СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
SIBERIAN FEDERAL UNIVERSITY

О. А. Антамошкин

ПРОГРАММНАЯ ИНЖЕНЕРИЯ

ТЕОРИЯ И ПРАКТИКА

Учебник

УМО

ГУМАНИТАРНЫЙ ИНСТИТУТ

ПРИКЛАДНАЯ ИНФОРМАТИКА

Олеслав Антамошкин

Программная инженерия. Теория и практика

«Сибирский федеральный университет»

2012

УДК 004.42(07)
ББК 32.973.2-018я73

Антамошкин О. А.

Программная инженерия. Теория и практика /
О. А. Антамошкин — «Сибирский федеральный университет»,
2012

ISBN 978-5-7638-2511-4

В учебнике освещены современные методы и средства программной инженерии, детально рассмотрен процесс разработки программного обеспечения (ПО), приведена теория управления разработкой ПО. В качестве средства разработки ПО представлен продукт Visual Studio Team System. Для закрепления студентами полученных теоретических знаний во второй половине учебника дан практикум. Предназначен для студентов, обучающихся по направлению подготовки 080801 «Прикладная информатика», а также может быть рекомендован студентам других специальностей, интересующимся как вопросами управления разработкой программного обеспечения, так и тематикой программной инженерии в целом.

УДК 004.42(07)
ББК 32.973.2-018я73

ISBN 978-5-7638-2511-4

Содержание

Введение 5
Теоретический курс 7
1. Методы и средства программной инженерии 7
1.1. Введение в программную инженерию 7
1.2. Технология программирования и ее основные этапы 10
Конец ознакомительного фрагмента. 16

О. А. Антамошкин Программная инженерия. Теория и практика

Введение

Иан Соммервилл в своей книге «Инженерия программного обеспечения» определяет программную инженерию как интегрирование принципов математики, информатики и компьютерных наук с инженерными подходами, разработанными для производства осязаемых материальных артефактов. Дисциплина программной инженерии включается в круг вопросов компьютинга (англ. computing) и может рассматриваться как инженерная область, имеющая более тесные связи со своей базовой дисциплиной – компьютерными науками, чем другие инженерные области [1]. Среди других инженерных дисциплин она качественно выделяется нематериальностью программного обеспечения и дискретной природой его функционирования. Основываясь на математике и компьютинге, программная инженерия занимается разработкой систематических моделей и надежных методов производства высококачественного программного обеспечения, и данный подход распространяется на все уровни – от теории и принципов до реальной практики создания программного обеспечения, которая лучше всего заметна сторонним наблюдателям.

Термин «инженерия программного обеспечения» появился впервые в 1968 г. на конференции НАТО «Инженерия программного обеспечения» и должен был спровоцировать размышления относительно текущего в то время «кризиса программного обеспечения». С тех пор он стал использоваться для определения как профессии, так и области исследований, посвященных созданию программного обеспечения, которое имеет более высокое качество, более доступно, поддерживаемо и быстрее строится.

Так как эта область относительно молода по сравнению со своими сестринскими областями инженерии, все еще продолжаются дебаты вокруг того, что представляет собой «инженерия программного обеспечения» и удовлетворяет ли она понятию инженерии. Этот спор развивается естественным образом, начавшись с попыток рассматривать создание программного обеспечения только как программирование. Разработка программного обеспечения – термин, иногда предпочитаемый практиками в промышленности, которые рассматривают разработку программного обеспечения как несравнимо более мощную и конструкционно-емкую методологию в сравнении с процессом написания кода программистом.

Все же несмотря на юность профессии, будущее области радужно. Так, Money Magazine и Salary.com оценили профессию разработчика программного обеспечения как лучшую работу в Америке в 2006 г.

В настоящее время существует как минимум три основополагающих документа в области программной инженерии. Это SWEBOK (Software Engineering Body of Knowledge) – документ, подготавливаемый комитетом Software Engineering Coordinating Committee, в который вовлечено сообщество IEEE Computer Society (IEEE-CS). Назначение SWEBOK – в объединении знаний по инженерии программного обеспечения (разработке программного обеспечения), Второй документ – Software Engineering Code of Ethics and Professional Practice, посвящен этическим и профессиональным стандартам для инженерии ПО, выпущен в 1998 г. Третий документ (SE2004), выпущенный в 2004 г., посвящен составлению учебного плана по программной инженерии.

