МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Омский государственный технический университет»

А. Г. Белик, В. Н. Цыганенко

ПРОЕКТИРОВАНИЕ И АРХИТЕКТУРА ПРОГРАММНЫХ СИСТЕМ

Учебное пособие

Омск
Издательство ОмГТУ
2016

УДК 004.4(075)
ББК 32.97я73
Б43

Рецензенты:

С. Н. Чуканов, д-р техн. наук, профессор,
зав. кафедрой «Компьютерные информационные
автоматизированные системы»
ФГБОУ ВО «Сибирская государственная
автомобильно-дорожная академия (СибАДИ)»;

П. Н. Надточий, канд. физ.-мат. наук,
зав. отделом разработки ПО ООО «Автоматика-Э»

Белик, А. Г.

Б43 Проектирование и архитектура программных систем : учеб. пособие / А. Г. Белик, В. Н. Цыганенко ; Минобрнауки России, ОмГТУ. – Омск : Изд-во ОмГТУ, 2016. – 96 с. : ил.

ISBN 978-5-8149-2258-8

На основе обзора литературных источников и нормативно-технической документации изложены методические подходы и технологии решения задач проектирования программного обеспечения; приведен аналитический обзор типовых решений по системной архитектуре, моделированию информационных баз и процессов; рассмотрены современные информационные технологии, применяемые в этой области; изложены основные сведения по управлению программными проектами, обеспечению качества и документированию программных систем.

Может быть использовано студентами следующих направлений бакалавриата и магистратуры: 09.03.01 «Информатика и вычислительная техника», 09.03.04 «Программная инженерия», 27.03.03 «Системный анализ и управление».

УДК 004.4(075)
ББК 32.97я73

*Печатается по решению редакционно-издательского совета
Омского государственного технического университета*

ISBN 978-5-8149-2258-8

ОГЛАВЛЕНИЕ

СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ

АСОИУ – автоматизированная система обработки информации и управления;
БД – база данных;
НСИ – нормативно-справочная информация;
ООП – объектно-ориентированное программирование;
ПО – программное обеспечение;
РАС – распределенная автоматизированная система;
ЭВМ – электронная вычислительная машина;
CASE – *computer aided software engineering*, технология автоматизации разработки программного обеспечения;
COM – *component object mode*, технологический стандарт создания программного обеспечения на основе взаимодействующих компонентов;
CORBA – *common objectrequest broker architecture*, общая архитектура брокера объектных запросов, технологический стандарт написания распределённых приложений;
EPM – *enterprise performance management*, управление эффективностью деятельности организации;
ERP – *enterprise resource planning*, планирование ресурсов предприятия;
HTTP – *hypertext transfer protocol*, протокол прикладного уровня передачи данных;
OMG – *object management group*, консорциум, занимающийся разработкой объектно-ориентированных технологий и стандартов;
RAD – *rapid application development*, технология быстрой разработки приложений;
PDM – *product data management*, управление данными об изделии;
UML – *unified modeling language*, язык графического описания для объектного моделирования в разработке программного обеспечения;
SADT – structured analysis and design technique, технология структурного анализа в моделировании систем;
SOAP – *simple object access protocol*, протокол обмена структурированными сообщениями в распределённой вычислительной среде;
TCP/IP – *transmission control protocol/internet protocol*, набор сетевых протоколов передачи данных, используемых в сетях, включая Интернет;
TQM – *totaly quality management*, общеорганизационный метод непрерывного повышения качества всех организационных процессов;
XML – *extensible markup language*, расширяемый язык разметки, позволяющий структурировать информацию разного типа;
WEB – всемирная паутина, система доступа к связанным между собой документам на различных компьютерах, подключённых к Интернету;
WSDL – *web services description language*, язык описания веб-сервисов и доступа к ним, основанный на языке XML.

ВВЕДЕНИЕ

Разработка программных систем является трудоемким, плохо формализуемым и одновременно творческим процессом. В условиях развитого рынка программного обеспечения разработчики вынуждены решать две во многом противоречивые задачи: с одной стороны, это создание программного комплекса в заданные сроки в соответствии с формальными требованиями заказчика и, с другой стороны, творческий процесс исследовательского поиска по его оптимальной реализации.

Для достижения эффективных результатов при решении этих задач инженерия программного обеспечения требует применения специализированных методик управления всеми стадиями жизненного цикла программной системы: анализа, проектирования, реализации и эксплуатации разработанного программного продукта.

Поскольку производство программных продуктов является одной из самых динамично развивающихся областей в сфере информационных технологий, что демонстрируется существенным ежегодным приростом объемов рынка реализации, необходимо развивать технологии, превращающие программирование из творческого поиска в инженерную науку.

В процессе создания программного обеспечения основными составляющими, которые оказывают наиболее существенное влияние на качество производимых продуктов, являются следующие «четыре П»: продукт, персонал, процесс и проект [1].

Продукт представляет собой объект, получаемый в результате деятельности разработчиков, состоящий из кода программы, совокупности данных, документации и различных артефактов.

Под персоналом понимается не только коллектив разработчиков различной специализации, но и другие участники процесса проектирования: заказчики, пользователи, инвесторы.

Процесс – это организованная совокупность стадий и этапов разработки: анализа, проектирования, разработки, внедрения и сопровождения. А под проектом понимается совокупность исследовательских и проектных действий и операций, необходимая для создания программного продукта.

Артефактами считаются любые дополнительные результаты, используемые или порождаемые в процессе разработки программного обеспечения. К ним относятся, как правило, элементы информации или аспекты

нестандартного поведения системы, возникающие в ходе тестирования или эксплуатации.

Наиболее распространенным подходом к анализу, проектированию и программированию в настоящее время является объектно-ориентированная парадигма. С ее использованием построено большинство инструментальных RAD-систем быстрой разработки приложений, позволяющих осуществлять максимально производительную разработку программных продуктов с соблюдением необходимых требований к качеству.

Для обеспечения процессов создания ПО в условиях, когда требуется не просто выполнить один конкретный проект, но создать производственный конвейер по разработке ПО, важное значение приобретает применение стандартов. Объектом таких стандартов являются промышленные технологии индустрии производства программных продуктов.

1. СИСТЕМНЫЙ ПОДХОД

1.1. Системы и их признаки

Для правильной организации эффективных процессов разработки ПО и обеспечения высокого качества программных продуктов важно отчетливо понимать и применять при анализе и проектировании принципы системного взаимодействия объектов и субъектов реального мира.

Понятие системы постоянно совершенствуется в процессе развития методологических основ исследования окружающей нас реальности. Основоположник теории систем Людвиг фон Берталанфи первоначально определил систему как комплекс взаимодействующих элементов, находящихся в определенных отношениях друг с другом и со средой. При этом под средой понимается все то, что не входит в систему, а система, наоборот, представляет собой конечное множество объектов, целевым образом выделенное из среды. При этом между средой и системой существует множество взаимных связей, с помощью которых реализуется процесс их взаимодействия.

Таким образом, при определении системы необходимо исходить из следующего основополагающего принципа: система – это целостное образование, представляющее собой организованное и динамическое множество элементов, свойств и отношений, имеющее системообразующие свойства и взаимодействующее с внешним окружением.

Один из вариантов модели взаимодействия предприятия «как системы» с элементами ее внешней среды [2] представлен на рис. 1.

Программная система – система, состоящая из частей – компонентов программного обеспечения (рис. 2). В свою очередь, программное обеспечение, которое существует в совокупности с техническим (аппаратным), математическим, информационным, организационным, методическим и др. [3], является одним из видов обеспечивающих подсистем автоматизированной системы обработки информации и управления (АСОИУ).

АСОИУ представляет собой систему обработки данных, основанную на использовании ЭВМ и связанную с управлением теми или иными объектами (предприятиями, организациями, технологическими процессами). Она предназначена для целенаправленного автоматизированного ведения производственных, организационно-административных и технологиче-

ских процессов с выдачей достоверной технико-экономической и технологической информации.

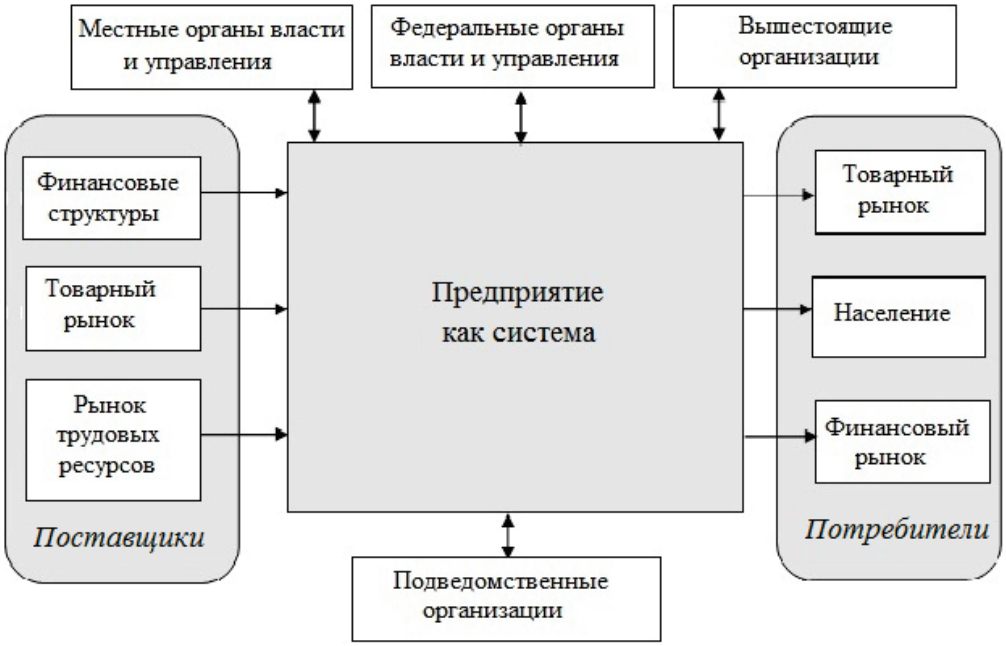

Рис. 1. Модель предприятия как система

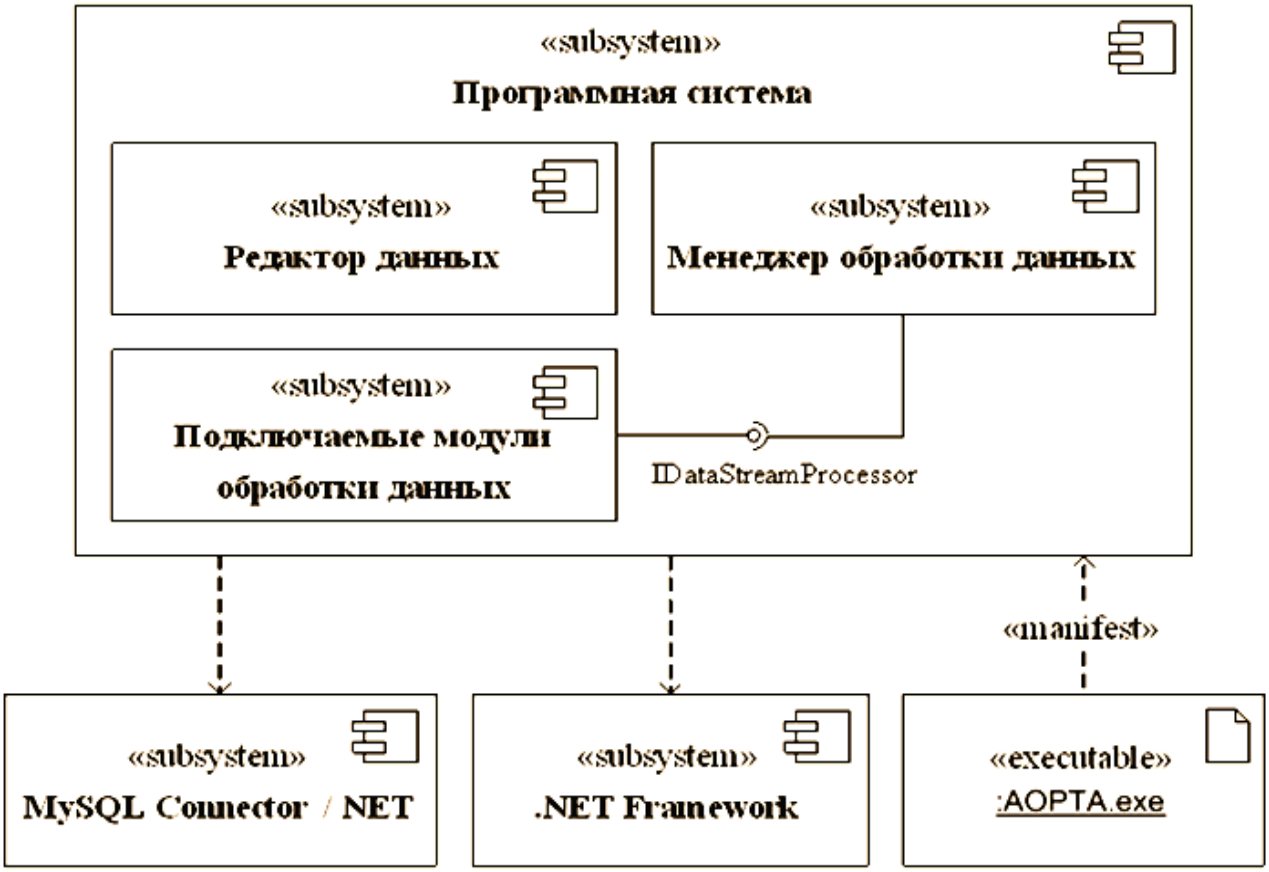

Рис. 2. Пример программной системы

В целом можно выделить следующие признаки любой системы, позволяющие различать их среди разнообразных объектов реального мира [4]:

– система представляет собой совокупность элементов, которые тоже могут рассматриваться как системы; в свою очередь, исходная система является частью более общей системы, таким образом, система представляется как органическая часть иерархии систем;

– система характеризуется наличием интегративных свойств, которые определяют систему в целом, однако не свойственны отдельным ее элементам;

– систему отличает наличие существенных, хотя и различных по типу, связей между элементами (совокупность разрозненных частей считать системой нельзя).

1.2. Системный подход как исследование

Системный подход представляет собой универсальное направление методологии исследования, основанного на рассмотрении объекта исследования как целостного множества взаимосвязанных элементов, т. е. как системы. При системном подходе основное внимание уделяется не анализу элементов как таковых, а изучению структуры объекта и взаимодействия составляющих систему элементов [4, 5]. Основными моментами, определяющими системность любого исследования, являются следующие:

– исследование феномена целостности и установление состава элементов целого;

– изучение закономерностей соединения элементов в систему и определение структуры объекта, что является ядром системного подхода;

– установление функций системы в тесной связи с анализом ее структуры, т. е. структурно-функциональный анализ системы;

– изучение и анализ генезиса системы, сфер ее влияния, связей с другими системами.

Системный подход как комплексная методология процесса исследования и анализа систем характеризуется следующими основными признаками:

– исследуемый объект оценивается как единое целое независимо от рассматриваемой точки зрения;

– решение проблем отдельных элементов подчиняется решению общих для всей системы проблем;

– познание объекта не ограничивается только механизмом функционирования, а распространяется и на установление внутренних закономерностей развития объекта;

– элементы системы, которые имеют второстепенное значение в одних условиях, при изменении обстоятельств могут оказаться существенными.

Главными принципами системного подхода являются:

– *единство* – система рассматривается как единое целое, являясь совокупностью составных частей;

– *целостность* – элементы системы могут быть разной направленности по функциональности и характеру поведения, но они одновременно должны быть совместимы между собой;

– *динамичность* – способность системы к изменению состояния под воздействием направленных или случайных факторов;

– *взаимозависимость системы и среды* – система проявляет свои свойства в процессе взаимодействия со средой;

– *иерархичность* – структурное упорядочивание частей, когда каждый элемент системы рассматривается как подсистема, а сама система – как элемент более сложной системы;

– *организованность* – формальное использование правил взаимодействия элементов для приведения в порядок составных частей и связей;

– *множественность состояний* – определение различных моделей, каждая из которых описывает определенное состояние системы;

– *декомпозиция* – возможность разделения объекта на составные части исходя из подцелей, возникающих из общей цели всей системы.

Сущность системного подхода определяется следующей совокупностью взаимосвязанных видов представления объекта:

– *элементного*, показывающего из каких элементов состоит система при ее отображении и исследовании;

– *структурного*, раскрывающего внутреннюю организацию системы, характер связей и способы взаимодействия компонентов;

– *функционального*, отвечающего на вопрос: «Какие функции выполняет сама система и образующие ее компоненты?»;

– *коммуникационного*, определяющего взаимосвязь данной системы с другими как по горизонтали (сотрудничество), так и по вертикали (соподчиненность);

– *интегративного*, устанавливающего общие механизмы сохранения, совершенствования и развития системы;

– *исторического*, объясняющего то, каким образом возникла система, какие этапы проходила в своем развитии и каковы перспективы ее дальнейшего совершенствования.

Наибольшая эффективность применения системного подхода к исследованию и анализу появляется при комплексном его использовании при решений как стратегических задач объекта, так задач определения структуры и поведения системы. Это позволяет рассматривать системный подход как триаду системных моделей (рис. 3), которая обеспечивает нужный эффект при реализации и эксплуатации программной системы.

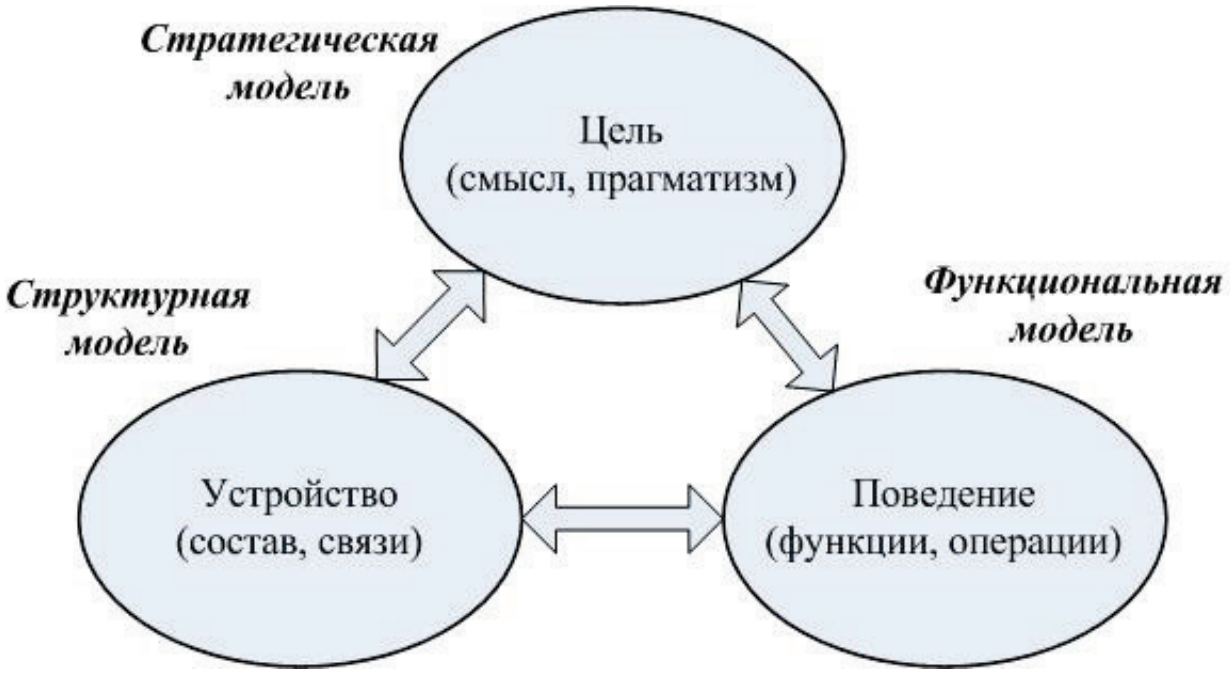

Рис. 3. Триада системного подхода

В триаде системного подхода учитывается, что:

– любая система создана для решения определенных задач и имеет предназначение, задаваемое ее целями;

– для реализации этого предназначения она должна уметь выполнять определенные функции, владеть некими аспектами деятельности;

– общие цели и задачи достигаются в структурном взаимодействии функциональных элементов, между которыми установлены связи.

Системный подход является еще и жизненной философией, которая помогает успешно решать проблемы повседневной жизни путем нахождения нестандартных решений, придерживаться «золотой середины» и избегать крайностей. Совершенствование системного мышления является непростым процессом, требующим интеллектуальных способностей. При этом нельзя ограничиваться только готовыми шаблонами поведения, следует использовать глубину мышления, интуицию и здравый смысл.

При решении проблем, связанных с программными системами, различают два подхода: улучшение и проектирование. Улучшением является преобразование, которое приближает систему к стандартным, или нормальным, условиям работы. Для улучшения систем используются методы, базирующиеся на научном методе, которые называют научной парадигмой. Проектирование представляет собой исследовательский и творческий процесс, который отвергает решения, лежащие в основе используемых форм, и требует получения новых. Методы, применяемые для проектирования систем, основаны на теории систем и называются системной парадигмой.

