. Кухарчук, В.М. Египко, Л.И. Струтинский и др. – К.: Наукова думка, 1972. – 240 с.
5. Автокод машины «Днепр». – Киев: ИК АН УССР, 1969. – 98 с.
6. *Ершов А.П.* Введение в теорию программирования. – М.: Наука, 1977. – 288 с.
7. *Ющенко Е.Л.* Адресное программирование. – Киев: Знание. – 1963. – 31 с.
8. *Ляпунов А.А.* О логических схемах программ // Проблемы кибернетики. – 1958. – Вып. 1.
9. *Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л.* Алгебра. Языки. Программирование. – Киев: Наукова думка. – 1974. – 287 с.
10. *Шура-Бура М.Р., Любимский Э.З.* Транслятор с языка Алгол-60 // ЖВМ и МФ. – 1964. – Т. 4. – № 1.
11. Сообщение об языке Алгол-60 / под ред. А.П. Ершова. – 1960.
12. *Попов В.Р., Степанов В.А., Стищева А.Г., Травникова Н.А.* Программирующая программа // ЖВМ и МФ. – 1965. – Т. 5. – № 2.
13. *Бабецкий Г.И. и др.* Система автоматизации программирования АЛЬФА // ЖВМ и МФ. – 1965. – Т. 5, № 2.
14. *Камынин С.С., Любимский Э.З.* Алгоритмический машино-ориентированный язык АЛМО // Алгоритмы и алгоритмические языки. – М.: ВЦ АН СССР, 1967. – Вып. 1.
15. *Жоголев Е.А., Росляков Г.С., Трифонов Н.П., Шура-Бура М.Р.* Система стандартных подпрограмм. – М.: ГИФМЛ, 1958. – 258 с.
16. *Иванников В.П., Гайсорян С.С., Томилин А.Н.* Ситемное программное обеспечение вычислительной системы «Электроника ССБИС //

SORUCOM.2014. Труды Международ. конференции «Развитие ВТ и ее ПО в России и в странах бывшего СССР. История и перспективы». 15–17 октября 2014. – Казань, Россия. – С. 117–125.
17. *Лавров С.С.* Основные понятия и конструкции ЯП. – М.: Финансы и статистика, 1982. – 80 с.
18. *Лаврищева Е.М., Борисенко Л.Г., Гришкевич К.И. и др.* Транслятор с языка Д-АЛГАМС для УВК «Днепр-2». – К.: ИК АН УССР, 1970. – 186 с.
19. *Лаврищева Е.М., Грищенко В.Н.* Связь разноязыковых модулей в ОС ЕС. – М.: Финансы и статистика, 1982. – 127 с.
20. *Терехов А.Н.* Проблемы идентификации и структура компилятора с языка АЛГОЛ 68. – http://www.math.cpbu.ru/usr/ant/all_articles/003
21. Терехов А.Н. АЛГОЛ 68 и его влияние на развитие программировния в СССР в России. – SORUCOM.2014. Труды Международ. конференции «Развитие ВТ и ее ПО в России и в странах бывшего СССР. История и перспективы». 15–17 октября 2014. – Казань, Россия. – С. 336–347.
22. Всесоюзная конференция по программированию «Теория и практика составления трансляторов с языков программирования», ВПК1, октябрь 1968. – Киев: Институт кибернетики АН УССР. – 231 с.
23. *Кахро М.И., Тыугу Э.Х.* Инструментальная система программирования на ЕС ЭВМ (ПРИЗ). – М.: Финансы и статистика, 1981.
24. *Тыугу Э.Х.* Концептуальное программирование. – М.: Наука, 1984. – 256 с.
25. *Редько В.Н.* Композиции программ и композиционное программирование // Программирование. – 1978. – № 5. – С. 17–26.
26. *Лаврищева Е.М., Грищенко В.Н.* Сборочное программирование. – Киев: Наукова думка, 1991. – 213 с.
27. *Жоголев Е.А.* Принципы построения многоязычной системы модульного программирования // Кибернетика. – 1974. – № 4. – С. 78–83.
28. *Лаврищева Е.М., Панчук А.Н., Семотюк В.П.* Технология создания ППП // Труды Всесоюз. конференции «Программная продукция как продукция производственно-технического назначения». – Тула, 1985.
29. *Орлов С.А.* Технология разработки ПО. – СПб.: Питер, 2002. – 483 с.
30. *Фуксман А.Л.* Технологические основы создания программных систем. – М.: Статистика, 1979. – 241 с.
31. *Глушков В.М.* Фундаментальные основы и технология программирования // Программирование. – 1980. – № 2. – С. 13–24.
32. *Глушков В.М., Капитонова Ю.В., Летичевский А.А.* О применении метода формализованных технических заданий к проектированию программ обработки структур данных // Программирование. – 1978. – № 6. – С. 5–12.
33. *Мищенко Н.М.* Расширение семантики входного языка расширяющейся системы программирования ТЕРЕМ // Труды конф. «Автоматизация производства пакетов прикладных программ». – Таллин, 1980. – С. 29–33.
34. *Глушков В.М., Стогний А.А., Лаврищева Е.М.* Система автоматизации производства программ (АПРОП). – Киев: ИК АН УССР, 1976. – 136 с.
35. *Лаврищева Е.М.* Вопросы объединения разноязыковых модулей в ОС ЕС. // Программирование. – 1978. – № 1. – С. 22–27.

