Министерство образования и науки Российской федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Томский государственный университет систем управления и
радиоэлектроники» (ТУСУР)

УДК 681.3.06

Боровской И.Г., Колесникова С.И., Матолыгин А.А.

Специализированная подготовка разработчиков бизнес приложений

Учебное пособие
Издание второе, дополненное

2012

В данном пособии рассматриваются вопросы проектирования и создания программных комплексов и создания электронных образовательных ресурсов, как средства подготовки специалистов среды наукоемкого бизнеса.

Оглавление

ВВЕДЕНИЕ

Современная экономика немыслима без информации. Тысячи предприятий, миллионы налогоплательщиков, триллионы рублей, биржевые котировки, реестры акционеров – все эти информационные потоки необходимо оценить, обработать, сделать необходимые выводы, принять правильное решение.

Активная информатизация общества, автоматизация технологических процессов, широкое использование вычислительной техники, средств связи и телекоммуникаций ставит перед современным менеджером, инженером и служащим целый комплекс взаимосвязанных задач по повышению эффективности бизнес-процессов принятия и выполнения решений.

В области разработки информационных технологий и программных систем групповое проектное обучение основывается на методологии «Программная инженерия», получившей бурное развитие как в России, так и за рубежом. С позиций программной инженерии процесс создания программной системы состоит из следующих основных этапов: маркетинг и определение целевого продукта; анализ предметной области и разработка требований; проектирование; разработка; тестирование; документирование; комплексные испытания; сопровождение.

В данном пособии рассматриваются вопросы проектирования и создания программных комплексов и создания электронных образовательных ресурсов, как средства подготовки специалистов среды наукоемкого бизнеса.

ГЛАВА 1. ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

В данной главе рассматриваются общие принципы технологии программирования. Даются базовые понятия, исследуется полный «цикл жизни» программы, а также освещаются основные вопросы, касающиеся создания надежных программных систем. Несмотря на некоторую «академичность» этой части, ее можно рассматривать как набор обоснованных правил и общих рекомендаций по созданию «хороших программ», без привязки к конкретной операционной системе. Следуйте этим правилам, чтобы избежать возможных затруднений при разработке собственных программ.

1.1. Общие сведения о технологии программирования. Задачи технологии программирования

Прежде всего, необходимо выяснить значения ряда основных терминов и понятий. Например, ввести понятие компьютерной программы как формализованное описание процесса обработки данных; уяснить значение термина программная система и многое другое.

1.1.1. Базовые определения

Итак, цель программирования состоит в описании процессов обработки некоторых *данных*.

Данные – это представление фактов и идей в формализованном виде, пригодном для передачи и переработке в некоем процессе, а информация – это смысл, который придается данным при их представлении [1].

Обработка данных – это выполнение систематической последовательности действий с данными. Данные представляются и хранятся на так называемых носителях данных.

Информационная среда – это совокупность носителей данных, используемых при какой-либо обработке данных.

Состояние информационной среды – это набор данных, содержащихся в какой-либо момент в информационной среде. Тогда понятие *процесса* можно определить как последовательность сменяющих друг друга состояний некоторой информационной среды. Описать процесс означает определить последовательность состояний заданной информационной среды.

Если необходимо, чтобы по заданному описанию требуемый процесс порождался автоматически на каком-либо компьютере, необходимо, чтобы это описание было формализованным. Такое описание называется *программой*.

Однако программа должна быть понятной и человеку, так как и при разработке программ, и при их использовании приходится выяснять, какой именно процесс она порождает. Поэтому программа составляется на удобном для человека формализованном *языке программирования*, с которого она автоматически переводится на язык соответствующего компьютера с помощью другой программы, называемой *транслятором*.

Разработчику, прежде чем составить программу на удобном для него языке программирования, приходится проделывать большую подготовительную работу по уточнению постановки задачи, выбору метода ее решения, выяснению специфики применения требуемой программы, определению общей организации разрабатываемой программы и многое другое. Использование этой информации может существенно упростить задачу понимания программы человеком, поэтому весьма полезно ее как-то фиксировать в виде отдельных документов, зачастую не формализованных, а рассчитанных только для восприятия человеком.

Как правило, программы разрабатываются в расчете на то, что ими будут пользоваться люди, не только не участвующие в их разработке, но и вообще далекие от вопросов программирования. Поэтому таким пользователям для освоения программы требуется определенная дополнительная

документация. Программа или логически связанная совокупность программ на носителях данных, снабженная программной документацией, называется *программной системой* (ПС).

Программа позволяет осуществлять некоторую автоматическую обработку данных на компьютере, тогда как программная документация позволяет понять, какие функции выполняет та или иная программа из состава ПС, как подготовить исходные данные и запустить требуемую программу в процесс ее выполнения, а также, что немало важно, как трактовать получаемые результаты. Программная документация несет и еще одну функцию, а именно, она помогает разобраться в самой программе, что необходимо, например, при ее модификации.

1.1.2. Невозможность доказательства отсутствия программных ошибок

Можно считать, что итоговым продуктом технологии программирования является программная система, содержащее программы, которые выполняют набор требуемых функций. Здесь под термином «программа» будем понимать *правильно работающую программу*, т.е. программу, которая не содержит ошибок. Сложность состоит в том, что понятие *ошибки* в программе трактуется неоднозначно. Согласно работам [2,3], будем считать, что в программе имеется ошибка, если программа не выполняет того, что разумно ожидать от нее пользователю. При этом «разумное ожидание пользователя» формируется на основании документации по применению этой программы. Отсюда следует, что понятие ошибки в программе является существенно *не формальным*. В ПС программы и документация взаимно увязаны, образуют некоторую целостность. Поэтому правильнее говорить об ошибке не в программе, а в ПС в целом.

Будем считать, что в программной системе имеется ошибка, если она не выполняет того, что разумно ожидать от нее пользователю. В частности,

разновидностью ошибки в ПС является несогласованность между программами ПС и документацией по их применению. В работе [3] выделяется в отдельное понятие частный случай ошибки в ПС, когда программа не соответствует своей функциональной спецификации, т.е. описанию, составляемому на этапе, который предшествует непосредственному программированию. Такая ошибка именуется *дефектом* программы. Однако выделение такой разновидности ошибки в отдельное понятие вряд ли оправданно, так как причиной ошибки может оказаться сама функциональная спецификация, а не уже готовая программа.

Итак, мы сталкиваемся с ситуацией, когда задание на ПС сформулировано неформально, а понятие ошибки в ПС не формализовано. Отсюда следует, что *в общем случае нельзя* доказать формальными методами, т.е. с привлечением строгого математического аппарата, правильность ПС.

Конечно, каждое ПС проходит этап опытного тестирования, когда некоторому набору исходных данных соответствует определенный ожидаемый результат. Однако, как указал Дейкстра [4], нельзя показать правильность ПС с помощью тестирования – тестирование может лишь выявить наличие в ПС ошибок, *но не их отсутствие*. Поэтому понятие правильной ПС неконструктивно в том смысле, что после окончания работы над созданием ПС мы не сможем достоверно убедиться, что достигли итоговой цели.

1.1.3. Надежность программной системы

В качестве альтернативы *правильного* ПС введем термин *надежное* ПС. Под надежностью ПС будем понимать его способность безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени с достаточно большой вероятностью. При этом под отказом в ПС понимают проявление в нем ошибки [2]. Таким образом, надежное ПС не исключает наличия в нем ошибок. Важно лишь то, что эти

ошибки при практическом применении данного ПС в заданных условиях проявлялись достаточно редко. Убедиться, что ПС обладает таким свойством можно при его опытном тестировании, а также при практическом применении. Таким образом, в реальности мы можем разрабатывать лишь *надежные*, но не *правильные* ПС.

ПС может обладать различной степенью надежности. Как определить эту степень? Так же, как в любой отрасли техники – степень надежности характеризуется вероятностью работы ПС без отказа в течение определенного периода времени [2]. Однако в силу специфических особенностей ПС определение этой вероятности наталкивается на ряд трудностей по сравнению с решением аналогичной задачи в технике.

При оценке степени надежности ПС следует также учитывать последствия каждого отказа. Некоторые ошибки в ПС могут вызывать лишь некоторые неудобства при его применении, тогда как другие ошибки могут иметь катастрофические последствия, например, угрожать полному разрушению информации на магнитном носителе. Поэтому для оценки надежности ПС иногда используют дополнительные показатели, учитывающие степень вреда для пользователя при проявлении конкретного отказа. Позже мы еще вернемся к этому вопросу.

1.1.4. Технология программирования как способ создания надежных программных систем

Под технологией программирования будем понимать совокупность производственных процессов, которая приводит к созданию требуемого ПС, а также описание этой совокупности процессов. Другими словами, под технологией программирования будем понимать определенную технологию разработки программных систем, включая сюда все процессы, начиная с момента зарождения идеи этого программного средства, не упуская из виду мер, касающихся создания необходимой программной документации.

Каждый процесс этой совокупности базируется на использовании определенных средств и методов. В частности, при создании компьютерных программ можно говорить о компьютерной технологии программирования. В литературе можно найти и другие определения технологии программирования.

Нужно заметить, что существуют два других понятия, довольно тесно связанных с понятием технологии программирования – это *программная инженерия* и *методология программирования*.

Программная инженерия определяется как систематический подход к разработке, эксплуатации, сопровождению и, наконец, изъятию из обращения программных систем. Главное различие между технологией программирования и программной инженерией как дисциплинами для изучения заключается в способе рассмотрения и систематизации материала. В технологии программирования основной акцент делается на изучении процессов разработки ПС, тогда как в программной инженерии большее внимание уделяется различным методам и инструментальным средствам разработки ПС с точки зрения достижения определенных целей.

Не следует также путать технологию программирования с методологией программирования [3]. В технологии программирования все методы рассматриваются «сверху», с точки зрения организации технологических процессов, а в методологии программирования – методы рассматриваются как бы «снизу», т.е. с точки зрения основ их построения.

Возвращаясь к технологии программирования и имея в виду, что надежность является неотъемлемым атрибутом ПС, мы будем рассматривать технологию программирования как способ разработки надежных ПС. При этом будут рассматриваться все процессы разработки ПС, начиная с момента возникновения замысла о каком-либо ПС, когда в качестве конечного продукта технологии принимается надежное ПС, которое *может* содержать ошибки.

Такой взгляд на технологию программирования будет существенно влиять на организацию технологических процессов, на выбор в них методов и инструментальных средств.

1.1.5. Этапы развития технологии программирования

Кратко рассмотрим этапы развития программирования, чтобы лучше понять движущие силы и перспективы дальнейшей эволюции технологии программирования.

В 50-е годы мощность компьютеров первого поколения была невелика, а программирование для них велось преимущественно в машинном коде. Главным образом решались научно-технические задачи оборонного характера, при этом задание на программирование содержало, как правило, достаточно точную математическую постановку задачи. Например, расчет траектории полета ракеты с учетом множества сопутствующих факторов. Использовалась интуитивная технология программирования, когда сразу же приступали к составлению программы по исходному заданию, при этом часто само задание несколько раз изменялось, что сильно увеличивало время разработки программы. Минимальная документация оформлялась уже после того, как программа начинала работать. Тем не менее, именно в этот период родилась фундаментальная для технологии программирования концепция модульного программирования [5], ориентированная на преодоление трудностей программирования в машинном коде. Появились первые языки программирования высокого уровня.

В 60-е годы можно было наблюдать бурное развитие и широкое использование языков программирования высокого уровня, таких как АЛГОЛ 60, ФОРТРАН, КОБОЛ, значение которых в технологии программирования явно преувеличивалась. Надежда на то, что эти языки решат все проблемы, возникающие в процессе разработки *больших программ*, не оправдалась. В результате повышения мощности компьютеров и накопле-

ния опыта программирования на языках высокого уровня быстро росла сложность решаемых на компьютерах задач, в результате чего обнаружилась ограниченность языков, проигнорировавших модульную организацию программ. Кроме того, стало понятно, что важно не только то, на каком языке мы программируем, но и то, как мы программируем [4]. Это было уже началом серьезных размышлений над методологией и технологией программирования. Появление в компьютерах второго поколения прерываний привело к развитию мультипрограммирования и созданию больших программных систем. Широко стала использоваться коллективная разработка, которая, однако, вскрыла ряд серьезных технологических проблем.

В 70-е годы получили широкое распространение информационные системы и базы данных. К середине 70-х годов стоимость хранения одного бита информации на компьютерных носителях стала меньше, чем на традиционных носителях. Это резко повысило интерес к компьютерным системам хранения данных. Появляются языки программирования высокого уровня, например, Ада, С и другие, учитывающие требования технологии программирования в полной мере [6]. Началось интенсивное развитие технологии программирования, прежде всего, в следующих направлениях:

- обоснование и широкое внедрение нисходящей разработки и структурного программирования;
- развитие абстрактных типов данных и модульного программирования, в частности, возникновение идеи разделения спецификации и реализации модулей и использование модулей, скрывающих структуры данных;
- исследование проблем обеспечения надежности и мобильности ПС;
- создание методики управления коллективной разработкой ПС;
- появление инструментальных программных средств поддержки технологии программирования.

В 80-х годах было характерно широкое внедрение персональных компьютеров во все сферы человеческой деятельности и, тем самым, создание обширного и разнообразного контингента пользователей ПС. Это привело к бурному развитию пользовательских интерфейсов и созданию четкой концепции качества ПС. Развиваются методы и языки спецификации ПС [7]. Выходит на передовые позиции *объектный подход* к разработке ПС [8]. Создаются различные инструментальные среды разработки и сопровождения ПС. Бурно развиваются концепции компьютерных сетей.

В 90-е годы международная компьютерная сеть широко охватила все человеческое общество, персональные компьютеры стали подключаться к ней как терминалы. Остро встала проблема защиты компьютерной информации и передаваемых по сети сообщений. Стали бурно развиваться компьютерная технология разработки ПС и связанные с ней формальные методы спецификации программ. Можно сказать, что в этот период начинается решающий этап полной информатизации и компьютеризации общества.

1.1.6. Технология программирования и информатизация общества

Технология программирования играла разную роль на разных этапах ее развития. По мере повышения мощности компьютеров и развития средств и методологии программирования росла и сложность решаемых задач с применением компьютеров, что и привело к повышенному вниманию к технологии программирования. Резкое удешевление стоимости компьютеров и, в особенности, стоимости хранения информации на магнитных носителях привело к широкому внедрению компьютеров практически во все сферы человеческой деятельности, что существенно изменило направленность технологии программирования. Человеческий фактор стал играть в ней решающую роль. Сформировалось достаточно ясное понятие

качества ПС, причем предпочтение стало отдаваться не столько эффективности ПС, сколько удобству работы с ним для пользователей, конечно, при равной надежности. Широкое использование компьютерных сетей привело к интенсивному развитию распределенных вычислений; появилась возможность для дистанционного доступа к удаленной информации и для электронного способа обмена сообщениями между людьми. Компьютерная техника из средства решения отдельных задач все более превращается в средство информационного моделирования реального и мыслимого мира. Все это ставит перед технологией программирования новые и достаточно трудные задачи.

1.2. Общие принципы разработки программных систем

1.2.1. Специфика разработки программных систем

1. Прежде всего, напомним о существующем противоречии, когда с одной стороны, мы имеем неформальный характер требований к ПС в виде постановки задачи, а также неформальное понятие программной ошибки, а с другой – формализованный основной объект разработки в виде итоговой программы. Тем самым разработка ПС содержит этапы формализации, а переход от неформального к формальному остается существенно неформальным.

2. Разработка ПС носит творческий характер, когда на каждом шаге приходится делать какой-либо выбор и принимать какое-либо решение. Она не сводится к выполнению какой-либо последовательности регламентированных действий. Тем самым эта разработка ближе к процессу проектирования каких-либо сложных устройств, но никак не к их массовому производству.

3. Вполне понятно, что продукт разработки представляет собой некоторую совокупность текстов в виде *статических объектов*, тогда как смысл или семантика этих текстов выражается процессами обработки дан-

ных и действиями пользователей, запускающих эти процессы, т.е. является *динамическим*. Это предопределяет выбор разработчиком ряда специфичных приемов, методов и средств.

4. Продукт разработки имеет и другую специфическую особенность – программная система при своем использовании не расходуется и не расходует дополнительных ресурсов.

1.2.2. Основные подходы при создании ПС

К настоящему время сложилось пять основных подходов к организации процесса создания и использования ПС [9].

Водопадный подход. При таком подходе разработка ПС состоит из цепочки этапов. На каждом этапе создаются документы, используемые на последующем этапе. В исходном документе фиксируются требования к ПС. В конце этой цепочки создаются программы, включаемые в ПС.

Исследовательское программирование. Этот подход предполагает быструю реализацию рабочих версий программ ПС, выполняющих лишь в первом приближении требуемые функции. После экспериментального применения реализованных программ производится их модификация с целью сделать их более полезными для пользователей. Этот процесс повторяется до тех пор, пока ПС не будет достаточно приемлемо для пользователей. Можно сказать, что исследовательское программирование исходит из взгляда на программирование как на искусство. Оно применяется тогда, когда водопадный подход не может быть использован из-за того, что не удается точно сформулировать требования к ПС.

Прототипирование. Этот подход моделирует начальную фазу исследовательского программирования вплоть до создания рабочих версий программ, предназначенных для проведения экспериментов с целью установить требования к ПС. В дальнейшем должна последовать разработка ПС

по установленным требованиям в рамках какого-либо другого подхода, например, водопадного.

Формальные преобразования. Этот подход включает разработку формальных спецификаций ПС и превращение их в программы путем корректных преобразований. На этом подходе базируется, например, компьютерная CASE-технология разработки ПС.

Сборочное программирование. Этот подход предполагает, что ПС конструируется из предопределенного набора компонент. Должна существовать некоторая библиотека таких компонент, каждая из которых может многократно использоваться в разных ПС. Процесс разработки ПС при данном подходе состоит скорее из сборки программ из компонент, чем из их программирования. В качестве примера реализации данного способа можно указать такие RAD системы, как Visual Basic и Delphi.

Из всех вышеперечисленных, водопадный подход представляет наибольший интерес, поскольку в этом подходе приходиться иметь дело с большинством процессов программной инженерии. Кроме того, в рамках этого подхода создается большинство больших программных систем. Именно этот подход рассматривается в качестве индустриального подхода разработки программного обеспечения.

1.2.3. Жизненный цикл программной системы

Под *жизненным циклом* ПС понимают весь период его разработки и эксплуатации, начиная от момента возникновения замысла ПС и кончая прекращением всех видов его использования [10]. Жизненный цикл охватывает довольно сложный процесс создания и использования ПС. Этот процесс может быть организован по-разному для разных классов ПС и в зависимости от особенностей коллектива разработчиков.

В рамках водопадного подхода выделяют три стадии жизненного цикла ПС (рис. 1.1), а именно: разработка ПС, производство программных изделий[1] и, наконец, эксплуатация ПС.

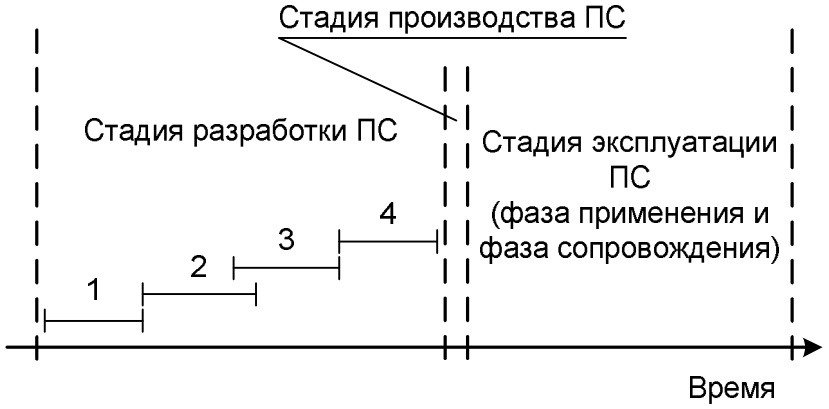

Рисунок 1.1. Стадии и этапы жизненного цикла ПС
1 – этап внешнего описания; 2 – конструирование; 3 – кодирование; 4 – аттестация

Стадия *разработки* ПС включает следующие этапы, при этом заметим, что всем этапам сопутствуют процессы документирования ПС:

- Этап внешнего описания ПС включает процессы, приводящие к созданию некоторого документа, который мы будем называть внешним описанием ПС. Этот документ является описанием поведения ПС с точки зрения внешнего по отношению к нему наблюдателя с фиксацией требований относительно его качества. Внешнее описание ПС начинается с анализа и определения требований к ПС со стороны заказчика, а также включает процессы спецификации этих требований.
- Конструирование ПС охватывает процессы разработки архитектуры ПС, разработки структур программ ПС и их детальную спецификацию.

1 Программное изделие – это экземпляр или копия разработанного ПС.

– Кодирование ПС включает процессы создания текстов программ на языках программирование, их отладку посредством тестирования. Зачастую этапы конструирования и кодирования перекрываются. Это означает, что кодирование некоторых частей программной системы может быть начато еще до завершения этапа конструирования.
– На этапе аттестации ПС производится оценка качества ПС. Если эта оценка оказывается приемлемой для практического использования ПС, то разработка ПС считается законченной.