1.3. Системный анализ программных систем

Системный анализ означает синтез идей и принципов теории систем и кибернетики с возможностями современной вычислительной техники и используется для изучения и моделирования объектов сложной природы (систем) [4, 5, 6]. В основе системного анализа лежат такие основополагающие принципы, как:

1) структурированность;
2) взаимосвязанность и согласованность проблем;
3) целеполагание и разрешимость;
4) допустимость, рациональность и оптимальность;
5) ориентация на качественный результат;
6) согласованность целей, средств и результатов;
7) стабильность и изменчивость.

Сутью *структурного принципа* является сохранение качественных характеристик системы при ее разделении на части или объединении частей. При этом используются два основных метода исследования:

– *декомпозиция* – метод анализа и получения оценок проблемы на основе изучения свойств ее частей;

– *агрегирование* – метод исследования на основе объединения подзадачи в единую задачу.

При последовательной декомпозиции совокупность составных частей образует так называемое *иерархическое дерево*, при этом процесс разбиения является итеративным. Выделение элементов одного уровня такой структуры следует проводить на основании следующих принципов:

– *существенности*, выделять следует элементы, существенные для цели анализа данного уровня;

– *однородности*, следует включать элементы, имеющие одинаковую важность по отношению к цели анализа данного уровня;

– *независимости*, элементы анализируемого уровня должны быть взаимно независимы.

Главной целью системного анализа является помощь в понимании и решении имеющейся проблемы, которая возникает при проектировании или управлении, путем ее перевода в задачу принятия решения. Представить систему в удобном для исследования виде и выступить в качестве инструмента концептуального проектирования позволяют модели системного анализа. Системный анализ программной системы использует три основных типа моделей: информационную, структурную и поведенческую, которые в процессе проектирования последовательно преобразуются в проектные решения по разработке базы данных, программной архитектуры и процедурных алгоритмов (рис. 4).

Далее проектные решения воплощаются в результате кодирования в программные модули, образующие после стадии тестирования готовую к эксплуатации программную систему.

Системный анализ как исследовательский процесс состоит из последовательности этапов, которые включены в замкнутый итерационный цикл управления, так как результаты анализа, реализованные в виде проектных решений, воздействуют на реальную автоматизированную систему управления, приводя ее в новое качественное состояние, которое требует системного исследования.

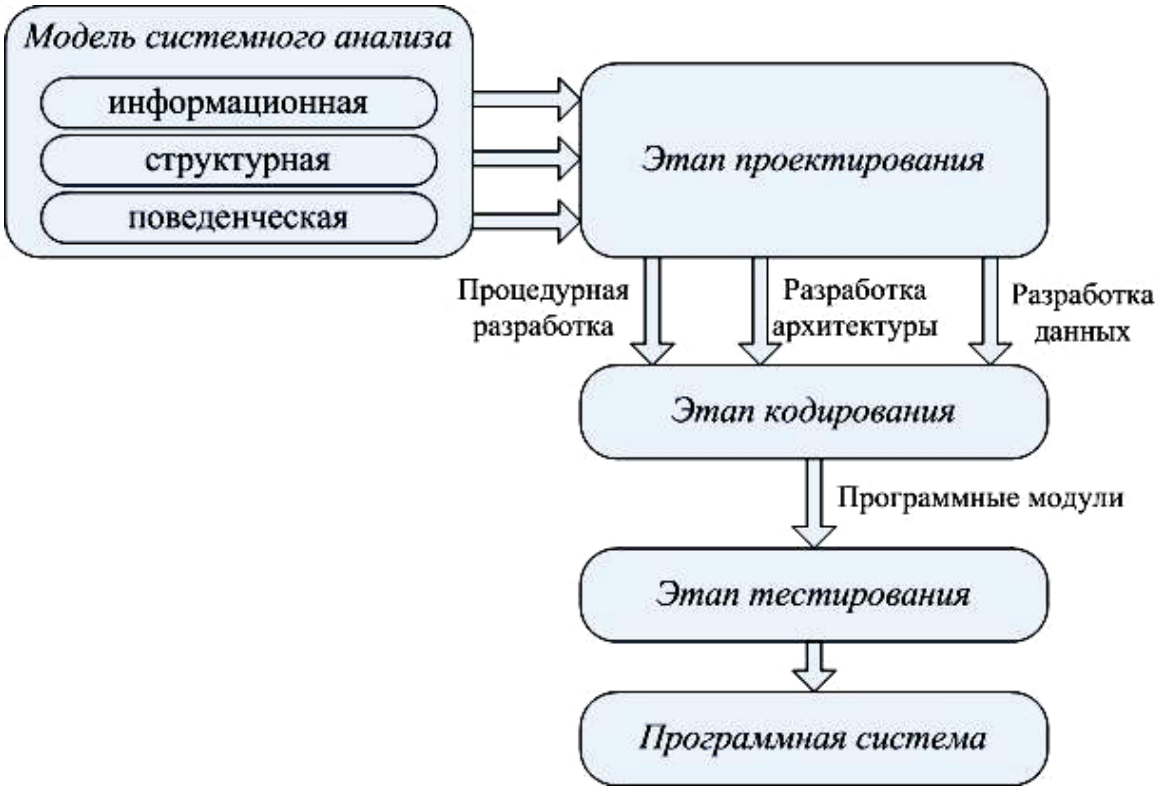

Рис. 4. Модели системного анализа при проектировании ПО

Основные этапы системного анализа, реализующие такую модель управления, представлены схематически на рис. 5.

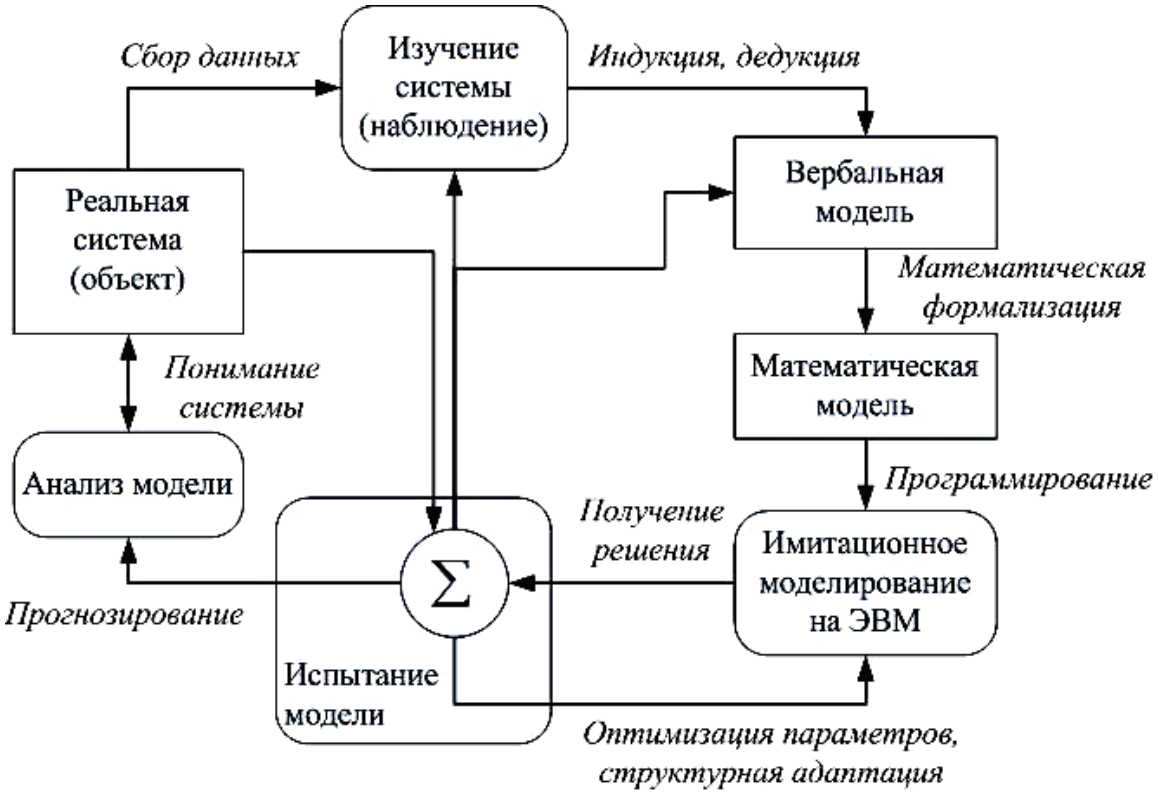

Рис. 5. Этапы системного анализа

В данной модели основными процессами, составляющими логическую последовательность проведения системного анализа, являются:

– изучение реального объекта путем наблюдения за его состоянием и формированием сначала вербальной (представление средствами естественного языка), а затем и математической модели;

– имитационное моделирование с использованием программ, реализующих сформированные модели, в результате которого формируется решение по управлению объектом;

– испытание полученной имитационной модели, в ходе которого определяются оптимальные параметры управления, осуществляется структурная адаптация, проводится уточнение самой модели, формируются прогнозы поведения исследуемой системы;

– анализ результатов моделирования как основной процесс интерпретации и понимания системной сущности моделируемого объекта с формированием управляющего воздействия, изменяющего его качественное состояние.

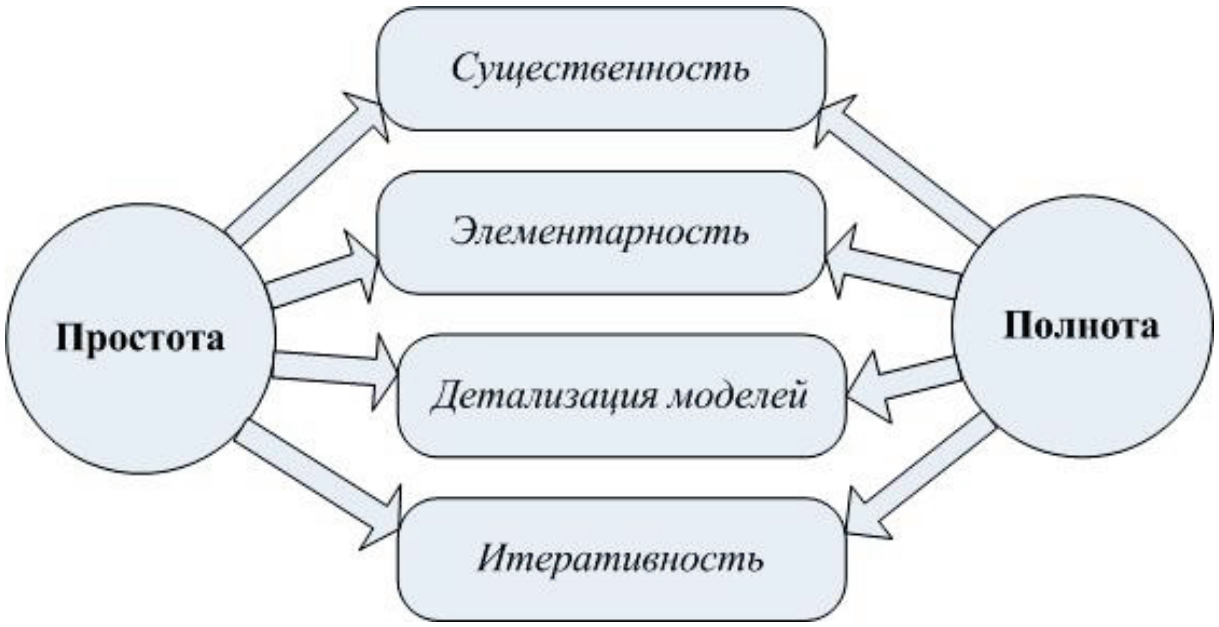

Рис. 6. Системный анализ как компромисс между простотой и полнотой

Следует отметить, что системный анализ можно рассматривать как процедуру формирования компромисса при моделировании между стремлением разработчиков, с одной стороны, получить максимально полную и точную копию исследуемого объекта, а с другой стороны, иметь наиболее простое и удобное в использовании представление системы (рис. 6). Дан-

ный компромисс позволяет получить оптимизационное решение по таким аспектам параметров создаваемой модели системы, как их существенность для задач анализа и элементарность, степень детализации и итеративность.

1.4. Автоматизированные системы обработки информации и управления

АСОИУ представляет собой *систему обработки данных*, основанную на использовании ЭВМ и связанную с управлением теми или иными *объектами* (предприятиями, организациями, технологическими процессами) [3]. Она предназначена для *целенаправленного автоматизированного ведения* производственных, организационно-административных и технологических процессов с формированием достоверной технико-экономической и технологической информации. Под *объектом управления* понимается совокупность технологического оборудования, персонала, технических, организационных, экономических и финансовых процессов.

В состав АСОИУ включаются следующие системные элементы:

– современные автоматические средства *сбора и обработки информации*;

– *человек* как субъект труда, принимающий содержательное решение в выработке решений по управлению;

– *реализация* в системе процесса обработки организационно-административной, технологической и технико-экономической информации;

– цели функционирования системы, заключающиеся в общем смысле в *оптимизации работы* объекта по заданному критерию.

Основными целями создания АСОИУ являются:

– экономия топлива, материалов и других производственных ресурсов;

– обеспечение нужного уровня безопасности работы объекта;

– повышение качества продукции или обеспечение заданных свойств производимых изделий;

– уменьшение затрат труда персонала;

– обеспечение оптимальной загрузки оборудования;

– оптимизация режимов выполнения технологического процесса;

– комфортность оперативно-обслуживающего персонала;

– повышение оперативности управления.

В целом АСОИУ создаются для достижения нужных значений критериев эффективности, т. е. признаков, на основе которых производится оценка качества системы. Существуют два типа критериев эффективности: критерии эффективности первого рода, которые характеризует *степень достижения цели* системой, и критерии второго рода, являющиеся оценками *эффективности пути* достижения цели. Чаще всего приходится иметь дело с векторным критерием, составляющие которого – самостоятельные независимые критерии. Такие системы называются *многокритериальными*.

К критериям эффективности предъявляются требования, обусловливающие возможность их использования при оценке качественных характеристик системы и применимость к проведению оптимизации управления объектом. Так, критерий эффективности должен:

– быть *количественным*, т. е. являться числом или набором чисел для векторного критерия;

– быть *простым и эффективным* в статическом смысле, т. е. разброс измеряемых значений относительно истинной величины должен быть невелик;

– иметь *физический смысл*, что снижает возможность ошибок при его использовании;

– быть *нормируемым*, т. е. сравниваемым с идеальным или заданным критерием и выраженным в процентах или долях единицы.

Совокупность действий, направленных на достижение определенной цели управления, называется *функцией* системы. К основным группам функций АСОИУ относятся *управляющие и информационные* функции системы, направленные на конкретного потребителя (объект, персонал, внешние системы), а также вспомогательные функции, обеспечивающие решение внутрисистемных задач по функционированию системы и не имеющие потребителя вне системы. Так, основными функциями организационно-экономических систем являются:

– *планирование*, т. е. выбор целей и путей их достижения;

– *оперативное управление*;

– *учет состояния* объекта управления;

– *контроль и анализ* состояния объекта управления;

– *регулирование* как действие по поддержанию плановой траектории управления объектом;

– *связи с внешней средой*.

Общая структура АСОИУ рассматривается как совокупность обеспечивающих подсистем независимо от сферы применения. К таким подсистемам относятся следующие:

1) *информационное обеспечение*, представляющее собой совокупность системы классификации, кодирования и документирования информации, моделей информационных потоков, методологии построения БД;

2) *техническое обеспечение*, являющееся комплексом технических средств, обеспечивающих работу информационной системы, в том числе и документация на эти средства и способы их использования;

3) *математическое и алгоритмическое обеспечение*, представляющее собой математические методы, модели и алгоритмы, такие как модели процессов управления, методы математического программирования, математической статистики, теории массового обслуживания и др.;

4) *программное обеспечение*, которое представляет собой комплекс программ для реализации целей и задач АСОИУ и обеспечения нормального функционирования технических средств;

5) *организационное обеспечение* – набор организационных мер, методов и средств, определяющих характер взаимодействия персонала с техническими средствами и между собой в процессе разработки и эксплуатации АСОИУ;

6) *правовое обеспечение*, состоящее из правовых норм, определяющих юридический статус АСОИУ и регламентирующих порядок получения и использования информации.

Существуют различные по составу и назначению АСОИУ, поэтому необходима их классификация. К основным признакам этой классификации относятся:

– структурированность задач;

– выполняемые функции;

– уровень управления;
– уровень автоматизации;
– сфера применения;
– характер использования задач;
– тип данных и др.

На рис. 7 приведены основные типы АСОИУ по функциональному признаку.

Рис. 7. Классификация АСОИУ по функциональному признаку

АСОИУ, как правило, имеют определенную иерархическую организацию. Например, если в качестве АСОИУ выступает комплексная система автоматизации предприятия, то управленческие или производственные задачи входящих в него отделов и подразделений решаются с использованием внутренних специализированных систем автоматизации. Подчиненность структурных элементов такой иерархии может иметь разную силу в зависимости от набора функций, выполняемых каждым подразделением в процессе своей деятельности.

2. ПРОЦЕСС РАЗРАБОТКИ ПРОГРАММНЫХ СИСТЕМ

2.1. Жизненный цикл ПО

Жизненный цикл ПО представляет собой непрерывный и упорядоченный набор видов деятельности, осуществляемый и управляемый в рамках проекта по разработке и эксплуатации ПО [7, 8]. Он начинается с момента появления идеи (замысла) создания ПО, принятия решения о необходимости его создания и завершается процессом полного изъятия из эксплуатации. В жизненном цикле программного обеспечения обычно выделяют следующие этапы:

1) системный анализ и обоснование необходимости разработки;
2) формирование требований;
3) проектирование;
4) программирование;
5) тестирование и отладка;
6) ввод программы в действие;
7) эксплуатация и сопровождение;
8) завершение эксплуатации.

Процессы жизненного цикла ПО в настоящее время регламентируются стандартом ISO 12207–1999 и показаны на рис. 8. Они объединены в три группы: основные, вспомогательные и организационные.

Рис. 8. Процессы жизненного цикла программного обеспечения по ISO 12207

Основной процесс разработки ПО в данной модели жизненного цикла представляет собой анализ, проектирование и реализацию программной системы. Он включает работы по созданию ПО в соответствии с заданными требованиями, оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для проведения тестирования и проверки работоспособности и качества программного продукта, организации обучения персонала.

2.2. Этапы проектирования программных систем

При планировании процесса разработки программной системы важное значение имеет выбор стратегии разработки при реализации проектных работ. Существуют три основных стратегии создания ПО [8]:

– *однократный проход* (водопадная стратегия), в ходе которого выполняется линейная последовательность этапов конструирования;

– *инкрементная стратегия*, заключающаяся в изначальном установлении всех пользовательских и системных требований, которые в дальнейшем реализуются в виде последовательности версий;

– *эволюционная стратегия*, при которой программная система также строится в виде последовательности версий, но в начале процесса определены не все требования, они уточняются в результате разработки версий.

Классический цикл проектирования реализуется водопадной (каскадной) моделью, представляющей последовательность этапов, при том, что переход на следующий этап происходит после завершения работ на текущем. В оригинальной каскадной модели Ройса [9] эти этапы расположены в таком порядке:

– определение требований;
– проектирование;
– конструирование (реализация, кодирование);
– воплощение;
– тестирование и отладка (в том числе верификация);
– инсталляция;
– поддержка.

Каскадная модель в чистом виде почти не используется из-за недостаточной гибкости процесса проектирования, однако она определяет логическую последовательность этапов проектирования, применяемую как в

итерационных, так и эволюционных моделях. В частности, широкое применение получила V-образная модель, которая была разработана как разновидность каскадной модели [8, 10]. Здесь каждая последующая фаза также начинается по завершению получения результативных данных предыдущей фазы. Однако в данной модели используется комплексный подход к определению фаз процесса проектирования ПО за счет выделения взаимосвязей, существующих между аналитическими фазами и фазами проектирования, предшествующими кодированию, и последующими фазами верификации и тестирования. Схематически данная модель представлена на рис. 9.