36. *Грищенко В.Н., Лаврищева Е.М.* О создании межъязыкового интерфейса для ОС ЕС // УсиМ. – 1979. – № 1. – С. 46–54.
37. *Коваль Г.И., Коротун Т.М., Лаврищева Е.М.* Об одном подходе к решению проблемы межмодульного и технологических интерфейсов. – Межотрасл. сб. АН ССР и Минвуза СССР, 1987.
38. *Коваль Г.И., Коротун Т.М., Лаврищева Е.М.* Программирование в системе виртуальных машин ЕС ЭВМ. – М.: Финансы и статистика. – 1990. – 254 с.
39. *Вельбицкий И.В., Ходаковский В.Н., Шолмов Л.И.* Технологический комплекс автоматизации программ на машинах ЕС ЭВМ и БЭСМ-6. – М.: Статистика, 1980. – 263 с.
40. *Летичевский А.А., Капитонова Ю.В.* Математическая теория построения многопроцессорных ЭВМ и создания системы для исследования и моделирования. Проект «Маяк», 1990.
41. *Молчанов И.Н.* Компьютеры МИР и МВК – вклад Глушкова в развитие ВТ / М.В. Глушков. Прошлое устремленное в будущее». – Киев: Академпериодика, 2013. – С. 98–103.
42. *Вельбицкий И.В.* Технология программирования. – Киев: Техника, 1984. – 278 с.
43. *Редько В.Н., Стукало А.С., Сергиенко И.В.* Основы построения ППП. – Киев: Наукова думка, 1992. – 324 с.
44. *Коваль Г.И., Коротун Т.М., Лаврищева Е.М.* Технологический интерфейс для разработки систем // Труды Международ. конференции «Интерфейс-СЭВ», 1987. – С. 97–110.
45. *Волховер В.Г., Иванов Л.А.* Производственные методы разработки программ. – М.: Финансы и статистика, 1983. – 208 с.
46. *Липаев В.В.* Технология проектирования комплексов программ. – М.: Радио и связь, 1982. – 240 с.
47. *Андон Ф.И., Лаврищева Е.М.* Тенденции развития технологии программирования 90-х. – Киев: УСиМ. – 1993. – № 3. – С. 22–30.
48. *Лаврищева Е.М.* Software engineering компьютерных систем. Парадигмы, технологи, CASE-средства. – Киев: Наукова думка, 2013. – 264 с.
49. *Липаев В.В., Позин Б.А., Штрик А.А.* Технология сборочного программирования. – М.: Радио и связь, 1992. – 276 с.
50. Система автоматизации производства программ с режимом мультидоступа (АПРОП-2) / Вишня А.Т., Грищенко В.Н., Лаврищева Е.М., Моренцов Е.И., Коваль Г.И., Коротун Т.М., Зинькович В.М. – ГосФап. – № П005508, ЯЩ 15001-33-01, 09.12.81. – 587 с.
51. *Ершов А.П.* Два облика программирования // Кибернетика. – 1982. – № 6. – С. 122–125.
52. *Ершов А.П.* Опыт интегрального подхода к актуальной проблеме ПО // Кибернетика. – 1984. – № 4. – С. 11–21.
53. *Лаврищева Е.М.* Технологическая подготовка разработки систем: препринт / Ин-т кибернетики им. В.М. Глушкова; 87–5. – Киев, 1987. – 29 с.
54. *Лаврищева Е.М.* Технологическая подготовка и программная инженерия // УСиМ. – 1988. – № 1. – С. 48–52.