Эти документы созданы совместными усилиями IEEE-CS и Association for Computing Machinery (ACM) и призваны определить следующее:

- необходимый набор знаний и рекомендуемые практики;
- этические и профессиональные стандарты;
- учебную программу для студентов, аспирантов и продолжающих обучение.

В рамках изучения дисциплины «Программная инженерия» сложно охватить весь объем накопленных в этой области знаний. Этот учебник представляет собой попытку собрать основные компоненты необходимых наборов знаний, умений и рекомендуемых практик в рассматриваемой дисциплине. В первую очередь он предназначен для студентов, обучающихся по направлению подготовки «Прикладная информатика». Также он будет небезынтересен людям, непосредственно занимающимся разработкой и проектированием различных информационных систем, в аспекте систематизации и актуализации знаний по программной инженерии.

Учебник состоит из теоретического курса и практикума. Для успешного освоения теории необходимо использовать так называемые карты памяти. Обратите внимание, что первые карты памяти нужно рисовать «вручную». Впоследствии, особенно при «массовом» процессе использования карт памяти, целесообразно пользоваться инструментами. Мы рекомендуем продукт Comapping (www.comapping.com).

Часть заданий и вопросов на отдельную тему часто необходимо выполнять после освоения всего курса вообще, так как они содержат ссылки на другие темы – как назад, так и вперед.

Относительно всего теоретического материала правомочно одно стандартное задание – нарисовать все содержание в виде одной карты памяти на листе формата А4 (можно А3, но не больше). Такие задания целесообразно выполнять сразу после изучения. Полезность такого подхода связана с тем, что такие карты памяти легко рисуются (при наличии навыка) и очень легко проверяются.

Часто задания с помощью карт памяти оказываются не совсем картами памяти. Исчезает центральный объект, вместо него появляется произвольный «плоский» граф. Также полезно оказывается рисовать имена связям. Мы все равно называем такие графы картами памяти, чтобы терминологически не усложнять ситуацию.

Теоретический курс

1. Методы и средства программной инженерии

Понятие программной инженерии. Основные определения: информатика, системотехника, бизнес-реинжиниринг. Программное обеспечение: определение, свойства.

1.1. Введение в программную инженерию

Чем программирование отличается от программной инженерии [2]? Тем, что первое является некоторой абстрактной деятельностью и может происходить во многих контекстах. Можно программировать для удовольствия, для того, чтобы научиться (например, на уроках, на семинарах в университете), можно программировать в рамках научных разработок, а можно заниматься промышленным программированием. Как правило, это происходит в команде. При этом необходимо точно понимать, что нужно заказчику, выполнить работу в определенные сроки и результат должен быть нужного качества – того, которое удовлетворит заказчика и за которое он заплатит. Чтобы удовлетворить этим требованиям, программирование «обрастает» различными дополнительными видами деятельности: разработкой требований, планированием, тестированием, конфигурационным управлением, проектным менеджментом, созданием различной документации (проектной, пользовательской и пр.).

Разработка программного кода предваряется анализом и проектированием (первое означает создание функциональной модели будущей системы без учета реализации для осознания программистами требований и ожиданий заказчика; второе – предварительный макет, эскиз, план системы на бумаге). Трудозатраты на анализ и проектирование, а также форма представления их результатов сильно варьируются в зависимости от видов проектов и предпочтений разработчиков и заказчиков.

Требуются также специальные усилия по организации процесса разработки. В общем виде это итеративно-инкрементальная модель, когда требуемая функциональность создается порциями, которые менеджеры и заказчик могут оценить, и тем самым есть возможность управления ходом разработки. Однако эта общая модель имеет множество модификаций и вариантов.