Стадия *производства программных изделий* – это процесс генерации или воспроизведения программ и программных документов ПС с целью их поставки пользователю для применения по назначению. Тогда под производством программных изделий будем понимать совокупность работ по обеспечению изготовления требуемого количества копий ПС в установленные сроки [11]. Вполне очевидно, что в жизненном цикле ПС эта стадия является вырожденной, так как она представляет рутинную работу, которая может быть выполнена автоматически и без ошибок. Этим она принципиально отличается от стадии производства технических устройств.

Стадия *эксплуатации* ПС охватывает процессы хранения, внедрения и сопровождения ПС. Она состоит из двух параллельно проходящих фаз – фазы применения ПС и фазы сопровождения ПС.

Применение ПС – это использование ПС для решения практических задач на компьютере путем выполнения ее программ.

Сопровождение ПС – это процесс сбора информации о качестве ПС в эксплуатации, устранения обнаруженных в нем ошибок, доработки ПС и его модификации, а также извещения пользователей о внесенных в него изменениях [11].

1.2.4. Понятие качества программной системы

Каждое ПС должно обладать целым рядом свойств, которые дают возможность успешно его использовать в течение длительного периода, другими словами – программная система должно обладать определенным качеством. Под *качеством* ПС будем понимать совокупность черт и характеристик ПС, которые влияют на его способность удовлетворять заданные потребности пользователей. Это не означает, что разные ПС должны обладать одной и той же совокупностью таких свойств в их наивысшей степени. Этому препятствует тот факт, что повышение качества ПС по одному из таких свойств часто может быть достигнуто лишь ценой изменения стоимости, сроков завершения разработки и снижения качества этого ПС по другим его свойствам. Качество ПС является удовлетворительным, когда оно обладает указанными свойствами в такой степени, чтобы гарантировать успешное его использование.

Совокупность свойств ПС, которая образует удовлетворительное для пользователя качество ПС, зависит от условий и характера эксплуатации этого ПС, т.е. от позиции, с которой должно рассматриваться качество этого ПС. Поэтому при описании качества ПС, прежде всего, должны быть указаны критерии отбора требуемых свойств ПС. В настоящее время критериями качества ПС принято считать следующее шесть параметров [12]:

- *Функциональность*, т.е. способность ПС выполнять набор функций, удовлетворяющих заданным или подразумеваемым потребностям пользователей. Набор указанных функций определяется во внешнем описании ПС.
- *Надежность* подробно обсуждалась в п.1.1.3.
- *Легкостью применения* обладают ПС, которые позволяют минимизировать усилия пользователя как при подготовке исходных данных, так и при использовании ПС в целом.

- *Эффективность* – это отношение уровня услуг, предоставляемых ПС пользователю при заданных условиях, к объему используемых ресурсов.
- *Сопровождаемостью* обладают ПС, которые позволяют минимизировать усилия как при внесении изменений для устранения обнаруженных ошибок, так и при его модификации.
- *Мобильность* – это способность ПС быть перенесенным из одной компьютерной платформы на другую при минимальных затратах на модификацию.

Из всех перечисленных, функциональность и надежность являются *обязательными* критериями качества ПС, причем обеспечение надежности будет красной нитью проходить по всем этапам и процессам разработки ПС. Остальные критерии используются в зависимости от потребностей пользователей в соответствии с требованиями к ПС.

1.2.5. Обеспечение надежности – основной критерий разработки программных систем

Рассмотрим теперь общие принципы обеспечения надежности ПС, что, как уже подчеркивалось, является основной заботой при разработке ПС. В работе [2] указаны четыре основных подхода к обеспечению надежности: 1) предупреждение ошибок; 2) самообнаружение ошибок; 3) самоисправление ошибок; 4) обеспечение устойчивости к ошибкам.

Целью подхода *предупреждения ошибок* является не допустить ошибок в готовых ПС. Указанные ранее пути возникновения ошибок при разработке ПС позволяют для достижения этой цели сконцентрировать внимание на следующих вопросах:

- борьба со сложностью;
- обеспечение точности перевода;
- преодоление барьера между пользователем и разработчиком;

– обеспечение контроля принимаемых решений.

Этот подход связан с организацией процессов разработки ПС, т.е. с технологией программирования. И хотя, как уже отмечалось, гарантировать отсутствие ошибок в ПС невозможно, но в рамках этого подхода можно достигнуть приемлемого уровня надежности ПС.

Остальные три подхода связаны с организацией самих продуктов технологии. Они учитывают возможность появления ошибки в программах.

Самообнаружение ошибки в программе означает, что программа содержит средства обнаружения отказа в процессе ее выполнения.

Самоисправление ошибки в программе означает не только обнаружение отказа в процессе ее выполнения, но и исправление последствий этого отказа, для чего в программе должны иметься соответствующие средства.

Обеспечение устойчивости программы к ошибкам означает, что в программе содержатся средства, позволяющие локализовать область влияния отказа программы, уменьшить его негативные последствия и предотвратить катастрофические последствия отказа. Однако применение этих подходов весьма затруднено, поскольку многие простые методы, используемые в технике в рамках этих методов, неприменимы в программировании. Например, широко применяемое в технике дублирование отдельных блоков и устройств в нашем случае не дает никакого эффекта. Кроме того, добавление в программу дополнительных фрагментов приводит к ее значительному усложнению, что, в конечном счете, препятствует применению метода предупреждения ошибок.

1.2.5.1. Методы борьбы со сложностью

Прежде всего, нужно уяснить, что же скрывается за термином «сложность». Следуя работе [4], будем выделять три интеллектуальные способности человека, очень важные при разработке ПС, а именно:

- Способность человека *к перебору* связана с возможностью последовательного переключения внимания с одного предмета на другой, позволяя узнавать искомый предмет. Эта способность весьма ограничена, в среднем человек может уверенно, не сбиваясь перебирать в пределах 1000 предметов.
- Средством преодоления этой ограниченности является другая способность *к абстракции*, благодаря которой человек может объединять разные предметы или экземпляры в одно понятие, заменять множество элементов одним элементом другого рода.
- Способность человека *к математической индукции* позволяет ему справляться с бесконечными последовательностями.

При разработке ПС человек имеет дело с системами. Под системой будем понимать совокупность взаимодействующих друг с другом элементов. ПС можно рассматривать как пример системы. Логически связанный набор программ является другим примером системы. Любая отдельная программа также является системой. Понять систему означает осмысленно перебрать все пути взаимодействия между ее элементами. В силу ограниченности человека к перебору будем различать простые и сложные системы.

Под *простой* будем понимать такую систему, в которой человек может уверенно перебирать все пути взаимодействия между ее элементами, а под *сложной* будем понимать такую систему, в которой он этого делать не в состоянии.

При разработке ПС мы не всегда можем уверенно знать обо всех связях между ее элементами из-за возможных ошибок. Поэтому полезно уметь оценивать сложность системы по числу ее элементов, т.е. числом потенциальных путей взаимодействия между ее элементами. Данная оценка имеет вид $n!$, где n – число элементов системы.

Систему назовем *малой*, если $n < 7$ ($6! = 720 < 1000$), систему назовем *большой*, если $n \geq 7$. Малая система всегда *проста*. Большая система также может оказаться простой, но в подавляющем числе случаев большая система является *сложной*. Задача технологии программирования – научиться делать большие системы простыми.

Заметим, что полученная оценка простых систем по числу элементов широко используется на практике. Так, для руководителя коллектива весьма желательно, чтобы в нем не было больше шести взаимодействующих между собой подчиненных. Весьма важно также следовать правилу: «все, что может быть сказано, должно быть сказано в шести пунктах или меньше». Этому правилу мы будем стараться следовать в настоящей книге – всякие перечисления взаимосвязанных утверждений будут соответствующим образом группироваться и обобщаться. Полезно этому правилу следовать и при разработке ПС.

Итак, возвращаясь к вопросу борьбы со сложностью систем, укажем два таких общих метода:

Обеспечение независимости компонент системы означает разбиение системы на такие части, между которыми должны остаться по возможности меньше связей. Одним из воплощений этого метода является модульное программирование.

Использование в системах иерархических структур позволяет локализовать связи между компонентами, допуская их лишь между компонентами, принадлежащими смежным уровням иерархии. Этот метод, по существу, означает разбиение большой системы на подсистемы, образующих

малую систему. Здесь существенно используется способность человека к абстрагированию.

1.2.5.2. Обеспечение точности перевода

Обеспечение точности перевода направлено на достижение однозначности интерпретации документов различными разработчиками и, конечно же, пользователями ПС. Это требует придерживаться при переводе определенной дисциплины решения задач. Так, весь процесс перевода можно разбить на следующие этапы:

- поймите задачу, поставленную заказчиком;
- составьте план, включая цели и методы решения;
- выполните план, проверяя правильность каждого шага;
- проанализируйте полученное решение.

1.2.5.3. Преодоление барьера между пользователем и разработчиком

Как обеспечить, чтобы ПС выполняла то, что пользователь разумно ожидает от нее? Для этого необходимо правильно понять, чего же хочет пользователь, а также узнать уровень его подготовки и окружающую пользователя обстановку. При разработке ПС следует привлекать предполагаемых пользователей для участия в процессах принятия решений, а также детально ознакомиться с особенностями его работы. Самое лучшее – побывайте в его «шкуре».

1.2.5.4. Контроль принимаемых решений

Обязательным шагом на каждом этапе разработки ПС должна быть проверка правильности принятых решений. Это позволит обнаруживать и исправлять ошибки на самой ранней стадии после ее возникновения, что существенно снижает стоимость ее исправления и повышает вероятность полного ее устранения.

С учетом специфики разработки ПС необходимо применять везде, где это возможно, следующие два метода:

Смежный контроль означает проверку полученного документа лицами, не участвующими в его разработке, с двух сторон – со стороны автора исходного для контролируемого процесса документа, и лицами, которые будут использовать полученный документ в качестве исходного в последующих технологических процессах. Такой контроль позволяет обеспечивать однозначность интерпретации полученного документа.

Сочетание статических и динамических методов контроля означает то, что нужно не только контролировать документ как таковой, но и проверять какой процесс обработки данных он описывает. Это отражает одну из специфических особенность ПС, которую мы отмечали ранее – статическая форма при динамическом содержании.

1.3. Архитектура программной системы

1.3.1. Понятие архитектуры программной системы

Архитектура ПС – это представление программной системы как системы, состоящей из некоторой совокупности взаимодействующих подсистем. В качестве таких подсистем выступают обычно отдельные программы. Разработка архитектуры является первым этапом борьбы со сложностью ПС, на котором реализуется принцип выделения относительно независимых компонент. Основные задачи разработки архитектуры ПС заключаются в следующем:

- выделение программных подсистем и отображение на них внешних функций, заданных во внешнем описании ПС;
- определение способов взаимодействия между выделенными программными подсистемами.

С учетом принимаемых на этом этапе решений производится дальнейшая конкретизация функциональных спецификаций.

1.3.2. Основные классы архитектур программных систем

В работе [2] приводится следующая классификация архитектур программных систем:

- цельная программа;
- комплекс автономно выполняемых программ;
- слоистая программная система;
- комплекс параллельно выполняемых программ.

Цельная программа представляет вырожденный случай архитектуры ПС, поскольку в состав ПС входит только одна программа. Такую архитектуру выбирают обычно в том случае, когда ПС должно выполнять одну какую-либо ярко выраженную функцию и ее реализация не представляется слишком сложной. Естественно, что такая архитектура не требует какого-либо описания, кроме фиксации класса архитектуры, так как отображение внешних функций на эту программу тривиально, а определять способ взаимодействия не требуется в силу отсутствия какого-либо внешнего взаимодействия программы, кроме как взаимодействия ее с пользователем, а последнее описывается в документации по применению ПС.

Комплекс автономно выполняемых программ состоит из набора программ так, что только одна из программ может быть активизирована пользователем, но при выполнении этой программы другие программы из данного набора не могут быть запущены до тех пор, пока не закончит выполнение активизированная программа. Кроме того, все программы этого набора применятся к одной и той же информационной среде.

Следовательно, программы этого набора не взаимодействуют на уровне управления, поскольку взаимодействие между ними осуществляется только через общую информационную среду.

Слоистая программная система состоит из некоторой упорядоченной совокупности программных подсистем, называемых слоями, такой, что:

- на каждом слое ничего не известно ни о свойствах, ни о существовании других, более высоких слоев;
- каждый слой может взаимодействовать с предшествующим, более низким слоем только через заранее определенный интерфейс, ничего не зная о внутреннем строении всех предшествующих слоев;
- каждый слой располагает определенными ресурсами, которые он либо скрывает от других слоев, либо предоставляет последующему слою некоторые их абстракции через указанный интерфейс.

Таким образом, в слоистой программной системе каждый слой может реализовать некоторую абстракцию данных. Связи между слоями ограничены передачей значений параметров обращения каждого слоя к смежному снизу слою и выдачей результатов этого обращения от нижнего слоя верхнему. Использование глобальных данных несколькими слоями невозможно.

В качестве примера использования такой архитектуры рассмотрим некую операционную систему, предложенную Дейкстра. Эта операционная система состоит из четырех слоев. На нулевом слое производится обработка всех прерываний и распределение центрального процессора программам, как процессам. Только этот уровень осведомлен о мультипрограммных аспектах системы. На первом слое осуществляется управление страничной организацией памяти. Всем вышестоящим слоям предоставляется виртуальная непрерывная, но не страничная память. На втором слое осуществляется связь с консолью (пультом управления) оператора. Только этот слой знает технические характеристики консоли. На третьем, последнем слое осуществляется буферизация входных и выходных потоков данных и реализуются так называемые абстрактные каналы ввода и вывода,

так что прикладные программы не знают технических характеристик устройств ввода и вывода.

Нетрудно увидеть, как идеи, выдвинутые Дейкстра более 20 лет назад, воплотились в архитектуре операционных систем UNIX и последних версий Windows.

Комплекс параллельно действующих программ представляет собой набор программ, способных взаимодействовать между собой, находясь одновременно в стадии выполнения. Это означает, что такие программы, загружены в оперативную память, активизированы и могут попеременно разделять во времени один или несколько центральных процессоров компьютера. Кроме того, программы могут осуществлять динамическое взаимодействие между собой, выполняя синхронизацию. Обычно взаимодействие между такими процессами производится путем передачи друг другу некоторых сообщений.

Простейшей разновидностью такой архитектуры является конвейер. Конвейер представляет собой последовательность программ, в которой стандартный вывод каждой программы, кроме самой последней, связан со стандартным вводом следующей программы этой последовательности. Конвейер обрабатывает некоторый поток сообщений. Каждое сообщение этого потока поступает на ввод первой программе, которая передает переработанное сообщение следующей программе, а сама начинает обработку очередного сообщения их входного потока. Таким же образом действует каждая следующая программа конвейера – получив сообщение от предшествующей программы и, обработав его, она передает переработанное сообщение следующей программе и приступает к обработке очередного сообщения. Последняя программа конвейера выводит результат работы всего конвейера в виде результирующего сообщения. Таким образом, в конвейере, состоящим из n программ, может одновременно находиться в обработке до n сообщений. Конечно, в силу того, что разные программы конвейера

могут затратить на обработку очередных сообщений разное время, необходимо обеспечить синхронизацию всех процессов.

1.3.3. Архитектурные функции

Для обеспечения взаимодействия между подсистемами в ряде случаев не требуется создавать какие-либо дополнительные программные компоненты. Для этого может быть достаточно заранее фиксированных соглашений и стандартных возможностей базового программного обеспечения, т.е. операционной системы. Так в комплексе автономно выполняемых программ для обеспечения взаимодействия достаточно спецификации общей внешней информационной среды и возможностей операционной системы для запуска программ. В слоистой программной системе может оказаться достаточным спецификации выделенных программных слоев и обычный аппарат обращения к процедурам.

Однако в ряде случаев для обеспечения взаимодействия между программными подсистемами может потребоваться создание дополнительных программных компонент. Так для управления работой комплекса автономно выполняемых программ часто создают специализированный командный интерпретатор, более удобный в данной предметной области для подготовки требуемой внешней информационной среды и запуска требуемой программы, чем базовый командный интерпретатор используемой операционной системы. В слоистых программных системах может быть создан особый аппарат обращения к процедурам слоя, например, обеспечивающий параллельное выполнение этих процедур. В комплексе параллельно действующих программ для управления портами сообщений требуется специальная программная подсистема. Такие программные компоненты реализуют не внешние функции ПС, а функции, возникшие в результате разработки архитектуры этого ПС. В связи с этим такие функции мы будем называть архитектурными.

1.4. Тестирование и отладка программной системы

1.4.1. Основные понятия

Отладка ПС – это комплекс мер, направленных на обнаружение и исправление ошибок в ПС с использованием процессов выполнения его программ.

Тестирование ПС – это процесс выполнения его программ на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется тестовым или просто тестом. Таким образом, отладку можно представить в виде многократного повторения трех процессов: тестирования, в результате которого может быть констатировано наличие в ПС ошибки, поиска места ошибки в программах и документации ПС и редактирования программ и документации с целью устранения обнаруженной ошибки.

1.4.2. -Принципы и виды отладки программной системы

Успех отладки ПС в значительной степени предопределяет рациональная организация тестирования. При отладке ПС отыскиваются и устраняются, в основном, те ошибки, наличие которых в ПС устанавливается при тестировании. Как было уже отмечено ранее, тестирование не может доказать правильность ПС, в лучшем случае оно может продемонстрировать наличие в нем ошибки. Другими словами, нельзя гарантировать, что тестированием ПС практически выполнимым набором тестов можно установить наличие каждой имеющейся в ПС ошибки. Поэтому возникает две задачи.

Первая задача состоит в подготовке такого набора тестов, которые позволят обнаружить в нем по возможности большее число ошибок. Однако чем дольше продолжается процесс тестирования и отладки в целом, тем

большей становится стоимость ПС. Отсюда вытекает вторая задача, состоящая в определении момента окончания отладки ПС. Признаком возможности окончания отладки является полнота охвата пропущенными через ПС тестами множества различных ситуаций, возникающих при выполнении программ ПС, и относительно редкое проявление ошибок в ПС на последнем отрезке процесса тестирования. Последнее определяется в соответствии с требуемой степенью надежности ПС, указанной в спецификации его качества.

Для оптимизации набора тестов, т.е. для подготовки такого набора тестов, который позволил бы при заданном их числе обнаруживать наибольшее число ошибок в ПС, необходимо заранее спланировать этот набор и использовать рациональную стратегию планирования тестов. Проектирование тестов можно начинать сразу же после завершения этапа внешнего описания ПС. Возможны различные подходы к выработке стратегии проектирования тестов, которые можно условно разместить между следующими двумя предельными подходами [2].

Левый крайний подход (рис. 1.2) заключается в том, что тесты проектируются только на основании изучения спецификаций ПС – внешнего описания, описания архитектуры и спецификации программных модулей. Строение модулей при этом никак не учитывается, т.е. они рассматриваются как черные ящики. Фактически такой подход требует полного перебора всех наборов входных данных, так как в противном случае некоторые участки программ ПС могут не работать при пропуске любого теста, а это значит, что содержащиеся в них ошибки не будут проявляться. Однако тестирование ПС полным множеством наборов входных данных практически неосуществимо.

Правый крайний подход заключается в том, что тесты проектируются на основании изучения текстов программ с целью протестировать все пути выполнения каждой программ ПС. Если принять во внимание нали-

чие в программах циклов с переменным числом повторений, то различных путей выполнения программ ПС может оказаться чрезвычайно много, так что их тестирование также будет практически неосуществимо.

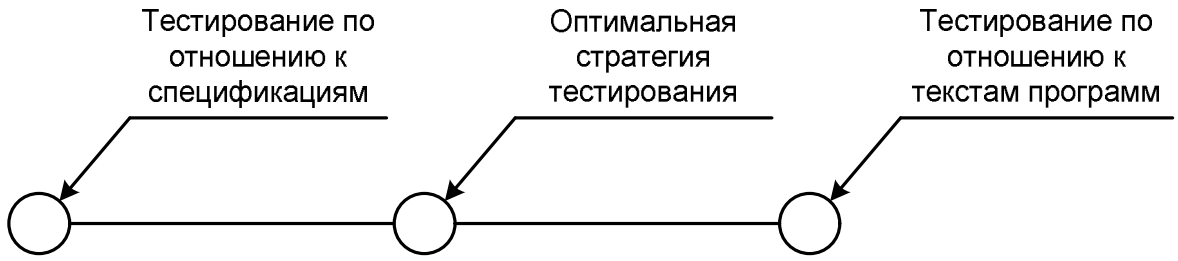

Рисунок 1.2. Спектр подходов к проектированию тестов

Оптимальная стратегия проектирования тестов расположена внутри интервала между этими крайними подходами, но ближе к левому краю. Она включает проектирование значительной части тестов по спецификациям, но требует также выполнения некоторых тестов и по текстам программ. При этом в первом случае эта стратегия базируется на принципах:

- на каждую используемую функцию или возможность требуется хотя бы один тест;
- на каждую область и на каждую границу изменения какой-либо входной величины необходим хотя бы один тест;
- на каждую особую, исключительную ситуацию, указанную в спецификациях, нужен хотя бы один тест.

Во втором случае эта стратегия базируется на принципе – каждая команда каждой программы ПС должна проработать хотя бы на одном тесте.