55. *Лаврищева Е.М., Панчук А.Н., Семотюк В.П.* Технология создания ППП // Труды Всесоюзн. конф. «Программная продукция как продукция производственно-технического назначения». – Тула, 1985.
56. *Эммерих В.* Конструирование распределенных объектов. Методы и средства программирования интероперабельных объектов в архитектурах OMG/CORBA, Microsoft COM и Java RMI. – М.: Мир, 2002. – 510 с.
57. *Роджерсон Д.* Основы COM. – MicrosoftPress / русск. перевод. – Microsoft Corporation, 1997. – 350 с.
58. *Siegel J.* CORBA Fundamentals and Programming. – Wiley Co. Publ. Group, John Wiley & Sons, Inc. – USA. – 1996. – 694 p.
59. *Островский А.В.* Подход к обеспечению взаимодействия сред Java MS.Net // Проблемы программирования. – 2011. – № 2. – С. 37–44.
60. *Андон Ф.И., Коваль Г.И., Коротун Т.М. и др.* Основы инженерии качества программных систем. – Наукова думка. – 2007. – 680 с.
61. *Лаврищева Е.М.* Методы программирования. Теория, инженерия, практика. – Киев: Наукова думка. – 2006. – 371 с.
62. *Лаврищева Е.М.* Парадигма интеграции в программной инженерии // Проблемы программирования. – 2000. – № 1–2. – С. 351–360.
63. *Лаврищева Е.М., Стеняшин А.Ю.* Подход к трансформации общих типов данных стандарта ISO/IEC 11404 для применения в гетерогенных средах // 2nd International Conference on High Performance Computing. – Харьков. – 2012. – С. 227–234.
64. *Лаврищева Е.М., Грищенко В.Н.* Сборочное программирование. Основы индустрии программных продуктов. – Киев: Наукова думка, 2009. – 371 с.
65. *Лаврищева Е.М., Рыжов А.Д.* Теория фундаментальных и общих типов данных для применения в Big Data // Международная конференция DAMDID/RCDL'2016. – М., 2016.

ПРИЛОЖЕНИЕ

Словарь терминов ТП и ПИ

Абстракция (Abstaction) – логическое представление существенных свойств программного объекта без деталей его реализации.

Аспект (Aspect) – точка зрения, с помощью которой можно рассмотреть какое-либо понятие, процесс, перспективу.

Аспектно-ориентированное программирование (АОП) (Aspect-oriented programming) – обеспечивает выделение *сквозной функциональности* и вывод ее за пределы бизнес-логики приложения. АОП сборочное основано на сборке полнофункциональных приложений из многоаспектных компонентов, инкапсулирующих различные варианты реализации.

Абстрактная архитектура (Abstract architecture) – декомпозиция структуры для выделенного спектра задач приложения на подсистемы или иерархию подсистем, на каждом уровне которой фиксируются возможные варианты

выделенных характеристик и ограничений, которые определяют соответствующие вариации выделенных компонентов.

Агрегация (Agregation) – объединение ряда понятий в новое понятие, существенные признаки которого могут быть или суммой признаков компонентов или существенно новых (типа отношение «часть–целое»).

Адаптивность (Adaptive) – атрибут, который характеризует усилия, необходимые для настройки программы к разным средам без выполнения специальных действий или средств.

Анализ требований (Analysis of requirements) – отображение функций системы и ее ограничений в модели требований приложения (домена).

Артефакт (Artifacts) – любой продукт деятельности специалистов при разработки программного обеспечения.