Разработку системы также необходимо выполнять с учетом удобств ее дальнейшего сопровождения, повторного использования и интеграции с другими системами. Это значит, что система разбивается на компоненты, удобные в разработке, годные для повторного использования и интеграции, а также имеющие необходимые характеристики по быстродействию. Для этих компонент тщательно прорабатываются интерфейсы. Сама же система документируется на многих уровнях, создаются правила оформления программного кода, оставляются многочисленные семантические следы, помогающие создать и сохранить, поддерживать единую, стройную архитектуру, единообразный стиль, порядок.

Все эти и другие дополнительные виды деятельности, выполняемые в процессе промышленного программирования и необходимые для успешного выполнения заказов, и будем называть программной инженерией (software engineering). Хотя еще в 70-х гг. прошлого столетия академиком А.П. Ершовым термин software engineering переводился на русский язык как «технология программирования», программная инженерия – более современный, но менее традиционный перевод этого же термина, предложенный в конце 90-х И.В. Поттосиным. Получа-

ется, что так мы обозначаем, во-первых, некоторую практическую деятельность, а во-вторых, специальную область знания, или, другими словами, научную дисциплину. Ведь для облегчения выполнения каждого отдельного проекта, для возможности использовать разнообразный положительный опыт, достигнутый другими командами и разработчиками, этот самый опыт подвергается осмыслению, обобщению и надлежащему оформлению. Так появляются различные методы и практики (best practices) – тестирования, проектирования, работы над требованиями, архитектурных шаблонов, а также стандарты и методологии, касающиеся всего процесса в целом (например, MSF, RUP, CMMI, Scrum). Вот эти-то обобщения и входят в программную инженерию как в область знания.

Необходимость в программной инженерии как в специальной области знаний была осознана мировым сообществом в конце 60-х гг. прошлого века, более чем на 20 лет позже рождения самого программирования, если считать таковым знаменитый отчет фон Неймана «First Draft of a Report on the EDVAC», обнародованный им в 1945 г. Рождением программной инженерии является 1968 г. – конференция НАТО «Software Engineering», г. Гармиш (ФРГ), которая целиком была посвящена рассмотрению этих вопросов. В сферу программной инженерии попадают все вопросы и темы, связанные с организацией и улучшением процесса разработки ПО, управлением коллективом разработчиков, созданием и внедрением программных средств поддержки жизненного цикла ПО. Программная инженерия использует достижения информатики, тесно связана с системотехникой, часто предваряется бизнес-реинжинирингом (рис. 1). Немного подробнее об этом контексте программной инженерии.

Информатика (computer science) – это свод теоретических наук, основанных на математике и посвященных формальным основам вычислимости. Сюда относят математическую логику, теорию грамматик, методы построения компиляторов, математические формальные методы, используемые в верификации и модельном тестировании и т.д. Трудно строго отделить программную инженерию от информатики, но в целом направленность этих дисциплин различна. Программная инженерия нацелена на решение проблем производства, информатика – на разработку формальных, математизированных подходов к программированию.

Системотехника (system engineering) объединяет различные инженерные дисциплины по разработке всевозможных искусственных систем – энергоустановок, телекоммуникационных систем, встроенных систем реального времени и т.д. Очень часто ПО оказывается частью таких систем, выполняя задачу управления соответствующим оборудованием. Такие системы называются программно-аппаратными, и участвуя в их создании, программисты вынуждены глубоко разбираться в особенностях соответствующей аппаратуры.

Бизнес-реинжиниринг (business reengineering) в широком смысле обозначает модернизацию бизнеса в определенной компании, внедрение новых практик, поддерживаемых соответствующими новыми информационными системами. При этом акцент может быть как на внутреннем переустройстве компании, так и на разработке нового клиентского сервиса (как правило, эти вопросы взаимосвязаны). Бизнес-реинжиниринг часто предваряет разработку и внедрение информационных систем на предприятии, так как требуется сначала навести определенный порядок в делопроизводстве, а лишь потом закрепить его информационной системой.

Рис. 1. Связь программной инженерии с другими областями

Программное обеспечение – это множество развивающихся во времени логических предписаний, с помощью которых коллектив людей управляет и использует многопроцессорную и распределенную систему вычислительных устройств.

Это определение, данное Харальдом Милсом, известным специалистом в области программной инженерии из компании IBM, заключает в себе следующее.