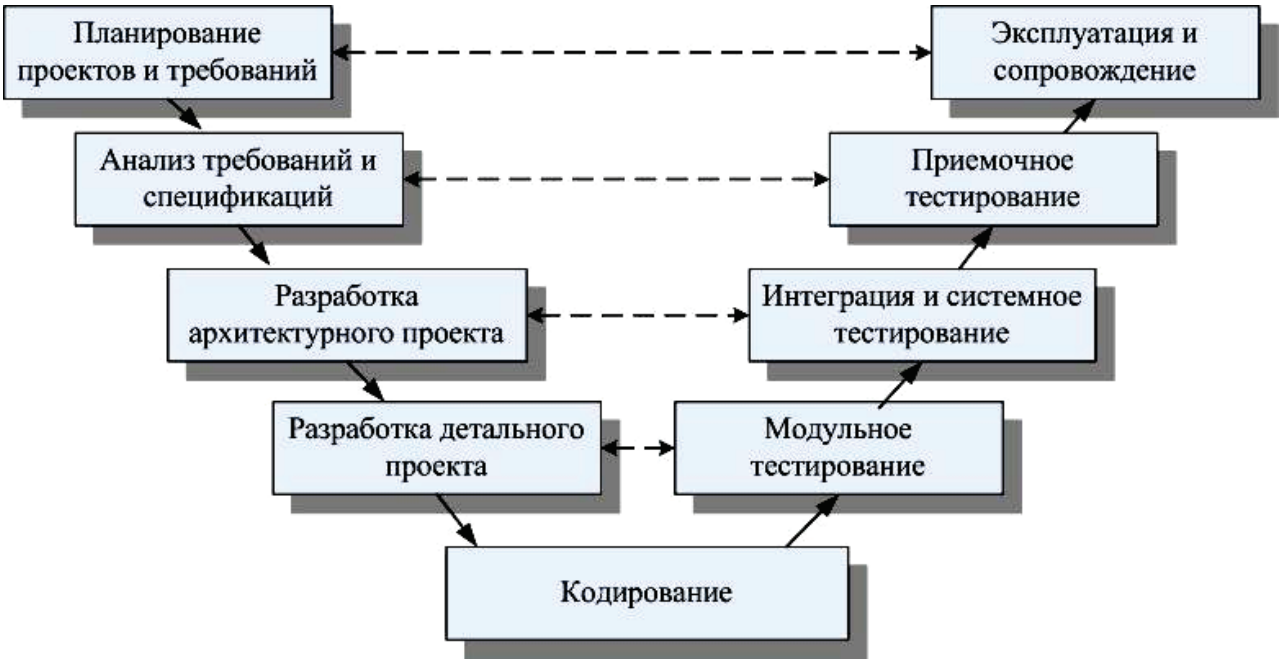

Рис. 9. Основные этапы процесса разработки ПО

Отметим, что наиболее трудоемким при разработке ПС является архитектурное и детальное проектирование, в ходе которого ведется разработка основных аспектов программного проекта:

– определение архитектуры программной системы как совокупности взаимодействующих компонентов;

– создание структуры данных, составляющих информационную базу;

– проектирование алгоритмов, отражающих детали поведения при реализации выполняемых процессов;

– разработка пользовательского интерфейса;

– документирование.

2.3. Управление программными проектами

Процесс проектирования программной системы является, как правило, достаточно сложным и противоречивым [11]. К основным источникам возникающих сложностей относятся:

– наличие и доступность высококвалифицированных специалистов на рынке труда;

– стабильность используемой технологической платформы, стабильность и функциональность инструментов разработки;

– эффективность используемых методов разработки, включая методы моделирования, проектирования, тестирования и управления версиями;

– наличие и доступность специалистов, обладающих экспертизой в прикладной области;

– используемая методология и ее соответствие данному проекту; сроки и финансирование проекта;

– множество других организационных и технических переменных.

Данные сложности приводят к проблемам при управлении программными проектами, которые могут определяться следующими факторами.

1. Многие процессы разработки неуправляемы, их исходные данные и желаемый результат неизвестны или определены очень нечетко.

2. Процесс достижения желаемого результата не поддается формализации, наиболее это характерно для разработки архитектуры и исчерпывающего тестирования продукта.

3. Идентифицированные процессы разработки сопровождаются неизвестным количеством неидентифицированных.

4. Требования к продукту часто меняются в течение жизненного цикла проекта, что требует сложной процедуры изменения и согласования требований.

5. Попытки предложить формальную, детализованную методологию разработки ПО оказываются безуспешны, потому что сам процесс разработки не поддается детализации и формализации.

6. Слепое следование методологиям, предполагающим управляемость и предсказуемость процессов разработки, приводит к непредсказуемым результатам проекта.

Любой проект разработки ПО имеет свою организационную модель, которая определяет распределение обязанностей, ответственности и полномочий среди исполнителей проекта, а также отношений отчетности. Чем меньше проект, тем больше ролей приходится совмещать одному исполнителю. Всех исполнителей типового проекта разработки ПО можно условно разделить на пять категорий в соответствии с их ролями:

1) *анализ* – сбор информации, исследование, составление и сопровождение требований к продукту;

2) *управление* – организация и управление производственными процессами;

3) *производство* – проектирование и разработка ПО;

4) *тестирование* – проверка и контрольные испытания ПО;

5) *обеспечение* – производство дополнительных продуктов и услуг.

Для того чтобы обеспечить успешность программного проекта, необходимо реализовать его выполнение в соответствии со спецификациями, в срок и в пределах бюджета.

2.4. CASE-технологии

Многочисленные факторы, определяющие сложности в реализации программных проектов, привели к появлению программно-технологических средств автоматизации процесса проектирования ПО – CASE-средств [7, 12]. При этом реализуется специальная CASE-технология создания и сопровождения программных систем, которая представляет собой методологию автоматизированного проектирования ПО, а также инструментальные средства реализации. Это позволяет в наглядной форме представлять модель предметной области, анализировать ее на всех этапах разработки и сопровождения системы, разрабатывать программные компоненты в соответствии с информационными запросами пользователей. CASE-технология включает в себя методы визуализации программных решений, с помощью которых на основе графической нотации строятся диаграммы, поддерживаемые инструментальной средой.

CASE-средства повышают производительность труда программистов и улучшают качество программного обеспечения. Это достигается тем, что они:

– обеспечивают автоматизированный контроль совместимости спецификаций проекта;

– уменьшают время создания прототипа системы и ускоряют процесс проектирования и разработки;

– автоматизируют формирование проектной документации для всех этапов жизненного цикла;

– частично генерируют коды программ для различных сред разработки;

– поддерживают технологии повторного использования компонентов;

– обеспечивают возможность восстановления проектной документации по имеющимся исходным кодам.

Автоматизация процессов разработки ПО предполагает широкое использование *компонентного подхода* к проектированию, т. е. реализацию его построения из отдельных компонентов, физически обособленных существующих частей, которые взаимодействуют между собой с использованием *стандартизованных двоичных интерфейсов*.

Отличием объектов – компонентов ПО от обычных программных модулей является то, что объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы и потом распространять их в двоичном виде без исходных текстов и применять в любом языке программирования, поддерживающем соответствующую технологию.

2.5. Методологии разработки ПО

В последнее время вопросу выбора методологии разработки программного обеспечения уделяется особое внимание, так как без правильной методологии даже небольшие проекты не могут быть успешными [7, 13].

На разных уровнях и по разным критериям выделяют пересекающиеся модели:

– водопадная (каскадная) модель, реализующая основное направление нисходящего структурного программирования;

– макетирование;

– спиральная (итерационная) модель разработки ПО;

– объектно-ориентированное программирование;

– гибкие (agile) технологии: экстремальное программирование (XP), Scrum, TDD, FDD;

– методология моделирования RUP (Rational Unified Process);

– методологии компонентного подхода (COM, CORBA, RAD).

3. МОДЕЛИРОВАНИЕ ПРОГРАММНЫХ СИСТЕМ

В практике проектирования программных систем широко применяется визуальное моделирование, которое осуществляется с использованием средств схематического или иного описания, анализа, проектирования и документирования решений по структурной организации ПО [7, 10]. Модели используются для того, чтобы разработчик и специалист по эксплуатации мог понять структуру и поведение программной системы, управление проектом осуществлялось максимально легко и прозрачно, с уменьшением возможных рисков.

Кроме того, моделирование дает возможность документировать принимаемые проектные решения и служит основой взаимодействия между участниками проекта, что способствует созданию качественного ПО. Документация по программной архитектуре представлена набором графических средств и нотаций представления моделей системы и их описанием. В описании указывается, из каких подсистем состоит система, из каких модулей формируется каждая подсистема. Графические схемы позволяют оценить структуру системы с разных сторон.

Модель системной архитектуры является основой для создания спецификации системных компонентов, причем при ее проектировании разрабатывается базовая структура, т. е. определяются основные элементы системы и взаимодействия между ними.

Процесс моделирования может быть разделен на несколько этапов: опрос экспертов, создание диаграмм и моделей, распространение документации, оценка адекватности моделей и принятие их для дальнейшего использования. Принятие решений по системной организации программной системы представляет собой многогранный, плохо формализованный и сложный процесс, в ходе которого широко используются системные исследования и анализ с использованием моделирования различных аспектов проектирования. При этом может использоваться широкий спектр методологий системного анализа и моделирования. К наиболее продуктивным из них следует отнести методологию структурного анализа и моделирования SADT . Она интегрирует процессы моделирования и управления конфигурациями проекта с использованием дополнительных языковых средств логической систематизации структуры и поведения, а также эффективной визуализации программных систем.

Основные правила формирования и визуализации структурных моделей базируются на методологии IDEF, которая предназначена для решения задач моделирования, отображения и анализа деятельностей разнообразных сложных систем в различных разрезах. Глубина и объем обследования процессов определяются самим разработчиком, что позволяет оптимизировать создаваемую модель, не перегружая ее излишними данными. В настоящее время данная методология содержит 15 стандартов (от IDEF 0 до IDEF 14), используемых для создания различных видов представления сложных систем при проведении структурного анализа.

В целом SADT представляет собой совокупность методов, правил и процедур, используемых для построения структурной модели объекта автоматизации. Модели SADT отображают функциональную, процессную или информационную структуры объекта, включая производимые элементами системы действия и связи между этими действиями. Основными концепциями методологии структурного анализа и проектирования являются:

– графическое представление структурных блоков модели, при этом графика диаграммы представляет функцию в виде прямоугольного блока, а интерфейсы входа/выхода изображаются дугами, входящими в блок и выходящими из него;

– строгость и точность, предполагающие выполнение формальных условий SADT.

При создании SADT-диаграммы нужно руководствоваться следующими правилами:

– ограничение количества блоков на каждом уровне декомпозиции (правило 3–6 блоков);

– связность диаграмм с помощью нумерации блоков;

– уникальность наименований (отсутствие повторяющихся названий);

– формальность синтаксических правил для графики блоков и дуг;

– разделение входов и управлений для определения роли данных;

– исключение влияния организационной структуры на функциональную модель.

Эта методология предназначена для моделирования широкого круга систем, а также определения требований и функций, которые затем явля-

ются основой для проектирования. Для уже существующих систем методология используется с целью структурного анализа функций, выполняемых системой, и указания механизмов, посредством которых они осуществляются.

3.1. Архитектура программной системы

Под архитектурой программной системы понимается совокупность основополагающих решений по структурной организации программной системы. Она включает в себя [7, 14]:

– выбор структурных элементов и их интерфейсов, с помощью которых обеспечивается их взаимодействие и совместное функционирование;

– соединение выбранных элементов структуры и поведения во всё более крупные системы;

– архитектурный стиль, который направляет всю организацию – все элементы, их интерфейсы, их сотрудничество и их соединение.

Архитектурное проектирование является начальным этапом процесса проектирования, когда определяются основные подсистемы, процессы и структура управления и взаимодействия. Его результатом должно стать формирование структуры программного обеспечения.

Архитектурный вид состоит из двух основных компонентов: структурных элементов и отношений между ними. При этом схематически архитектура может быть представлена в логическом или физическом виде.

Если структурные элементы представляют собой концептуальные (логические) компоненты ПО, такие как прецеденты, классы, процессы, состояния и др., то такая модель программной архитектуры будет соответствовать логической. Физическая же архитектура будет являться реализацией логической структуры ПС, когда ее элементами станут физически существующие компоненты, такие как программы, базы и файлы данных, интерфейсы и т. д.

На рис. 10 приведен пример логической структурной схемы программной архитектуры на примере системы управления эффективностью деятельности организации (EPM-система). На рис. 11 представлена физическая модель программной архитектуры WEB-системы.

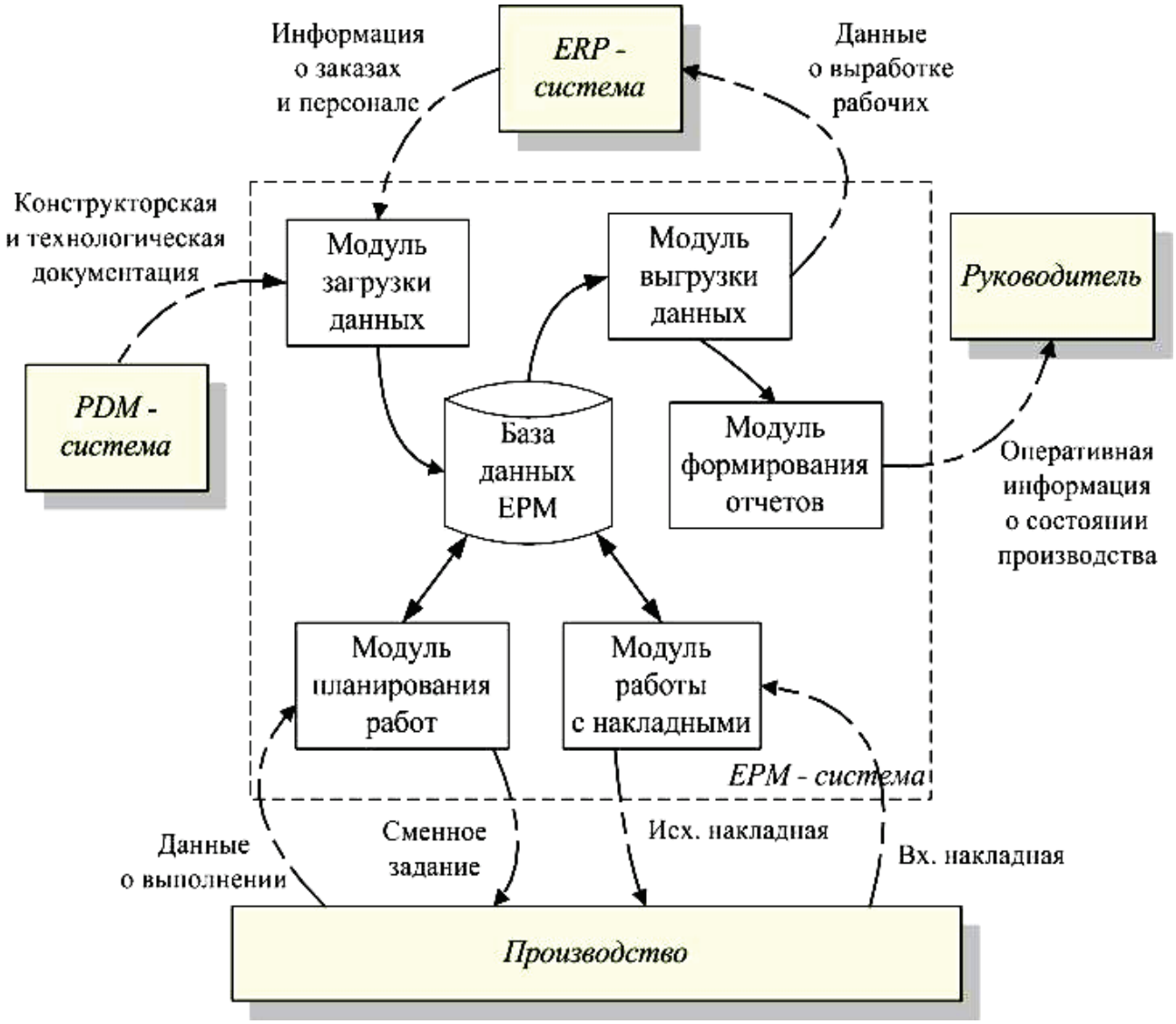

Рис. 10. Логическая программная архитектура EPM-системы

Выделим основные этапы, которые являются общими для любых процессов проектирования системной архитектуры:

1) *структурирование системы*, на этом этапе система структурируется в виде набора относительно независимых подсистем, при этом определяются виды взаимодействия между ними;

2) *моделирование управления*, при этом создается базовая модель управления отношениями между элементами системы;

3) *модульная декомпозиция* – этап, на котором определенные на первом этапе подсистемы разбиваются на отдельные процессы (модули), а также определяются их типы и взаимные связи.

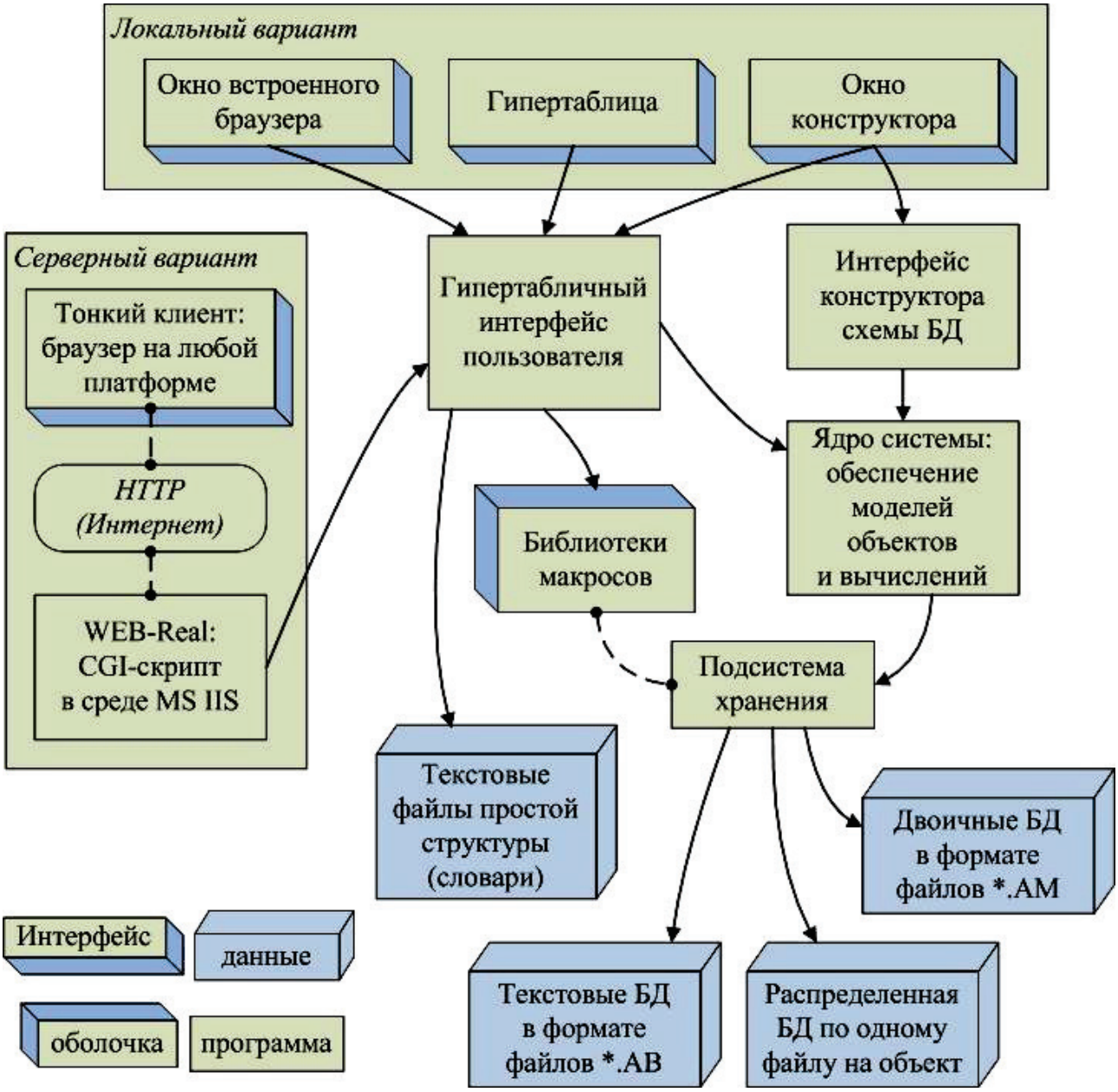

Рис. 11. Физическая программная архитектура WEB-системы

Как правило, разрабатывается четыре типа моделей архитектуры ПО:

1) *статическая* структурная модель, которая отображает подсистемы или компоненты, разрабатываемые на следующих этапах независимо;

2) *динамическая* модель процессов, представляющая организацию процессов во время функционирования системы;

3) *интерфейсная* модель, устанавливающая сервисы, предоставляемые каждой подсистемой через общесистемный интерфейс;

4) модели *отношений*, показывающие взаимодействия между элементами системы, такие как потоки данных.

Модели архитектуры определяются и рядом нефункциональных системных требований, к которым следует отнести следующие.

1. *Производительность*. Требование высокой производительности системы предполагает такую архитектуру, где за все критические операции отвечало бы минимальное число подсистем с ограниченным взаимодействием между ними. В таких случаях лучше использовать компоненты в виде крупных модулей, а не мелкие структурные элементы.

2. *Защищенность*, для обеспечения которой требуется архитектура с многоуровневой структурой, где наиболее критические системные элементы защищены на внутренних уровнях, а проверка их безопасности осуществляется на более высоком уровне.

3. *Безопасность*, для обеспечения которой архитектуру следует спроектировать так, чтобы при этом за операции, влияющие на безопасность системы, должно отвечать как можно меньше подсистем.

4. *Надежность*, повысить которую можно путем разработки архитектуры с включением избыточных компонентов, которые можно было бы заменять и обновлять, не останавливая общее функционирование системы.

5. *Удобство сопровождения* предполагает, что системная архитектура системы должна основываться на уровне мелких структурных компонентов, которые можно легко заменять и модифицировать. Программные модули, создающие данные, необходимо отделить от модулей, их использующих. Кроме того, нужно избегать структур совместного использования данных.