Кроме того, оптимальную стратегию проектирования тестов можно конкретизировать на основании следующего правила – для каждого программного документа, включая тексты программ, должны проектироваться свои тесты с целью выявления в нем ошибок. Во всяком случае, этот принцип необходимо соблюдать в соответствии с определением ПС и содержа-

нием понятия технологии программирования как технологии разработки надежных ПС.

Обычно различают два основных вида отладки – автономную и комплексную отладку ПС. *Автономная отладка* ПС означает последовательное раздельное тестирование различных частей программ, входящих в ПС, с поиском и исправлением в них фиксируемых при тестировании ошибок. Она фактически включает отладку каждого программного модуля и отладку сопряжения модулей. *Комплексная отладка* означает тестирование ПС в целом с поиском и исправлением фиксируемых при тестировании ошибок во всех документах, включая тексты программ ПС. К таким документам относятся определение требований к ПС, спецификация качества ПС, функциональная спецификация ПС, описание архитектуры ПС и тексты программ ПС.

1.4.3. Заповеди отладки программной системы

В этом разделе дадим общие рекомендации по организации отладки ПС. Но сначала следует отметить один феномен, который подтверждает важность предупреждения ошибок на предыдущих этапах разработки – по мере роста числа обнаруженных и исправленных ошибок в ПС растет также относительная вероятность существования в нем *необнаруженных ошибок* [2]. Это объясняется тем, что при росте числа ошибок, обнаруженных в ПС, уточняется и наше представление об общем числе допущенных в нем ошибок, а значит, в какой-то мере, и о числе необнаруженных еще ошибок.

Итак, сформулируем рекомендации, которых необходимо придерживаться при организации отладки.

Правило 1. Считайте тестирование ключевой задачей разработки ПС, поручайте его самым квалифицированным и одаренным программистам; нежелательно тестировать свою собственную программу.

Правило 2. Хорошим считается тест, для которого высока вероятность обнаружения ошибки, а не тот, который демонстрирует правильную работу программы.

Правило 3. Готовьте тесты как для правильных, так и для неправильных данных.

Правило 4. Документируйте запуски тестов через компьютер; детально изучайте результаты каждого теста; избегайте уникальных тестов, которые нельзя повторить.

Правило 5. Никогда не упрощайте программу только затем, чтобы облегчить ее тестирование.

Правило 6. Пропускайте заново все тесты, связанные с проверкой работы какой-либо программы ПС, если в нее были внесены изменения, например, в результате устранения ошибки.

1.4.4. Автономная отладка программной системы

При автономной отладке ПС каждый модуль на самом деле тестируется в некотором программном окружении, кроме случая, когда отлаживаемая программа состоит только из одного модуля. Это окружение состоит из других модулей, часть которых является модулями отлаживаемой программы, которые уже отлажены, а часть – модулями, управляющими отладкой или отладочными модулями. Таким образом, при автономной отладке тестируется всегда некоторая программа, построенная специально для тестирования отлаживаемого модуля. Эта программа лишь частично совпадает с отлаживаемой программой, кроме случая, когда отлаживается последний модуль отлаживаемой программы. В процессе автономной отладки ПС производится наращивание тестируемой программы отлаженными модулями, при переходе к отладке следующего модуля в его программное окружение добавляется последний отлаженный модуль. Такой процесс наращивания программного окружения отлаженными модулями

называется *интеграцией программы* [2]. Отладочные модули, входящие в окружение отлаживаемого модуля, зависят от порядка, в каком отлаживаются модули этой программы, от того, какой модуль отлаживается и, возможно, от того, какой тест будет пропускаться.

При восходящем тестировании это окружение будет содержать только один отладочный модуль, который будет головным в тестируемой программе. Такой отладочный модуль называют *ведущим* или драйвером [2]. Ведущий отладочный модуль подготавливает информационную среду для тестирования отлаживаемого модуля, т.е. формирует ее состояние, требуемое для тестирования этого модуля, в частности, путем ввода некоторых тестовых данных, осуществляет обращение к отлаживаемому модулю и после окончания его работы выдает необходимые сообщения. При отладке одного модуля для разных тестов могут составляться разные ведущие отладочные модули.

При нисходящем тестировании окружение отлаживаемого модуля в качестве отладочных модулей содержит отладочные имитаторы или заглушки некоторых еще не отлаженных модулей. К таким модулям относятся, прежде всего, все модули, к которым может обращаться отлаживаемый модуль, а также еще не отлаженные модули, к которым могут обращаться уже отлаженные модули, включенные в это окружение. Некоторые из этих имитаторов при отладке одного модуля могут изменяться для разных тестов.

На практике в окружении отлаживаемого модуля могут содержаться отладочные модули обоих типов, если используется смешанная стратегия тестирования. Это связано с тем, что как восходящее, так и нисходящее тестирование имеет свои достоинства и свои недостатки.

К достоинствам восходящего тестирования относятся: 1) простота подготовки тестов; 2) возможность полной реализации плана тестирования модуля.

Это связано с тем, что тестовое состояние информационной среды готовится непосредственно перед обращением к отлаживаемому модулю.

Недостатками восходящего тестирования являются следующие его особенности: 1) тестовые данные готовятся, как правило, не в той форме, которая рассчитана на пользователя; 2) большой объем отладочного программирования; 3) необходимость специального тестирования сопряжения модулей.

К достоинствам нисходящего тестирования относятся следующие его особенности: 1) большинство тестов готовится в форме, рассчитанной на пользователя; 2) во многих случаях относительно небольшой объем отладочного программирования; 3) отпадает необходимость тестирования сопряжения модулей.

Недостатком нисходящего тестирования является то, что тестовое состояние информационной среды перед обращением к отлаживаемому модулю готовится косвенно, оно является результатом применения уже отлаженных модулей к тестовым данным или данным, выдаваемым имитаторами. Это затрудняет подготовку тестов и требует высокой квалификации разработчика тестов, а также делает затруднительным или даже невозможным реализацию полного плана тестирования отлаживаемого модуля.

Указанный недостаток иногда вынуждает разработчиков применять только восходящее тестирование. Однако чаще применяют некоторые модификации нисходящего тестирования либо некоторую комбинацию нисходящего и восходящего тестирования. Исходя из того, что нисходящее тестирование, в принципе, является предпочтительным, остановимся на приемах, позволяющих в какой-то мере преодолеть указанные трудности.

Прежде всего, необходимо организовать отладку программы таким образом, чтобы как можно раньше были отлажены модули, осуществляющие ввод данных, тогда тестовые данные можно готовить в форме, рассчи-

танной на пользователя, что существенно упростит подготовку последующих тестов. Далеко не всегда этот ввод осуществляется в головном модуле, поэтому приходится в первую очередь отлаживать цепочки модулей, ведущие к модулям, осуществляющим указанный ввод. Пока модули, осуществляющие ввод данных, не отлажены, тестовые данные поставляются некоторыми имитаторами – они либо включаются в имитатор как его часть, либо вводятся этим имитатором.

При нисходящем тестировании некоторые состояния информационной среды, при которых требуется тестировать отлаживаемый модуль, могут не возникать при выполнении отлаживаемой программы ни при каких входных данных. В этих случаях можно было бы вообще не тестировать отлаживаемый модуль, так как обнаруживаемые при этом ошибки не будут проявляться при выполнении отлаживаемой программы ни при каких входных данных. Однако так поступать не рекомендуется, так как при изменениях отлаживаемой программы, например, при сопровождении ПС, не использованные для тестирования отлаживаемого модуля состояния информационной среды могут уже возникать, что требует дополнительного тестирования этого модуля. Для осуществления тестирования отлаживаемого модуля в указанных ситуациях иногда используют подходящие имитаторы, чтобы создать требуемое состояние информационной среды. Чаще же пользуются модифицированным вариантом нисходящего тестирования, при котором отлаживаемые модули перед их интеграцией предварительно тестируются отдельно. В этом случае в окружении отлаживаемого модуля появляется ведущий отладочный модуль, наряду с имитаторами модулей, к которым может обращаться отлаживаемый модуль. Однако более целесообразной представляется другая модификация нисходящего тестирования. После завершения нисходящего тестирования отлаживаемого модуля для достижимых тестовых состояний информационной среды следует его от-

дельно протестировать для остальных требуемых состояний информационной среды.

Часто применяют также комбинацию восходящего и нисходящего тестирования, которую называют методом *сандвича* [2]. Сущность этого метода заключается в одновременном осуществлении как восходящего, так и нисходящего тестирования, пока эти два процесса тестирования не встретятся на каком-либо модуле, где-то в середине структуры отлаживаемой программы. Этот метод при разумном порядке тестирования позволяет воспользоваться достоинствами как восходящего, так и нисходящего тестирования, а также в значительной степени нейтрализовать их недостатки.

Весьма важным при автономной отладке является тестирование сопряжения модулей. Дело в том, что спецификация каждого модуля программы, кроме головного, используется в этой программе в двух ситуациях – при разработке текста этого модуля и при написании обращения к этому модулю в других модулях программы. И в том, и в другом случаях в результате ошибки может быть нарушено требуемое соответствие заданной спецификации модуля. Такие ошибки требуется обнаруживать и устранять. Для этого и предназначено тестирование сопряжения модулей. При нисходящем тестировании тестирование сопряжения осуществляется попутно каждым пропускаемым тестом, что считают достоинством нисходящего тестирования. При восходящем тестировании обращение к отлаживаемому модулю производится не из модулей отлаживаемой программы, а из ведущего отладочного модуля. В связи с этим существует опасность, что последний модуль может приспособиться к некоторым дефектам отлаживаемого модуля. Поэтому, приступая в процессе интеграции программы к отладке нового модуля, приходится тестировать каждое обращение к ранее отлаженному модулю с целью обнаружения несогласованности этого обращения с телом соответствующего модуля. Таким образом, при-

ходится частично повторять в новых условиях тестирование ранее отлаженного модуля, при этом возникают те же трудности, что и при нисходящем тестировании.

Автономное тестирование модуля целесообразно осуществлять в четыре последовательно выполняемых шага [2].

Шаг 1. На основании спецификации отлаживаемого модуля подготовьте тесты для каждой возможности и каждой ситуации, для каждой границы областей допустимых значений всех входных данных и для каждой области изменения данных, для каждой области недопустимых значений всех входных данных и каждого недопустимого условия.

Шаг 2. Проверьте текст модуля, чтобы убедиться, что каждое направление любого разветвления будет пройдено хотя бы на одном тесте.

Шаг 3. Проверьте текст модуля, чтобы убедиться, что для каждого цикла существуют тесты, обеспечивающие, по крайней мере, три следующие ситуации – тело цикла не выполняется ни разу, тело цикла выполняется один раз и тело цикла выполняется максимальное число раз.

Шаг 4. Проверьте текст модуля, чтобы убедиться, что существуют тесты, проверяющие чувствительность к отдельным особым значениям входных данных.

Обязательно добавьте недостающие тесты, если обнаружилось, что какой-либо из шагов не выполнен.

1.4.5. Комплексная отладка программной системы

Вполне очевидно, при комплексной отладке тестируется ПС в целом, причем тесты готовятся по каждому из документов программного средства. Тестирование этих документов производится, как правило, в порядке, обратном их разработке. Исключение составляет лишь тестирование документации по применению, которая разрабатывается по внешнему описанию параллельно с разработкой текстов программ – это тестирование луч-

ше производить после завершения тестирования внешнего описания. Тестирование при комплексной отладке представляет собой применение ПС к конкретным данным, которые в принципе могут возникнуть у пользователя, в частности, все тесты готовятся в форме, рассчитанной на пользователя, но, возможно, в моделируемой, а не в реальной среде. Например, некоторые недоступные при комплексной отладке устройства ввода и вывода могут быть заменены их программными имитаторами.

Тестирование архитектуры ПС. Целью тестирования является поиск несоответствия между описанием архитектуры и совокупностью программ ПС. К моменту начала тестирования архитектуры ПС должна быть уже выполнена автономная отладка каждой подсистемы. Ошибки реализации архитектуры могут быть связаны, прежде всего, с взаимодействием этих подсистем, в частности, с реализацией архитектурных функций. Поэтому хотелось бы проверить все пути взаимодействия между подсистемами ПС. При этом желательно хотя бы протестировать все цепочки выполнения подсистем без повторного вхождения последних. Если заданная архитектура представляет ПС в качестве малой системы из выделенных подсистем, то число таких цепочек будет вполне обозримо.

Тестирование внешних функций. Целью тестирования является поиск расхождений между функциональной спецификацией и совокупностью программ ПС. Несмотря на то, что все эти программы автономно уже отлажены, указанные расхождения могут быть, например, из-за несоответствия внутренних спецификаций программ и их модулей функциональной спецификации ПС в целом. Как правило, тестирование внешних функций производится так же, как и тестирование модулей на первом шаге, т.е. как черного ящика.

Тестирование качества ПС. Целью тестирования является поиск нарушений требований качества, сформулированных в спецификации качества ПС. Это наиболее трудный и наименее изученный вид тестирова-

ния. Ясно лишь, что далеко не каждый примитив качества ПС может быть испытан тестированием. Завершенность ПС проверяется уже при тестировании внешних функций. На данном этапе тестирование этого примитива качества может быть продолжено, если требуется получить какую-либо вероятностную оценку степени надежности ПС. Однако методика такого тестирования еще требует своей разработки. Могут тестироваться такие примитивы качества, как точность, устойчивость, защищенность, временная эффективность, в какой-то мере – эффективность по памяти, эффективность по устройствам, расширяемость и, частично – независимость от устройств. Каждый из этих видов тестирования имеет свою специфику и заслуживает отдельного рассмотрения.

Тестирование документации по применению ПС. Целью тестирования является поиск несогласованности документации по применению и совокупностью программ ПС, а также выявление неудобств, возникающих при применении ПС. Этот этап непосредственно предшествует подключению пользователя к завершению разработки ПС, поэтому весьма важно разработчикам сначала самим воспользоваться ПС так, как это будет делать пользователь [2]. Все тесты на этом этапе готовятся исключительно на основании только документации по применению ПС. Прежде всего, должны тестироваться возможности ПС как это делалось при тестировании внешних функций, но только на основании документации по применению. Должны быть протестированы все неясные места в документации, а также все примеры, использованные в документации. Далее тестируются наиболее трудные случаи применения ПС с целью обнаружить нарушение требований относительности легкости применения ПС.

Тестирование определения требований к ПС. Целью тестирования является выяснение, в какой мере ПС не соответствует предъявленному определению требований к нему. Особенность этого вида тестирования заключается в том, что его осуществляет заказчик ПС с целью преодоления

барьера между разработчиком и пользователем [2]. Обычно это тестирование производится с помощью контрольных задач, для которых известен результат решения. В тех случаях, когда разрабатываемое ПС должно придти на смену другой версии ПС, которая решает хотя бы часть задач разрабатываемого ПС, тестирование производится путем решения общих задач с помощью как старого, так и нового ПС с последующим сопоставлением полученных результатов. Иногда в качестве формы такого тестирования используют опытную эксплуатацию ПС, т.е. ограниченное применение нового ПС с анализом использования результатов в практической деятельности.

1.5. Обеспечение функциональности и надежности программного средства

1.5.1. Функциональность и надежность как обязательные критерии качества программного средства

В предыдущих разделах были рассмотрены все этапы разработки ПС, кроме его аттестации. При этом мы не касались вопросов обеспечения качества ПС в соответствии с его спецификацией качества. Занимаясь реализацией функциональной спецификации ПС, мы тем самым обсудили основные вопросы обеспечения критерия функциональности. Объявив надежность ПС основным его атрибутом, мы выбрали *предупреждение ошибок* в качестве основного подхода для обеспечения надежности ПС и обсудили его воплощение на разных этапах разработки ПС. Таким образом, проявлялся тезис об обязательности функциональности и надежности ПС как критериев его качества.

Тем не менее, в спецификации качества ПС могут быть дополнительные характеристики этих критериев, обеспечение которых требуют специального обсуждения. Этим вопросам и посвящен настоящий раздел.

Ниже обсуждается обеспечение примитивов качества ПС, выражающих критерии функциональности и надежности ПС.

1.5.2. Обеспечение завершенности программного средства

Завершенность ПС является общим примитивом качества ПС для выражения и функциональности и надежности ПС, причем для функциональности она является единственным примитивом.

Функциональность ПС определяется его функциональной спецификацией. Завершенность ПС как примитив его качества является мерой того, в какой степени эта спецификация реализована в разрабатываемом ПС. Обеспечение этого примитива в полном объеме означает реализацию каждой из функций, определенной в функциональной спецификации, со всеми указанными там деталями и особенностями. Все рассмотренные ранее технологические процессы показывают, как это может быть сделано.

Однако в спецификации качества ПС могут быть определены несколько уровней реализации функциональности ПС: может быть определена некоторая упрощенная (начальная или стартовая) версия, которая должна быть реализована в первую очередь; могут быть также определены и несколько промежуточных версий. В этом случае возникает дополнительная технологическая задача: организация наращивания функциональности ПС. Здесь важно отметить, что разработка упрощенной версии ПС не есть разработка его *прототипа*. Прототип разрабатывается для того, чтобы лучше понять условия применения будущего ПС, уточнить его внешнее описание. Он рассчитан на избранных пользователей и поэтому может сильно отличаться от требуемого ПС не только выполняемыми функциями, но и особенностями пользовательского интерфейса. Упрощенная же версия разрабатываемого ПС должна быть рассчитана на *практически полезное* применение любыми пользователями, для которых предназначено это ПС. Поэтому главный принцип обеспечения функционально-

сти такого ПС заключается в том, чтобы с самого начала разрабатывать ПС таким образом, как будто требуется ПС в полном объеме, до тех пор, когда разработчики будут иметь дело непосредственно с теми частями или деталями ПС, реализацию которых можно отложить в соответствии со спецификацией его качества. Тем самым, и внешнее описание и описание архитектуры ПС должно быть разработано в полном объеме. Можно отложить лишь реализацию тех программных подсистем (определенных в архитектуре разрабатываемого ПС), функционирования которых не требуется в начальной версии этого ПС. Реализацию же самих программных подсистем лучше всего производить методом целенаправленной конструктивной реализации, оставляя в начальной версии ПС подходящие имитаторы тех программных модулей, функционирование которых в этой версии не требуется. Допустима также упрощенная реализация некоторых программных модулей, опускающая реализацию некоторых деталей соответствующих функций. Однако такие модули с технологической точки зрения лучше рассматривать как своеобразные их имитаторы (хотя и далеко продвинутые).

Достигнутый при обеспечении функциональности ПС уровень его завершенности на самом деле может быть не таким, как ожидалось, из-за ошибок, оставшихся в этом ПС. Можно лишь говорить, что требуемая завершенность достигнута с некоторой вероятностью, определяемой объемом и качеством проведенного тестирования. Для того чтобы повысить эту вероятность, необходимо продолжить тестирование и отладку ПС. Однако, оценивание такой вероятности является весьма специфической задачей, которая пока еще ждет соответствующих теоретических исследований.

1.5.3. Обеспечение точности программного средства

Обеспечение этого примитива качества связано с действиями над значениями вещественных типов (точнее говоря, со значениями, представ-

ляемыми с некоторой погрешностью). Обеспечить требуемую точность при вычислении значения той или иной функции – значит получить это значение с погрешностью, не выходящей за рамки заданных границ. Видами погрешности, методами их оценки и методами достижения требуемой точности (*приближенными вычислениями*) занимается вычислительная математика. Здесь мы лишь обратим внимание на некоторую структуру погрешности: погрешность вычисленного значения (*полная погрешность*) зависит от:

- *погрешности используемого метода* вычисления, в которую включается и неточность используемой модели;
- *неустранимой погрешности* – погрешности представления используемых данных;
- *погрешности округления* – неточности выполнения используемых в методе операций.

1.5.4. Обеспечение автономности программного средства

Вопрос об автономности программного средства решается путем принятия решения о возможности использования в разрабатываемом ПС какого-либо подходящего базового программного обеспечения. Надежность имеющегося в распоряжении разработчиков базового программного обеспечения для целевого компьютера может не отвечать требованиям к надежности разрабатываемого ПС. Поэтому от использования такого программного обеспечения приходиться отказываться, а его функции в требуемом объеме приходится реализовывать в рамках разрабатываемого ПС. Аналогичное решение приходится принимать при жестких ограничениях на используемые ресурсы (по критерию эффективности ПС). Такое решение может быть принято уже в процессе разработки спецификации качества ПС, иногда – на этапе конструирования ПС.

1.5.5. Обеспечение устойчивости программного средства

Этот примитив качества ПС обеспечивается с помощью так называемого *защитного программирования*. Вообще говоря, защитное программирование применяется при программировании модуля для повышения надежности ПС в более широком смысле. Как утверждает Майерс [2], «защитное программирование основано на важной предпосылке: худшее, что может сделать модуль, – это принять неправильные входные данные и затем вернуть неверный, но правдоподобный результат». Для того чтобы этого избежать, в текст модуля включают проверки его входных и выходных данных на их корректность в соответствии со спецификацией этого модуля, в частности, должны быть проверены выполнение ограничений на входные и выходные данные и соотношений между ними, указанные в спецификации модуля. В случае отрицательного результата проверки возбуждается соответствующая исключительная ситуация. Для обработки таких ситуаций в конец этого модуля включаются фрагменты второго рода – обработчики соответствующих исключительных ситуаций. Эти обработчики помимо выдачи необходимой диагностической информации, могут принять меры либо по исключению ошибки в данных (например, потребовать их повторного ввода), либо по ослаблению влияния ошибки (например, во избежание поломки устройств, управляемых с помощью данного ПС, при аварийном прекращении выполнения программы осуществляют мягкую их остановку).