Архитектура системы (Systems architecture) – определение системы в терминах вычисляемых составляющих (подсистем) и интерфейсов между ними, которые отображает правила декомпозиции проблемы на составляющие.

Ассоциация (Association) – общее отношение, которое утверждает наличие связи между понятиями, без уточнения зависимости их от содержания и объемов.

Безотказность (Faultless) – атрибут надежности, который определяет отсутствие или частоту отказов через ошибки в программном обеспечении.

Валидация (Validation) – обеспечение соответствия разработки продукта требованиям ее заказчиков.

Вариабельность (Variability) – свойство продукта (системы) к расширению, изменению, приспособлению или конфигурированию с целью использования в определенном контексте и обеспечения его эволюции.

Верификация (Verification) – проверка правильности проекта в ходе его разработки.

Взаимодействие (Co-operation) – обмен сообщениями между объектами.

Восстанавливаемость (Restored) – атрибут надежности, который указывает на возможность программы к перезапуску для повторного выполнения и возобновления данных после отказов.

Водопадная или каскадная модель (Cascade model) – схема работ, по которой каждая работа выполняется один раз и в том порядке, в котором они перечислены в модели жизненного цикла.

Дефект (Fault) – результат ошибок разработчика во входных или проектных спецификациях, а также в кодах программ, эксплуатационной документации и т.п.

Динамический метод тестирования (Dynamic method of testing) – выполнение программ с целью установления причин ошибок с ожидаемыми реакциями на эти ошибки.

Диаграмма UML (Diagram) – графическое представление элементов моделируемой системы (класса, сценария, последовательности, сотрудничества, активности, состояний, реализации и т.п.).

Домен (Domain) – спектр задач предметной области, которые допускают общие способы их решения.

Защищенность (Protected) – атрибут функциональности, который указывает на возможность предотвращения несанкционированного доступа (случайный или преднамеренный) к программам и данным.

Изменяемость (Changeable) – показатель сопровождения, определяющий усилия, которые расходуются на модификацию, удаление ошибок или внесение изменений в связи с ошибками и новыми условиями среды функционирования.

Инженерия (Engineering) – применение научных результатов, которые позволяют извлекать пользу от свойств материалов и источников энергии.

Инженерия требований (Engineering requirements) – сбор, анализ требований заказчика и представления их в нотации, которая является понятной и согласованной как с заказчиком, так и с исполнителем.

Инженерия качества (Engineering quality) – процесс предоставления программным продуктам заданных в требованиях характеристик качества и их оценивание.

Интероперабельность (Variabilities) – атрибут функциональности, который обеспечивает взаимозамену совместно действующих компонентов в программной системе с необходимой инсталляцией или адаптацией.

Информационная система (Infomation systems) – система, которая проводит сбор, обработку, сохранение и выработку информации людьми с использованием автоматических процессоров.

Индустрия (Industry) – производство разных видов продуктов массового применения и средств их изготовления для получения прибыли.

Инструмент (Instrument) – это программное, языковое или методическое средство, применяемое для получения состояния объекта в некотором законченном виде. В зависимости от этапа жизненного цикла и состояния объекта для работы используются языки, трансляторы, генераторы и т.п.

Императивное программирование (Imperative programming) основано на идеи абстрактного вычислителя, выполняющего последовательность команд программы, образуя некоторое состояние, которое изменяется при переходе к другой последовательности команд.

Качество программного обеспечения (Software quality) – совокупность свойств, определяющих возможность программного обеспечения удовлетворить заказчика, который сформулировал эти свойства и требования к разработке.

Компонент (Component) – тип, класс, проектное решение, документация или другой продукт программной инженерии, специально приспособленный для повторного использования.

Компонентная разработка (Component Development) – конструирование программного обеспечения из готовых конструкций путем композиции готовых компонентов повторного использования КПИ из репозиториев.

Конкретизация (Concretization) – добавление существенных признаков, благодаря которых содержание понятия расширяется, а объем понятия сужается.

Конфигурация (Configuration) – структура программной системы из компонентов и набора вариантов модулей.

Концептуальное моделирование (Conceptual design) – процесс построения модели проблемы, ориентированной на понимание ее человеком.