Логические предписания – это не только сами программы, но и различная документация (например, по эксплуатации программ) и шире – определенная система отношений между людьми, использующими эти программы в рамках некоторого процесса деятельности.

Современное ПО предназначено, как правило, для одновременной работы со многими пользователями, которые могут быть значительно удалены друг от друга в физическом пространстве. Таким образом, вычислительная среда (персональные компьютеры, серверы и т.д.), в которой ПО функционирует, оказывается распределенной.

Задачи, решаемые современным ПО, часто требуют разных вычислительных ресурсов в силу различной специализации этих задач, из-за большого объема выполняемой работы, а также из соображений безопасности. Например, появляется сервер базы данных, сервер приложений и пр. Следовательно, вычислительная среда, в которой ПО функционирует, оказывается многопроцессорной.

ПО развивается во времени – исправляются ошибки, добавляются новые функции, выпускаются новые версии, меняется его аппаратная база.

Таким образом, ПО является сложной динамической системой, включающей в себя технические, психологические и социальные аспекты. ПО заметно отличается от других видов систем, создаваемых (созданных) человеком – механических, социальных, научных и пр., и имеет следующие особенности, выделенные Фредериком Бруксом в его знаменитой статье «Серебряной пули нет»:

1. *Сложность* программных объектов, которая существенно зависит от их размеров. Как правило, большее ПО (большее количество пользователей, больший объем обрабатываемых данных, более жесткие требования по быстродействию и пр.) с аналогичной функциональностью – это другое ПО. Классическая наука строила простые модели сложных явлений, и это удавалось, так как сложность не была характеристической чертой рассматриваемых явлений. Сравнение программирования именно с наукой, а не с театром, кинематографом, спортом и другими областями человеческой деятельности оправдано, поскольку оно возникло главным образом из математики, а первые его плоды – программы – предназначались для использования при научных расчетах. Кроме того, большинство программистов имеют естественно-научное, математическое или техническое образование, именно поэтому парадигмы научного мышления широко используются при программировании – явно или неявно.

2. *Согласованность* – ПО основывается не на объективных посылках (подобно тому, как различные системы в классической науке подкреплены постулатами и аксиомами), а должно быть согласовано с большим количеством интерфейсов, с которыми впоследствии оно должно

взаимодействовать. Эти интерфейсы плохо поддаются стандартизации, поскольку они базируются на многочисленных и плохо формализуемых человеческих соглашениях.

3. *Изменяемость* – ПО легко изменить и, как следствие, требования к нему постоянно меняются в процессе разработки. Это создает много дополнительных трудностей при его разработке и эволюции.

4. *Нематериальность*, или, как было в оригинале, *незримость* – ПО невозможно увидеть, оно виртуально. Поэтому, например, трудно воспользоваться технологиями, основанными на предварительном создании чертежей, успешно используемыми в других промышленных областях (например, в строительстве, машиностроении). Там на чертежах в схематичном виде воспроизводятся геометрические формы создаваемых объектов. Когда объект создан, эти формы можно увидеть, а ПО увидеть невозможно.

1.2. Технология программирования и ее основные этапы

Программирование – сравнительно молодая и быстро развивающаяся отрасль науки и техники. Опыт ведения реальных разработок и совершенствования имеющихся программных и технических средств постоянно переосмысливается, в результате чего появляются новые методы, методологии и технологии, которые, в свою очередь, служат основой более современных средств разработки программного обеспечения. Исследовать процессы создания новых технологий и определять их основные тенденции целесообразно, сопоставляя эти технологии с уровнем развития программирования и особенностями имеющихся в распоряжении программистов программных и аппаратных средств.

Технологией программирования называют совокупность методов и средств, используемых в процессе разработки программного обеспечения. Как любая другая технология, технология программирования представляет собой набор технологических инструкций. Они включают:

- указание последовательности выполнения технологических операций;
- перечисление условий, при которых выполняется та или иная операция;
- описания самих операций, в которых для каждой операции определены исходные данные, результаты, а также инструкции, нормативы, стандарты, критерии и методы оценки и т.п.