Применение защитного программирования модулей приводит к снижению эффективности ПС как по времени, так и по памяти. Поэтому необходимо разумно регулировать степень применения защитного программирования в зависимости от требований к надежности и эффективности ПС, сформулированным в спецификации качества разрабатываемого ПС. Входные данные разрабатываемого модуля могут поступать как непосредственно от пользователя, так и от других модулей. Наиболее употреби-

тельным случаем применения защитного программирования является применение его для первой группы данных, что и означает реализацию устойчивости ПС. Это нужно делать всегда, когда в спецификации качества ПС имеется требование об обеспечении устойчивости ПС. Применение защитного программирования для второй группы входных данных означает попытку обнаружить ошибку в других модулях во время выполнения разрабатываемого модуля, а для выходных данных разрабатываемого модуля – попытку обнаружить ошибку в самом этом модуле во время его выполнения. По существу, это означает частичное воплощение подхода самообнаружения ошибок для обеспечения надежности ПС. Этот случай защитного программирования применяется крайне редко – только в том случае, когда требования к надежности ПС чрезвычайно высоки.

1.5.6. Обеспечение защищенности программных средств

Различают следующие виды защиты ПС от искажения информации:

- защита от сбоев аппаратуры;
- защита от влияния «чужой» программы;
- защита от отказов «своей» программы;
- защита от ошибок оператора (пользователя);
- защита от несанкционированного доступа;
- защита от защиты.

Защита от сбоев аппаратуры. В настоящее время этот вид защиты является не очень злободневной задачей (с учетом уровня достигнутой надежности компьютеров). Но все же полезно знать ее решение. Это обеспечивается организацией т.н. «двойных или тройных просчетов». Для этого весь процесс обработки данных, определяемый ПС, разбивается по времени на интервалы так называемыми «опорными точками». Длина этого интервала не должна превосходить половины среднего времени безотказной работы компьютера. В начале каждого такого интервала во вторич-

ную память записывается с некоторой контрольной суммой копия состояния изменяемой в этом процессе памяти («опорная точка»). Для того чтобы убедиться, что обработка данных от одной опорной точки до следующей (т.е. один «просчет») произведена правильно (без сбоев компьютера), производится два таких «просчета». После первого «просчета» вычисляется и запоминается указанная контрольная сумма, а затем восстанавливается состояние памяти по опорной точке и делается второй «просчет». После второго «просчета» вычисляется снова указанная контрольная сумма, которая затем сравнивается с контрольной суммой первого «просчета». Если эти две контрольные суммы совпадают, второй просчет считается правильным, в противном случае контрольная сумма второго «просчета» также запоминается и производится третий «просчет» (с предварительным восстановлением состояния памяти по опорной точке). Если контрольная сумма третьего «просчета» совпадет с контрольной суммой одного из первых двух «просчетов», то третий просчет считается правильным, в противном случае требуется инженерная проверка компьютера.

Защита от влияния «чужой» программы. При появлении мультипрограммного режима работы компьютера в его памяти может одновременно находиться в стадии выполнения несколько программ, попеременно получающих управление в результате возникающих прерываний (т.н. квазипараллельное выполнение программ). Одна из таких программ (обычно: операционная система) занимается обработкой прерываний и управлением мультипрограммным режимом. Здесь под «чужой» программой понимается программа (или какой-либо программный фрагмент), выполняемая параллельно (или квазипараллельно) по отношению к защищаемой программе (или ее фрагменту). Этот вид защиты должна обеспечить, чтобы эффект выполнения защищаемой программы не зависел от того, какие программы выполняются параллельно с ней, и относится, прежде всего, к функциям операционных систем. Различают две разновидности этой защиты:

– защита от отказов «чужой» программы;
– защита от злонамеренного влияния «чужой» программы.

Защита от отказов «чужой» программы означает, что на выполнение функций защищаемой программой не будут влиять отказы (проявления ошибок), возникающие в параллельно выполняемых программах. Для того чтобы управляющая программа (операционная система) могла обеспечить защиту себя и других программ от такого влияния, аппаратура компьютера должна реализовывать следующие возможности:

– защиту памяти;
– два режима функционирования компьютера: привилегированный и рабочий (пользовательский);
– два вида операций: привилегированные и ординарные;
– корректную реализацию прерываний и начального включения компьютера;
– временно́е прерывание.

Защита памяти означает возможность программным путем задавать для каждой программы недоступные для нее участки памяти. В привилегированном режиме могут выполняться любые операции (как ординарные, так и привилегированные), а в рабочем режиме – только ординарные. Попытка выполнить привилегированную операцию, а также обратиться к защищенной памяти в рабочем режиме вызывает соответствующее прерывание. К привилегированным операциям относятся операции изменения защиты памяти и режима функционирования, а также доступа к внешней информационной среде. Корректная реализация прерываний и начального включения компьютера означает обязательную установку привилегированного режима и отмену защиты памяти. В этих условиях управляющая программа (операционная система) может полностью защитить себя от влияния отказов других программ. Для этого достаточно, чтобы:

- все точки передачи управления при начальном включении компьютера и при прерываниях принадлежали этой программе;
- она не позволяла никакой другой программе работать в привилегированном режиме (при передаче управления любой другой программе должен включаться только рабочий режим);
- она полностью защищала свою память (содержащую, в частности, всю ее управляющую информацию, включая так называемые вектора прерываний) от других программ.

Тогда никто не помешает ей выполнять любые реализованные в ней функции защиты других программ (в том числе и доступа к внешней информационной среде). Для облегчения решения этой задачи часть такой программы помещается в постоянную память. Наличие временно́го прерывания позволяет управляющей программе защититься от зацикливания в других программах (без такого прерывания она могла бы просто лишиться возможности управлять).

Защита от злонамеренного влияния «чужих» программ означает, что изменение внешней информационной среды, предоставленной защищаемой программе, со стороны другой, параллельно выполняемой программы будет невозможно или сильно затруднено без ведома защищаемой программы. Для этого операционная система должна обеспечить подходящий контроль доступа к внешней информационной среде. Необходимым условием обеспечение такого контроля является обеспечения защиты от злонамеренного влияния «чужих» программ хотя бы самой операционной системы. В противном случае такой контроль можно было бы обойти путем изменения операционной системы со стороны «злонамеренной» программы.

Этот вид защиты включает, в частности, и защиту от т.н. «компьютерных вирусов», под которыми понимают фрагменты программ, способные в процессе своего выполнения внедряться (копироваться) в другие

программы (или в отдельные программные фрагменты). «Компьютерные вирусы», обладая способностью к размножению (к внедрению в другие программы), при определенных условиях вызывают изменение эффекта выполнения «зараженной» программы, что может привести к серьезным деструктивным изменениям ее внешней информационной среды. Операционная система, будучи защищенной от влияния «чужих» программ, может ограничить доступ к программным фрагментам, хранящимся во внешней информационной среде. Так, например, может быть запрещено изменение таких фрагментов любыми программами, кроме некоторых, которые знает операционная система, или, другой вариант, может быть разрешено только после специальных подтверждений программы (или пользователя).

Защита от отказов «своей» программы. Обеспечивается надежностью ПС, на что ориентирована вся технология программирования, обсуждаемая в настоящем пособии.

Защита от ошибок пользователя. Здесь идет речь не об ошибочных данных, поступающих от пользователя ПС, – защита от них связана с обеспечением устойчивости ПС, а о действиях пользователя, приводящих к деструктивному изменению состояния внешней информационной среды ПС, несмотря на корректность используемых при этом данных. Защита от таких действий, частично, обеспечивается выдачей предупредительных сообщений о попытках изменить состояние внешней информационной среды ПС с требованием подтверждения этих действий. Для случаев же, когда такие ошибки совершаются, может быть предусмотрена возможность восстановления состояния отдельных компонент внешней информационной среды ПС на определенные моменты времени. Такая возможность базируется на ведении (формировании) архива состояний (или изменений состояния) внешней информационной среды.

Защита от несанкционированного доступа. Каждому пользователю ПС предоставляет определенные информационные и процедурные ресурсы

(услуги), причем у разных пользователей ПС предоставленные им ресурсы могут отличаться, иногда очень существенно. Этот вид защиты должен обеспечить, чтобы каждый пользователь ПС мог использовать только то, что ему предоставлено (санкционировано). Для этого ПС в своей внешней информационной среде может хранить информацию о своих пользователях и предоставленным им правах использования ресурсов, а также предоставлять пользователям определенные возможности формирования этой информации. Защита от несанкционированного доступа к ресурсам ПС осуществляется с помощью т.н. *паролей* (секретных слов). При этом предполагается, что каждый пользователь знает только свой пароль, зарегистрированный в ПС этим пользователем. Для доступа к выделенным ему ресурсам он должен предъявить ПС свой пароль. Другими словами, пользователь как бы «вешает замок» на предоставленные ему права доступа к ресурсам, «ключ» от которого имеется только у этого пользователя.

Различают две разновидности такой защиты:

- простая защита от несанкционированного доступа;
- защита от взлома защиты.

Простая защита от несанкционированного доступа обеспечивает защиту от использования ресурсов ПС пользователем, которому не предоставлены соответствующие права доступа или который указал неправильный пароль. При этом предполагается, что пользователь, получив отказ в доступе к интересующим ему ресурсам, не будет предпринимать попыток каким-либо несанкционированным образом обойти или преодолеть эту защиту. Поэтому этот вид защиты может применяться и в ПС, которая базируется на операционной системе, не обеспечивающей полную защиту от влияния «чужих» программ.

Защита от взлома защиты – это такая разновидность защиты от несанкционированного доступа, которая существенно затрудняет преодоление этой защиты. Это связано с тем, что в отдельных случаях могут быть

предприняты настойчивые попытки взломать защиту от несанкционированного доступа, если защищаемые ресурсы представляют для кого-то чрезвычайную ценность. Для такого случая приходится предпринимать дополнительные меры защиты. Во-первых, необходимо обеспечить, чтобы такую защиту нельзя было обойти, т.е. должна действовать защита от влияния «чужих» программ. Во-вторых, необходимо усилить простую защиту от несанкционированного доступа использованием в ПС специальных программистских приемов, в достаточной степени затрудняющих подбор подходящего пароля или его вычисление по информации, хранящейся во внешней информационной среде ПС. Использование обычных паролей оказывается недостаточной, когда речь идет о чрезвычайно настойчивом стремлении добиться доступа к ценной информации. Если требуемый пароль («замок») в явном виде хранится во внешней информационной среде ПС, то «взломщик» этой защиты относительно легко может его достать, имея доступ к этому ПС. Кроме того, следует иметь в виду, что с помощью современных компьютеров можно осуществлять достаточно большой перебор возможных паролей с целью найти подходящий.

Защититься от этого можно следующим образом. Пароль (секретное слово или просто секретное целое число) X должен быть известен только владельцу защищаемых прав доступа и он не должен храниться во внешней информационной среде ПС. Для проверки прав доступа во внешней информационной среде ПС хранится другое число $Y = F(X)$, однозначно вычисляемое ПС по предъявленному паролю X. При этом функция F может быть хорошо известной всем пользователям ПС, однако она должна обладать таким свойством, что восстановление слова X по Y практически невозможно: при достаточно большой длине слова X (например, в несколько сотен знаков) для этого может потребоваться астрономическое время. Такое число Y будем называть *электронной* (*компьютерной*) *подписью* владельца пароля X, а значит, и защищаемых прав доступа.

Другой способ защиты от взлома защиты связан с защитой сообщений, пересылаемых по компьютерным сетям. Такое сообщение может представлять команду на дистанционный доступ к ценной информации, и этот доступ отправитель сообщения хочет защитить от возможных искажений. Например, при осуществлении банковских операций с использованием компьютерной сети. Использование компьютерной подписи в такой ситуации недостаточно, так как защищаемое сообщение может быть перехвачено «взломщиком» (например, на «перевалочных» пунктах компьютерной сети) и подменено другим сообщением с сохранением компьютерной подписи (или пароля).

Защиту от такого взлома защиты можно осуществить следующим образом. Наряду с функцией *F*, определяющей компьютерную подпись владельца пароля *X*, в ПС определены еще две функции: *Stamp* и *Notary*. При передаче сообщения отправитель, помимо компьютерной подписи $Y = F(X)$, должен вычислить еще другое число $S = Stamp(X,R)$, где X – пароль, а R – текст передаваемого сообщения. Здесь также предполагается, что функция Stamp хорошо известна всем пользователям ПС и обладает таким свойством, что по *S* практически невозможно ни восстановить число *X*, ни подобрать другой текст сообщения *R* с заданной компьютерной подписью *Y*. При этом передаваемое сообщение (вместе со своей защитой) должно иметь вид: (*R Y S*), причем *Y* (компьютерная подпись) позволяет получателю сообщения установить истинность клиента, а *S* как бы скрепляет защищаемый текст сообщения R с компьютерной подписью *Y*. В связи с этим будем называть число *S* *электронной* (*компьютерной*) *печатью*. Функция $Notary(R,Y,S)$ проверяет истинность защищаемого сообщения: (R,Y,S).

Эта позволяет получателю сообщения однозначно установить, что текст сообщения *R* принадлежит владельцу пароля *X*.

Защита от защиты. Защита от несанкционированного доступа может создать нежелательную ситуацию для самого владельца прав доступа к ресурсам ПС – он не сможет воспользоваться этими правами, если забудет (или потеряет) свой пароль («ключ»). Для защиты интересов пользователя в таких ситуациях и предназначена защита от защиты. Для обеспечения такой защиты ПС должно иметь привилегированного пользователя, называемого *администратором ПС.* Администратор ПС должен, в частности, отвечать за функционирование защиты ПС: именно он должен формировать контингент пользователей данного экземпляра ПС, предоставляя каждому из этих пользователей определенные права доступа к ресурсам ПС. В ПС должна быть привилегированная операция (для администратора), позволяющая временно снимать защиту от несанкционированного доступа для пользователя с целью фиксации требуемого пароля («замка»).

1.6. Обеспечение качества программного средства

1.6.1. Общая характеристика процесса обеспечения качества программного средства

Спецификация качества определяет основные ориентиры (цели), которые на всех этапах разработки ПС так или иначе влияют при принятии различных решений на выбор подходящего варианта. Однако каждый примитив качества имеет свои особенности такого влияния, тем самым, обеспечение его наличия в ПС может потребовать своих подходов и методов разработки ПС или отдельных его частей. Кроме того, уже отмечалась ранее противоречивость критериев качества ПС, а также и выражающих их примитивов качества: хорошее обеспечение одного какого-либо примитива качества ПС может существенно затруднить или сделать невозможным обеспечение некоторых других из этих примитивов. Поэтому существенная часть процесса обеспечения качества ПС состоит из поиска приемлемых компромиссов. Эти компромиссы частично должны быть определены

уже в спецификации качества ПС: модель качества ПС должна конкретизировать требуемую степень присутствия в ПС каждого его примитива качества и определять приоритеты достижения этих степеней.

Обеспечение качества осуществляется в каждом технологическом процессе: принятые в нем решения в той или иной степени оказывают влияние на качество ПС в целом. В частности и потому, что значительная часть примитивов качества связана не столько со свойствами программ, входящих в ПС, сколько со свойствами документации. В силу отмеченной противоречивости примитивов качества весьма важно придерживаться выбранных приоритетов в их обеспечении. При этом следует придерживаться двух общих принципов:

- сначала необходимо обеспечить требуемую функциональность и надежность ПС, а затем уже доводить остальные критерии качества до приемлемого уровня их присутствия в ПС;
- нет никакой необходимости и, может быть, даже вредно добиваться более высокого уровня присутствия в ПС какого-либо примитива качества, чем тот, который определен в спецификации качества ПС.

Обеспечение функциональности и надежности ПС было рассмотрено в предыдущем разделе. Ниже обсуждается обеспечение других критериев качества ПС.

1.6.2. Обеспечение легкости применения программного средства

Легкость применения, в значительной степени, определяется составом и качеством пользовательской документации, а также некоторыми свойствами, реализуемыми программным путем.

С пользовательской документацией связаны такие примитивы качества ПС, как *П-документированность* и *информативность*. Обеспечением ее качества занимаются обычно технические писатели. Этот вопрос будет

обсуждаться позднее. Здесь лишь следует заметить, что там речь будет идти об автономной по отношению к программам документации. В связи с этим следует обратить внимание на широко используемый в настоящее время подход информирования пользователя в интерактивном режиме (в процессе применения программ ПС). Такое информирование во многих случаях оказывается более удобным для пользователя, чем с помощью автономной документации, так как позволяет пользователю без какого-либо поиска вызывать необходимую информацию за счет использования контекста ее вызова. Такой подход к информированию пользователя является весьма перспективным.

Программным путем реализуются такие примитивы качества ПС как коммуникабельность, устойчивость и защищенность. Обеспечение устойчивости и защищенности уже было рассмотрено ранее. Коммуникабельность обеспечивается соответствующей реализацией обработки исключительных ситуаций и созданием подходящего пользовательского интерфейса.

Возбуждение исключительной ситуации во многих случаях означает, что возникла необходимость информировать пользователя о ходе выполнения программы. При этом выдаваемая пользователю информация должна быть простой для понимания. Однако исключительные ситуации возникают обычно на достаточно низком уровне модульной структуры программы, а создать понятное для пользователя сообщение можно, как правило, на более высоких уровнях этой структуры, где известен контекст, в котором были активизированы действия, приведшие к возникновению исключительной ситуации. Обработка исключительных ситуаций внутри модуля уже обсуждалась. Для обработки возникшей исключительной ситуации в другом модуле приходится принимать не простые решения. Применяемый часто способ передачи информации о возникшей исключительной ситуации по цепочке обращений к программным модулям (в обратном

направлении) является тяжеловесным: он требует дополнительных проверок после возврата из модуля и часто усложняет само обращение к этим модулям за счет задания дополнительных параметров. Приемлемым решением является включение в операционную среду выполнения программ (в исполнительную поддержку) возможностей прямой передачи этой информации обработчикам исключительных ситуаций по динамически формируемой очереди таких обработчиков.

Пользовательский интерфейс представляет средство взаимодействия пользователя с ПС. При разработке пользовательского интерфейса следует учитывать потребности, опыт и способности пользователя [9]. Поэтому потенциальные пользователи должны быть вовлечены в процесс разработки такого интерфейса. Большой эффект здесь дает его прототипирование. При этом пользователи должны получить доступ к прототипам пользовательского интерфейса, а их оценка различных возможностей используемого прототипа должна существенно учитываться при создании окончательного варианта пользовательского интерфейса.

В силу большого разнообразия пользователей и видов ПС существует множество различных стилей пользовательских интерфейсов, при разработке которых могут использоваться разные принципы и подходы. Однако следующие важнейшие принципы следует соблюдать всегда [9]:

- пользовательский интерфейс должен базироваться на терминах и понятиях, знакомых пользователю;
- пользовательский интерфейс должен быть единообразным;
- пользовательский интерфейс должен позволять пользователю исправлять собственные ошибки;
- пользовательский интерфейс должен позволять получение пользователем справочной информации: как по его запросу, так и генерируемой ПС.

В настоящее время широко распространены командные и графические пользовательские интерфейсы.

Командный пользовательский интерфейс предоставляет пользователю возможность обращаться к ПС с некоторым заданием (запросом), представляемым некоторым текстом (командой) на специальном командном языке (языке заданий). Достоинствами такого интерфейса является возможность его реализации на дешевых алфавитно-цифровых терминалах и возможность минимизации требуемого от пользователя ввода с клавиатуры. Недостатками такого интерфейса являются необходимость изучения командного языка и достаточно большая вероятность ошибки пользователя при задании команды. В связи с этим командный пользовательский интерфейс обычно выбирают только опытные пользователи. Такой интерфейс позволяет им осуществлять быстрое взаимодействие с компьютером и предоставляет возможность объединять команды в процедуры и программы, например, язык Shell операционной системы Unix.

Графический пользовательский интерфейс предоставляет пользователю возможности:

- обращаться к ПС путем выбора на экране подходящего графического или текстового объекта;
- получать от ПС информацию на экране в виде графических и текстовых объектов;
- осуществлять прямые манипуляции с графическими и текстовыми объектами, представленными на экране.

Графический пользовательский интерфейс позволяет:

- размещать на экране множество различных окон, в которые можно выводить информацию независимо;
- использовать графические объекты, называемые пиктограммами (или иконами), для обозначения различных информационных объектов или процессов;

– использовать экранный указатель для выбора объектов (или их элементов), размещенных на экране; экранный указатель управляется (перемещается) с помощью клавиатуры или мыши.

Достоинством графического пользовательского интерфейса является возможность создания удобной и понятной пользователю модели взаимодействия с ПС (панель управления, рабочий стол и проч.) без необходимости изучения какого-либо специального языка. Однако его разработка требует больших трудозатрат, сравнимых с трудозатратами по создания самого ПС. Кроме того, возникает серьезная проблема по переносимости ПС на другие операционные системы, так как графический интерфейс существенно зависит от возможностей (графической пользовательской платформы), предоставляемых операционной системой для его создания.

Графический пользовательский интерфейс обобщает такие виды пользовательского интерфейса, как интерфейс типа меню и интерфейс прямого манипулирования.