3.2. Структурное моделирование процессов

На первом этапе разработки программной архитектуры система разбивается на несколько взаимодействующих подсистем. На абстрактном уровне это показывается графически с помощью структурной схемы, в которой подсистемы представлены отдельными блоками. Стрелками, связывающими блоки, обозначаются потоки данных или потоки управления [8, 12].

Характерными представителями методов визуализации структурных моделей системы являются такие методологии структурного анализа

SADT, как функциональное моделирование IDEF0 и моделирование потоков данных DFD.

В подходах IDEF0 структурной единицей системы является функциональный блок, выполняющий некоторое преобразование и связанный с внешним окружением и другими функциональными блоками интерфейсами (рис. 12).

Интерфейсы (связи) представлены четырьмя типами:

1) слева в блок входят стрелки интерфейсов, отражающих входную информацию, необходимую для выполнения функционального преобразования;

2) справа выходят стрелки выходной информации;

3) сверху входят интерфейсы управляющей информации, регламентирующей поведение блока;

4) снизу указываются интерфейсы механизмов реализации функции, под которыми понимается уникальный способ ее использования различными пользователями или техническими устройствами или системами, являющимися источниками, потребителями или контроллерами выполняемой функции.

Рис. 12. Функциональный блок модели IDEF0

Каждый функциональный блок в методологии IDEF0 может быть подвергнут декомпозиции, результатом которой является представление этого блока новой диаграммой, состоящей из функциональных блоков нижнего уровня структурной иерархии.

Диаграммы потоков данных (DFD) используются в качестве основного средства моделирования функциональных требований к системной архитектуре. С их помощью структурные элементы разбиваются на процессы (поведенческие компоненты) и представляются в виде связанной потоками данных сети. Типами структурных элементов DFD являются:

– *процесс (система, подсистема)* – структурный элемент, представляющий продуцирование выходных потоков данных из входных в соответствии с действием, задаваемым назначением процесса, который может быть подвергнут декомпозиции и отображен следующей в иерархии моделирования диаграммой потоков данных;

– *накопитель данных* определяет логические хранилища данных, которые должны использоваться процессами;

– *внешняя сущность* представляет собой объект вне контекста системы, являющийся источником или приемником внешних данных.

На рис. 13 приведен пример диаграммы первого уровня для системы определения допустимых скоростей движения железнодорожных составов.

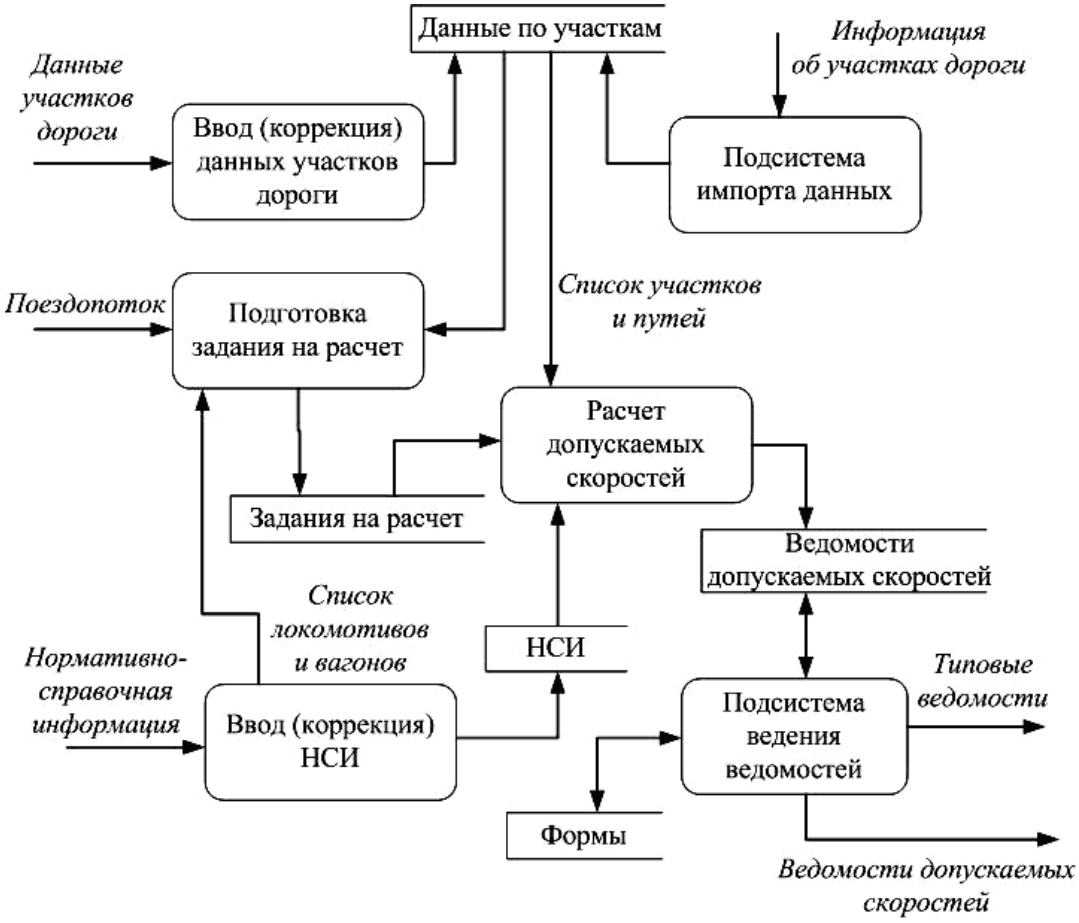

Рис. 13. Диаграмма потоков данных

Процесс моделирования потоков данных может быть представлен следующей последовательностью этапов:

1) *идентификация внешних объектов*, с которыми система должна быть связана, основных видов информации, циркулирующей между системой и внешними объектами;

2) *разработка контекстной диаграммы*, на которой система представляется процессом, внешние объекты – внешними сущностями, связанные с системой потоками данных;

3) *формирование диаграммы потоков данных* первого уровня на базе декомпозиции контекстной диаграммы, которая, как правило, представляет основные подсистемы как процессы, накопители данных, обслуживающие всю систему, потоки данных между этими элементами;

4) *декомпозиция каждой подсистемы* (процесса) текущей диаграммы с помощью детализирующей диаграммы или спецификации процесса; построение спецификации процесса в случае, если некоторую функцию сложно или невозможно выразить очередной комбинацией процессов;

5) *построение спецификации процесса* в случае, если некоторую функцию сложно или невозможно выразить очередной комбинацией процессов.

Наиболее существенным и важным моментом для построения эффективной программной архитектуры с использованием методов структурного анализа является мотивация разделения системы на составляющие ее компоненты. Рассмотрим основные аргументы, используемые проектировщиками при принятии таких архитектурных решений.

Для обеспечения эффективной работы системы между ними должен идти интенсивный обмен информацией, который можно организовать следующими способами:

– хранение всех совместно используемых данных в центральной базе данных, доступной всем подсистемам. Модель, основанную на совместном использовании БД, называют *моделью репозитория*;

– хранение данных каждой подсистемы в собственную БД, при этом информационный обмен между подсистемами происходит путем *передачи сообщений*.

В большинстве случаев совместное использование больших объемов данных является более эффективным, так как не требуется осуществлять

направленную передачу данных из одной подсистемы в другие. Однако при этом необходимо находить компромисс между требованиями, предъявляемыми к каждой подсистеме, из-за снижения их производительности, при этом для подсистем, в которых создаются данные, не важно, как эти данные используются в других подсистемах. Кроме того, для систем, где генерируются большие объемы информации, проблематичной становится их модернизация, так как переход на новую модель данных будет дорогостоящим и сложным.

Отметим, что в системах с репозиторием такие средства, как резервное копирование, обеспечение безопасности, управление доступом и восстановление данных, также централизованы, так как входят в систему управления репозиторием. Модель репозитория является и более прозрачной, при этом если новые подсистемы совместимы с согласованной моделью данных, их легко интегрировать.

Хотя в структурных моделях нет никакой информации по управлению, разработчик программной архитектуры обязан организовать подсистемы согласно установленной модели управления, дополняющей функциональную модель. При этом на уровне архитектуры проектируется поток управления между подсистемами.

Выделяются два основных типа управления:

– *централизованное управление*, когда определенная подсистема полностью отвечает за управление, т. е. она запускает и завершает работу остальных подсистем;

– *управление, основанное на событиях*, при котором реагировать на внешние события может любая подсистема, при этом эти события могут происходить как в других подсистемах, так и во внешнем окружении системы.

При централизованном управлении одна из систем назначается главной, она управляет работой других подсистем. Существует два класса таких моделей: *модель вызова-возврата* и *модель диспетчера*. Первая представляет модель организации вызова программных процедур «сверху вниз», когда управление от процедур верхних уровней иерархии через вызовы передается на более нижние уровни. Такую модель можно применять в последовательных системах.

Модель диспетчера характерна для параллельных систем, где один системный компонент является диспетчером, он управляет запуском, завершением и координированием других процессов системы. При этом диспетчеризация может выполняться параллельно с другими процессами. Модель диспетчера можно применять и в последовательных системах, когда управляющая программа вызывает отдельные модули в зависимости от значений некоторых переменных состояния.

В моделях управления, основанного на событиях, управляющее воздействие определяется значениями системных переменных состояния. Существуют также системы, в которых событие представляет собой передачу сообщения всем подсистемам, которые обрабатывают данное событие и отвечают на него.

Еще одной событийной моделью является модель управления прерываниями, которые обычно используются в системах реального времени, когда внешние прерывания регистрируются обработчиком прерываний и обрабатываются некоторым системным компонентом.

После того как разработка системной структуры завершается, следует этап декомпозиции подсистем на модули. При этом могут использоваться две модели модульной декомпозиции:

– *объектно-ориентированная модель*, в которой система представляется набором взаимодействующих объектов [16];

– *процессная (процедурная) модель* потоков данных, где система состоит из процессов, которые получают на входе данные и преобразуют их некоторым образом в выходные данные.

В объектно-ориентированной модели структурными элементами являются объекты, имеющие собственное состояние и определенные операции с этими состояниями. Объектная архитектура формирует систему в виде совокупности слабо связанных объектов с четко определенными интерфейсами, при этом объекты вызывают сервисы, предоставляемые другими объектами.

В модели потоков данных структурные элементы выполняют функциональные преобразования, образующие некоторую последовательность отдельных шагов обработки данных. В ней данные поступают на вход системы, проходят через все последовательные или параллельные преоб-

разования и достигают выхода. Иногда такой подход называют *конвейерным*.

Еще одной разновидностью системных архитектурных моделей, характерных для конкретной предметной области, являются *проблемно-зависимые архитектуры*, среди которых выделяют два основных типа:

– *модели классов систем*, которые реализуют классы реальных систем на основе их реальных характеристик и используются в системах реального времени, таких как системах сбора данных, мониторинга и др.;

– *базовые модели*, являющиеся более абстрактными и предоставляющие информацию по общей структуре некоторого типа систем.

Среди моделей классов систем наиболее известна модель компилятора, в которой используются компоненты с разными архитектурными моделями, например архитектура потоков данных, в которой таблица идентификаторов служит хранилищем совместно используемых данных и может интегрироваться с подсистемой редактирования структур и документов, организованной в соответствии с моделью репозитория.

Базовые архитектурные модели отображают наиболее идеализированные архитектуры, отражающие особенности систем конкретной предметной области. Примером такой архитектуры является модель взаимодействия открытых систем OSI. Основное назначение базовых моделей заключается в том, чтобы являться эталоном для сравнения программных систем в некоторой предметной области, т. е. выступать в роли стандарта при оценке различных систем.

3.3. Структурное моделирование данных

Большинство информационных хранилищ используют технологию реляционных баз данных, поскольку она предлагает надежные, проверенные и эффективные средства хранения и управления большими объемами данных. Важнейшим вопросом, связанным с конструированием хранилищ данных, является структура базы данных как логическая, так и физическая. Создание логической схемы информационной базы требует всеобъемлющего моделирования автоматизируемой деятельности [7, 18, 19]. Цель моделирования данных состоит в обеспечении разработчика ПС концептуальной схемой БД в виде одной или нескольких локальных мо-

делей, которые относительно легко преобразуются в систему информационных баз.

Самым известным и популярным средством моделирования данных являются диаграммы «сущность – связь» (ERD), позволяющие осуществить детализацию накопителей данных в моделях потоков данных, а также документировать информационные аспекты автоматизированной системы, важные для предметной области объекты (сущности), их свойства (атрибуты) и связи с другими объектами (отношения).

Сущность (Entity) характеризует множество однотипных экземпляров реальных или абстрактных объектов (людей, событий, состояний, предметов), обладающих общими атрибутами или характеристиками. Любой информационный объект может быть представлен одной сущностью, при этом имя которой должно отражать тип объекта, а не его конкретный экземпляр. Сущность должна обладать уникальным идентификатором, который и используется для однозначной идентификации экземпляра данного типа сущности.

Сущность должна обладать следующими свойствами:

– иметь уникальное имя;

– иметь один или несколько атрибутов, которые могут принадлежат сущности или наследоваться через связь;

– иметь один или несколько атрибутов, однозначно идентифицирующих каждый экземпляр.

Сущность может иметь произвольное количество связей с другими сущностями. *Связью (Relationship*) называется поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области.

Атрибутом (Attribute) является любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут – это тип характеристики или свойства, ассоциированных с множеством реальных или абстрактных объектов. На *ER-диаграмме* (диаграмме «сущность – связь» ER – это аббревиатура Entity-Relationship) атрибуты ассоциируются с конкретными сущностями так, чтобы экземпляр сущности мог обладать его единственным определенным значением.

Среди методов визуализации моделей данных путем построения ER-диаграмм наиболее популярными методами являются метод Баркера и метод IDEF1X, входящий в методологию SADT [8, 12]. Рассмотрим основные особенности решения этой задачи на примере последней.

IDEF1X является методом разработки реляционных БД и использует условную нотацию, специально разработанную для удобного построения концептуальной схемы, которая называется универсальное представление структуры данных, независимое от конечной реализации БД и аппаратной платформы.

В логической модели данных описываются информационные объекты предметной области и взаимосвязи между ними, а физическая модель данных представляет собой реализацию логической модели как структуры систем хранения выявленных объектов данных.

Пример диаграммы «сущность – связь» представлен на рис. 14 моделью информационной базы системы учета заказов по обслуживанию компьютерной техники.

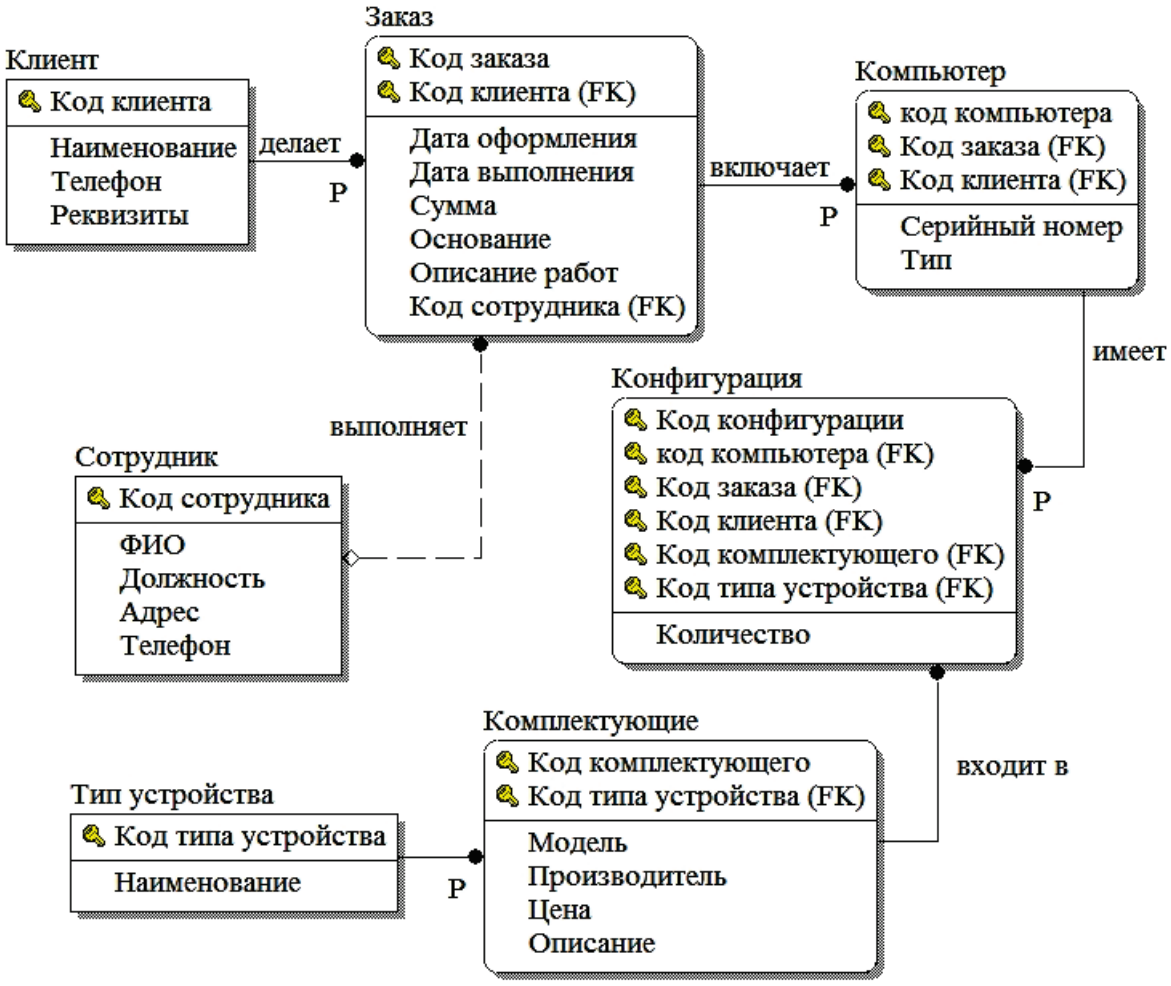

Рис. 14. Логическая диаграмма «сущность – связь» модели данных

Создание модели данных начинается с разработки логической модели, после чего проектировщик может выбрать необходимую СУБД и создать соответствующую физическую модель.

3.4. Моделирование поведения и алгоритмизация

Поведенческое моделирование сложных систем используют для определения динамики их функционирования [7, 20]. Методы и средства моделирования системного поведения должны удовлетворять набору различных, часто противоречивых требований. Укажем некоторые из них:

– модель должна быть достаточно детальной для того, чтобы являться основой для написания текста программы;

– модель должна быть компактной и обозримой для того, чтобы служить средством обсуждения в процессе разработки системы и для обмена идеями;

– модель не должна зависеть от особенностей реализации, используемых аппаратных средств, языка программирования, а также применяемых технологий проектирования БД;

– модели поведения должны быть знакомыми и привычными для большинства программистов и не противоречить основным парадигмам программирования.

Поскольку удовлетворить сразу всем требованиям в полной мере практически невозможно, средства моделирования поведения являются результатом многочисленных компромиссов и предполагают различные формы представления. Выбор конкретного инструмента диктуется типом поведения, которое нужно описать. Методы моделирования поведения можно разделить на четыре группы:

– описание поведения с явным выделением состояний;

– описание поведения с явным выделением потоков данных и управления между элементами системы;

– описание поведения как последовательности сообщений во времени;

– описание параллельного поведения.

Одним из способов отображения поведения систем является методология IDEF3, которая является стандартом документирования информа-

ционных, организационных или технологических процессов и является удобным инструментом для наглядного исследования и моделирования их сценариев.

Применяются два основных типа диаграмм в IDEF3, представляющие описание процесса в разных ракурсах:

– диаграммы описания последовательности этапов процесса (*Process Flow Description Diagrams, PFDD*),

– диаграммы состояния объекта в процессе и его трансформаций (*Object State Transition Network, OSTN*).

Центральными компонентами модели PFDD являются единицы работы – *Unit of Work (UOW)*, которые можно просто называть работами (*Activity*). Связи на диаграмме показывают взаимоотношения работ. А для визуального представления логики взаимодействия при слиянии и разветвлении или при объединении используются «перекрестки» (*Junction*).

Пример диаграммы IDEF3 показан на рис. 15 для описания последовательности этапов процесса подготовки и публикации видеолекций.