Кроме набора операций и их последовательности, технология также определяет способ описания проектируемой системы, точнее модели, используемой на конкретном этапе разработки.

Различают технологии, используемые на конкретных этапах разработки или для решения отдельных задач этих этапов, и технологии, охватывающие несколько этапов или весь процесс разработки. В основе первых, как правило, лежит ограниченно применимый *метод*, позволяющий решить конкретную задачу, в основе вторых – базовый метод или *подход*, определяющий совокупность методов, используемых на разных этапах разработки, или *методологию*.

Чтобы разобраться в существующих технологиях программирования и определить основные тенденции их развития, целесообразно рассматривать эти технологии в историческом контексте, выделяя этапы развития программирования как науки.

Первый этап – «стихийное» программирование. Этот этап охватывает период от момента появления первых вычислительных машин до середины 60-х гг. XX в. В этот период практически отсутствовали сформулированные технологии и программирование фактически было искусством. Первые программы имели простейшую структуру. Они состояли из собственно программы на машинном языке и обрабатываемых ею данных. Сложность программ в машинных кодах ограничивалась способностью программиста одновременно мысленно отслеживать последовательность выполняемых операций и местонахождение данных при программировании.

Появление ассемблеров позволило вместо двоичных или 16-ричных кодов использовать символические имена данных и мнемоники кодов операций. В результате программы стали более «читаемыми».

Создание языков программирования высокого уровня, таких, как FORTRAN и ALGOL, существенно упростило программирование вычислений, снизив уровень детализации операций. Это, в свою очередь, позволило увеличить сложность программ.

Революционным было появление в языках средств, дающих возможность оперировать подпрограммами (идея написания подпрограмм возникла гораздо раньше, но отсутствие средств поддержки в первых языковых средствах значительно снижало эффективность их применения). Подпрограммы можно было сохранять и использовать в других программах. В результате были созданы огромные библиотеки расчетных и служебных подпрограмм, которые по мере надобности вызывались из разрабатываемой программы.

Типичная программа того времени состояла из основной программы, области *глобальных данных* и набора подпрограмм (в основном библиотечных), выполняющих обработку всех данных или их части.

Слабым местом такой архитектуры было то, что при увеличении количества подпрограмм возрастала вероятность искажения части глобальных данных какой-либо подпрограммой. Например, подпрограмма поиска корней уравнения на заданном интервале по методу деления отрезка пополам меняет величину интервала. Если при выходе из подпрограммы не предусмотреть восстановления первоначального интервала, то в глобальной области окажется неверное значение интервала. Чтобы сократить количество таких ошибок, было предложено в подпрограммах размещать *локальные данные*.

Сложность разрабатываемого программного обеспечения при использовании подпрограмм с локальными данными по-прежнему ограничивалась возможностью программиста отслеживать процессы обработки данных, но уже на новом уровне. Однако появление средств поддержки подпрограмм позволило осуществлять разработку программного обеспечения нескольким программистам параллельно.

В начале 60-х гг. XX в. разразился «кризис программирования». Он выражался в том, что фирмы, взявшиеся за разработку сложного программного обеспечения, такого, как операционные системы, срывали все сроки завершения проектов. Проект устаревал раньше, чем был готов к внедрению, увеличивалась его стоимость, и в результате многие проекты так никогда и не были завершены.

Объективно все это было вызвано несовершенством технологии программирования. Прежде всего стихийно использовалась разработка «снизу вверх» – подход, при котором вначале проектировали и реализовывали сравнительно простые подпрограммы, из которых затем пытались построить сложную программу. В отсутствие четких моделей описания подпрограмм и методов их проектирования создание каждой подпрограммы превращалось в непростую задачу, интерфейсы подпрограмм получались сложными, и при сборке программного продукта выявлялось большое количество ошибок согласования. Исправление таких ошибок, как правило, требовало серьезного изменения уже разработанных подпрограмм, что еще более осложняло ситуацию, так как при этом в программу часто вносились новые ошибки, которые также необходимо было исправлять… В конечном счете процесс тестирования и отладки программ занимал более 80 % времени разработки, если вообще когда-нибудь заканчивался. На повестке дня самым серьезным образом стоял вопрос разработки технологии создания сложных программных продуктов, снижающей вероятность ошибок проектирования.