1.6.3. Обеспечение эффективности программного средства

Эффективность ПС обеспечивается принятием подходящих решений на разных этапах его разработки, начиная с разработки его архитектуры. Особенно сильно на эффективность ПС (особенно по памяти) влияет выбор структуры и представления данных. Но и выбор алгоритмов, используемых в тех или иных программных модулях, а также особенности их реализации (включая выбор языка программирования) может существенно повлиять на эффективность ПС. При этом постоянно приходится разрешать противоречие между временно́й эффективностью и эффективностью по памяти (ресурсам). Поэтому весьма важно, чтобы в спецификации качества были явно указаны приоритеты или количественное соотношение между показателями этих примитивов качества. Следует также иметь в виду, что разные программные модули по-разному влияют на эффективность

ПС в целом: одни модули могут сильно влиять на временну́ю эффективность и практически не влиять на эффективность по памяти, а другие могут существенно влиять на общий расход памяти, не оказывая заметного влияния на время работы ПС. Более того, это влияние (прежде всего, в отношении временно́й эффективности) заранее (до окончания реализации ПС) далеко не всегда можно правильно оценить.

С учетом сказанного, рекомендуется придерживаться следующих принципов для обеспечения эффективности ПС [2]:

- сначала нужно разработать надежное ПС, а потом уж заниматься доведением его эффективности до требуемого уровня в соответствии с его спецификацией качества;
- для повышения эффективности ПС, прежде всего, нужно использовать оптимизирующий компилятор – это может обеспечить требуемую эффективность;
- если эффективность ПС не удовлетворяет спецификации его качества, то найдите самые критические модули с точки зрения требуемой эффективности ПС; эти модули и попытайтесь оптимизировать в первую очередь путем их ручной переделки;
- не следует заниматься оптимизацией модуля, если этого не требуется для достижения требуемой эффективности ПС.

Для отыскания критических модулей с точки зрения временно́й эффективности ПС потребуется получить распределение по модулям времени работы ПС путем соответствующих измерений во время выполнения ПС. Это может быть сделано с помощью динамического анализатора (специального программного инструмента), который может определить частоту обращения к каждому модулю в процессе применения ПС.

1.6.4. Обеспечение сопровождаемости программного средства

Обеспечение сопровождаемости ПС сводится к обеспечению изучаемости ПС и к обеспечению его модифицируемости.

Изучаемость (подкритерий качества) ПС определяется составом и качеством документации по сопровождению ПС и выражается через такие примитивы качества ПС как С-документированность, информативность, понятность, структурированность и удобочитаемость. Последние два примитива качества и, в значительной степени, понятность связаны с текстами программных модулей. Вопрос о документации по сопровождению будет обсуждаться позднее. Здесь мы лишь сделаем некоторые общие рекомендации относительно текстов программ (модулей).

При окончательном оформлении текста программного модуля целесообразно придерживаться следующих рекомендаций, определяющих практически оправданный стиль программирования [2]:

- используйте в тексте модуля комментарии, проясняющие и объясняющие особенности принимаемых решений; по возможности, включайте комментарии, хотя бы в краткой форме, на самой ранней стадии разработки текста модуля;
- используйте осмысленные, мнемонические и устойчиво различимые имена (оптимальная длина имени – 4–12 литер, цифры – в конце), не используйте сходные имена и ключевые слова;
- соблюдайте осторожность в использовании констант (уникальная константа должна иметь единственное вхождение в текст модуля: при ее объявлении или, в крайнем случае, при инициализации переменной в качестве константы);
- не бойтесь использовать необязательные скобки – они обходятся дешевле, чем ошибки;

– размещайте не больше одного оператора в строке; для прояснения структуры модуля используйте дополнительные пробелы (отступы) в начале каждой строки; этим обеспечивается удобочитаемость текста модуля;
– избегайте трюков, т.е. таких приемов программирования, когда создаются фрагменты модуля, основной эффект которых не очевиден или скрыт (завуалирован), например, побочные эффекты функций.

Структурированность текста модуля существенно упрощает его понимание. Обеспечение этого примитива качества подробно обсуждалось ранее. Удобочитаемость текста модуля может быть обеспечена автоматически путем применения специального программного инструмента – форматера.

Модифицируемость (подкритерий качества) ПС определяется, частично, некоторыми свойствами документации, и свойствами, реализуемые программным путем, и выражается через такие примитивы качества ПС как расширяемость, модифицируемость, структурированность и модульность.

Расширяемость обеспечивается возможностями автоматически настраиваться на условия применения ПС по информации, задаваемой пользователем. К таким условиям относятся, прежде всего, конфигурация компьютера, на котором будет применяться ПС (в частности, объем и структура его памяти), а также требования конкретного пользователя к функциональным возможностям ПС (например, требования, которые определяют режим применения ПС или конкретизируют структуру информационной среды). К этим возможностям можно отнести и возможность добавления к ПС определенных компонент. Для реализации таких возможностей в ПС часто включается дополнительная компонента (подсистема), называемая инсталлятором. Инсталлятор осуществляет прием от

пользователя необходимой информации и настройку ПС по этой информации. Обычно решение о включении в ПС такой компоненты принимается в процессе разработки архитектуры ПС.

Модифицируемость (примитив качества) обеспечивается такими свойствами документации и свойствами, реализуемые программным путем, которые облегчают внесение изменений и доработок в документацию и программы ПС ручным путем (возможно, с определенной компьютерной поддержкой). В спецификации качества могут быть указаны некоторые приоритетные направления и особенности развития ПС. Эти указания должны быть учтены при разработке архитектуры ПС и модульной структуры его программ. Общая проблема сопровождения ПС – обеспечить, чтобы все его компоненты (на всех уровнях представления) оставались согласованными в каждой новой версии ПС. Этот процесс обычно называют управлением конфигурацией (configuration management).Чтобы помочь управлению конфигурацией, необходимо, чтобы связи и зависимости между документами и их частями фиксировать в специальной документации по сопровождению. Эта проблема усложняется, если в процессе доработки может находиться сразу несколько версий ПС (в разной степени завершенности). Тогда без компьютерной поддержки довольно трудно обеспечить согласованность документов в разных конфигурациях. Поэтому в таких случаях в ПС включается дополнительная компонента (подсистема), называемая конфигуратором. С такой компонентой связывают специальную базу данных (или специальный раздел в базе данных), в которой фиксируются связи и зависимости между документами и их частями для всех версий ПС. Обычно решение о включении в ПС такой компоненты принимается в процессе разработки архитектуры ПС. Для обеспечения этого примитива качества в документацию по сопровождению включают специальное руководство, которое описывает, какие части ПС являются аппаратно- и про-

граммно-зависимыми, и как возможное развитие ПС учтено в его строении (конструкции).

Структурированность и модульность упрощают ручную модификацию программ ПС.

1.6.5. Обеспечение мобильности

Проблема мобильности возникает из-за того, что быстрое развитие компьютерной техники и аппаратных средств делает жизненный цикл многих больших программных средств (программных систем) намного продолжительнее периода «морально» оправданного существования компьютеров и аппаратуры, для которых первоначально создавались эти программные средства. Поэтому обеспечение критерия мобильности для таких ПС является весьма важной задачей.

Мобильность ПС определяется такими примитивами качества ПС как независимость от устройств, автономность, структурированность и модульность.

Если бы ПС обладало таками примитивами качества, как независимость от устройств и автономность, и его программы были бы представлены на машинно-независимом языке программирования, то перенос ПС в другую среду обеспечивался бы перетрансляцией (перекомпиляцией) его программ в этой среде. Однако трудно представить реальное ПС, обладающее таким качеством. Тем не менее, таким качеством могут обладать отдельные части программ ПС и даже весьма значительные. А это уже явный намек на то, каким путем следует добиваться мобильности ПС.

Если ПС зависит от устройств (аппаратуры), то в спецификации качества должна быть описана эта компьютерно-аппаратная среда (будем ее называть аппаратной платформой [12]). Избавиться от этой зависимости можно за счет такого примитива качества ПС как автономность. Как правило, ПС строится в рамках некоторой операционной системы (ОС), кото-

рая может спрятать специфику аппаратной платформы и, тем самым, сделать ПС независимым от устройств. Но тогда ПС не будет обладать свойством автономности. В этом случае в спецификации качества должна быть описана эта программная среда, над которой строится ПС (будем эту среду называть операционной платформой [12]). Таким образом, мобильность ПС будет непосредственно связано с мобильностью используемой ОС: перенос ПС на другую аппаратную платформу осуществляется автоматически, если будет осуществлен перенос на эту платформу используемой ОС. Но обеспечение мобильности ОС является самостоятельной и довольно трудной задачей.

Таким образом, для обеспечения мобильности ПС нужно решить две задачи:

- выделение по возможности наибольшей части программ ПС, обладающей свойствами независимости от устройств и автономности (другими словами, независимой от аппаратно-операционной платформы);
- обеспечение сопровождаемости для остальных частей программ ПС.

Для решения этих задач целесообразно выбрать в качестве архитектуры ПС слоистую систему (рис. 1.3):

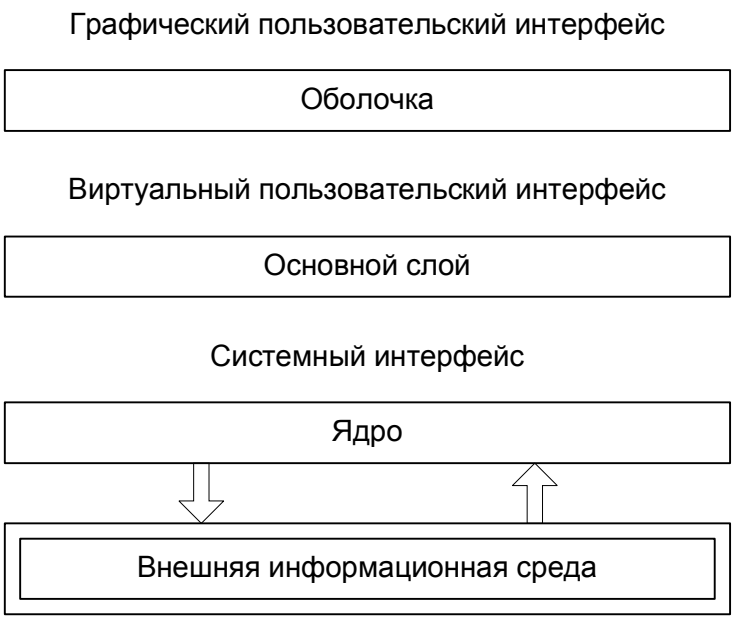

Рисунок 1.3. Слоистая система ПС

Основной слой, реализующий основные функции ПС, должен быть независимым от аппаратно-операционной платформы. Выделяется также слой (часто называемый ядром ПС), который включает программные модули, зависящие от аппаратно-операционной платформы. Этот слой должен обеспечивать, в частности, доступ к внешней информационной среде ПС. Между этими слоями должен быть определен интерфейс, независимый от аппаратно-операционной платформы и обеспечивающий правила обращения из основного слоя к модулям ядра. Будем называть этот интерфейс системным. Использование графических пользовательских интерфейсов требует выделение еще одного программного слоя, зависящего от той части аппаратно-операционной платформы (графической пользовательской платформы), на которой строятся пользовательские интерфейсы. Будем называть этот слой оболочкой ПС. Между оболочкой и основным слоем также должен быть определен интерфейс, независимый от графической пользовательской платформы и обеспечивающий правила обращения из оболочки к модулям основного слоя.

Модульность ПС позволяет сформировать указанные слои, выделяя программные модули с требуемыми свойствами и распределяя их между указанными слоями. Модульность и структурированность оболочки и ядра позволяют обеспечить эти слои свойством модифицируемости. При этом желательно, чтобы каждый модуль этих слоев был ориентирован на реализацию каких-либо функций управления четко выделенной компоненты аппаратно-операционной среды. Для этого используются такие методы как унификация интерфейсов, стандартизация протоколов и проч. [12].

ЛИТЕРАТУРА

1. Гоулд И.Г., Тутилл Дж.С. Терминологическая работа IFIP (Международная федерация по обработке информации) и ICC (Международный вычислительный центр) // Журн. вычисл. матем. и матем. физ., 1965, №2. – С. 377–386.

2. Майерс Г. Надежность программного обеспечения. – М.: Мир, 1980.

3. Турский В. Методология программирования. – М.: Мир, 1981.

4. Дейкстра Э. Дисциплина программирования. – М.: Мир, 1978.

5. Жоголев Е.А. Система программирования с использованием библиотеки подпрограмм / Система автоматизация программирования. – М.: Физматгиз, 1961. – С. 15–52.

6. Кауфман В.Ш. Языки программирования. Концепции и принципы. – М.: Радио и связь, 1993.

7. Требования и спецификации в разработке программ. – М.: Мир, 1984.

8. Буч Г. Объектно-ориентированное проектирование с примерами применения. – М.: Конкорд, 1992.

9. Sommerville I. Software Engineering. – Addison-Wesley Publishing Company, 1992.

10. Зиглер К. Методы проектирования программных систем. – М.: Мир, 1985.

11. Жоголев Е.А. Введение в технологию программирования. – М.: «ДИАЛОГ-МГУ», 1994.

12. Липаев В.В. Качество программного обеспечения. – М.: Финансы и статистика, 1983.

ГЛАВА 2. ОСНОВЫ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Решение профессиональных задач предполагает знание принципов системного, структурного анализа и функционального моделирования информационных систем (построения диаграмм, разработки структуры ИС), принципов моделирования данных; умение проектировать базы данных ИС; используя процедурное и объектно-ориентированное программирование уметь проектировать отдельные виды обеспечения ИС; обеспечивать возможности развития и адаптации профессионально-ориентированных информационных систем на всех стадиях их жизненного цикла: создания информационно-логических моделей объектов, разработки нового программного и информационного обеспечения в предметной области; стыковки информационных систем из разных предметных областей в связи с появляющимися новыми задачами; перевода систем на новые аппаратные и информационные платформы.

Цель курса «Основы проектирования информационных систем» – формирование навыков самостоятельного практического применения современных средств и методов проектирования информационных систем (ИС), на основе использования визуального проектирования и CASE – средств. В данной главе изложен следующий материал:

1. Проектирование информационной системы.
2. Требования к эффективности и надежности проектных решений.
3. Основные компоненты технологии проектирования ИС.
4. Методы и средства проектирования ИС.
5. Краткая характеристика применяемых технологий проектирования.
6. Требования, предъявляемые к технологии проектирования ИС и выбор технологии проектирования ИС. Стадии и этапы процесса проекти-

рования ИС. Состав работ на предпроектной стадии, состав проектной документации.

Приведены примеры проектирования информационной системы.

2.1. Проектирование информационной системы. Понятия и структура проекта ИС

2.1.1. Основные понятия и определения

Информационная система (ИС) — коммуникационная система по сбору, передаче, обработке информации для принятия решений и реализации функций управления (в интересах достижения поставленной цели).

Под *автоматизированной информационной системой* (АИС) понимается прикладная программная подсистема, ориентированная на сбор, хранение, поиск и обработку текстовой и/или фактографической информации.

Каждая ИС включает компоненты:

- структура системы – множество элементов системы и взаимосвязей между ними (например, организационная и производственная структура фирмы);
- функции каждого элемента системы;
- вход и выход каждого элемента системы (материальные или информационные потоки, поступающие в систему или выводимые ею);
- цели и ограничения системы.

Проект информационной системы – это ограниченное по времени целенаправленное изменение отдельной системы с четко определенными целями, установленными требованиями к срокам завершения, результатам, риску, рамкам расходования средств и ресурсов, организационной структуре.

Тип проекта определяется по основным сферам деятельности, в которых осуществляется проект. Можно выделить пять основных типов проекта [1–3]: технический, организационный, экономический, социальный, смешанный.

ИС имеют ряд существенных отличий от стандартных прикладных программ и систем. В зависимости от предметной области ИС могут весьма значительно различаться по своим функциям, архитектуре, реализации.

Проектирование информационных систем всегда начинается с определения цели проекта. Основная задача любого успешного проекта заключается в том, чтобы на момент запуска системы и в течение всего времени ее эксплуатации можно было обеспечить:

- требуемую функциональность системы и степень адаптации к изменяющимся условиям ее функционирования;
- требуемую пропускную способность системы;
- требуемое время реакции системы на запрос;
- безотказную работу системы в требуемом режиме, иными словами – готовность и доступность системы для обработки запросов пользователей;
- простоту эксплуатации и поддержки системы;
- необходимую безопасность.

Производительность является главным фактором, определяющим эффективность системы. Хорошее проектное решение служит основой высокопроизводительной системы.

Задачи и функции ИС. ИС предназначены для решения задач обработки данных, автоматизации работ, выполнения поиска информации и отдельных задач, основанных на методах искусственного интеллекта.

Задачи обработки данных обеспечивают обычно рутинную обработку и хранение информации с целью выдачи (регулярной или по запросам)

сводной информации, которая может потребоваться для управления объектом.

Автоматизация работ предполагает наличие в ЭИС системы ведения картотек, системы обработки текстовой информации, системы машинной графики, системы электронной почты и связи.

Поисковые задачи имеют свою специфику, и информационный поиск представляет собой интегральную задачу, которая рассматривается независимо от экономики или иных сфер использования найденной информации.

Алгоритмы *искусственного интеллекта* необходимы для задач принятия управленческих решений, основанных на моделировании действий специалистов предприятия при принятии решений.

В процессе разработки ИС[2] решаются *две основные задачи*:
1) разработка базы данных, предназначенной для хранения информации;
2) разработка графического интерфейса пользователя клиентских приложений.

2.1.2. Преимущества электронного документооборота

Документ – это совокупность трех составляющих [3]:

- физическая регистрация информации;
- форма представления информации;
- активизация определенной деятельности.

Именно некоторая деятельность и превращает информацию в документ. Но документ перестает существовать, если в дальнейшем не подразумевает процедуры обработки.

Документ [4] – слабоструктурированная совокупность блоков или объектов информации, понятная человеку. В общем случае обойтись без

---

[2] Разработка ИС относится к техническим проектам

документов пока нельзя. Сам по себе документ, независимо от того, обычная ли это бумага или электронный бланк, проблем корпорации не решает – первичны бизнес-процессы и четкий контроль за выполнением проекта.

Документ занимает определенное место в процессе некоторой деятельности на границе разделяемых функций исполнения. Поэтому правильно рассматривать *документ как инструмент распределения функций между работниками*.

К *основным преимуществам электронного документооборота* можно отнести следующие:

– полный контроль за перемещением и эволюцией документа, регламентация доступа и способ работы пользователей с различными документами и их отдельными частями (рис. 2.1);

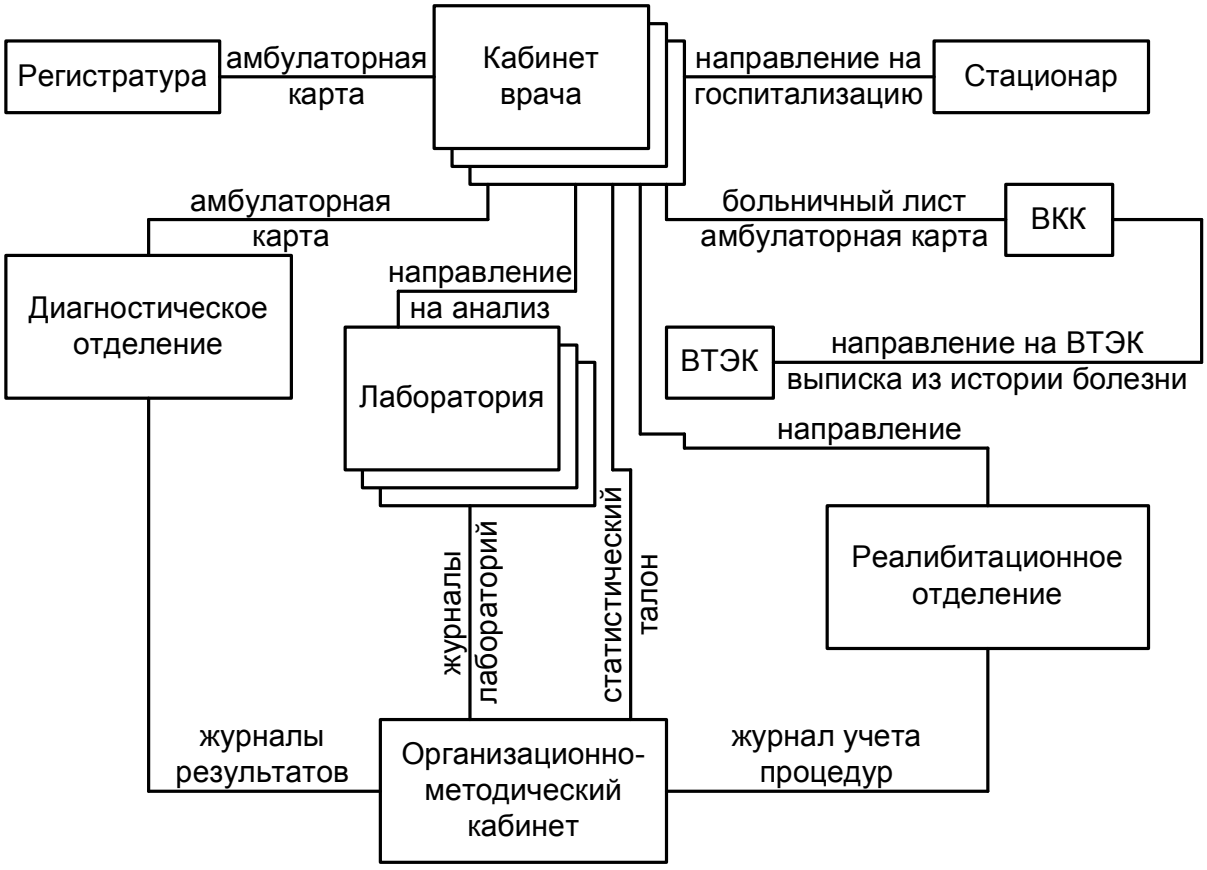

Рисунок 2.1. Схема информационного сопровождения пациента поликлиники

- уменьшение расходов на управление за счет высвобождения (на 90% и более) людских ресурсов, занятых различными видами обработки бумажных документов, снижение бюрократической волокиты за счет маршрутизированного перемещения документов и жесткого контроля за порядком и сроками прохождения документов;
- быстрое создание новых документов из существующих;
- поддержка одновременной работы многих пользователей с одним и тем же документом, предотвращение его потери или порчи;
- сокращение времени поиска нужных документов.