Компьютерная инженерия (Computer Engineering) – это теория и принципы построения компьютеров, фреймворков, суперкомпьютеров, вычислительных кластеров и их системного программного обеспечения.

Компонентное программирование (Component Programming) – это подход к созданию компонентных программ из готовых КПИ путем их сборки на всех этапах жизненного цикла (ЖЦ).

Компонентное сборочное программирование (Component assembling programming) – компонентное программирование основано на распространении классов в бинарном виде и предоставление доступа к методам класса через строго определенные интерфейсы. Это программирование поддерживают технологические подходы COM, CORBA, .Net.

Контейнер (Container) – оболочка, внутри которой реализованы функции в виде экземпляров компонентов, обеспечивает взаимодействие с сервером через стандартные интерфейсы (функция, Home интерфейс). Экземпляры обращаются друг к другу через системные сервисы данного контейнера или другого.

Логическое программирование (Logical Programming) – программирование в терминах фактов и правил вывода с использованием языков формальных исчислений.

Метод программный (Programs method) – способ или планомерный подход к достижению той цели, которая ставится перед объектом разработки (метод – нисходящий, восходящий, модульный, сборочный и др.).

Метод сборки (Method of assembling) – способ соединения разноязычных объектов в языках программирования (ЯП), основанный на теории спецификации и отображении типов и структур данных ЯП, представленных алгебраическими системами.

Модель жизненного цикла (Life Cycle model) – типичная схема последовательности работ на этапах разработки программного продукта.

Модель процессов (Process Model) – действия, которые сопровождают изменения состояний объектов.

Модель качества (Quality model) – четырехуровневая модель, которая отображает характеристики, атрибуты (показатели), метрики и оценочные элементы программного обеспечения.

Модульное программирование (Modular programming) – метод разработки программ, основанный на разбиении проблемы на независимые части – модули, которые реализуют отдельные функции и связаны между собой через интерфейсы (входные и выходные данные) для взаимодействия с другими модулями, собранными в большую программу.

Модуль (Module) – независимая функциональная часть программы, к которой можно обращаться как к самостоятельной единице через внешний интерфейс.

Надежность (Reliability) – набор атрибутов, которые указывают на возможность программного обеспечения превращать входные данные в результаты при условиях, которые зависят от периода времени (износ и старение не учитываются) и наличия отказов в результате внутренних дефектов или нарушения условий его применения.

Нефункциональные требования (Unfunctional requirements) – требования, которые характеризуют организационные, исполнительные, операционные и безопасные аспекты работы программной системы.

Нисходящее программирование (программирование «сверху вниз») – методика разработки программ, при которой разработка начинается с определения

целей проблемы, после чего идет их детализация и описание в языках программирования.

Объект (Object) – базовое понятие в объектно-ориентированном программировании, которое обладает свойствами наследования, инкапсуляции и полиморфизма. Объекты с общими свойствами и методами образуют класс, являясь в нем экземплярами класса. Они взаимодействуют между собой через сообщения и используют библиотеки классов общего применения.

Объектно-ориентированная модель (Object-oriented model) – представление программной системы в виде совокупности объектов, которые взаимодействуют между собой и имеют определенные свойства и поведение.

Объектно-ориентированное программирование (Object-oriented programming) – это подход, при котором предметная область рассматривается как набор самостоятельных объектов, заданных в виде ориентированного графа, в вершинах которого находятся объекты, а дуги задают связи между ними. Является разновидностью сборочного программирования.

Онтология (Ontology) – совокупность примитивных понятий, терминологии и парадигмы интерпретации в сфере проблем, которые должны быть решены в новой разработке.

Оценка качества (Quality estimation) – действия, направленные на определение степени удовлетворения программного обеспечения потребностям соответственно их назначения.

Ошибка (Mistake) – недостатки в операторах программы или в технологическом процессе разработки, которые приводят к неправильной интерпретации входной информации, а следовательно, и к неправильному решению.

Парадигма АОП (Paradigm AOP) фиксирует *модель* встраивания аспекта в систему (JPM, от Join Point Model), которая включает элементы парадигмы: срезы, фрагменты, точки присоединения, аспекты и др.