Анализ причин возникновения большинства ошибок позволил сформулировать новый подход к программированию, который был назван «структурным».

Второй этап – структурный подход к программированию (60–70-е гг. XX в.). Структурный подход к программированию представляет собой совокупность рекомендуемых

технологических приемов, охватывающих выполнение всех этапов разработки программного обеспечения. В основе структурного подхода лежит *декомпозиция* (разбиение на части) сложных систем с целью последующей реализации в виде отдельных небольших (до 40–50 операторов) подпрограмм. С появлением других принципов декомпозиции (объектного, логического и т.д.) данный способ получил название *процедурной* декомпозиции.

В отличие от используемого ранее процедурного подхода к декомпозиции, структурный подход требовал представления задачи в виде иерархии подзадач простейшей структуры. Проектирование, таким образом, осуществлялось «сверху вниз» и подразумевало реализацию общей идеи, обеспечивая проработку интерфейсов подпрограмм. Одновременно вводились ограничения на конструкции алгоритмов, рекомендовались формальные модели их описания, а также специальный метод проектирования алгоритмов – метод пошаговой детализации.

Поддержка принципов структурного программирования была заложена в основу так называемых *процедурных* языков программирования. Как правило, они включали основные «структурные» операторы передачи управления, поддерживали вложение подпрограмм, локализацию и ограничение области «видимости» данных. Среди наиболее известных языков этой группы стоит назвать PL/1, ALGOL-68, Pascal, C.

Одновременно со структурным программированием появилось огромное количество языков, базирующихся на других концепциях, но большинство из них не выдержало конкуренции. Какие-то языки были просто забыты, идеи других были использованы в следующих версиях развиваемых языков.

Дальнейший рост сложности и размеров разрабатываемого программного обеспечения потребовал развития *структурирования данных*. Как следствие этого в языках появляется возможность определения пользовательских типов данных. Одновременно усилилось стремление разграничить доступ к глобальным данным программы, чтобы уменьшить количество ошибок, возникающих при работе с глобальными данными. В результате появилась и начала развиваться технология модульного программирования.

Модульное программирование предполагает выделение групп подпрограмм, использующих одни и те же глобальные данные в отдельно компилируемые *модули* (библиотеки подпрограмм), например, модуль графических ресурсов, модуль подпрограмм вывода на принтер. Связи между модулями при использовании данной технологии осуществляются через специальный интерфейс, в то время как доступ к реализации модуля (телам подпрограмм и некоторым «внутренним» переменным) запрещен. Эту технологию поддерживают современные версии языков Pascal и C (C++), языки Ада и Modula.

Использование модульного программирования существенно упростило разработку программного обеспечения несколькими программистами. Теперь каждый из них мог разрабатывать свои модули независимо, обеспечивая взаимодействие модулей через специально оговоренные межмодульные интерфейсы. Кроме того, модули в дальнейшем без изменений можно было использовать в других разработках, что повысило производительность труда программистов.

Практика показала, что структурный подход в сочетании с модульным программированием позволяет получать достаточно надежные программы, размер которых не превышает 100 000 операторов. Узким местом модульного программирования является то, что ошибка в интерфейсе при вызове подпрограммы выявляется только при выполнении программы (из-за раздельной компиляции модулей обнаружить эти ошибки раньше невозможно). При увеличении размера программы обычно возрастает сложность межмодульных интерфейсов, и с некоторого момента предусмотреть взаимовлияние отдельных частей программы становится практически невозможно. Для разработки программного обеспечения большого объема было предложено использовать объектный подход.

Третий этап – объектный подход к программированию (с середины 80-х до конца 90-х гг. XX в.). Объектно-ориентированное программирование – технология создания сложного программного обеспечения, основанная на представлении программы в виде совокупности *объектов*, каждый из которых является экземпляром определенного типа (*класса*), а классы образуют иерархию с *наследованием* свойств. Взаимодействие программных объектов в такой системе осуществляется путем передачи *сообщений*.