Использование АИС может рассматриваться в качестве базы для общего совершенствования управления предприятием. При этом управление предприятием реализует следующие основные функции:

- обслуживание клиентов;
- разработка продукции;
- учет и контроль за деятельностью предприятия;
- финансовое обеспечение деятельности предприятия и т.д.

Комплексная автоматизация этих функции требует создания единого информационного пространства предприятия, в котором сотрудники и руководство могут осуществлять свою деятельность, руководствуясь едиными правилами представления и обработки информации в документном и бездокументном виде.

Для этого в рамках предприятия требуется создать единую ИС по управлению информацией или единую систему управления документами, включающую возможности:

- удаленной работы, когда члены одного коллектива могут работать в разных комнатах здания или в разных зданиях;

- доступа к информации, когда разные пользователи должны иметь доступ к одним и тем же данным без потерь в производительности и независимо от своего местоположения в сети;
- средств коммуникации, например: электронная почта, факс, печать документов;
- сохранения целостности данных в общей базе данных;
- полнотекстового и реквизитного поиска информации;
- открытость системы, когда пользователи должны иметь доступ к привычным средствам создания документов и к уже существующим документам, созданным в других системах;
- защищенность информации;
- удобства настройки на конкретные задачи пользователей;
- масштабируемости системы для поддержки роста организаций и защиты вложенных инвестиций и т.д.

2.1.3. Области применения и примеры реализации информационных систем

Бухгалтерский учет – классическая и наиболее часто реализуемая на сегодняшний день область применения информационных технологий (ИТ). Задача бухгалтерского учета довольно легко формализуется, так что разработка систем автоматизации бухгалтерского учета не представляет технически сложной проблемы. В то же время разработка систем автоматизации бухгалтерского учета весьма трудоемка, так как а) к системам бухгалтерского учета предъявляются повышенные требования в отношении надежности, максимальной простоты и удобства в эксплуатации; б) постоянные изменения в бухгалтерском и налоговом учете.

Управление финансовыми потоками. Внедрение ИТ в управление финансовыми потоками также обусловлено критичностью этой области управления предприятия к ошибкам. Неправильно построив систему рас-

четов с поставщиками и потребителями, можно спровоцировать кризис наличности даже при налаженной сети закупки, сбыта и хорошем маркетинге. И наоборот, точно просчитанные и жестко контролируемые условия финансовых расчетов могут существенно увеличить оборотные средства фирмы.

Управление складом, ассортиментом, закупками. Автоматизация процесса анализа движения товара позволит решать задачу получения максимальной прибыли (отследив и зафиксировав те двадцать процентов ассортимента, которые приносят восемьдесят процентов прибыли).

Оптимальное **управление производственным процессом** представляет собой очень трудоемкую задачу. Автоматизированное решение подобной задачи дает возможность грамотно планировать, учитывать затраты, проводить техническую подготовку производства, оперативно управлять процессом выпуска продукции в соответствии с производственной программой и технологией.

Очевидно, что чем крупнее производство, тем большее число бизнес-процессов участвует в создании прибыли, а значит, использование информационных систем жизненно необходимо.

Управление маркетингом подразумевает сбор и анализ данных о фирмах-конкурентах, их продукции и ценовой политике, а также моделирование параметров внешнего окружения для определения оптимального уровня цен, прогнозирования прибыли и планирования рекламных кампаний. Решения большинства этих задач могут быть формализованы и представлены в виде ИС, позволяющей существенно повысить эффективность маркетинга.

Документооборот является очень важным процессом деятельности любого предприятия. Хорошо отлаженная система учетного документооборота отражает реально происходящую на предприятии текущую производственную деятельность и дает управленцам возможность воздейство-

вать на нее. Поэтому автоматизация документооборота позволяет повысить эффективность управления.

ИС, решающая **задачи оперативного управления предприятием**, строится на основе базы данных, в которой фиксируется вся возможная информация о предприятии. ИС оперативного управления включает в себя массу программных решений по автоматизации бизнес-процессов, имеющих место на конкретном предприятии. Процесс принятия управленческого решения состоит из фаз:

- получение информации;
- переработка информации;
- анализ, подготовка и принятие решения.

Все эти этапы самым тесным образом связаны с документационным обеспечением процессов управления, проектирования и производства (рис. 2.2).

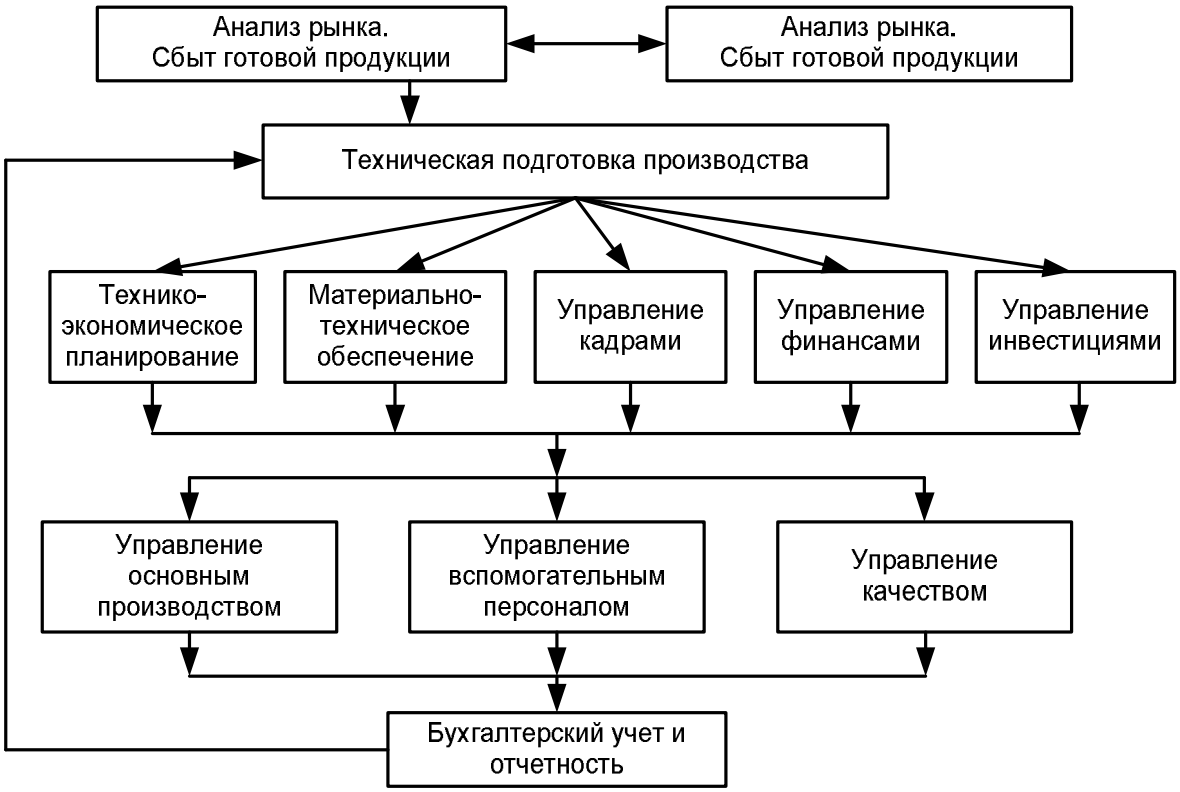

Рисунок 2.2. Пример информационной системы промышленного предприятия

2.1.4. Требования, предъявляемые к информационным системам

1. **Гибкость** или способность к адаптации и дальнейшему развитию подразумевают возможность приспособления ИС к новым условиям, новым потребностям предприятия. Выполнение этих условий возможно, если на этапе разработки ИС использовались общепринятые средства и методы документирования.

2. **Надежность** подразумевает ее функционирование без искажения информации, потери данных по «техническим причинам». Требование надежности обеспечивается созданием резервных копий хранимой информации, выполнения операций протоколирования, поддержанием качества каналов связи и физических носителей информации, использованием современных программных и аппаратных средств. Сюда же следует отнести защиту от случайных потерь информации в силу недостаточной квалификации персонала.

3. **Эффективность** обеспечивается оптимизацией данных и методов их обработки, применением оригинальных разработок, идей, методов проектирования (в частности, спиральной модели проектирования ИС, о которой речь пойдет ниже). Система является эффективной, если с учетом выделенных ей ресурсов она позволяет решать возложенные на нее задачи в минимальные сроки.

4. **Безопасность**. Под безопасностью подразумевается свойство системы, в силу которого посторонние лица не имеют доступа к информационным ресурсам организации. Согласно действующему в России законодательству, ответственность за вред, причиненный ненадлежащим качеством работ или услуг, несет исполнитель, то есть в нашем случае разработчик ИС. Поэтому ненадлежащее обеспечение безопасности ИС заказчика в худшем случае обернется для исполнителя судебным преследованием, в лучшем – потерей клиента и утратой деловой репутации.

Требование безопасности обеспечивается современными средствами разработки ИС, современной аппаратурой, методами защиты информации, применением паролей и протоколированием, постоянным мониторингом состояния безопасности операционных систем и средств их защиты.

2.1.5. Жизненный цикл информационных систем

Жизненный цикл информационных систем (ЖЦ ИС) – это непрерывный процесс, начинающийся с момента принятия решения о необходимости создания ИС и заканчивающийся в момент полного ее изъятия из эксплуатации.

Структура ЖЦ ПО по стандарту ISO/IEC[3] 12207 базируется на трех группах процессов:

- основные процессы ЖЦ (приобретение, поставка, разработка, эксплуатация, сопровождение);
- вспомогательные процессы, обеспечивающие выполнение основных процессов (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, оценка, аудит, решение проблем);
- организационные процессы (управление проектами, создание инфраструктуры проекта, определение, оценка и улучшение самого ЖЦ, обучение).

Среди *основных процессов* ЖЦ наибольшую важность имеют три: разработка, эксплуатация и сопровождение.

Разработка ИС включает в себя все работы по созданию информационного ПО и его компонентов в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для тестирования разработанных

[3] ISO – International Organization of Standardization – Международная организация по стандартизации, IEC – International Electrotechnical Commission – Международная комиссия по электротехнике

программных продуктов, и разработку материалов, необходимых для организации обучения персонала и т.д.

Эксплуатация включает в себя работы по внедрению компонентов ИС в эксплуатацию, в том числе конфигурирование базы данных и рабочих мест пользователей, обеспечение пользователей эксплуатационной документацией, проведение обучения персонала и т.д., и непосредственно эксплуатацию, в том числе локализацию проблем и устранение причин их возникновения, модификацию ПО в рамках установленного регламента, подготовку предложений по совершенствованию, развитию и модернизации системы.

Сопровождение включает в себя техническую поддержку ИС.

Стадии (этапы) ЖЦ:

1. *Планирование и анализ требований* (предпроектная стадия или стадия системного анализа): исследование и анализ существующей ИС, определение требований к создаваемой ИС, оформление технико-экономического обоснования (ТЭО) и технического задания (ТЗ) на разработку ИС.

2. *Проектирование (техническое проектирование, логическое проектирование)*: разработка в соответствии со сформулированными требованиями состава автоматизируемых функций (функциональная архитектура) и состава обеспечивающих подсистем (системная архитектура), оформление технического проекта ИС.

Второй и третий этапы нередко объединяют в одну стадию, называемую технорабочим проектированием или системным синтезом.

3. *Реализация (рабочее проектирование, физическое проектирование, программирование)*. Разработка и настройка программ, наполнение базы данных, создание рабочих инструкций для персонала, оформление рабочего проекта.

4. *Внедрение (тестирование, опытная эксплуатация)*: комплексная отладка подсистем, обучение персонала, поэтапное внедрение ИС в эксплуатацию (по подразделениям экономического объекта), оформление акта о приемо-сдаточных испытаниях ИС.

5. *Эксплуатация ИС (сопровождение, модернизация)*: сбор статистики о функционировании ИС, исправление ошибок и недоработок, оформление требований к модернизации ИС и ее выполнение (повторение стадий 2–5).

Модели жизненного цикла ПО ИС. Под моделью ЖЦ ИС понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач, выполняемых на протяжении ИС. Модель ЖЦ зависит от специфики ИС и специфики условий, в которых последняя создается и функционирует.

Каскадная модель (рис. 2.3) предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе.

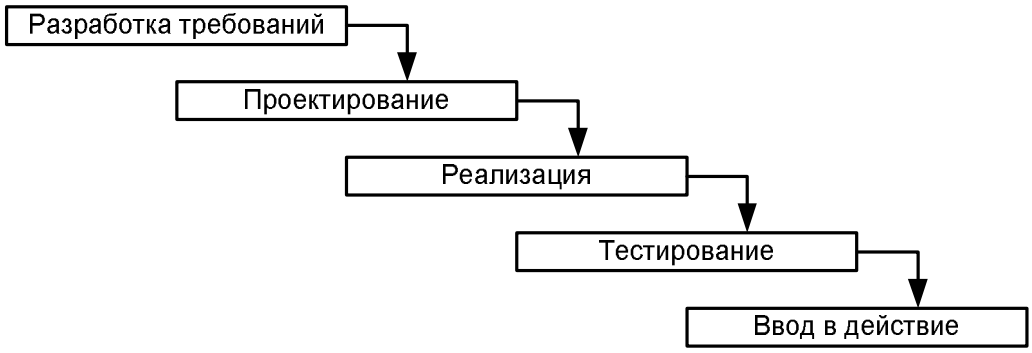

Рисунок 2.3. Каскадная модель ЖЦ ИС

Итерационная модель (поэтапная модель с промежуточным контролем, рис. 2.4). Разработка ИС ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют учитывать реально существующее взаимовлияние результатов разработки на различных

этапах; время жизни каждого из этапов растягивается на весь период разработки.

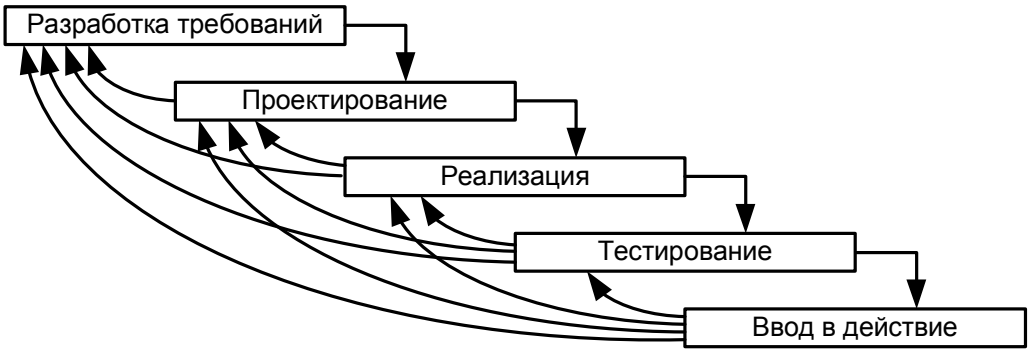

Рисунок 2.4. Поэтапная модель с промежуточным контролем

Спиральная модель (рис. 2.5). Виток спирали определяет очередную версию продукта, в которой уточняются требования проекта, определяется его качество и планируются работы следующего витка. Особое внимание уделяется начальным этапам разработки - анализу и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (макетирования).

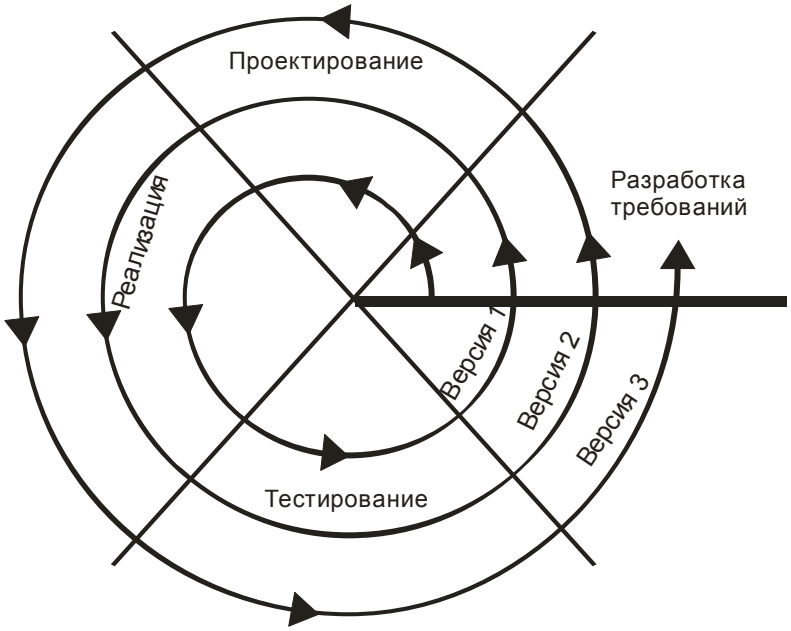

Рисунок 2.5. Спиральная модель ЖЦ ИС

2.1.6. Этапы разработки автоматизированных информационных систем

Запланируем комплекс работ по созданию информационной системы в соответствии с типовыми этапами разработки АИС, краткая характеристика которых приведена в табл. 2.1, а последовательность трансформации бизнес модели в объекты базы данных в табл. 2.2.

Таблица 2.1.

Этапы проектирования АИС и их характеристики

№	Наименование этапа	Основные характеристики
1.	Разработка и анализ бизнес-модели	Определяются основные задачи АИС, проводится декомпозиция задач по модулям и определяются функции, с помощью которых решаются эти задачи. Описание функций осуществляется на языке производственных (описание процессов предметной области), функциональных (описание форм обрабатываемых документов) и технических требований (аппаратное, программное, лингвистическое обеспечение АИС). *Метод решения*: Функциональное моделирование. *Результат*: 1. Концептуальная модель АИС, состоящая из описания предметной области, ресурсов и потоков данных, перечень требований и ограничений к технической реализации АИС. 2. Аппаратно-технический состав создаваемой АИС.
2.	Формализация бизнес-модели, разработка логической модели бизнес-процессов	Разработанная концептуальная модель формализуется в виде логической модели АИС. *Метод решения*: Разработка диаграммы «сущность-связь» (ER (Entity-Relationship) – CASE-диаграммы). *Результат*: Разработанное информационное обеспечение АИС: схемы и структуры данных для всех уровней модульности АИС, документация по логической структуре АИС, сгенерированные скрипты для создания объектов БД.

Продолжение таблицы 2.1.

№	Наименование этапа	Основные характеристики
3.	Выбор лингвистического обеспечения, разработка программного обеспечения АИС.	Разработка АИС: выбирается лингвистическое обеспечение (среда разработки – инструментарий), проводится разработка программного и методического обеспечения. Разработанная на втором этапе логическая схема воплощается в реальные объекты, при этом логические схемы реализуются в виде объектов базы данных, а функциональные схемы – в пользовательские формы и приложения. *Метод решения*: Разработка программного кода с использованием выбранного инструментария. *Результат*: Работоспособная АИС.
4.	Тестирование и отладка АИС	На данном этапе осуществляется корректировка информационного, аппаратного, программного обеспечения, проводится разработка методического обеспечения (документации разработчика, пользователя) и т.п. *Результат*: Оптимальный состав и эффективное функционирование АИС. Комплект документации: разработчика, администратора, пользователя.
5.	Эксплуатация и контроль версий	Особенность АИС созданных по архитектуре клиент сервер является их многоуровневость и многомодульность, поэтому при их эксплуатации и развитии на первое место выходят вопросы контроля версий, т.е. добавление новых и развитие старых модулей с выводом из эксплуатации старых. Например, если ежедневный контроль версий не ведется, то, как показала практика, БД АИС за год эксплуатации может насчитывать более 1000 таблиц, из которых эффективно использоваться будет лишь 20–30%. *Результат*: Наращиваемость и безизбыточный состав гибкой, масштабируемой АИС

Замечание. На обнаружение ошибок, допущенных на стадии системного проектирования (2-я и 3-я стадии), расходуется примерно в два раза больше времени, чем на последующих фазах, а их исправление обходится

в пять раз дороже. Поэтому на начальных стадиях проекта разработку следует выполнять особенно тщательно. Наиболее часто на начальных фазах допускаются следующие ошибки:

- ошибки в определении интересов заказчика;
- концентрация на маловажных, сторонних интересах;
- неправильная интерпретация исходной задачи;
- неправильное или недостаточное понимание деталей;
- неполнота функциональных спецификаций (системных требований);
- ошибки в определении требуемых ресурсов и сроков;
- редкая проверка на согласованность этапов и отсутствие контроля со стороны заказчика (нет привлечения заказчика).