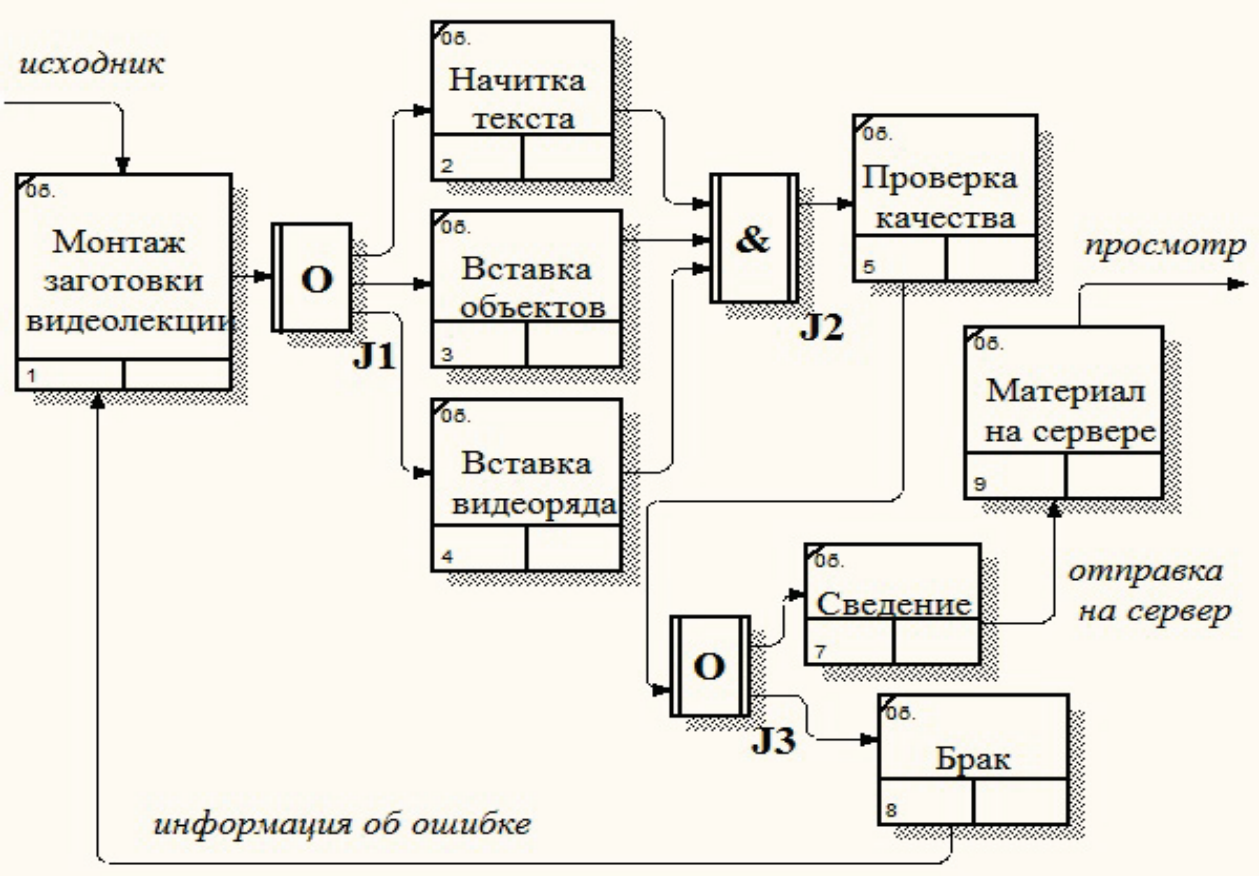

Рис. 15. Диаграмма описания последовательности этапов процесса IDEF3

Практически все типы поведенческих диаграмм умеют в той или иной степени отражать возможность параллельного выполнения различных операций, для чего предусмотрены элементы слияния и разветвления потока управления. Если модели поведения процессов программной системы следует рассматривать как приближенное описание поведения, то для формального, т. е. максимально точного и специфицированного описания поведенческого аспекта используются алгоритмы.

Наиболее развиты средства моделирования поведения систем в языке UML, где предусмотрено несколько различных средств (диаграмм) для описания поведения:

– *диаграммы прецедентов* описывают организацию поведения системы;

– *диаграммы последовательностей* акцентируют внимание на временной упорядоченности сообщений;

– *диаграммы кооперации* сфокусированы на структурной организации объектов, посылающих и получающих сообщения;

– *диаграммы состояний* описывают изменение состояния системы как конечного автомата в ответ на события;

– *диаграммы деятельности* демонстрируют передачу управления от одной деятельности к другой.

Часто поведение объектов разумно рассматривать в терминах их жизненного цикла, т. е. последовательностью изменений состояния объекта. Для такого описания используются модели, которые изображаются диаграммами состояний автомата. При этом состояния конечного автомата соответствуют возможным состояниям объекта, т. е. различным наборам значений атрибутов объекта, а переходы происходят в результате возникновения различных событий.

Когда программа представляет собой простую последовательность операторов (линейная программа), то операторы в программе исполняются по очереди в естественном порядке. При этом поток управления просто совпадает с заданной линейной последовательностью. Однако обычно возникают отклонения от линейного порядка следования операторов потому, что:

– на поток управления влияют специальные управляющие конструкции: операторы перехода, условные операторы, операторы цикла и др.;

– при вызове подпрограммы выполнение операторов программы приостанавливается, управление передается в подпрограмму, т. е. в поток управления попадают операторы подпрограммы. При выходе из подпрограммы выполнение операторов программы возобновляется.

Аналогичная ситуация возникает при синхронном обмене сообщениями между взаимодействующими в программе объектами.

Если поток управления представляет собой последовательность элементарных шагов, требуемых для выполнения отдельного метода или реализации сложного варианта использования, то для его описания удобно использовать диаграммы деятельности.

Помимо потока управления также используется поток данных как отражение связи выходных результатов одних действий с входными аргументами других действий. В языке UML [17, 21, 22] для этих целей предусмотрены *диаграммы деятельности и взаимодействия*.

Взаимодействие нескольких программных объектов между собой путем обмена временной последовательностью сообщений описывается такими диаграммами UML, как диаграммы взаимодействия в одной из двух практически эквивалентных форм (*диаграммой коммуникации и диаграммой последовательности*).

Для описания алгоритмов существует несколько способов, при этом каждый из них имеет свои достоинства и недостатки и предназначен для использования в различных ситуациях. Так, при описании алгоритмов, предназначенных для выполнения компьютером, используются языки программирования, а для описания алгоритмов, выполняемых человеком, применяются другие способы.

Выбор средств и методов для записи алгоритма зависит, прежде всего, от назначения самого алгоритма, а также от того, кто будет его исполнителем. Алгоритм описывается в виде последовательности шагов. На каждом шаге определяется состав выполняемых действий и направление дальнейших вычислений. При этом, если на текущем шаге не указывается, какой шаг должен выполняться следующим, то осуществляется переход к следующему шагу.

Алгоритмы могут быть записаны в виде:

– словесных правил;

– псевдокода;

– блок-схем;

– программ и других форм.

Словесные правила представляют собой по существу обычный язык, но с тщательным отбором слов и фраз, не допускающих лишних слов, двусмысленностей и повторений. Дополняется язык обычными математическими обозначениями и некоторыми специальными соглашениями.

Наиболее наглядным и простым способом описания алгоритма является графический способ описания алгоритма, т. е. это способ представления поведения с помощью общепринятых графических фигур, называемых *блок-схемами*, каждая из которых описывает один или несколько шагов алгоритма. Внутри блока записывается описание команд или условий. Для указания последовательности выполнения блоков используют линии связи. Последовательность блоков и линий образуют блок-схему алгоритма.

Алгоритм, записанный непосредственно на языке программирования, называется *программой*.

Псевдокод – это компактный и часто неформальный язык описания алгоритмов, в котором используются ключевые слова языков программирования, но опускаются несущественные подробности и специфический синтаксис. Обычно отсутствуют детали, несущественные для понимания алгоритма человеком, такие как описания переменных, системно-зависимый код и подпрограммы. Главная цель использования псевдокода – обеспечить понимание алгоритма человеком, сделать описание более воспринимаемым, чем исходный код на языке программирования.

4. АРХИТЕКТУРЫ РАСПРЕДЕЛЕННЫХ ПРОГРАММНЫХ СИСТЕМ

Основным критерием при определении понятия распределенной автоматизированной системы (РАС) является разделение ее функций между несколькими компьютерами. При таком подходе к распределенной можно отнести любую программную систему, обработка данных в которой разделена между двумя и более компьютерами. Э. Таненбаум [23] дает более узкое и точное определение, согласно которому распределенная система определяется как набор соединенных каналами связи независимых компьютеров, которые с точки зрения пользователя выглядят единым целым программным комплексом.

В таких системах функции одного уровня программного приложения могут быть разнесены между несколькими компьютерами, и в то же время программное обеспечение, установленное на одном компьютере, может отвечать за выполнение функций, относящихся к разным уровням. Поэтому подход к определению распределенной системы, считающей ее простой совокупностью компьютеров, является достаточно условным. Для более точного описания и реализации распределенных систем используется понятие программной компоненты. Под ней понимается единица программного обеспечения, исполняемая на одном компьютере в пределах одного процесса и предоставляющая определенный набор сервисов, которые используются через внешний интерфейс другими компонентами, выполняющимися как на том же компьютере, так и на удаленных компьютерах. Совокупность компонент пользовательского интерфейса предоставляют интегрированный сервис конечному пользователю.

4.1. Характеристики современных распределенных систем

Чтобы достигнуть основной цели своего функционирования – обеспечения эффективного выполнения запросов пользователя – распределенная система должна удовлетворять ряду необходимых требований, который можно представить приведенным нижеследующим набором характеристик [23, 24].

1. *Открытость.* Внутри РАС все протоколы взаимодействия компонентов в идеале должны быть основаны на общедоступных стандартах, что позволяет использовать для их создания универсальные средства разработки и операционные системы. При этом каждый компонент должен содержать точную и полную спецификацию своих сервисов.

2. *Масштабируемость*. Наиболее важный аспект этой характеристики состоит в потенциальной возможности добавления в РАС новых компонентов и компьютеров для увеличения производительности системы и обеспечения балансировки нагрузки на серверы системы. К масштабированию относят также обеспечение эффективного распределения ресурсов сервера, обслуживающего запросы клиентов.

3. *Поддержание логической целостности данных.* Данное свойство предполагает, что запрос пользователя в распределенной системе должен корректно выполняться целиком или не выполняться вообще. Должны быть исключены ситуации, когда часть компонентов системы корректно обработали поступивший запрос, а часть – нет.

4. *Устойчивость.* Под этой характеристикой понимается возможность автоматического перераспределения функций внутри системы при выходе из строя одного из узлов, что обеспечивается дублированием несколькими компьютерами одних и тех же функций. В идеале это означает практически полное отсутствие ситуации, когда выход из строя одного любого компьютера приводило бы к невозможности обслужить пользовательский запрос.

5. *Безопасность.* Данные, передаваемые между компонентами, должны быть защищены как от искажения, так и от просмотра третьими сторонами, а функции системы должны использоваться авторизированными на это компонентами или пользователями.

6. *Эффективность*. Под свойством понимается минимизация накладных расходов, связанных с распределенным характером системы. Эффективность может противоречить безопасности, открытости и надежности системы, поэтому требование эффективности обычно является наименее приоритетным. Так, на поддержку логической целостности данных в распределенной системе можно потратить значительные ресурсы времени и памяти, но система с недостоверными данными вряд ли вообще будет нужна пользователям.

4.2. Проблемы проектирования распределенных систем

В связи с тем, что распределенная система в сравнении с локальной имеет более сложный характер взаимодействия программных компонентов с технической, информационной и программной сторон, процессы их проектирования и эксплуатации имеют ряд недостатков, к которым относятся следующие.

1. *Сложность.* РАС имеют более сложную организацию по сравнению с централизованными системами, что обусловливает определенные трудности при понимании, оценивании и тестировании их свойств. В частности, производительность системы зависит не от скорости работы отдельного процессора, а от полосы пропускания сети и совокупной скорости обработки информации разными процессорами. На производительность системы может радикально повлиять перемещение ресурсов из одной части в другую.

2. *Безопасность.* Поскольку доступ к системе можно получить с разных компьютеров системы, то сообщения в сети могут просматриваться и перехватываться. Поэтому в РАС намного сложнее поддерживать информационную безопасность.

3. *Управляемость.* Так как система может состоять из разнотипных компьютеров с разными версиями операционных систем, то ошибки, возникшие на одной машине, могут распространиться на другие с непредсказуемыми последствиями. Поэтому для того, чтобы управлять и поддерживать систему в рабочем состоянии, требуется значительно больше усилий.

4. *Непредсказуемость.* При возникновении различных событий реакция РАС может быть непредсказуемой, так как она зависит и от полной загрузки системы, и от ее организации и сетевой нагрузки. Причем эти параметры могут периодически изменяться, поэтому и время, потраченное на выполнение запроса пользователя, может существенно различаться.

Перечисленные трудности вызывают и соответствующие проблемы проектирования РАС, которые можно разделить на следующие группы.

1. *Идентификация ресурсов*. Так как ресурсы распределенной системы размещаются на разных компьютерах, то систему имен ресурсов следует продумать так, чтобы пользователи без труда открывали и ссылались на необходимые им ресурсы. Примером является система унифицированного указателя ресурсов URL, которая определяет адреса WEB-страниц.

Если не иметь легко воспринимаемую и универсальную систему идентификации, то большая часть ресурсов окажется недоступной.

2. *Коммуникации*. Наиболее эффективным способом организации взаимодействия между компьютерами для большинства распределенных систем является Internet и реализация протоколов TCP/IP. Однако в случаях, когда на производительность, надежность и другие характеристик накладываются специальные требования, можно воспользоваться и другими, альтернативными способами системных коммуникаций.

3. *Качество системного сервиса*. Под ним обычно понимается производительность, работоспособность и надежность сервиса, предлагаемого системой. Качество сервиса в РАС определяется целым рядом дополнительных факторов, таких как распределение системных процессов, системные и аппаратные средства, возможности адаптации системы.

4. *Архитектура ПО РАС*. Определяет распределение системных функций по компонентам системы, а также развертывание этих компонентов по процессорам. Для поддержки высокого качества системного сервиса выбор правильной архитектуры может оказаться решающим фактором.

4.3. Основные типы архитектур РАС

Архитектура РАС строится на основании принципов разделения, которые можно разделить на два типа: функциональное и естественное. *Функциональное разделение* основывается на выделении специфичных задач, решаемых узлами, например клиент – сервер, хост – терминал, сборка данных – обработка данных – предоставление данных [23, 24]. При *естественном разделении* учитываются требования, определяемые структурой самой задачи автоматизации, например обслуживание сети супермаркетов, сеть для поддержки коллективной работы и др.

Основными мотивами разделения являются:

1) *обмен информацией между системами*, при котором распределение нагрузки и балансировку загрузкой процессоров системы стремятся осуществлять так, чтобы оптимизировать общую загрузку системы;

2) *усиление мощности*, когда различные узлы должны работать над одной задачей так, чтобы система, содержащая набор процессоров, по мощности могла приближаться к суперкомпьютеру;

3) *физический мотив*, при котором система строится в предположении, что узлы физически разделены исходя из требования к надежности и устойчивости к сбоям;

4) *экономические мотивы*, так как набор дешевых чипов может обеспечить лучшие показатели отношения цена/производительность, чем мэйнфрэйм;

5) *специализация компонентов*, при которой происходит их упрощение и удешевление.

В результате разделения формируется программная архитектура PAC, которая может быть отнесена к одному из двух основных типов:

– *архитектура клиент – сервер* как модель, представленная набором системных сервисов, которые предоставляются серверами клиентам, при этом по назначению и характеру использования серверы и клиенты значительно отличаются друг от друга;

– *архитектура распределенных объектов* как модель, в которой между серверами и клиентами не делается никаких различий, такая система представляется как набор взаимодействующих объектов, место расположения которых не имеет особого значения.

В распределенных системах разные компоненты могут быть реализованы на различных языках программирования и выполняться на разных типах процессоров. Модели данных, способы представления информации и протоколы взаимодействия также необязательно однотипны в PAC. Поэтому в них должен присутствовать такой слой ПО, который был бы способен управлять разнотипными частями при гарантии системного взаимодействия и обмена данными между ними. Это *промежуточное ПО* должно выполнять роль посредника между разными частями распределенных компонентов системы.

PAC проектируются на основе объектно-ориентированного подхода, конструируются из слабо интегрированных частей, каждая из которых может непосредственно взаимодействовать как с пользователем, так и с другими частями системы. Такие компоненты должны по возможности реагировать и на независимые события. Программные компоненты, разработанные на основе таких принципов, являются естественными частями PAC.

Архитектура распределенных объектов представляется наиболее общим и универсальным подходом в проектировании PAC, при котором

различия между клиентом и сервером стираются. Компонентами такой системы являются объекты, реализующие сервисы. При этом не делается различий между клиентом (пользователем сервиса) и сервером (поставщиком сервиса).

Основным недостатком таких архитектур является то, что они сложнее в проектировании, чем клиент-серверные системы, имеющие более естественный подход к разработке РА, так как отражают естественность общественных взаимоотношений, когда одни люди пользуются услугами других, специализацией которых является предоставление конкретных услуг. Кроме того, индустрия ПО пока не накопила достаточно опыта в проектировании и разработке крупных модульных систем распределенных объектов.

4.4. Клиент-серверные архитектуры

Типовой моделью взаимодействия между разными рабочими станциями является модель, в которой одна из сторон, клиент, инициирует обмен данными, посылая запрос другой стороне, серверу. Сервер обрабатывает этот запрос и при необходимости отправляет ответ клиенту, содержащий нужные данные (рис. 16). Такое взаимодействие может быть как синхронным, когда клиентская программа ожидает завершения обработки своего запроса сервером, так и асинхронным, при котором клиент после отправки запроса серверу продолжает свое функционирование без ожидания ответа сервера.

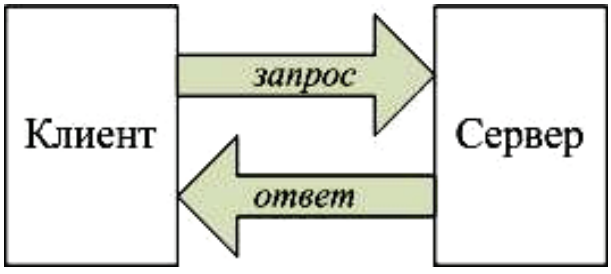

Рис. 16. Модель взаимодействия клиент – сервер

Модель клиент-серверной архитектуры может использоваться как основа для описания различных взаимодействий, которые важно знать при проектировании компонентов ПО, образующего РАС.

Логическая модель такой системы включает в себя три основных уровня: интерфейс пользователя, логику приложений и доступ к данным, как показано на рис. 17.

Рассматривая типичное распределенное приложение, которое разделено на логические уровни в соответствии с моделью, показанной на рис. 17, отметим, что пользователь взаимодействует с программной системой через интерфейс пользователя, база данных хранит используемые данные, описывающие предметную область, а уровень логики приложения реализует алгоритмы процессов обработки информации.

Рис. 17. Логические уровни
распределенного приложения

Обычно разных пользователей системы интересует возможность обращения к одним и тем же данным, поэтому наиболее простым разделением функций между несколькими компьютерами будет отделение логических уровней системы от той ее части, которая отвечает за доступ к данным, и клиентских частей, реализующих интерфейс пользователя.

При этом логика приложения может быть расположена либо на сервере, либо на клиентском компьютере, либо разделена между ними.

Архитектуру программных систем с разделением логики приложений между клиентом и сервером называют простой клиент-серверной или двухзвенной (рис. 18). Формально подобные системы могут считаться простейшими представителями распределенных систем. Если большая часть логики приложений располагается на клиентских машинах, то такая модель называется архитектурой «толстого клиента», а если наоборот, то это архитектура «тонкого клиента».

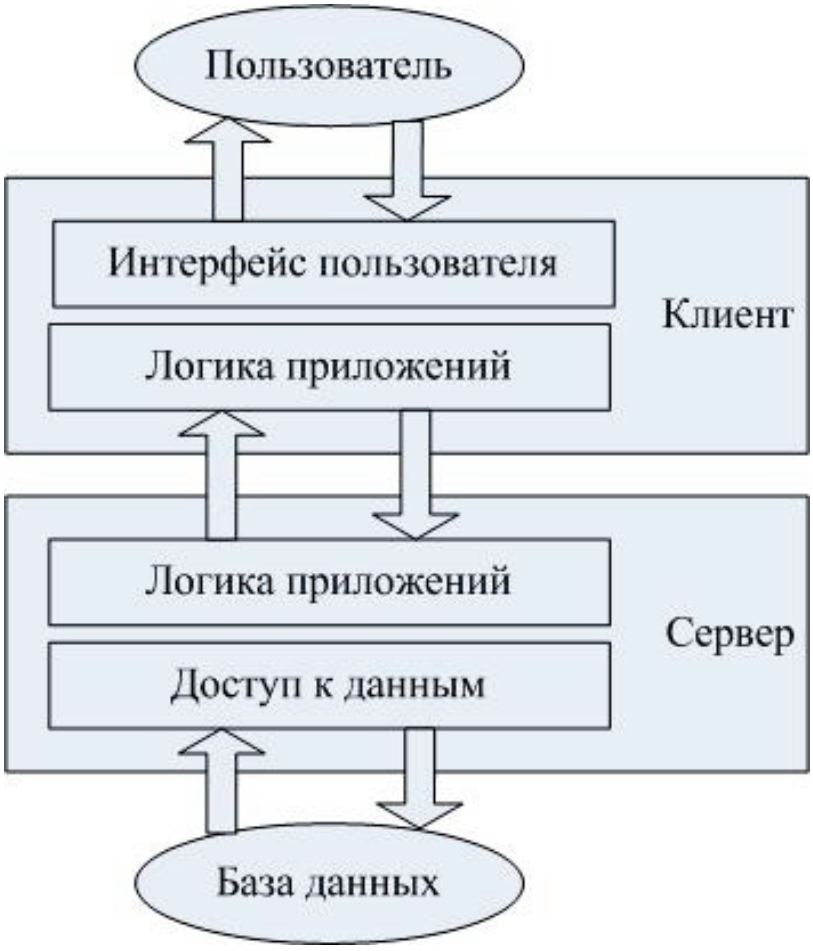

Рис. 18. Двухзвенная архитектура

Развитием двухзвенной является трехзвенная архитектура, в которой интерфейс пользователя, логика приложения и доступ к данным выделены в самостоятельные составляющие части, работающие на независимых компьютерах (рис. 19). Запрос пользователя в таких системах последовательно обрабатывается клиентской программой, сервером приложений и сервером баз данных.