Объектная структура программы впервые была использована в языке имитационного моделирования сложных систем Simula, появившемся еще в 60-х гг. XX в. Естественный для языков моделирования способ представления программы получил развитие в другом специализированном языке моделирования – языке Smalltalk (70-е гг. XX в.), а затем был использован в новых версиях универсальных языков программирования, таких, как Pascal, C++, Modula, Java.

Основным достоинством объектно-ориентированного программирования по сравнению с модульным программированием является «более естественная» декомпозиция программного обеспечения, которая существенно облегчает его разработку. Это приводит к более полной локализации данных и интегрированию их с подпрограммами обработки, что позволяет вести практически независимую разработку отдельных частей (объектов) программы. Кроме этого, объектный подход предлагает новые способы организации программ, основанные на механизмах наследования, полиморфизма, композиции, наполнения. Эти механизмы позволяют конструировать сложные объекты из сравнительно простых. В результате существенно увеличивается показатель повторного использования кодов и появляется возможность создания библиотек классов для различных применений.

Бурное развитие технологий программирования, основанных на объектном подходе, позволило решить многие проблемы. Так были созданы среды, поддерживающие *визуальное программирование*, например, Delphi, C++ Builder, Visual C++ и т.д. При использовании визуальной среды у программиста появляется возможность проектировать некоторую часть, например, интерфейсы будущего продукта, с применением визуальных средств добавления и настройки специальных библиотечных компонентов. Результатом визуального проектирования является заготовка будущей программы, в которую уже внесены соответствующие коды.

Использование объектного подхода дает много преимуществ, однако его конкретная реализация в объектно-ориентированных языках программирования, таких, как Pascal и C++, имеет существенные недостатки:

• фактически отсутствуют стандарты компоновки двоичных результатов компиляции объектов в единое целое даже в пределах одного языка программирования: компоновка объектов, полученных разными компиляторами C++, в лучшем случае проблематична, что приводит к необходимости разработки программного обеспечения с использованием средств и возможностей одного языка программирования высокого уровня и одного компилятора, а значит, требует одного языка программирования высокого уровня и одного компилятора, а также наличия исходных кодов используемых библиотек классов;

• изменение реализации одного из программных объектов, как минимум, связано с перекомпиляцией соответствующего модуля и перекомпоновкой всего программного обеспечения, использующего данный объект.

Таким образом, при использовании этих языков программирования сохраняется зависимость модулей программного обеспечения от адресов экспортируемых полей и методов, а также структур и форматов данных. Эта зависимость объективна, так как модули должны взаимодействовать между собой, обращаясь к ресурсам друг друга. Связи модулей нельзя разорвать, но можно попробовать стандартизировать их взаимодействие, на чем и основан компонентный подход к программированию.

Четвертый этап – компонентный подход и CASE-технологии (с середины 90-х гг. XX в. до нашего времени). Компонентный подход предполагает построение программного обеспечения из компонентов – физически отдельно существующих частей программного обеспечения, которые взаимодействуют между собой через *стандартизованные двоичные интерфейсы*. В отличие от обычных объектов, объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы, распространять в двоичном виде (без исходных текстов) и использовать в любом языке программирования, поддерживающем соответствующую технологию. В настоящее время рынок объектов стал реальностью: в Интернете существуют узлы, предоставляющие большое количество компонентов, рекламой компонентов забиты журналы. Это позволяет программистам создавать продукты, хотя бы частично состоящие из повторно использованного кода, т.е. использовать технологию, хорошо зарекомендовавшую себя в области проектирования аппаратуры.

Компонентный подход лежит в основе технологий, разработанных на базе COM (Component Object Model – компонентная модель объектов), и технологии создания распределенных приложений CORBA (Common Object Request Broker Architecture – общая архитектура с посредником обработки запросов объектов). Эти технологии используют сходные принципы и различаются лишь особенностями их реализации.