Переносимость (Bearable) – группа свойств, которая обеспечивает приспособляемость к переносу из одной среды функционирования в другие, а также усилие для перенесения программного обеспечения к новой среде или сети.

План тестирования (Plan of testing) – описание стратегии, ресурсов и графика тестирования отдельных компонентов и системы в целом.

Повторно используемый компонент (Reuses) – КПИ – фрагмент знаний о прошедшем опыте разработки системы программирования, которую можно адаптировать при создании новых систем.

Программа (program) – это объект разработки, который не является осязаемым (нельзя пощупать, взвесить и т.п.) человеком, а доступен пониманию ЭВМ. Готовая программа – это продукт, реализующий определённые функции ПрО, процесс проектирования и разработки которого осуществлялся соответствующими методами, средствами и инструментами.

Программная инженерия (Software Engineering) – это система методов, средств и дисциплин планирования, разработки, эксплуатации и сопровождения программного обеспечения, готового к внедрению. Основные принципы ПИ – продуктивность, индустрия и качество (см. ядро знаний SWEBOK (Software Engineering body of Knowledge, www.swebok.com, 2001 г.).

Продуктовая линия (Product Line/Product family) – группа продуктов или услуг, которые имеют общее управляемое множество свойств, удовлетворяющие потребностям определенного сегмента рынка или вида деятельности (group of products or services sharing a common, managed set of features that satisfy specific needs of a selected market or mission).

Правильность (Rightness), точность (exactness) – атрибут функциональности, который показывает, как обеспечивается достижение теоретически правильных и согласованных результатов.

Прикладная система (Application System) – конкретный продукт программной инженерии, предназначенный для выполнения конкретных сценариев конечного пользователя.

Программирование (Programming) – см. Реализация программной системы средствами языков программирования.

Процесс приобретения (Acquisition process) – действия, которые инициируют жизненный цикл системы и определяют организацию-покупателя автоматизированной системы, программной системы или сервиса.

Процесс разработки (Development Process) – действия организации-разработчика программного продукта, которые включают инженерию требований к системе, проектированию, кодировке и тестированию.

Процесс поставки (Delivery process) – действия по передачи разработанного продукта покупателю.

Процесс эксплуатации (Exploitation process) – действия по обслуживанию системы во время ее использования – консультации пользователей, изучение их пожеланий и др.

Пространство решений (Space of Task) – это знание о действующих компонентах и их конфигурации, средствах их конструирования, соединения или встраивания в соответствующий член семейства с оценкой их избыточности. Элементы фактически решают функциональные задачи данной ПрО.

Процесс сопровождения (Accompaniment process) – действия по управлению модификациями, поддержкой актуального состояния и функциональной пригодности, инсталляции или изъятия версии системы у пользователя.

Проектирование (Planning) – превращение требований в последовательность проектных решений системы.

Проектирование концептуальное (Planning conceptual) – уточнение понимания функций системы и согласование требований.

Проектирование архитектуры (Planning architecture) – определение главных структурных элементов системы, которую строят.

Производство продуктов (Production of products) – это процессы ТЛ, посредством которых реализуется программный продукт определенного назначения для массового применения.

Пространство проблемы (Space of Problem) – это понятия ПрО ПС, характеристики объектов, новые компоненты функции или готовых КПИ. Они связаны моделью характеристик, содержащей изменяемые параметры разных членов семейства и операции взаимодействия между собой элементов члена семейства.

Реализация программной системы (System Realization) – превращение проектных решений в работоспособную программную систему, которая реализует заданные функции (кодировка, конструирование, программирование).

Риск (risk) – совокупность угроз, которые могут привести к неблагоприятным последствиям и убыткам, а также к изменениям состояния системы.

Сервис (Service) – системный ресурс, который реализует некоторую функцию, в том числе и бизнес-функцию. Содержит независимый интерфейс с другими сервисами и ресурсами. Обеспечивает реализацию задачи интеграции программ и сервисов разной природы и используется как провайдер.

Сервисно-ориентированное программирование (Service Oriented programming) – это создание сервисно-ориентированной архитектуры (Service Oriented Architecture), доступ к сервисам которой происходит через web-языки, интерфейсы и протоколы.