Рис. 19. Трехзвенная архитектура

Помимо рассмотренных типов архитектуры имеются и более сложные системы, содержащие более трех звеньев. В таких системах автоматизации логикой приложений может быть распределена между несколькими серверными частями системы, каждая из которых может выполняться на отдельном компьютере. Например, реализация логики приложения системы розничных продаж торгового предприятия должна использовать запросы к логике приложения сторонних фирм – поставщиков товаров, системы электронных платежей или банки, которые предоставляют потребительские кредиты (рис. 20).

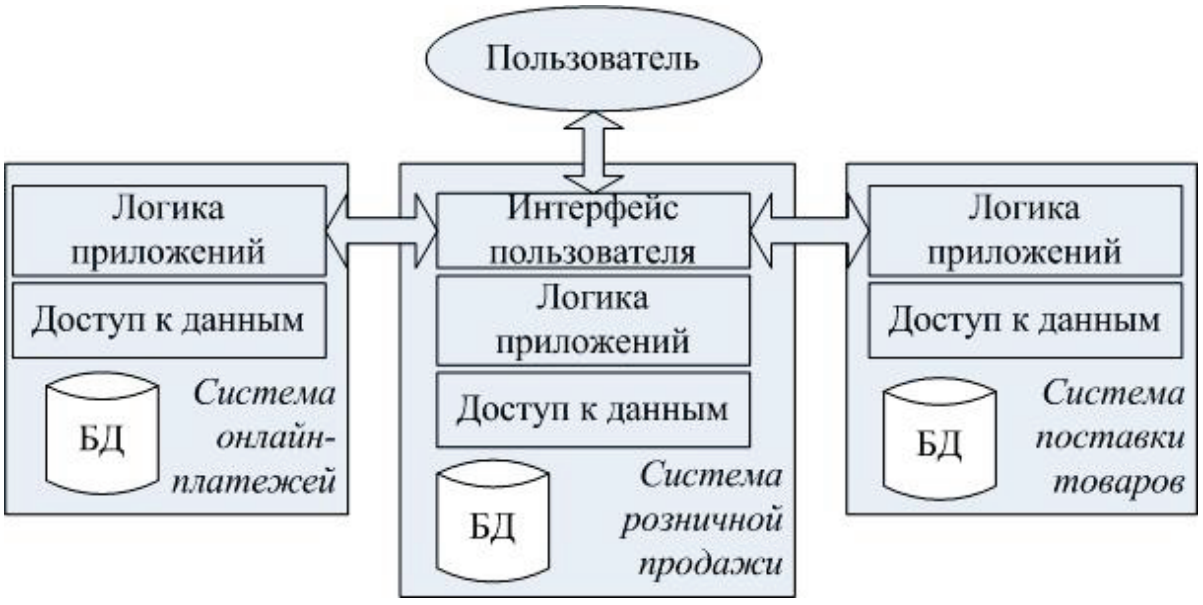

Рис. 20. Распределенная система розничных продаж

4.5. Технологии проектирования РАС

В инфраструктуру системной поддержки проектирования РАС входят:

– язык описания интерфейсов;

– транслятор с этого языка;

– библиотеки, реализующие функциональность, необходимую для удаленного вызова процедуры.

Часть ее используется при разработке программ, использующих удаленный вызов процедур, другая часть необходима для реализации во время выполнения такого вызова. Аналогично работают и другие виды взаимодействия распределенных компонентов.

Системная поддержка РАС может иметь следующие формы организации системного взаимодействия.

1. *Системы на базе удаленного вызова процедур*. Являются наиболее общим способом системного взаимодействия. Они содержат программы для прозрачного преобразования обычных вызовов процедур в удаленные. Такие системы лежат в основе почти всех других форм системного программного обеспечения, включая сетевые службы.

2. *Транзакционные мониторы*, которые относятся к наиболее надежной и стабильной технологии интеграции прикладных распределенных систем. Они могут рассматриваться как системы удаленного вызова процедур с транзакционными расширениями.

3. *Брокеры объектов.* Появились в результате применения объектно-ориентированного подхода к системам на базе систем удаленного вызова. Такие системы более развиты, чем системы удаленного вызова, хотя и мало от них отличаются. Наиболее распространены брокеры объектов, построенные на основе подхода CORBA, предложенного группой OMG при том, что основная часть их функциональности уже была реализована в транзакционных мониторах. Они были переведены на объектно-ориентированные языки программирования. В настоящее время на смену технологии CORBA пришли стандартизованные протоколы WEB-сервисов, такие как XML, WSDL, SOAP и др.

4. *Системы обмена сообщениями*. При сближении транзакционных мониторов и брокеров объектов возникли новые системы, которые стали называться мониторами объектов. В них синхронное взаимодействие ста-

ло не обязательным. Основой этих систем послужили транзакционные мониторы с сохранными очередями. В настоящее время системы обмена сообщениями представляют собой платформы, обеспечивающие транзакционный доступ к очередям, сами сохранные очереди и большое количество различных примитивов для чтения и записи в локальные и удаленные очереди.

5. *Брокеры сообщений.* Представляют собой обособленную форму систем на основе обмена сообщениями, в которые введены дополнительные возможности преобразования и фильтрации сообщений при прохождении через систему очередей. На основе анализа содержания сообщений такие системы могут динамически выбирать их получателей.

5. ОБЪЕКТНОЕ МОДЕЛИРОВАНИЕ И ПРОГРАММИРОВАНИЕ

В начальный период развития компьютерной техники, когда мощность компьютеров была недостаточно высокой, к исследованию модельного мира применялись естественные подходы к разработке программных систем, которые имели реляционный (процедурный, функциональный и другие) характер. Их сущность состоит в систематическом использовании декомпозиции функций или процессов для описания и построения структурных моделей. При этом сами моделируемые объекты представлялись фрагментарно в том объеме, который необходим для выполнения этих функций, и в форме, удобной для реализации этих функций. Это обеспечивает эффективную реализация требуемых функций, однако не создает цельного и адекватного представления моделируемого пространства.

В конечном счете и пользователю, и разработчику ПО необходимо наблюдение за изменением состояний объектов модельного мира в результате их взаимодействий, что требует использования комплексных информационных моделей таких объектов и создания программных средств, предоставляющих пользователю доступ к ним как на уровне свойств, так и на уровне поведения. Наиболее полно решению этой задачи отвечает объектный подход к разработке ПС, сущность которого состоит в систематическом использовании объектной декомпозиции при описании и построении ПС. При этом функции, выполняемые таким ПС, выражаются через отношения объектов, так что их декомпозиция существенно зависит от декомпозиции объектов, обладающих своей информационной структурой и поведением.

При объектно-ориентированном подходе структурный анализ требований к системе сводится к разработке ее модели, т. е. формальному описанию, в котором выделены основные объекты, составляющие систему, и отношения между ними [17, 18].

5.1. Объектный подход и объектная декомпозиция

Объектный подход ориентирован на описание объектов автоматизации с построением их комплексных информационно-поведенческих моделей. При этом многие процессы разработки ПС приобретают специфические объектные черты, такие как [21, 22]:

– использование новой системы понятий, позволяющих описывать объекты и их классы;

– декомпозиция объектов, являющаяся основным средством упрощения программной системы;

– использование внепрограммных абстракций для упрощения процессов разработки;

– предпочтение (приоритет) разработки структуры данных перед реализацией функций.

В объектной модели отношение между объектами обобщается в отношения между этими классами. В самом общем смысле объект – это сущность, идентифицирующая реальный предмет моделирования, а класс представляет собой описание множества однотипных объектов. При этом каждый объект обладает:

– *идентичностью*, т. е. его можно поименовать и отличать от иных объектов;

– *состоянием*, характеризуемым некоторой совокупностью данных;

– *поведением*, показывающим, что с ним можно делать или что он сам может делать с другими объектами.

Информационные характеристики класса называют атрибутами, а сервисы, предоставляемые объектом класса им самим или по требованию другого объекта, называются *операциями*. Отношения между двумя и более объектами называют *связями*, а их обобщение (отношение между классами) обычно называют ассоциациями. *Ассоциации* определяют допустимые связи между объектами. Различают следующие виды ассоциаций:

– взаимодействия состояний объектов;

– агрегирования (структурирования) объектов;

– абстрагирования (порождения) классов.

Процесс представления предметной области задачи в виде совокупности объектов, обменивающихся сообщениями, называется объектной декомпозицией. Объектная декомпозиция, так же как и процедурная, может применяться многократно или быть многоуровневой. Это значит, что каждый объект может рассматриваться как система, которая состоит из элементов, взаимодействующих друг с другом через передачу сообщений. При многоуровневой декомпозиции на каждом уровне мы получаем объекты с более простым поведением, что позволяет разрабатывать системы повышенной сложности по частям.

5.2. Язык объектного моделирования UML

Язык UML представляет собой универсальный язык визуального моделирования, разработанный для спецификации, визуализации, системного проектирования и документирования отдельных компонент ПО и программных систем на основе объектного подхода. Он эффективен при построении концептуальных и логических визуальных моделей сложных программных систем разного назначения [7, 17]. Поддержкой языка как открытого стандарта занимается консорциум OMG. Последней версией, опубликованной в июне 2015 года, является UML 2.5.

В нем реализованы основные базовые принципы, которые могут быть применены большинством программистов и разработчиков, использующих методы объектно-ориентированного анализа и программирования. Эти базовые понятия могут комбинироваться и расширяться для того, что разработчики объектных моделей имели возможность самостоятельно разрабатывать модели сложных программных систем.

Эффективное использование языка UML базируется на понимании общих принципов моделирования сложных систем и использовании специфики проведения объектно-ориентированного анализа и программирования. В частности, одним из базовых принципов объектного моделирования систем является принцип абстрагирования. Он предусматривает включение в создаваемую модель только те аспекты программной системы, которые имеют непосредственное отношение к выполнению системой своего назначения и состава выполняемых функций. Второстепенные детали при этом не учитываются для того, чтобы не усложнять процесс исследования формируемой модели.

Следующим принципом построения объектных моделей сложных систем является *принцип многомодельности*, основанный на том, что невозможно с достаточной степенью адекватности описывать различные аспекты сложной системы с использование единственной модели. Это предполагает включение в достаточно полную модель сложной системы определенного числа взаимосвязанных представлений (*views*), каждое из которых наиболее точно и достоверно отражает заданный аспект структуры или поведения проектируемой системы.

В объектном моделировании на языке UML используется и такой важный принцип прикладного системного анализа, как принцип *иерар-*

хичности формирования моделей сложных систем. Данный принцип предусматривает представление процесса моделирования на разных уровнях абстрагирования или детализации в рамках фиксированных представлений. Исходная модель при этом имеет наиболее общее *метапредставление*, которое формируется в начале этапа проектирования и не содержит несущественных деталей и аспектов моделируемой системы.

Таким образом, процесс объектного анализа и моделирования должен реализовываться как иерархический переход от наиболее абстрактных представлений концептуального уровня к более частным и детальным видам проектирования логического и физического уровней. На каждом из этапов этого процесса данные модели последовательно пополняются новым, уточняющим набором деталей, что приводит к более адекватному отражению различных аспектов реализации сложной программной системы. Общая система моделей UML показана на рис. 21.

Рис. 21. Общая схема представлений при объектном анализе и проектировании

UML-диаграмма – это специализированный язык графического описания, предназначенный для объектного моделирования в сфере разработки различного программного обеспечения. Данный язык имеет широкий

профиль и представляет собой открытый стандарт, в котором используются различные графические обозначения, чтобы создать абстрактную модель системы. UML создавался для того, чтобы обеспечить определение, визуализацию, документирование, а также проектирование всевозможных программных систем.

Всего в языке UML определены девять типов диаграмм.

Диаграммы классов (class diagram), которые предназначены для отображения классов, интерфейсов, объектов и их кооперaций, связанных различными отношениями. При моделировании систем на основе объектно-ориентированного подхода этот тип диаграмм применяется наиболее часто. Он соответствует статическому представлению системы с точки зрения проектирования.

Диаграммы объектов (object diagram) используются для отображения объектов и связей между ними. Объекты представляют собой статические копии экземпляров сущностей, показываемых на диаграммах классов. Так же как и диаграммы классов диаграммы объектов относятся к статическому отображению системы с точки зрения проектирования или процессов, но рассчитаны на программную или макетную реализацию.

Диаграммы прецедентов (use case diagram) представляют прецеденты (варианты использования) и актеров, участвующих в их выполнении, также связанных между собой отношениями. Этот тип диаграмм относится к статическому виду системы и дает точку зрения вариантов ее использования или процессов. Диаграммы прецедентов особенно важны при концептуальном моделировании системы.

Диаграммы взаимодействия предназначены для представления способов и характера информационного взаимодействия между объектами и показывают, в частности, сообщения, с помощью которых объекты обмениваются данными. Используются два частных случая этих диаграмм: *диаграммы последовательностей* (sequence diagram), отражающие временной порядок обмена сообщениями, и *диаграммы кооперации* (collaboration diagram), на которых отображается структурная модель объектов, обменивающихся сообщениями. Эти типы диаграмм являются изоморфными, т. е. они могут свободно трансформироваться друг в друга.

Диаграммы состояний (statechart diagram) отображают автомат, включающий в свою структуру состояния, переходы, события и виды опе-

раций. Эти диаграммы относятся к динамическому представлению программной системы, особенно они необходимы при моделировании изменения состояний интерфейсов, классов или кооперации. Поведение объекта в диаграмме состояний определяется последовательностью событий, что важно для моделирования реактивных систем.

Диаграммы деятельности (activity diagram) являются частным случаем диаграмм состояний. На них представляется поток управления путем переходов от одной деятельности к другой. Этот вид диаграмм относится к динамическим видам системы и является незаменимым при моделировании процессов ее функционирования, так как описывает поток управления между объектами.

Диаграммы компонентов (component diagram) используются для представления организации связанной совокупности программных компонентов. Они относятся к статистическому представлению системы с точки зрения реализации, могут быть связаны с диаграммами классов в силу следующего обстоятельства: один компонент обычно включает реализацию одного или нескольких классов, интерфейсов или кооперации.

Диаграммы развертывания (deployment diagram) представляют конфигурацию узлов программной системы с указанием размещенных в них компонентов. Эти диаграммы относятся к статическому представлению системы и представляют точку зрения развертывания. Они тесно связаны с диаграммами компонентов, так как на узле размещается один или несколько компонентов.

5.3. Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) представляет собой метод программирования, основанный на использовании объектов в качестве основных элементов программ [22]. В объектно-ориентированных языках программирования понятие объекта реализовано в виде интегрированной совокупности свойств (данных, характеризующих данный объект), операций по их обработке (функций изменения свойств), а также событий, на которые данный объект может реагировать.

Основным фундаментальным понятием ООП является *класс*, который представляет собой типизированный шаблон, на основе которого может

быть создан конкретный исполняемый экземпляр программного объекта. Класс объединяет свойства и методы, определяя сущность и поведение объектов. Объединение данных и процедур обработки в одном объекте с использованием элементов защиты от внешнего использования получило название *инкапсуляции* и является одним из основополагающих принципов ООП. Кроме того, к важнейшим принципами ООП относятся наследование, полиморфизм и модульность.

Под *наследованием* понимают такой способ организации классов, при котором новый класс может быть создан на базе существующих, причем класс-потомок сохраняет (наследует) все свойства базового класса – родителя.

Полиморфизм – это свойство, которое означает, что наследуемые объекты содержат информацию о том, какие методы они должны использовать в случае их программного вызова в ситуации неизменности имен элементов класса при наследовании. Иными словами, это концепция, реализующая «множество методов в одном интерфейсе» в зависимости от того, в каком месте дерева иерархии классов они находятся.

Под *модульностью* понимают такую организацию объектов, при которой они заключают в себе полное определение их характеристик, причем никакие определения методов и свойств не могут располагаться вне класса, это позволяет свободно копировать и внедрять один объект в другой.

К основным современным языкам объектно-ориентированного программирования относятся C++, C#, Object Pascal, Java, Ruby, Python и др. С середины 90-х годов прошлого века объектно-ориентированные языки включаются в системы визуального программирования с применением технологии RAD. В такой системе разработки приложений интерфейсная часть программы создается в диалоговом режиме, практически без непосредственного написания программных операторов. К наиболее распространенным и популярным инструментальным системам визуального проектирования относятся:

– *Embarcadero RAD Studio* – среда быстрой разработки приложений для Microsoft Windows компании Embarcadero Technologies., которая объединяет популярные системы программирования Delphi и C++ Builder в единую интегрированную среду;

– *Microsoft Visual Studio* – среда разработки для всех платформ, поддерживаемых Windows: Windows Mobile, Windows CE, Xbox, .NET Framework, Windows Phone .NET Compact Framework и Silverlight, включающая такие компоненты поддержки различных языков, как Visual Basic.NET, Visual C++, Visual C#, Visual F#, Microsoft SQL Server;

– *Eclipse* – свободная интегрированная среда разработки модульных кросс-платформенных приложений на языке программирования Java производства компании Eclipse Foundation, которая включает в настоящее время и расширения для поддержки других языков, таких как C/C++, Fortran, Perl, PHP, Python, Ruby, 1C V8.

6. ОБЕСПЕЧЕНИЕ КАЧЕСТВА ПРОГРАММНЫХ ПРОДУКТОВ

Качество программных систем – это совокупность его черт и характеристик, которые влияют на их способность удовлетворять заданные потребности пользователя [7, 13]. Это не значит, что программные средства можно однозначно оценить некоторым фиксированным набором свойств с высокими значениями показателей качества. Во-первых, потому что показатели качества являются противоречивыми, т. е. улучшение одних показателей качества может приводить к ухудшению других. А во-вторых, ПО можно считать качественным тогда, когда гарантируется успешное его использование в течение заданного срока эксплуатации, т. е. качество необходимо оценивать комплексно, хотя отдельные показатели могут считаться неудовлетворительными.

В последние годы сформировалась комплексная система управления качеством продукции *TQM (Totaly Quality Management)*, которая концептуально близка к предшествующей более общей системе на основе стандартов ISO серии 9000. Данная система ориентирована на удовлетворение требований потребителя за счет постоянного улучшения процессов проектирования и эксплуатации на основе управления по фактическому состоянию проекта. Системный подход в процессах управления качеством поддержан рядом специализированных инструментальных средств, ориентированных на управление производством программной продукции.

Применение этого комплекса может служить основой для систем обеспечения качества программных средств, однако требуется корректировка, адаптация или исключение некоторых положений стандартов применительно к принципиальным особенностям технологий и характеристик этого вида продукции. Кроме того, при реализации систем качества ПО необходимо привлечение ряда стандартов, формально не относящихся к этой серии и регламентирующих показатели качества, жизненный цикл, верификацию и тестирование, испытания, документирование и другие особенности жизненного цикла программ.

Для управления качеством необходим постоянный контроль качества разрабатываемого ПО через метрики качества, такие как плотность дефектов, размер переделок, среднее время между отказами и др., а также контроль качества отдельных стадий процессов проектирования, составляющих целостный процесс создания программной системы.

У каждого программного проекта есть своя специфика, требующая и различного набора методов обеспечения качества [11]. Так, особо критичные проекты могут требовать тщательнейшего тестирования всех тест-кейсов (наборов условий, при которых тестировщик будет определять, удовлетворяется ли заранее определённое требование). В проектах с высокой трудоемкостью тестирования больше внимания следует уделять инспекциям. Инновационные программные проекты нуждаются в тщательном предварительном прототипировании, а критичные к ресурсам проектируемые системы потребуют более строгого стресс-тестирования и т. д.

6.1. Проблемы надежности и качества программных систем

В настоящее время ни разработчики программных продуктов, ни пользователи не могут абсолютно обоснованно сравнивать качество существующих программных продуктов по экономическим критериям и оценивать, насколько один программный продукт эффективнее другого в эксплуатационных расходах, производительности, затратах живого труда и машинного времени.

Можно выделить следующие существующие проблемы в разработке программного обеспечения:

– существующее в ряде случаев несоответствие процессов разработки ПО международным стандартам;

– наличие ошибок в CASE-инструментах, используемых для разработки программных продуктов;

– сжатые сроки выполнения проекта;

– недостаточно опытные разработчики;

– плохая организация процессов разработки ПО;

– недопонимание той функциональности программы, которую желает видеть заказчик.