Технология COM фирмы Microsoft является развитием технологии OLE I (Object Linking and Embedding – связывание и внедрение объектов), которая использовалась в ранних версиях Windows для создания составных документов. Технология COM определяет *общую парадигму взаимодействия программ любых типов*: библиотек, приложений, операционной системы, т.е. позволяет одной части программного обеспечения использовать функции (*службы*), предоставляемые другой, независимо от того, функционируют ли эти части в пределах одного процесса, в разных процессах на одном компьютере или на разных компьютерах. Модификация COM, обеспечивающая передачу вызовов между компьютерами, называется DCOM (Distributed COM – распределенная COM).

По технологии COM приложение предоставляет свои службы, используя специальные объекты – *объекты* COM, которые являются экземплярами *классов* COM. Объект COM, так же как обычный объект, включает поля и методы, но в отличие от обычных объектов может реализовывать несколько интерфейсов, обеспечивающих доступ к его полям и функциям. Это достигается за счет организации отдельной таблицы адресов методов для каждого интерфейса (по типу таблиц виртуальных методов). При этом интерфейс обычно объединяет несколько однотипных функций. Кроме того, классы COM поддерживают *наследование интерфейсов*, но не поддерживают *наследования реализации*, т.е. не наследуют код методов, хотя при необходимости объект класса-потомка может вызвать метод родителя.

Каждый интерфейс имеет имя, начинающееся с символа «I» и глобальный уникальный идентификатор IID (Interface IDentifier). Любой объект COM обязательно реализует интерфейс ILJnknown (на схемах этот интерфейс всегда располагают сверху). Использование этого интерфейса позволяет получить доступ к остальным интерфейсам объекта.

Объект всегда функционирует в составе *сервера* – динамической библиотеки или исполняемого файла, которые обеспечивают функционирование объекта. Различают три типа серверов:

• внутренний – реализуется динамическими библиотеками, которые подключаются к приложению-клиенту и работают в одном с ними адресном пространстве, это наиболее эффективный сервер, кроме того, он не требует специальных средств;

• локальный – создается отдельным процессом (модулем, exe), который работает на одном компьютере с клиентом;

• удаленный – создается процессом, который работает на другом компьютере.

Например, Microsoft Word является локальным сервером. Он включает множество объектов, которые могут использоваться другими приложениями.

Для обращения к службам клиент должен получить указатель на соответствующий интерфейс. Перед первым обращением к объекту клиент посылает запрос к библиотеке COM, хранящей информацию обо всех зарегистрированных в системе классах COM-объектов, и передает ей имя класса, идентификатор интерфейса и тип сервера. Библиотека запускает необходимый сервер, создает требуемые объекты и возвращает указатели на объекты и интерфейсы. Получив указатели, клиент может вызывать необходимые функции объекта.

Взаимодействие клиента и сервера обеспечивается базовыми механизмами COM или DCOM, поэтому клиенту безразлично местонахождение объекта. При использовании локальных и удаленных серверов в адресном пространстве клиента создается *proxy-объект* – заместитель объекта COM, а в адресном пространстве сервера COM – заглушка, соответствующая клиенту. Получив задание от клиента, заместитель упаковывает его параметры и, используя службы операционной системы, передает вызов заглушке. Заглушка распаковывает задание и передает его объекту COM. Результат возвращается клиенту в обратном порядке.

На базе технологии COM и ее распределенной версии DCOM были разработаны компонентные технологии, решающие различные задачи разработки программного обеспечения.

OLE-automation, или просто *Automation* (автоматизация) – технология создания программируемых приложений, обеспечивающая программируемый доступ к внутренним службам этих приложений. Вводит понятие *диспинтерфейса* (dispinterface) – специального интерфейса, облегчающего вызов функций объекта. Эту технологию поддерживает, например, Microsoft Excel, предоставляя другим приложениям свои службы.

ActiveX – технология, построенная на базе OLE-automation, которая предназначена для создания программного обеспечения как сосредоточенного на одном компьютере, так и распределенного в сети. Предполагает использование визуального программирования для создания компонентов – элементов управления ActiveX. Полученные таким образом элементы управления можно устанавливать на компьютер дистанционно с удаленного сервера, причем устанавливаемый код зависит от используемой операционной системы. Это позволяет применять элементы управления ActiveX в клиентских частях приложений Интернет.

Конец ознакомительного фрагмента.