Таблица 2.2.

Последовательность трансформации бизнес-модели в объекты базы данных

Последовательность работ				
Разработка и анализ бизнес-модели. Определение и назначение функций ИС	Моделирование процессов предметной области	Разработка модели сущность-связь. Разработка CASE-диаграммы	Исследование бизнес-процессов аналогичной модели	Формирование объектов БД
→	→	→	→	
Методы функционального анализа	Oracle Designer, Power Designer, Power Soft и др.	Oracle Designer, Erwin, BPwin и др.	Oracle Designer, Silverun-RDM и др.	SQL и др.средства выполнения SQL-скриптов
Средства разработки				

Порядок выполнения практических работ по проектированию ИС может быть следующим.

1. В процессе выполнения практического задания проводится анализ и оформление результатов обследования деятельности предприятия (орга-

низации), и на его основе разрабатываются документы, необходимые для настройки типовой ИС. По итогам проведения обследования обычно формируются следующие документы:

– *Предварительная информация*. Предполагается, что в начале обследования проведен предварительный сбор информации о компании, по итогам которого получены следующие данные:
  - краткая информация о компании (профиль клиента);
  - цели проекта;
  - подразделения и пользователи системы.

На основе предварительной информации сформировано и согласовано с заказчиком общее представление о проекте:

– *Видение выполнения проекта* и границы проекта должно быть реализовано в документе, кратко описывающем, в каких подразделениях и в какой функциональности будет внедряться ИС.
– *Отчет об обследовании* содержит следующие разделы:
  1) Анализ существующего уровня автоматизации: составляется список программного обеспечения, используемого в компании, и приводятся данные об использовании этих пакетов в каждом из подразделений организации.
  2) Общие требования к ИС: формулируются общие требования к функциональности разрабатываемой системы.
  3) Формы документов: устанавливается перечень и структура документов, которые должны формироваться системой.
  4) Описание системы учета включает в себя следующие документы:
     - учетная политика компании;
     - план счетов и используемых аналитик;

- список типовых хозяйственных операций и их отражение в проводках.

5) Описание справочников: по каждому справочнику, проектируемому в системе, дается описание необходимой иерархической структуры.
6) Организационная диаграмма: используется для отражения организационной структуры подразделений предприятия и их зон ответственности.
7) Описание состава автоматизируемых бизнес-процессов: все бизнес-процессы компании должны быть перечислены в общем списке и каждый должен иметь свой уникальный номер.
8) Диаграммы прецедентов: для выделения автоматизируемых бизнес-процессов и их основных исполнителей используются диаграммы прецедентов.
9) Физическая диаграмма: служит для того, чтобы описать взаимодействие организации на верхнем уровне с внешними контрагентами.
10) Описания бизнес-процессов (книга бизнес-процессов содержит подробное описание автоматизируемых бизнес-процессов. Модели бизнес-процессов позволяют выделить отдельные операции, выполнение которых должно поддерживаться разрабатываемой ИС).

2. На последнем этапе осуществляется отображение модели предметной области на функциональность типовой системы – выбираются модули системы для поддержки выделенных операций, определяются особенности их настройки, выявляется необходимость разработки дополнительных программных элементов.

2.1.7. Классификация информационных систем

Информационные системы можно классифицировать по целому ряду различных признаков. В основу рассматриваемой классификации положены наиболее существенные признаки, определяющие функциональные возможности и особенности построения современных систем [1–3]. В зависимости от объема решаемых задач, используемых технических средств, организации функционирования, информационные системы делятся на ряд групп (классов) (рис. 1.6). Данная классификация не претендует на полноту, но является наиболее часто используемой:

По масштабу информационные системы подразделяются на следующие группы:

1. *Одиночные информационные системы* реализуются на автономном персональном компьютере (сеть не используется). Такая система может содержать несколько простых приложений, связанных общим информационным фондом, и рассчитана на работу одного пользователя или группы пользователей, разделяющих по времени одно рабочее место. Подобные приложения создаются с помощью так называемых настольных, или локальных, систем управления базами данных (СУБД). Среди локальных СУБД наиболее известными являются Clarion, Clipper, FoxPro, Paradox, dBase и Microsoft Access.

2. *Групповые информационные системы* ориентированы на коллективное использование информации членами рабочей группы и чаще всего строятся на базе локальной вычислительной сети. При разработке таких приложений используются серверы баз данных (называемые также SQL-серверами) для рабочих групп. Существует довольно большое количество различных SQL-серверов как коммерческих, так и свободно распространяемых. Среди них наиболее известны такие серверы баз данных, как Oracle, DB2, Microsoft SQL Server, InterBase, Sybase, Informix.

3. *Корпоративные информационные системы* являются развитием систем для рабочих групп, они ориентированы на крупные компании и могут поддерживать территориально разнесенные узлы или сети. В основном они имеют иерархическую структуру из нескольких уровней. Для таких систем характерна архитектура клиент-сервер со специализацией серверов или же многоуровневая архитектура. При разработке таких систем могут использоваться те же серверы баз данных, что и при разработке групповых информационных систем. Однако в крупных информационных системах наибольшее распространение получили серверы Oracle, DB2 и Microsoft SQL Server.

По **способу организации** групповые и корпоративные ИС подразделяются на следующие классы:

- системы на основе архитектуры файл-сервер;
- системы на основе архитектуры клиент-сервер;
- системы на основе многоуровневой архитектуры;
- системы на основе Интернет/интранет-технологий.

По **типу хранимых данных** различают следующие ИС (рис. 2.6):

- *фактографические*: предназначены для хранения и обработки структурированных данных в виде чисел и текстов. Над такими данными можно выполнять различные операции.
- *документальные*: в таких системах информация представлена в виде документов, состоящих из наименований, описаний, рефератов и текстов. Поиск по неструктурированным данным осуществляется с использованием семантических признаков; отобранные документы предоставляются пользователю; обработка данных в таких системах практически не производится.

По **степени автоматизации информационных процессов** информационные системы делятся на:

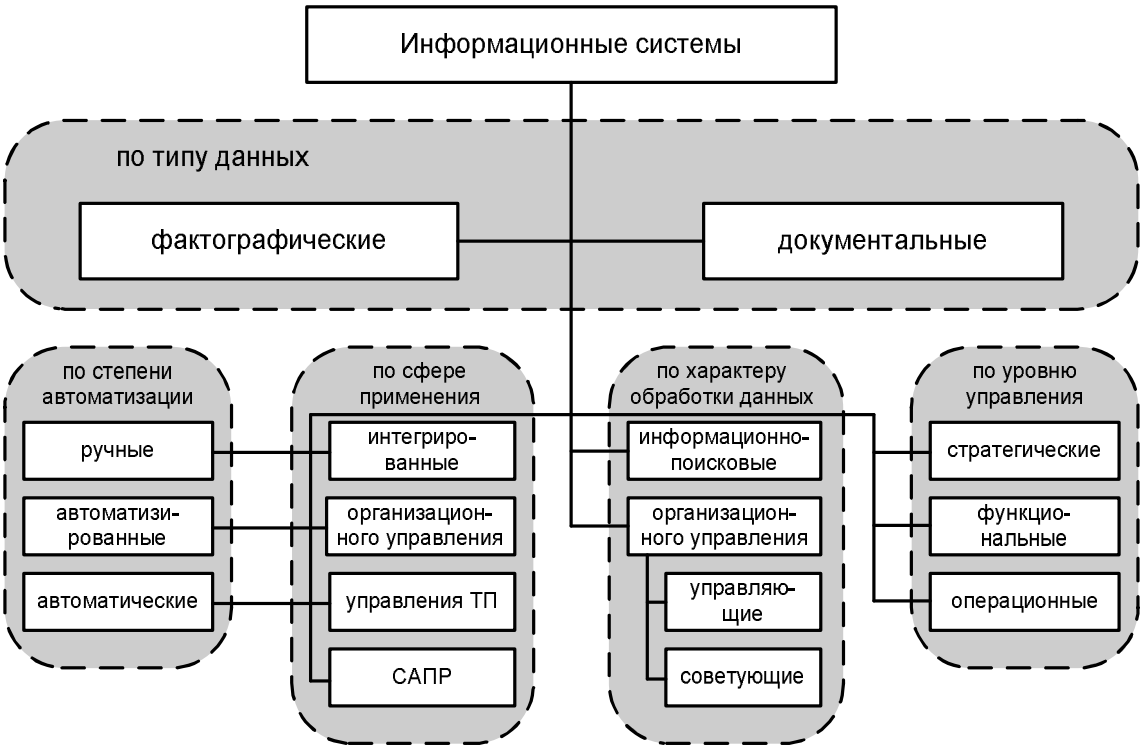

Рисунок 2.6. Классификация информационных систем

- *ручные*: характеризуются отсутствием современных технических средств переработки информации и выполнением всех операций человеком;
- *автоматические*: характеризуются выполнением всех операций по переработке информации без участия человека;
- *автоматизированные*: предполагают участие в процессе обработки информации и человека, и технических средств, причем главная роль в выполнении рутинных операций обработки данных отводится компьютеру. Именно этот класс систем соответствует современному представлению понятия «информационная система» и будет рассмотрен в данном разделе.

По **характеру обработки данных** различают следующие ИС:

- *информационно-поисковые*: производят ввод, систематизацию, хранение, выдачу информации по запросу пользователя без сложных преобразований данных (например, ИС библиотечного об-

служивания, резервирования и продажи билетов на транспорте, бронирования мест в гостиницах и пр.). В информационных системах организационного управления преобладает режим оперативной обработки транзакций (OnLine Transaction Processing (OLTP)) для отражения актуального состояния предметной области в любой момент времени, а пакетная обработка занимает весьма ограниченную часть. Для систем OLTP характерен регулярный (возможно, интенсивный) поток довольно простых транзакций, играющих роль заказов, платежей, запросов и т.п. Важными требованиями для них являются:

- высокая производительность обработки транзакций;
- гарантированная доставка информации при удаленном доступе к БД по телекоммуникациям;
- *информационно-решающие*: осуществляют, кроме того, операции переработки информации по определенному алгоритму. По характеру использования выходной информации такие системы принято делить на *управляющие* и *советующие*.

Результирующая информация *управляющих ИС* непосредственно трансформируется в принимаемые человеком решения. Для этих систем характерны задачи расчетного характера и обработка больших объемов данных (например, ИС планирования производства или заказов, бухгалтерского учета).

Советующие ИС вырабатывают информацию, которая принимается человеком к сведению и учитывается при формировании управленческих решений, а не инициирует конкретные действия. Эти системы имитируют интеллектуальные процессы обработки знаний, а не данных (экспертные системы).

В зависимости от **сферы применения** различают следующие классы ИС:

- *ИС организационного управления* – предназначены для автоматизации функций управленческого персонала (промышленных предприятий, гостиниц, банков, магазинов и пр.). Основные функции таких ИС: оперативный контроль и регулирование, оперативный учет и анализ, перспективное и оперативное планирование, бухгалтерский учет, управление сбытом, снабжением и другие экономические и организационные задачи.
- *ИС управления технологическими процессами* (ТП) – служат для автоматизации функций производственного персонала по контролю и управлению производственными операциями. В таких системах обычно предусматривается наличие развитых средств измерения параметров технологических процессов (температуры, давления, химического состава и т.п.), процедур контроля допустимости значений параметров и регулирования технологических процессов.
- *ИС автоматизированного проектирования* (САПР) – предназначены для автоматизации функций инженеров-проектировщиков, конструкторов, архитекторов, дизайнеров при создании новой техники или технологии. Основными функциями подобных систем являются: инженерные расчеты, создание графической документации (чертежей, схем, планов), создание проектной документации, моделирование проектируемых объектов.
- *Интегрированные (корпоративные) ИС* – используются для автоматизации всех функций фирмы и охватывают весь цикл работ от планирования деятельности до сбыта продукции. Они включают в себя ряд модулей (подсистем), работающих в едином информационном пространстве и выполняющих функции поддержки соответствующих направлений деятельности.

2.1.7.1. Классификация автоматизированных информационных систем

Рассмотрим классификацию автоматизированных информационных систем (АИС), сегодня чаще говорят просто ИС, подразумевая, что все современные ИС базируются на ЭВМ и являются автоматизированными (рис. 2.7).

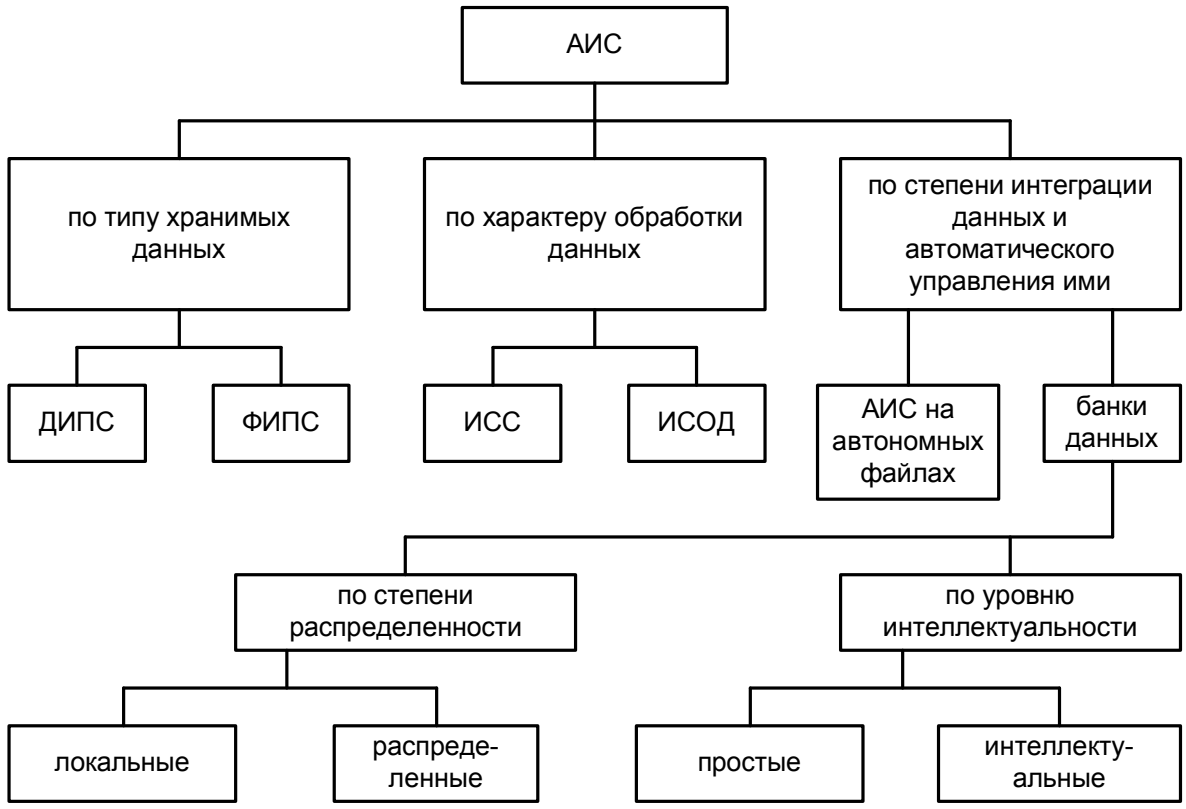

Рисунок 2.7. Схема классификации АИС

ДИПС – документированные информационно-поисковые системы. Предназначены для хранения и обработки документальных данных: адресов хранения документов, их наименований, описаний и рефератов, реже текстов документов. Примером ДИПС являются библиотечные, библиографические АИС.

ФИПС – фактографические информационно-поисковые системы, хранят и обрабатывают фактографическую информацию: структурирован-

ные данные в виде чисел и текстов. Над такими данными можно выполнять различные операции.

ИСС – информационно-справочные системы (запросно-ответные или просто справочные). Выполняют поиск и вывод информации без ее обработки. Эти системы работают в интерактивном режиме.

ИСОД – информационные системы обработки данных: представляют собой сочетание ИСС с системой обработки данных. Обработка найденных данных выполняется комплексом предусмотренных в системе прикладных программ либо с помощью средств, указываемых пользователями в запросах.

Важнейшим признаком классификации АИС является степень интеграции данных и автоматизации управления ими. В ранних системах – АИС на автономных файлах – принцип интеграции данных практически не реализован, либо интеграция является сравнительно низкой. Такие системы применяются и в настоящее время, однако они эффективны только в случае узкого, специализированного использования небольшим кругом лиц. Как большой класс АИС с высокой степенью интеграции данных выделяют банки данных. По сравнению с АИС на автономных файлах, в банках данных хранимая информация сосредоточена в едином информационном массиве – базе данных (БД), и процесс манипулирования данными автоматизирован.

По степени распределенности банки данных классифицируются на локальные системы, информация которых сосредоточена в одной или нескольких БД, но на одной ЭВМ, и распределенные, в которых информация хранится в отдельных БД, размещенных в узлах вычислительной сети. По уровню интеллектуальности банки данных делятся на простые системы, не использующие средств искусственного интеллекта, и на интеллектуальные банки данных, получившие специальное название – банки знаний. В их основе кроме БД лежат базы знаний.

2.1.8. Информационная модель и методы моделирования архитектуры проектируемой информационной системы

В условиях рынка все большее число компаний осознают преимущества использования информационных систем. В некоторых случаях ИС – это не только набор услуг, но и важнейший компонент бизнеса, как, например, система резервирования билетов или средства предоставления финансовой информации. Чтобы получить выгоду от использования информационной системы, ее следует создавать в короткие сроки и с уменьшенными затратами. Информационная система должна быть легко сопровождаемой и управляемой. Создание информационной системы предприятия – достаточно сложный и многоступенчатый процесс, который, весьма часто, содержит фазу информационного моделирования. *Информационная модель* – это спецификация структуры данных и бизнес правил (правил предметной области).

Методики разработки архитектуры ИС. Разработка информационных систем становится в последние годы распространенной задачей, решаемой федеральными, региональными органами управления и коммерческими структурами: банками, торговыми домами и т.п. Проекты такого рода по масштабу могут быть небольшими, средними, большими и сверхбольшими (уникальными). Типичными примерами уникальных проектов являются информатизация Центрального Банка РФ, Государственной налоговой службы, управления космическими аппаратами и т.п. Обычно большие и уникальные проекты характеризуются многоуровневой структурой объектов автоматизации, их разнесенностью в пространстве, сложной технологией функционирования, базой данных большого объема и/или высокой сложности, временем жизни, превышающим срок жизни оборудования, использованием в информационной системе части уже имеющегося оборудования.

Наиболее адекватной представляется разработка архитектуры сложных прикладных информационных систем на основе *концепции открытых систем*. Основная цель создания систем как открытых состоит в возможности экономически и технически эффективного объединения в единую гетерогенную систему разных видов оборудования и программного обеспечения на основе применения стандартизованных интерфейсов между компонентами системы. Такой подход потенциально позволяет повторно использовать наиболее наукоемкий продукт – программные средства – на разных вычислительных платформах без перепрограммирования и тем самым экономить значительные финансовые средства. С другой стороны, такой подход позволяет поэтапно наращивать вычислительную мощность прикладной системы в соответствии, как с потребностями пользователя, так и с его финансовыми возможностями.

2.1.9. Методы проектирования информационных систем

Методы проектирования ИС можно классифицировать по степени использования средств автоматизации, типовых проектных решений, адаптивности к предполагаемым изменениям (табл. 2.3):

– *ручное* (проектирование компонентов ИС осуществляется без использования специальных инструментальных программных средств, а программирование – на алгоритмических языках);

Таблица 2.3.

Классификация методов проектирования ИС

Использование средств автоматизации	Использование типовых проектных решений	Использование адаптивности к предполагаемым изменениям
Ручное Компьютерное	Оригинальное Типовое	Реконструкция Параметризация Реструктуризация модели

- *компьютерное*, при котором производится генерация или конфигурирование (настройка) проектных решений на основе использования специальных инструментальных программных средств.
- *оригинальное* (индивидуальное), когда проектные решения разрабатываются «с нуля» в соответствии с требованиями к АИС. Характеризуется тем, что все виды проектных работ ориентированы на создание индивидуальных для каждого объекта проектов, которые в максимальной степени отражают все его особенности;
- *типовое*, предполагающее конфигурирование ИС из готовых типовых проектных решений (программных модулей). Выполняется на основе опыта, полученного при разработке индивидуальных проектов. Типовые проекты, как обобщение опыта для некоторых групп организационно-экономических систем или видов работ, в каждом конкретном случае связаны со множеством специфических особенностей и различаются по степени охвата функций управления, выполняемым работам и разрабатываемой проектной документации;
- *реконструкция*, когда адаптация проектных решений выполняется путем переработки соответствующих компонентов (перепрограммирования программных модулей);
- *параметризация*, когда проектные решения настраиваются (генерируются) в соответствии с изменяемыми параметрами;
- *реструктуризации модели*, когда изменяется модель проблемной области, на основе которой автоматически заново генерируются проектные решения.