Известно большое количество показателей, используемых для характеристики качества программных систем, а также сравнительной оценки их потребительской ценности. По мнению экспертов, наиболее важными из них являются следующие показатели:

– *функциональная полнота*;

– *завершенность* разработки;

– *быстродействие* как затраты времени на решение задачи пользователя;

– уровень требований к *комплексу технических средств* (занимаемый объем памяти, быстродействие процессора, свободное пространство на диске);

– *степень и простота настройки* на техническую среду, сетевые и периферийные устройства;

– *стоимость* установки, обучения и обслуживания;

– *комплексность* решения задачи;

– *возможность перенастройки* на новые условия применения, например, в связи с изменением законодательства, реструктуризацией фирмы и т. д.;

– возможность *работы в сети*;

– *качество помощи* пользователю в процессе работы, например наличие ситуативной, контекстно-зависимой и гипертекстовой помощи с оглавлением;

– требования к уровню *квалификации пользователя*;

– *трудоемкость* освоения и внедрения;

– качество *пользовательского интерфейса*.

Важной задачей программной инженерии является контроль качества кода, которое невозможно контролировать без числовых показателей. Для формирования объективного представления о программном коде необходимо использовать взаимоувязанный набор метрик, которые должны отражать целостное представление о качестве программного кода. Качественный код характеризуется следующими свойствами:

– не наделен избыточной сложностью;

– не имеет избыточной связности составных элементов системы;

– достаточно хорошо структурирован;

– имеет обоснованные пропорции для объема, т. е. функциональные модули не должны быть слишком большими или маленькими.

Чтобы проанализировать качество программного кода, потребуется измерить его сложность, связность, структурированность и объемные характеристики. После того как каждая метрика будет вычислена, необходимо объединить эти показатели в группу и получить обобщенный результат для каждой группы. При этом могут использоваться различные математические методы и подходы.

6.2. Тестирование и рефакторинг программного кода

Тестирование кода позволяет на протяжении всего жизненного цикла программной системы гарантировать соответствие всех атрибутов программных проектов заданным параметрам качества [7, 13, 20]. Основная цель тестирования состоит в определении отклонений при реализации функциональных требований, степени их значимости, обнаружении ошибок в ходе выполнения программ и своевременном их исправлении.

На протяжении всего процесса разработки ПО необходимо применять комплексные методы проверки качества кода, включающие различные типы тестирования. При этом нужно гарантировать соответствие заданным показателям качества всех промежуточных версий проектируемой системы. Применяются как автоматические, так и ручные тесты. Основными видами тестирования ПО являются следующие.

1. *Модульное тестирование* (тестирование программных модулей) используется для проверки правильности функционирования функций, подпрограмм и методов классов ПО. Модульные тесты создаются и реализуются проектировщиками непосредственно в процессе написания кода. Как правило, модульное тестирование применяется как для проверки самого кода приложения, так и для проверки функционирования БД.

2. *Исследовательское тестирование* предназначается для проверки качества кода в случае, когда тестировщик пытается интуитивно исследовать возможности программной системы и обнаружить и зафиксировать неизвестные ошибки, не имея заранее определенных тестовых сценариев.

3. *Интеграционное тестирование* применяется для проверки правильности совместного взаимодействия компонентов программного продукта.

4. *Функциональное тестирование* предназначено для проверки конкретных функциональных требований к ПО и осуществляется в случае добавления к системе новых функций.

5. *Нагрузочное тестирование* используется для проверки работоспособности программной системы при максимальной (предельной) входной нагрузке.

6. *Регрессионное тестирование* применяется в случае внесения изменений в ПО для того, чтобы проверить корректность работы тех компо-

нентов системы, которые потенциально могут взаимодействовать с измененным компонентом.

7. *Комплексное тестирование* используется для проверки полноты всех функциональных и нефункциональных требований программной системы.

8. *Приемочное тестирование* является завершающим функциональным испытанием, которое должны подтвердить, что созданная программная система соответствует требованиям и ожиданиям пользователей и заказчиков. Приемочные тесты пишутся независимыми специалистами по контролю качества и тестировщиками.

Современные инструментальные среды разработки приложений, такие как *MS VisualStudio* последних версий, предоставляют разработчикам программного обеспечения возможности создавать модульные и нагрузочные тесты, а также тесты для проверки пользовательского интерфейса.

Для этого используются шаблоны тестовых проектов, к которым можно отнести следующие:

– макет модульного теста, который позволяет создавать тесты проверки модулей в процессе проектирования;

– макет с WEB-тестами производительности и нагрузочными тестами;

– макет с кодированными тестами пользовательского интерфейса.

Кроме того, в среду разработки вводится инструментарий тестировщика, который предназначен для управления процессом тестирования ПО, включая планирование, тестирование, мониторинг и контроль.

Примером такого инструментария является подсистема *Microsoft Test Manager* в среде разработки в MS VisualStudio. С помощью таких средств специалисты по тестированию могут разрабатывать планы и управлять процессом тестирования.

При формировании плана тестирования в него можно добавлять как наборы тестов и тестовые случаи, так и конфигурации, необходимые для тестирования. Конфигурации применяются для определения среды, в которой будут реализовываться сформированные наборы тестов. При этом, как правило, имеется возможность выполнять тестирование как в ручном, так и автоматическом режиме. Результаты тестирования сохраняются в базе данных, что позволяет их анализировать и создавать различные отчеты. Ошибки, выявляемые в процессе тестирования, регистрируются, до-

кументируются и передаются разработчикам для устранения. При внесении изменений в программный код возникает необходимость в создании регрессионных тестов. При этом план регрессионного тестирования может формироваться автоматически, выявляя, какие тесты должны быть повторно выполнены.

Как уже отмечалось, качество программного кода во многом зависит от того, насколько трудно или легко вносить изменения в код, а также насколько он доступен для понимания и осмысления. При этом разработанное приложение может успешно выполнять требуемые функции, но иметь проблемы с внесением изменений или пониманием созданного кода. Такое ПО нельзя назвать качественным, так как на этапе сопровождения могут возникнуть проблемы с его модификацией при изменении пользовательских требований.

Одним из наиболее эффективных методов улучшения качества кода программных приложений является *рефакторинг*. Согласно определению М. Фаулера [25], он представляет собой процесс изменения программной системы так, чтобы её внешнее поведение не изменялось, а внутренняя структура улучшалась. Это означает, что для создания качественного программного продукта в процессе проектирования необходимо обеспечивать не только выполнение функциональных требований, но и нефункциональных, таких как удобство сопровождения, предполагающее возможность и простоту внесения изменений в код программы, а также обеспечение легкости его понимания.

Некачественное оформление (дизайн) кода можно определить по целому ряду признаков, к которым можно отнести следующие:

– *жесткость*, характеристику программы, определяющую степень сложности внесения изменений в код;

– *хрупкость*, свойство ПО повреждаться в нескольких местах при внесении единственного изменения;

– *косность*, характеристику того, что код содержит структуры или фрагменты, которые могли бы оказаться полезными в других программных системах, но усилия и риски, сопряженные с попыткой отделить эти части от оригинальной системы, слишком велики;

– *ненужная сложность*, свойство, которое характеризует факт наличия в программе элементов, которые не используются в текущий момент;

– *ненужные повторения*, характеристику, показывающую наличие повторяющихся фрагментов кода;

– *непрозрачность*, свойство, оценивающее трудность кода для понимания.

Для того чтобы создать качественное оформление текста программы, следует применять некоторые принципы и паттерны проектирования [13, 26]. К таким принципам относятся:

– принцип *единственной обязанности*, определяющий то, что у класса должна быть только одна причина для изменения. По этой причине классу целесообразно поручать только одну обязанность;

– принцип *открытости/закрытости*, устанавливающий необходимость программных элементов (классов, модулей, функций) быть открытыми для расширения, но закрытыми для модификации;

– принцип *подстановки*, определяющий наличие возможности вместо базового класса подставлять любой его подтип;

– принцип *инверсии зависимостей*, базирующийся на следующих положениях:

- модули верхнего уровня не могут зависеть от модулей нижнего уровня, но и те и другие должны зависеть от абстракций;
- абстракции не должны зависеть от деталей, а детали, наоборот, должны зависеть от абстракций;

– принцип *разделения интерфейсов*, устанавливающий обязанность клиентов знать только об абстрактных интерфейсах, обладающих свойством сцепления. Сцепление модулей является мерой взаимозависимости модулей по данным и определяется как способом передачи данных между модулями, так и свойствами передаваемых данных.

Паттерны проектирования предлагают универсальные, проверенные практикой решения. Среди обширного перечня различных паттернов следует выделять и использовать те, которые рекомендуется применять при гибкой разработке программного обеспечения. При этом использование паттернов при разработке позволяет разрабатывать ПО, которое можно легко модифицировать и сопровождать.

Отметим, что для проведения рефакторинга следует использовать только надежные тесты, которые обеспечивают контроль реализации функциональных требований при улучшении дизайна программного кода.

6.3. Качество программного обеспечения по ISO 9126

Стандарт ISO 9126 (ГОСТ Р ИСО / МЭК 9126–93) «Информационная технология. Оценка программного продукта. Характеристики качества и руководство по их применению» является международным стандартом, определяющим оценочные характеристики качества ПО [7, 13]. Российский аналог этого стандарта – ГОСТ 28195–99 «Оценка качества программных средств».

Стандарт ISO 9126 состоит из четырех частей, в которых излагаются следующие категории:

- модель качества;
- внешние метрики;
- внутренние метрики;
- метрики качества в использовании.

Модель качества классифицирует качество ПО с шестью структурными наборами характеристик – показателей качества ПО. Эти показатели, в свою очередь, детализируются атрибутами (субхарактеристиками), как показано на рис. 22.

Рис. 22. Показатели качества по ISO 9126

К основным характеристикам, определяющим модель качества программной системы, относятся:

1) *функциональность* – соответствие функциональных возможностей ПО набору требуемого пользователем предназначения;

2) *надёжность* – способность ПО сохранять необходимый уровень качества функционирования в заданных условиях за определенный период времени;

3) *практичность* (применимость) – характеризует объемы работ, требуемых для исполнения и индивидуальной оценки такого исполнения определенным или предполагаемым кругом пользователей;

4) *эффективность* – оценивает соотношение между уровнем качества функционирования ПО и временной производительностью (скоростью работы) в сочетании с объемом используемых ресурсов;

5) *сопровождаемость* – показатель, определяемый объемом работ, требуемых для проведения конкретных изменений (модификаций) в процессах сопровождения и продолжающейся разработки;

6) *мобильность* – способность ПО к перенесению из одного аппаратно-программного окружения в другое.

В целом представление качества ПО является иерархически многоуровневым, где:

– первый уровень отражает комплекс перечисленных выше шести характеристик (показателей) качества ПО;

– второму уровню соответствуют атрибуты для каждого показателя качества, детализирующие его разные аспекты и использующиеся при оценке качества (рис. 22);

– третий уровень определен для измерения и оценки качества с помощью метрик, каждая из которых определяется как комбинация метода измерения атрибута и шкалы измерения значений атрибутов;

– четвертый уровень представляет собой оценочные элементы метрики (веса), используемые для получения количественного значения или качественной оценки конкретного атрибута показателя ПО.

Приведенные характеристики и атрибуты качества ПО используются для систематического описывания требований к нему, определяя, какие конкретные свойства программ хотят обеспечить заинтересованные сто-

роны по заданной характеристике. Эти требования должны определять следующие аспекты качества:

– что ПО должно делать, например позволять клиенту оформлять заказы и обеспечить их доставку;

– обеспечивать контроль качества строительства и контролировать проблемные места;

– насколько оно должно быть надежно, например работать 7 дней в неделю и 24 часа в сутки;

– никакие введенные пользователями данные при отказе не должны теряться;

– насколько этим ПО должно быть удобно пользоваться, например пользователь, зная название товара и имея средние навыки работы в Интернете, должен находить нужный ему товар за не более чем 1 мин;

– насколько ПО должно быть эффективно, например поддерживать обслуживание до 10000 запросов в секунду, или время отклика на запрос при максимальной загрузке не должно превышать 2 с;

– насколько удобным должно быть его сопровождение, например добавление в систему нового вида запросов не более 3 человеко-дней;

– насколько оно должно быть переносимо, например ПО должно работать на операционных системах Linux, Windows 10 и MacOS X;

– приложение должно взаимодействовать с документами в форматах MS Word и HTML.

7. ДОКУМЕНТИРОВАНИЕ ПРОГРАММНЫХ СИСТЕМ

7.1. Документация в АСОИУ

При создании сложной программной системы требуется серьезная организация процесса проектирования, неотъемлемой частью которой является создание программной документации, которая подвержена постоянным изменениям, вносимым различными специалистами [7, 27]. Проектирование и эксплуатация программных систем сопровождается документированием объектов и процессов их жизненного цикла для обеспечения возможности успешного функционирования и развития функций программных средств и баз данных на любых этапах проекта, для обеспечения удобного интерфейса между разработчиками и пользователями.

Управление документированием ПО должно непрерывно поддерживать полноту документов, их корректность и согласованность с программным продуктом. Кроме того, нужно обеспечивать возможность достоверного, формально точного взаимодействия всех участников программного проекта между собой, с создаваемым продуктом и с нормативными документами для обеспечения целенаправленного развития и совершенствования программных средств в составе АСОИУ.

Программная документация должна быть адекватна формальным требованиям, актуальному состоянию текстов, диаграмм и объектных кодов программ. Она должна проверяться, контролироваться и удостоверяться (подписываться) ответственными руководителями и заказчиками проекта. Следует иметь в виду, что ошибки и дефекты документов не менее опасны для использования программной системы, чем ошибки в структуре, интерфейсах, файлах текстов программ и в содержании данных. Поэтому к структуре, полноте, правильности и качеству программной документации нужно относиться не менее тщательно, чем к разработке и модификациям проектных схем, текстов программ и описаниям данных.

Под процессом документирования понимается совокупность действий по записи информации, произведенной при реализации процессов жизненного цикла. При этом выполняется набор операций, в которых программные документы нужно планировать, проектировать, разрабатывать, производить, редактировать и сопровождать в интересах заинтересованных лиц проекта: менеджеров, системных аналитиков, инженеров, про-

граммистов, а также пользователей программного продукта и специалистов по системному администрированию.

По назначению и ориентации на выполняемые задачи и группы пользователей программную документацию можно разделить на две основные группы:

– *технологическую документацию* процессов разработки и обеспечения жизненного цикла ПО, которая включает подробные технические описания проектных решений и готовится для специалистов, осуществляющих проектирование, разработку и сопровождение программных систем, обеспечивая возможность детального освоения, развития и корректировки программ и данных на всем жизненном цикле ПО;

– *эксплуатационную документацию* программного продукта – самого программного комплекса и результатов разработки, которая создается для пользователей, позволяя им осваивать и квалифицированно применять эти программные средства при решении своих конкретных функциональных задач.

Общая схема разработки и использования программной документации представлена на рис. 23.

Техническое задание и пояснительная записка к нему разрабатываются на этапе постановки задачи. Фактически они являются формальными выходными документами данного этапа. В них фиксируются требования к разрабатываемой программной системе. В случае, если заказчик и разработчик программы работают в разных организациях, техническое задание становится обязательной частью юридического договора. В завершающих стадиях проектирования, при тестировании программы и принятии заключения о ее работоспособности и приемке в эксплуатацию, именно техническое задание будет определять требования, подлежащие инспектированию и контролю.

На этапе разработки формируются формальное описание и тексты программных модулей (исходный код), содержащие принятые на разных этапах технического и рабочего проектирования решения. На заключительном этапе разработки формируется информационно-справочная система, включаемая в состав программного комплекса, и руководства пользователей. Текст программы используется при регистрации программы как объекта интеллектуальной собственности. При этом он полностью или

частично депонируется, т. е. сохраняется и размещается как отдельный экземпляр документа в электронной форме в глобальной информационной сети. Эта копия в дальнейшем может быть использована для установления авторства.

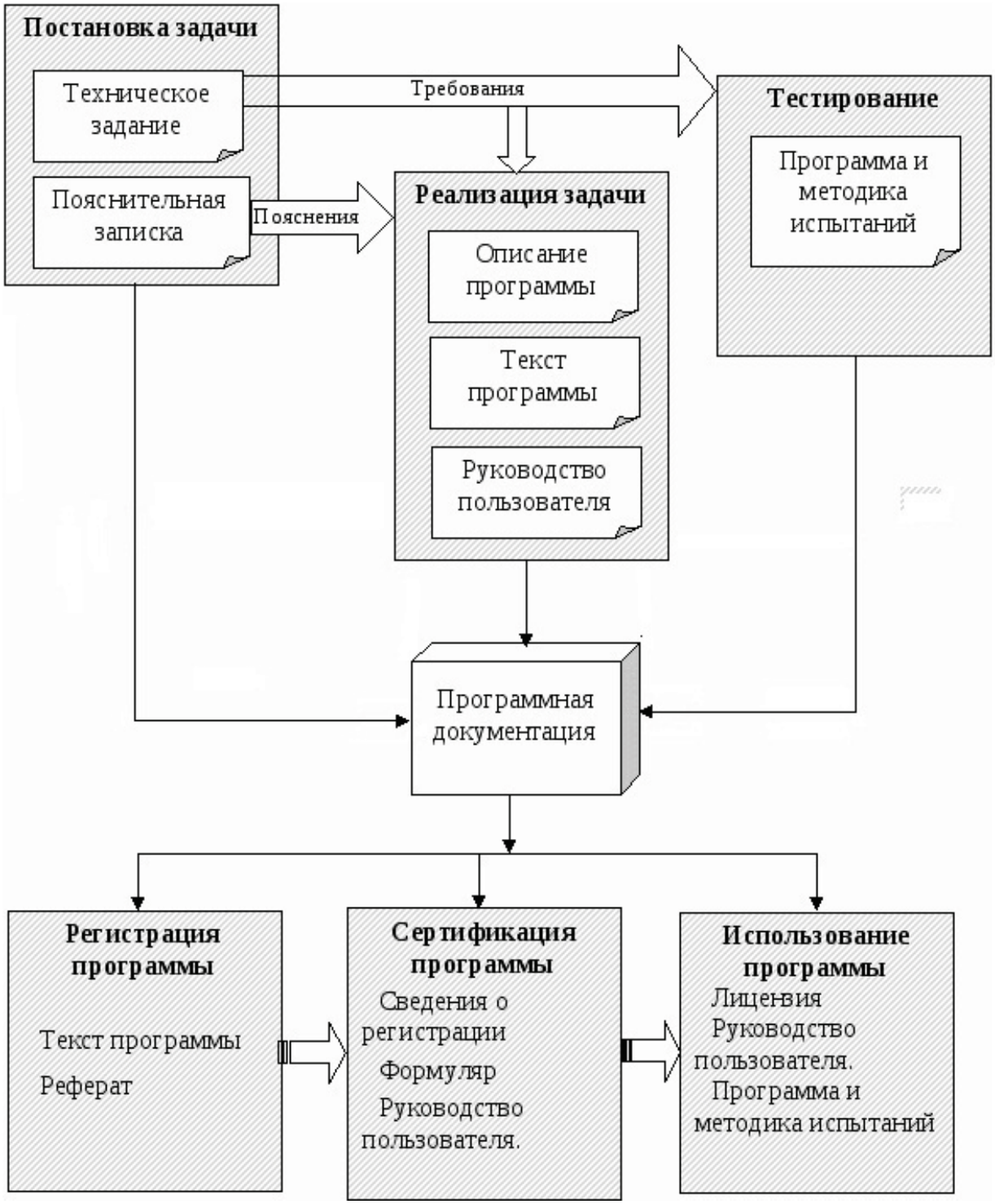

Рис. 23. Схема создания и использования программной документации

7.2. Требования к программной документации

Документация является органически составной частью программной системы и требует определенных ресурсов для ее создания и эффективного применения. Описание технического проекта, исходный код и другие программные документы в совокупности должны полностью соответствовать структуре и содержанию программной системы и быть достаточными для ее освоения, применения и изменения. В связи с этим программные документы должны быть:

– корректными и точными в изложении;

– строго адекватными функционированию программ и содержанию БД;

– изложенными систематически, структурированно, технически грамотно и понятно;

– доступными и удобными для использования специалистами и пользователями различной квалификации, рангов и назначения.

В программных документах должны качественно и полно отображаться процессы и продукты в жизненном цикле программной системы с нужной степенью достоверности информации для взаимодействия заказчиков, пользователей и разработчиков.

Важной задачей документирования является связывание объективными экономическими категориями взаимодействия разных специалистов в типовой производственной цепочке: заказчик – разработчик – программист – пользователь документации. Поэтому программный продукт как продукт потребления, а также его документация должны быть связаны со всеми процессами взаимодействия в этой цепочке совокупностью экономических и технических характеристик, использующих в той или иной степени основные экономические показатели, такие как реальные затраты материальных и финансовых ресурсов, труда и времени специалистов.