Системная инженерия (System Engineering) – это теория, методы и принципы построения информационных, компьютерных АС и систем управления ими.

Синтезирующее программирование (Synthesizing programming) – постановка задачи в виде спецификации модели вычислений, по которой производится синтез программы.

Структурное программирование (Structured programming) – методология и технология разработки по методу «сверху вниз» отдельных блоков системы, структура которых задается элементами: последовательность, ветвление и цикл.

Событие (Event) – явление, которое провоцирует изменение определенного состояния и переход к другому состоянию в домене или системе.

Сборочное программирование (Assmbling Programming) – это метод сборки разнородных готовых программных ресурсов (модулей, объектов, компонентов, reuses, сервисов и др.) многократного применения через их интерфейсные данные, которыми они обмениваются между собой.

Сертификация продукта (Certification of product) – процесс, который подтверждает соответствие идентифицированной программной продукции (процесса или услуг) конкретному стандарту или техническим условиям с оценкой специальным знаком или свидетельством.

Семейства систем (Families Systems) – множество прикладных систем с общими функциональными свойствами.

Спецификация (Specification) – представление объекта разработки с помощью формальных правил ЯП, критериев качества и ограничений на использование объектов.

Совместимость (Compatibility) – показатель переносимости, которая определяет возможность использования новых программ в среде действующего программного обеспечения.

Сопровождение (Accompaniment) – любые работы по внесению изменений в программную систему после ее передачи пользователю для эксплуатации.

Схема сборки (Chart of assembling) – это граф, в вершинах которого находятся объекты сборки, а на дугах – их интерфейсы для взаимодействия друг с другом и обмена между ними данными.

Сценарий (Scenario) – один из возможных путей использования системы.

Тест (Test) – некоторая программа, предназначенная для проверки работоспособности другой программы и выявления в ней ошибочных ситуаций.

Тестовые данные (Data Of Testing) – данные, готовящиеся определенными генераторами на основе документов программ или по спецификациям требований, которые используются для проверки работы программной системы.

Тестирование (Testing) – средство семантической отладки (проверки) программы, которая заключается в проработке последовательности разных контрольных наборов тестов, для получения известного результата.

Технология (Technology) – это совокупность методов, способов, приемов, средств автоматизации и порядка их использования при производстве некоторого продукта.

Технологическая линия (Technological line, ТЛ) – набор процессов разработки функций объекта, представленных совокупностью автоматизированных операций, которые последовательно и систематически преобразуют состояния объектов для получения заключительного состояния – готового программного продукта с заданными показателями качества.

Техника функционального моделирования SADT (Structured analysis and design technique) основана на спецификации функций системы в виде диаграмм, задающих фрагменты описания программ и их интерфейсов (входные и выходные данные) с другими функциональными программами.

Функциональные требования (Functional requirements) – требования, которые определяют цели и функции реализуемой системы.

Функциональность (Functionality) – это совокупность свойств, которые определяют возможность программного обеспечения выполнять перечень функций в заданной среде, которые отвечают требованиям.

Целостность (Integrity) – возможность системы сохранять стабильность в работе и не иметь риска.

Черный ящик – метод тестирования реализованных функций путем проверки несоответствия между реальным поведением функций и ожидаемым поведением согласно с входными спецификациями требований.

Характеристики качества (Quality descriptions) – функциональность (functionality), надежность (reliability), удобство (usability), эффективность (efficiency), сопроводженность (maintainability), переносимость (portability) и т.п.

UML (Unified Modeling Language) – язык для спецификации, визуализации, конструирования и документирования артефактов программных систем, а также для моделирования бизнеса.

Экстремальное программирование (Extreme Programming, XP) – многократно проходит цикл создания системы от анализа потребностей заказчика до комплексного тестирования и изготовления готовой системы. В систему последовательно добавляются различные протестированные «возможности» (facilities), согласованные с заказчиком, и цикл повторяется.

Экземпляризация – зависимость между параметризуемым абстрактным классом – шаблоном (template) и реальным классом, который инициирован при определении параметров шаблона.