Количество, полнота содержания и сложность необходимой совокупности программных документов в первую очередь зависят от масштаба – размера проекта программной системы, поэтому целесообразно эти показатели оценивать в начале жизненного цикла. При этом необходимо детально учитывать требуемые ресурсы на создание, хранение и представление программных документов применительно к разным различным

классам и назначению программных средств (встроенных, коммерческих, технологических, административных, учебных или научных). Особое внимание в последнее время уделяется возможности совершенствования и детализации документов в условиях итерационного характера развития программных систем в течение длительного времени и в многочисленных версиях. При этом могут и должны модифицироваться и документы, отражающие состояние процессов и компонентов проектов, что предполагает гибкое и точное их изменение – сопровождение и конфигурационное управление версиями и редакциями. Эти процессы должны поддерживаться CASE-средствами документирования, адекватными тем инструментальным средам, которые применяются для непосредственной разработки комплекса программ и данных. В них должна быть включена поддержка оперативного управления и контроля процессами создания документов, регистрации и утверждения версии.

Для выполнения оценки и обоснования спецификаций требований комплекса программных документов нужны такие исходные данные, как:

– обобщенные характеристики использованных ресурсов;

– технико-экономические показатели завершенных разработок (прототипов);

– оценки влияния на программные документы функциональности и других характеристик объектов и среды разработки;

– осуществленные планы документирования, сводные перечни выполненных работ, сведения о проведенных ранее оценках и разработках программных документов;

– цели и содержание выполненных работ в процессе создания предыдущих программных комплексов и соответствующего набора различных документов;

– структура и содержание полного комплекта документов, явившегося результатом выполнения работ в рамках конкретного проекта.

Составлять спецификацию требований к программной документации следует так, чтобы все заинтересованные в проекте лица могли в ней разобраться и применять, для этого:

– названия разделов, подразделов и отдельные требования должны быть согласованными и четко соответствовать структуре документа;

– рекомендуется использовать способы визуального выделения (полужирное начертание, подчеркивание, курсив и различные шрифты) последовательно, но в разумных пределах;

– в состав документа необходимо вводить оглавление, а также алфавитный указатель для того, чтобы облегчить поиск необходимой информации;

– необходимо нумеровать все рисунки и таблицы и ссылаться на них, используя присвоенные номера.

Приведем ниже содержание некоторых, наиболее существенных для проектирования и эксплуатации программной системы, документов.

Техническое задание на проектирование программного средства должно включать:

1) общие сведения:

– титульный лист с утверждающими и согласующими подписями;

– полное наименование проекта программной системы;

– назначение и цель разработки (развития) программной системы;

– основание для выполнения и финансирования проекта;

– организация – заказчик проекта;

– организация – головной исполнитель и соисполнители проекта;

– общие сроки выполнения проекта;

2) общие технические требования, стандарты и базовые нормативные документы для выполнения проекта;

3) характеристики системы информатизации или управления:

– общие требования к системе и внешней среде;

– общие требования от системы к структуре программного средства;

– требования к программному средству в целом;

– требования к функциям и основным задачам проекта;

– требования к внешней среде проекта;

– требования к классам и характеристикам пользователей;

4) детальные спецификации требований к функциям, компонентам и эксплуатационным характеристикам программной системы:

– требования к структуре и функционированию системы;

– требования к надежности и безопасности применения;

– требования к защите информации;

– требования к стандартизации и унификации;

– требования к интерфейсам между компонентами, с внешней средой и с пользователями;

5) специальные требования к аппаратной и операционной платформам для реализации ПС;

6) требования к содержанию, оформлению и обозначениям эксплуатационной и технологической документации;

7) требования к составу и содержанию работ по внедрению программной системы в эксплуатацию;

8) этапы, сроки и график выполнения основных работ;

9) ожидаемые результаты применения системы и форма их представления;

10) порядок контроля, испытаний и приемки результатов проекта;

11) предложения по применению и развитию проекта.

Пояснительная записка к эскизному или детальному проекту программного средства должна содержать следующую информацию:

1) общие положения;

2) наименование проектируемой программной системы и наименования документов, их номера и даты утверждения, на основании которых ведется проектирование программной системы;

3) цели, назначение, области использования программного комплекса;

4) перечень организаций, участвующих в разработке АСОИУ;

5) сроки выполнения основных этапов и проекта в целом;

6) сведения об использованных при проектировании нормативных документах;

7) сведения о научно-исследовательских работах и изобретениях, примененных при разработке проекта;

8) последовательность создания функциональных компонентов программного комплекса и их объем;

9) описание процессов функционирования и использования программной системы;

10) состав функций и отдельных операций с учетом обеспечения взаимосвязи и совместимости процессов деятельности;

11) учетные данные пользователей;

12) требования к организации работ пользователей в условиях функционирования системы;

13) основные технические решения, использованные в процессе проектирования программной системы и ее компонентов;

14) структура программного комплекса в целом и по компонентам, средства и способы организации взаимодействия для информационного обмена между компонентами;

15) условия обмена информацией с другими взаимодействующими системами, обеспечение их совместимости;

16) функции комплексов задач, реализуемых системой и ее компонентами;

17) состав и структура обрабатываемой информации, ее размеры и способы организации;

18) режимы функционирования и проверки работы программной системы и ее компонентов;

19) виды машинных носителей, входные и выходные документы и сообщения с указанием последовательности обработки информации;

20) состав программных средств и инструментов проектирования, технология, языки программирования и их реализации;

21) численность, квалификация и функции обслуживающего персонала и возможных пользователей, режимы работы и порядок взаимодействия специалистов;

22) сведения о реализации заданных требований к потребительским характеристикам качества системы и ее компонентов;

23) описание организации БД, основных компонентов и процессов проектирования версий программного продукта;

24) описание логической и физической структуры БД, структуры компонентов и программной системы в целом;

25) состав, форматы и содержание данных версий компонентов и процессов программного проекта;

26) взаимосвязи между компонентами и версиями программного проекта и данными;

27) описание принятого варианта расположения компонентов программной системы и данных на конкретных машинных носителях и рабочих станциях;

28) мероприятия по подготовке системы к вводу в действие и к эксплуатации:

– мероприятия по адаптации и приведению исходной информации к виду, пригодному для программной обработки;

– мероприятия по обучению и инспекции квалификации обслуживающего персонала и основных пользователей;

29) мероприятия по организации необходимых подразделений и рабочих мест пользователей.

Руководство системного администратора программного средства содержит такие сведения:

1) описание процесса запуска системы управления и комплекса программ либо непосредственно с центрального компьютера, либо другим централизованным способом, либо через сеть;

2) описание аппаратных и программных средств, требуемых для работы системы;

3) технические характеристики используемых аппаратных устройств;

4) обзор структуры, назначения и функционирования каждого компонента комплекса программ;

5) перечень входных команд, команд доступа к программам и реакции системы на их выполнение;

6) аварийные сообщения и другие выходные данные, формируемые для контроля комплекса программ;

7) оценка типового времени выполнения основных функций программного комплекса;

8) последовательность действий, производимых для запуска системы и комплекса программ;

9) перечень требуемых библиотек поддержки обработки информации и интерфейсов системы;

10) форма и средства регистрации дефектов и ошибок, возникающих в процессе эксплуатации программной системы;

11) перечень процедур, выполняемых системным администратором при установке системы для конкретного выбранного окружения и конкретной конфигурации системы.

Общее описание *руководства пользователей* программного средства включает следующую информацию:

1) порядок действий пользователя для установки и использования программной системы;

2) краткое описание функций и характеристик программного продукта;

3) описание внешней программной среды;

4) перечень файлов, включая файлы баз данных, необходимых для применения системы;

5) порядок действий для продолжения или возобновления функционирования программного комплекса в случаях возникновения отказов, отключений и других непредвиденных ситуаций;

6) характер организации и функционирования системы с точки зрения пользователя;

7) детальное описание процедур, позволяющих фиксировать дефекты и ошибки;

8) детальные, пошаговые действия пользователя при включении системы и дальнейшей работе с ней;

9) ссылки на другие руководства системы и комплекса программ;

10) перечень и пояснение выводимых системой сообщений.

7.3. Стандартизация программной документации

Общие требования к составу и содержанию документов, поддерживающих разработку программной системы, представлены в ряде стандартов разного уровня, а также в корпоративных инструкциях и публикациях по управлению проектами. Состав и формы программных документов могут широко варьироваться в зависимости от класса и характеристик про-

ектируемой системы, а также от используемых при этом методов и технологий.

Стандарт ISO 9294 (ГОСТ Р ИСО/МЭК ТО 9294–93) представляет собой руководство по документированию ПО, предназначенное для менеджеров, отвечающих за создание программных продуктов, и используемое для оказания помощи в управлении разработкой и эффективном документировании программных проектов. Данный стандарт содержит рекомендуемые стратегии, процедуры, ресурсы и планы, которыми должны заниматься руководители проектов для эффективного создания комплектов документов ПО.

Руководители и администраторы программных проектов должны осуществлять организацию работ по документированию и поддержке этих работ, для чего необходимо управление и стимулирование персонала при проведении документирования, а также обеспечение его требуемыми ресурсами. В процессе проектирования административно-управленческому персоналу необходимо оценивать ход работы, анализировать возникающие проблемы и обеспечивать развитие процесса документирования.

Технологическая документация, описывающая процесс разработки, определяет требования, которым должна удовлетворять проектируемая система, определяет то, как проект контролируют и обеспечивают качество. Она должна включать подробное техническое описание программной системы, в том числе спецификацию программной логики, взаимосвязей, форматов хранения данных. Технологическая документация представляет собой средство связи между всеми лицами, вовлеченными в процесс разработки, так как описывает подробности решений, принимаемых в требованиях к проекту, программированию и тестированию, а также обязанности группы разработки, поясняя кто, что и когда делает с учетом роли объекта работ, документации и каждого специалиста в процессе разработки. Эта документация образует основу сопровождения и описывает в хронологическом порядке историю разработки программной системы. Если технологические документы разработки отсутствуют, неполны или устарели, то руководители и менеджеры теряют важное средство для мониторинга состояния проекта.

Эксплуатационная документация обеспечивает информацию, которая необходима для процессов эксплуатации, сопровождения, модернизации, преобразования и передачи программного изделия пользователю. Она:

– включает необходимую учебную и справочную информацию, сведения для специалистов, использующих или эксплуатирующих программную продукцию;

– облегчает системным администраторам, не разрабатывавшим программную систему, ее сопровождение и модернизацию;

– способствует продаже или приемке программной продукции.

Эксплуатационная документация должна содержать следующие инструктивные материалы:

– руководства для пользователей, выполняющих ввод данных, восстановление информации и решение служебных задач с помощью программного комплекса;

– руководства для операторов, которые применяют программный комплекс в АСОИУ;

– руководства для сопровождающих программистов и системных администраторов;

– руководства для руководителей, которые следят за использованием комплекса программ.

Типовыми эксплуатационными документами являются:

– учебные и справочные материалы;

– руководства по сопровождению программного комплекса;

– брошюры и информационные листовки, посвященные рекламе программного продукта.

Документация управления проектом содержит:

– графики выполнения работ для каждой стадии процесса разработки, а также отчеты об их изменениях;

– отчеты о согласованиях при внесении изменений в программы;

– отчеты о решениях, связанных с ходом проектирования;

– распределение обязанностей специалистов.

Руководители и менеджеры программных проектов обязаны применять стандарты качества ПО в соответствии с различными типами документов и проектов, а также устанавливать, как и когда будут достигнуты и

поддержаны нужные показатели качества. Под качеством документации понимается: качество содержания, структура представляемой информации, использование иллюстраций.

Кроме того, в проектируемых форматах нужно учитывать, будут ли документы переводиться для международного распространения.

Еще одним стандартом, который позволяет регламентировать процессы документирования ПО, является стандарт ISO 12182 (ГОСТ Р ИСО/МЭК ТО 12182–2002). В нем излагается схема классификации программных средств, охватывающая существенные характеристики и атрибуты программных систем, их виды и классы.

Классификация, определяемая этим стандартом, предназначена для определения классов конкретных программных средств, а также взаимосвязей программных процессов, продуктов и их документов со стандартами программной инженерии. В данном стандарте устанавливается схема классификации, которая помогает:

– уточнить области применения и характер использования программной системы и документации на нее;

– выбрать стандарты и шаблоны документов, применимые к конкретному проекту;

– определить классификационные характеристики новых стандартов.

Введенная в этом стандарте классификация может служить в качестве концептуальной схемы построения системы документации. При этом разработчики и заказчики могут применять собственные подходы к практическому использованию этой классификации. Данная классификация не основана на формальных требованиях к потребностям разработчиков ПО и пользователей, поэтому применение данного стандарта в практической деятельности не является обязательным.

В стандарте ISO 12207 (ГОСТ Р ИСО/МЭК 12207–2010 «Процессы жизненного цикла программных средств») менеджменту программной документации посвящен специальный раздел 7.2.1. Кроме того, в данном стандарте при описании всех процессов и работ жизненного цикла ПО отражены конкретные требования к документированию соответствующих работ.

В частности, стандартом устанавливается, что результатом успешного осуществления процесса менеджмента программной документации являются:

– разработка стратегии идентификации документации, которая реализуется в течение жизненного цикла программного продукта или услуги;

– определение стандартов, которые необходимо применять при разработке программной документации;

– определение документации, которая производится основным или вспомогательным процессом жизненного цикла ПО;

– рассмотрение и утверждение содержания и целей всей документации программного проекта;

– разработка и обеспечение доступности к документации в соответствии с определенными стандартами;

– сопровождение документации в соответствии с определенными критериями.

Стандарт ГОСТ Р 51904 «Программное обеспечение встроенных систем. Общие требования к разработке и документированию» определяет документы, которые создаются в течение всего жизненного цикла ПО и позволяют реализовать процессы и модификацию программного средства.

В этом стандарте подчеркивается, что программные документы создаются в течение всего жизненного цикла ПО для того, чтобы планировать требуемые действия, управлять ими, объяснять, определять, регистрировать выполнение требуемых действий или обеспечивать доказательство процессов. Заказчик осуществляет выбор необходимого и экономически обоснованного состава и содержания документов для потребностей конкретной разработки.

Основными характеристиками документов жизненного цикла ПО согласно ГОСТ Р 51904 являются:

– *однозначность*: информация в документе должна быть изложена в терминах, допускающих единственную интерпретацию, уточненную, если необходимо, соответствующими определениями;

– *полнота*: информация должна включать в себя необходимые, релевантные требования, а также описательные материалы, при этом используемые рисунки и таблицы должны сопровождаться необходимыми обо-

значениями, а термины и единицы измерений должны быть полностью определены;

– *верифицируемость*: информация программного документа должна быть проверяема на корректность специалистом или инструментальным средством;

– *согласованность*: информация не должна иметь противоречий внутри документа;

– *модифицируемость*: информация должна быть структурирована и иметь такой стиль, чтобы изменения могли быть выполнены в необходимом объеме, согласованно и корректно без нарушения структуры;

– *трассируемость*: информация программного документа должна быть такой, чтобы для каждого ее компонента был определен первоисточник;

– *форма документа* должна обеспечивать эффективный поиск и просмотр документов жизненного цикла ПО в процессе обслуживания системы.

Документы жизненного цикла ПО могут иметь различные формы. Например, они могут быть подготовлены как компьютерный файл, хранящийся на магнитных носителях, или как отображение на удаленном терминале. Программная документация может быть оформлена в виде отдельных документов, может объединять несколько документов или быть разделена на несколько документов.

ЗАКЛЮЧЕНИЕ

Успешное проектирование программных систем определяется большим количеством факторов, к числу которых относятся:

– знание теоретических основ современных методологий и технологий создания программных комплексов;

– правильное применение методологических основ выполнения основных видов проектных работ;

– практические навыки и опыт работы разработчиков;

– способность проектировщиков к творческому и нетривиальному подходу при принятии решений;

– грамотная организация управления проектными работами и коллективом разработчиков.

Эффект от использования этих факторов возможен только в результате системного видения последствий применения автоматизированных систем обработки информации управления в деятельности предприятий и в социальной сфере. При этом важно уметь анализировать и прогнозировать не только положительные стороны внедрения систем автоматизации в форме экономической выгоды, но и негативные, такие как сокращение потребности в живом труде, уменьшение требований к квалификации и интеллектуальным способностям работников и т. д.

Таким образом, создание программных систем имеет многогранный комплексный характер и требует от менеджеров, системных аналитиков, разработчиков архитектуры и информационных баз, программистов и тестировщиков не только знаний технических сторон проектирования, но и творческих способностей, умения документировать, отстаивать и обосновывать принимаемые решения, моральных и этических качеств.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Брауде, Эрик Дж. Технология разработки программного обеспечения / Эрик Дж. Брауде. – СПб. : Питер, 2004. – 655 с.

2. Ехлаков, Ю. П. Исследование систем управления : конспект лекций / Ю. П. Ехлаков. – Томск : ТУСУР, 1998. – 112 с.

3. Хетагуров, Я. А. Проектирование автоматизированных систем обработки информации и управления (АСОИУ) : учебник / Я. А. Хетагуров. – М. : Высш. школа, 2006. – 243 с.

4. Тарасенко, Ф. Л. Прикладной системный анализ : учеб. пособие / Ф. Л. Тарасенко. – М. : КНОРУС, 2010. – 224 с.

5. Антонов, А. В. Системный анализ : учеб. для вузов / А. В. Антонов. – М. : Высш. шк., 2004. – 454 с.

6. Орлов, С. А. Технологии разработки программного обеспечения / С. А. Орлов, Б. Я. Цилькер. – СПб. : Питер, 2012. – 608 с.

7. Белик, А. Г. Теория и технология программирования : учеб. пособие / А. Г. Белик, В. Н. Цыганенко. – Омск : Изд-во ОмГТУ, 2013. – 88 с.

8. Гвоздева, Т. В. Проектирование информационных систем : учеб. пособие / Т. В. Гвоздева, Б. А. Баллод. – Ростов н/Д : Феникс, 2009. – 508 с.

9. Боэм, Б. У. Инженерное проектирование программного обеспечения / Б. У. Боэм. – М. : Радио и связь. – 1985. – 512 с.

10. Гагарина, Л. Г. Разработка и эксплуатация АИС : учеб. пособие / Л. Г. Гагарина, Д. В. Киселев, Е. Л. Федотова. – М. : ИД «Форум» : ИНФРА-М, 2007. – 384 с.

11. Кантор, М. Управление программными проектами. Практическое руководство по разработке успешного программного обеспечения / М. Кантор. − М. : Вильямс, 2002. − 176 с.

12. Цыганенко, В. Н. CALS/CASE-технологии проектирования информационных систем : конспект лекций / В. Н. Цыганенко. – Омск : Изд-во ОмГТУ, 2007. – 88 с.

13. Крылов, Е. В. Техника разработки программ : в 2 кн. Кн. 2. Технология, надежность, качество программного обеспечения : учебник / Е. В. Крылов, В. А. Острейковский, Н. Г. Типикин. – М. : Высш. шк., 2008. – 469 с.

14. Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер. − М. : Вильямс, 2007. − 544 с.

15. Крёнке, Д. Теория и практика построения баз данных / Д. Крёнке. – СПб. : Питер, 2008. – 800 с.

16. Цыганенко, В. Н. Компьютерные системы поддержки принятия решений : конспект лекций / В. Н. Цыганенко, А. Г. Белик. – Омск : Изд-во ОмГТУ, 2007. – 96 с.

17. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон. – М. : ДМК, 2006. – 496 с.

18. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – СПб. : Питер, 2002. – 368 с.

19. Мартин, Р. С. Быстрая разработка программ. Принципы, примеры, практика / Р. С. Мартин, Дж. В. Ньюкирк, Р. С. Косс. – М. : Вильямс, 2004. – 752 с.

20. Криспин, Л. Гибкое тестирование. Практическое руководство для тестировщиков ПО и гибких команд / Л. Криспин, Д. Грегори. – М. : Вильямс, 2010. – 464 с.

21. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч. – М. : Вильямс, 2008. – 720 с.

22. Хорев, П. Б. Технология объектно-ориентированного программирования : учеб. пособие / П. Б. Хорев. – М. : Издательский центр «Академия», 2008. – 448 с.

23. Таненбаум, Э. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. Ван Стеен. – СПб. : Питер, 2003. – 877 с.

24. Лупин, С. А. Технологии параллельного программирования / С. А. Лупин, М. А. Посыпкин. – М. : ИНФРА-М, 2011. – 208 с.

25. Фаулер, М. Рефакторинг. Улучшение существующего кода / М. Фаулер. – М. : Символ-Плюс, 2003. – 432 с.

26. Белик, А. Г. Информационные технологии анализа данных : учеб. пособие / А. Г. Белик, В. Н. Цыганенко. – Омск : Изд-во ОмГТУ, 2015. – 80 с.

27. Глаголев, В. Разработка технической документации / В. Глаголев. – СПб. : Питер, 2008. – 192 с.

Учебное издание

Белик Алевтина Георгиевна

Цыганенко Валерий Николаевич

ПРОЕКТИРОВАНИЕ И АРХИТЕКТУРА ПРОГРАММНЫХ СИСТЕМ

Учебное пособие

Редактор *К. В. Муковоз*

Компьютерная верстка *О. Г. Белименко*

Сводный темплан 2016 г.
Подписано в печать 08.07.16. Формат $60\times84^{1}/_{16}$. Отпечатано на дупликаторе.
Бумага офсетная. Усл. печ. л. 6,00. Уч.-изд. л. 6,00.
Тираж 100 экз. Заказ 366.

Издательство ОмГТУ. 644050, г. Омск, пр. Мира, 11; т. 23-02-12.
Типография ОмГТУ.