Выделяют два основных класса используемых технологий проектирования ИС [3–7]: *каноническую* и *индустриальную* технологии. Индустриальная технология проектирования, в свою очередь, разбивается на два подкласса: *автоматизированное* (использование CASE-технологий) и

типовое (параметрически-ориентированное или модельно-ориентированное) проектирование. Использование индустриальных технологий не исключает использования в отдельных случаях канонических.

В основе **канонического проектирования** лежит каскадная модель жизненного цикла ИС[4], рассмотренная выше.

Стадии и этапы создания ИС, выполняемые организациями-участниками, прописываются в договорах и технических заданиях на выполнение работ:

Стадия 1. *Формирование требований к ИС*. На начальной стадии проектирования выделяют следующие этапы работ:

- обследование объекта и обоснование необходимости создания ИС;
- формирование требований пользователей к ИС;
- оформление отчета о выполненной работе и тактико-технического задания на разработку.

Стадия 2. *Разработка концепции ИС*:

- изучение объекта автоматизации;
- проведение необходимых научно-исследовательских работ;
- разработка вариантов концепции ИС, удовлетворяющих требованиям пользователей;
- оформление отчета и утверждение концепции.

Стадия 3. *Техническое задание*:

- разработка и утверждение технического задания на создание ИС.

Стадия 4. *Эскизный проект*:

- разработка предварительных проектных решений по системе и ее частям;
- разработка эскизной документации на ИС и ее части.

---

[4] в соответствии с применяемым в нашей стране ГОСТ 34.601-90 «Автоматизированные системы. Стадии создания»

Стадия 5. *Технический проект*:

- разработка проектных решений по системе и ее частям;
- разработка документации на ИС и ее части;
- разработка и оформление документации на поставку комплектующих изделий;
- разработка заданий на проектирование в смежных частях проекта.

Стадия 6. *Рабочая документация*:

- разработка рабочей документации на ИС и ее части;
- разработка и адаптация программ.

Стадия 7. *Ввод в действие*:

- подготовка объекта автоматизации;
- подготовка персонала;
- комплектация ИС поставляемыми изделиями (программными и техническими средствами, программно-техническими комплексами, информационными изделиями);
- строительно-монтажные работы;
- пусконаладочные работы;
- проведение предварительных испытаний;
- проведение опытной эксплуатации;
- проведение приемочных испытаний.

Стадия 8. *Сопровождение ИС*:

- выполнение работ в соответствии с гарантийными обязательствами;
- послегарантийное обслуживание.

Типовое проектирование ИС предполагает создание системы из готовых типовых элементов. Основополагающим требованием для применения методов типового проектирования является возможность декомпозиции проектируемой ИС на множество составляющих компонентов (подси-

стем, комплексов задач, программных модулей и т.д.). Для реализации выделенных компонентов выбираются имеющиеся на рынке типовые проектные решения, которые настраиваются на особенности конкретного предприятия.

Типовое проектное решение (ТПР) – это тиражируемое (пригодное к многократному использованию) проектное решение.

Принятая классификация ТПР основана на уровне декомпозиции системы. Выделяются следующие классы ТПР:

- *элементные ТПР* – типовые решения по задаче или по отдельному виду обеспечения задачи (информационному, программному, техническому, математическому, организационному);
- *подсистемные ТПР* – в качестве элементов типизации выступают отдельные подсистемы, разработанные с учетом функциональной полноты и минимизации внешних информационных связей;
- *объектные ТПР* – типовые отраслевые проекты, которые включают полный набор функциональных и обеспечивающих подсистем ИС.

Для реализации типового проектирования используются два подхода: параметрически-ориентированное и модельно-ориентированное проектирование.

Параметрически-ориентированное проектирование включает следующие этапы: определение критериев оценки пригодности пакетов прикладных программ (ППП) для решения поставленных задач, анализ и оценка доступных ППП по сформулированным критериям, выбор и закупка наиболее подходящего пакета, настройка параметров (доработка) закупленного ППП.

Критерии оценки ППП делятся на следующие группы:

- назначение и возможности пакета;
- отличительные признаки и свойства пакета;

- требования к техническим и программным средствам;
- документация пакета;
- факторы финансового порядка;
- особенности установки пакета;
- особенности эксплуатации пакета;
- помощь поставщика по внедрению и поддержанию пакета;
- оценка качества пакета и опыт его использования;
- перспективы развития пакета.

Модельно-ориентированное проектирование заключается в адаптации состава и характеристик типовой ИС в соответствии с моделью объекта автоматизации.

Технология проектирования в этом случае должна обеспечивать единые средства для работы как с моделью типовой ИС, так и с моделью конкретного предприятия.

Типовая ИС в специальной базе метаинформации – *репозитории* – содержит модель объекта автоматизации, на основе которой осуществляется конфигурирование ПО. Таким образом, модельно-ориентированное проектирование ИС предполагает, прежде всего, построение модели объекта автоматизации с использованием специального программного инструментария. Возможно также создание системы на базе типовой модели ИС из репозитория, который поставляется вместе с программным продуктом и расширяется по мере накопления опыта проектирования ИС для различных отраслей и типов производства.

Реализация типового проекта предусматривает выполнение следующих операций:

- установку глобальных параметров системы;
- задание структуры объекта автоматизации;
- определение структуры основных данных;

- задание перечня реализуемых функций и процессов;
- описание интерфейсов;
- описание отчетов;
- настройку авторизации доступа;
- настройку системы архивирования.

2.1.10. Профили открытых информационных систем

Области применения современных ИС, например, ИС крупных предприятий, ИС органов государственного управления, ИС учреждений науки и образования, предъявляют к ним весьма высокие требования. Эти требования связаны, прежде всего, с необходимостью интеграции в единой системе задач, которые ранее на предприятии могли быть решены автономно (как независимые «островки» автоматизации разных процессов производства, планирования, управления, снабжения и сбыта), интеграции разных информационных технологий (обработки данных, обработки текстов, обработки изображений, машинной графики и т.д.). Другими словами, современные ИС уровня предприятия являются по своей сути *интегрированными системами*. Кроме того, требуется обеспечивать интеграцию двух или более ИС, когда они должны взаимодействовать между собой, реализуя связанные бизнес-процессы разных предприятий, например, при организации цепочек поставок, отношений с потребителями продукции и бизнес-партнерами и т.д.

Требования интеграции влекут за собой резкий рост сложности систем. Однако, с другой стороны, в современных условиях рыночной экономики ужесточаются ограничения на сроки создания и внедрения ИС, материальные и финансовые ресурсы, которые предприятие может выделить на эти работы.

Непрерывные изменения в деятельности предприятий, связанные с конкурентной борьбой на рынке, изменения нормативно-правовой базы

этой деятельности (особенно в России в переходный период) влекут за собой необходимость иметь возможность изменений состава прикладных функций ИС, адекватных изменяющимся условиям деятельности и росту потребностей пользователей ИС в информационном обеспечении. Эти изменения прикладных функций определенных подсистем ИС не должны затрагивать другие подсистемы, иначе потребовалось бы перепроектировать всю систему, что уже не представляется возможным.

Компромисс всех этих противоречивых требований достигается применением принципов открытых систем при создании, сопровождении и развитии современных ИС уровня предприятия. Обеспечение таких свойств открытых систем, как *расширяемость* (изменяемость) состава прикладных функций ИС, *интероперабельность* (способность к взаимодействию приложений разных подсистем в пределах одной интегрированной ИС или нескольких ИС между собой), *переносимость* приложений между разными аппаратно-программными платформами, *масштабируемость* (при изменении размерности решаемых задач, числа пользователей, обслуживаемых ИС), *дружественность пользовательского интерфейса*, неразрывно связано с применением соответствующих стандартов. При этом определение набора базовых стандартов, которые комплексно специфицируют интерфейсы, протоколы взаимодействия и форматы обмена данными и др. составляет предмет, так называемой, *функциональной стандартизации* [6]. Такой набор называют профилем системы, а после его утверждения – *функциональным стандартом*. Это позволяет применять стандартизованные проектные решения при построении ИС (аналогично методам крупноблочного строительства зданий и сооружений) с тем, чтобы снизить затраты и сократить сроки создания и внедрения ИС в условиях роста их сложности и наращивания функций.

Введенное в [6] понятие «профили» определяет их как подмножество и/или комбинации базовых стандартов информационных технологий, не-

обходимые для реализации требуемых наборов функций. Для определения места и роли каждого базового стандарта в профиле требуется концептуальная модель. Такая модель, называемая OSE/RM (Open System Environment/Reference Model), предложена в [7].

Таким образом, придание конкретной ИС перечисленных выше свойств открытых систем реализуется с помощью разработки ее профиля (функционального стандарта). В соответствии с этим открытые системы по определению IEEE определены как системы, в которых реализован «исчерпывающий и согласованный набор базовых международных стандартов информационных технологий и профилей функциональных стандартов, которые специфицируют интерфейсы, службы и поддерживающие форматы [данных], чтобы обеспечить интероперабельность и мобильность приложений, данных и персонала».

Каждую сложную интегрированную ИС, как уникальную ИС какого-либо предприятия или организации, так и типовую тиражируемую ИС для определенной области применения, предлагается сопровождать ее профилем, включающем в себя совокупность базовых стандартов и спецификаций, которым должны отвечать как ИС в целом, так и ее составные части.

Категории и виды профилей ИС. В зависимости от сферы распространения профилей ИС рассматриваются следующие их категории:

- *профили конкретных ИС*, определяющие стандартизованные проектные решения в пределах проекта данной ИС и имеющие статус документации проекта в части нормативных требований или статус стандарта предприятия, для которого создается эта ИС;
- *профили группы типовых тиражируемых ИС*, предназначенных для определенной области применения, имеющие статус отраслевого (ведомственного) стандарта для этой области или статус стандарта организации, разрабатывающей и поставляющей такие ИС (системного интегратора).

– *стратегические профили для определенной области применения ИС*, определяющие ориентацию информатизации этой области на долгосрочный период, например, профили переносимости приложений между разными ИС в этой области.

Принципы построения и структура профиля ИС. Необходимость стандартизации интерфейсов и протоколов для области телекоммуникаций была понята еще 25 лет назад. В отрасли связи сложились подходы и методология, без которых немыслимо было бы построение сетей передачи данных, локальных и глобальных вычислительных сетей.

Для этих целей предложена эталонная модель среды открытых систем – OSE/RM[5] (рис. 2.8).

В крупном плане концептуальная модель предусматривает разбиение ИС на приложения (прикладные программные комплексы), реализующие заданные функции ИС, и среду, обеспечивающую подготовку и выполнение (runtime) приложений. Между ними определяются стандартизованные интерфейсы прикладного программирования (API).

Кроме того, определяются стандартизованные интерфейсы взаимодействия данной ИС с внешней для нее средой - другими ИС и сетью Интернет и/или корпоративными сетями (EEI).

Спецификации функций компонентов ИС рассматриваются по четырем функциональным группам:

– функции, обслуживающие интерфейс ИС с пользователями;
– функции организации процессов обработки данных (системные функции среды);
– функции представления и хранения данных;
– коммуникационные функции.

---

[5] Модель OSE/RM закреплена документами ISO/IEC

Эти функции могут быть реализованы как приложениями, так и компонентами среды ИС. Их спецификации составляют плоскость основных функций ИС.

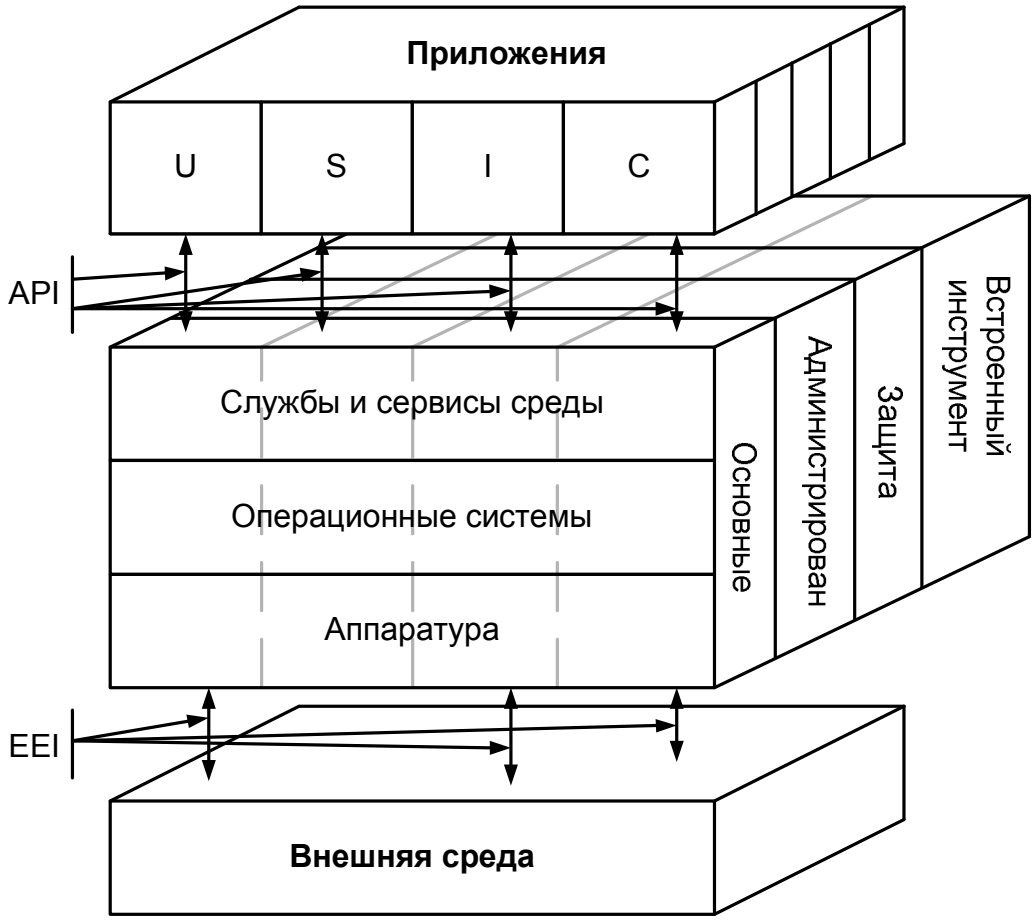

Рисунок 2.8. Эталонная модель среды открытых систем – OSE/RM (Open System Environment)

Функции системного и сетевого администрирования распределены между компонентами среды и приложений. Они образуют вторую плоскость концептуальной модели, в которую включаются управление приложениями, управление средствами пользовательского интерфейса, управление базами данных, управление процессами, обеспечиваемое операционными системами, управление коммуникационной сетью или отдельными узлами сети, управление средствами защиты информации.

Функции средств защиты информации в ИС также распределены между разными компонентами ИС. Часть из них реализуется штатными

средствами, встроенными в операционные системы, СУБД, ПО промежуточного слоя (например, в мониторы транзакций), а часть обеспечивается специальными средствами защиты. Поэтому в концептуальную модель введена третья плоскость – функции защиты информации. Четвертую плоскость составляют функции инструментальных средств, встроенных в ИС для поддержки ее эксплуатации и сопровождения.

2.2. Методологии, технологии и инструментальные средства проектирования

Методология проектирования ИС предполагает наличие некоторой концепции, принципов проектирования, реализуемых набором методов, которые, в свою очередь, должны поддерживаться некоторыми средствами.

Организация проектирования предполагает определение методов взаимодействия проектировщиков между собой и с заказчиком в процессе создания проекта ИС, которые могут также поддерживаться набором специфических средств.

Технология проектирования ИС – это совокупность методологии и средств проектирования ИС, а также методов и средств его организации.

Технология проектирования задается регламентированной последовательностью технологических операций, выполняемых на основе того или иного метода.

Предметом любой выбираемой технологии проектирования должно служить отражение взаимосвязанных процессов проектирования на всех стадиях жизненного цикла ИС. К основным требованиям, предъявляемым к выбираемой технологии проектирования, относятся следующие:

- созданный проект должен отвечать требованиям заказчика;
- максимальное отражение всех этапов жизненного цикла проекта;

- обеспечение минимальных трудовых и стоимостных затрат на проектирование и сопровождение проекта;
- технология должна быть основой связи между проектированием и сопровождением проекта;
- рост производительности труда проектировщика;
- надежность процесса проектирования и эксплуатации проекта;
- простое ведение проектной документации.

2.2.1. Модели структурного проектирования

Структурный подход к анализу и проектированию ИС заключается в рассмотрении ее с общих позиций с последующей детализацией и представлением в виде иерархической структуры. На верхнем уровне иерархии обычно представляется функциональное описание системы.

При проведении структурного анализа и проектирования для повышения наглядности используется *графическое представление функций ИС и отношений между данными*.

Наиболее распространенными моделями и диаграммами графического представления являются следующие:

- диаграммы СУЩНОСТЬ-СВЯЗЬ или ER-диаграммы – Entity-Relationship-Diagrams (ERD) служат для наглядного представления схем баз данных;
- диаграммы потоков данных – Data Flow Diagrams (DFD) служат для иерархического описания модели системы;
- метод структурного анализа и проектирования – Structured Analysis and Design Technique (SADT), служащий для построения функциональной модели объекта;
- схемы описания иерархии ВХОД-ОБРАБОТКА-ВЫХОД – Hierarchy plus Input-Processing-Output (HIPO) служат для описания ре-

лизуемых программой функций и циркулирующих внутри нее потоков данных;

– диаграммы Варнье-Орра служат для описания иерархической структуры системы с выделением элементарных составных частей, выделением процессов и указанием потоков данных для каждого процесса.

Названные модели позволяют получить описание ИС, а их состав зависит от требуемой полноты ее описания.

Моделирование данных – это процесс описания информационных структур и бизнес-правил для определения потребностей информационной системы.

Модель данных является визуальным представлением структур данных, данных и бизнес-правил для СУБД. Обычно она разрабатывается как часть более крупного проекта по разработке программного обеспечения. Модель данных состоит из двух компонент – логической и физической моделей. В большинстве случаев первой создается логическая модель, затем – модель физическая.

Логический уровень – это абстрактный взгляд на данные, когда данные представляются так, как выглядят в реальном мире, и могут называться так, как они называются в реальном мире, например «Постоянный клиент», «Отдел» или «Фамилия сотрудника». Объекты модели, представляемые на логическом уровне, называются сущностями и атрибутами. Логическая модель данных может быть построена на основе другой логической модели, например на основе модели процессов. Логическая модель данных является универсальной и никак не связана с конкретной реализацией СУБД.

Физическая модель данных, напротив, зависит от конкретной СУБД, фактически являясь отображением системного каталога. В физической модели содержится информация обо всех объектах БД. Поскольку стандартов

на объекты БД не существует (например, нет стандарта на типы данных), физическая модель зависит от конкретной реализации СУБД. Следовательно, одной и той же логической модели могут соответствовать несколько разных физических моделей. Если в логической модели не имеет значения, какой конкретно тип данных имеет *атрибут*, то в физической модели важно описать всю информацию о конкретных физических объектах (таблицах, колонках, индексах, процедурах и т.д.).

2.2.1.1. Стандарт моделирования данных IDEF1X. ER-диаграммы

Стандарт моделирования данных IDEF1X использует три уровня логических моделей, используемых для охвата требований бизнес-информации: *диаграмму зависимостей сущностей* (ERD); *модель, основанную на ключах* (Key Based, KB), и *полностью определенная модель* (Fully Attributed Diagram, FA). ERD и KB модели также называются «моделями данных области» потому, что охватывают обширные области бизнеса. Полной противоположностью является модель FA – «модель данных проекта» потому, что она, как правило, описывает только часть всей структуры информации [1–5].

Модель уровня сущностей является моделью нижнего уровня и применяется для работы проектировщика информационной системы с экспертами моделируемой системы. Модель включает сущности и связи между ними, которые отражают основные бизнес-правила предметной области, и допускает присутствие всех типов связей (определенных, неопределенных, типа «Категория»). Графическое представление этой модели называется ER-диаграммой.

Модель уровня ключей является дальнейшим развитием ER модели и содержит более подробное представление данных. Она содержит описание всех сущностей, связей между ними, первичных и внешних ключей. Эта модель не допускает наличия неопределенных связей и требует их предва-

рительного преобразования в определенные связи. Модель является переходным звеном от модели уровня сущностей к полному логическому описанию предметной области. Графическое представление этой модели называется KB-диаграммой (Key Based Diagram).

Полноатрибутная модель является дальнейшим развитием модели уровня ключей и представляет собой законченное (в рамках конкретного проекта) описание предметной области. Модель содержит описание всех сущностей, связей и атрибутов, выделенных при анализе предметной области. Построением этой модели завершается процесс концептуального проектирования, а модель в дальнейшем может быть использована при построении внутренней модели базы данных реляционного типа. Графическое представление этой модели называется FA-диаграммой (рис. 2.9).

В базе данных отражается информация об определенной предметной области. При проектировании баз данных предметная область отображается моделями данных нескольких уровней абстракций. В целом все модели данных можно разделить на три категории:

- *концептуальная модель* – описание предметной области, выполненное с использованием специальных языковых средств, без ориентации на используемые в дальнейшем программные и технические средства. Данная модель содержит исходную информацию о предметной области, необходимую для проектирования структуры базы данных (эта информация не зависит от особенностей СУБД);
- *внутренняя модель* – описание предметной области, выполненное в рамках целевой СУБД;
- физическая модель – определяет используемые запоминающие устройства и способы физической организации данных на них [2].

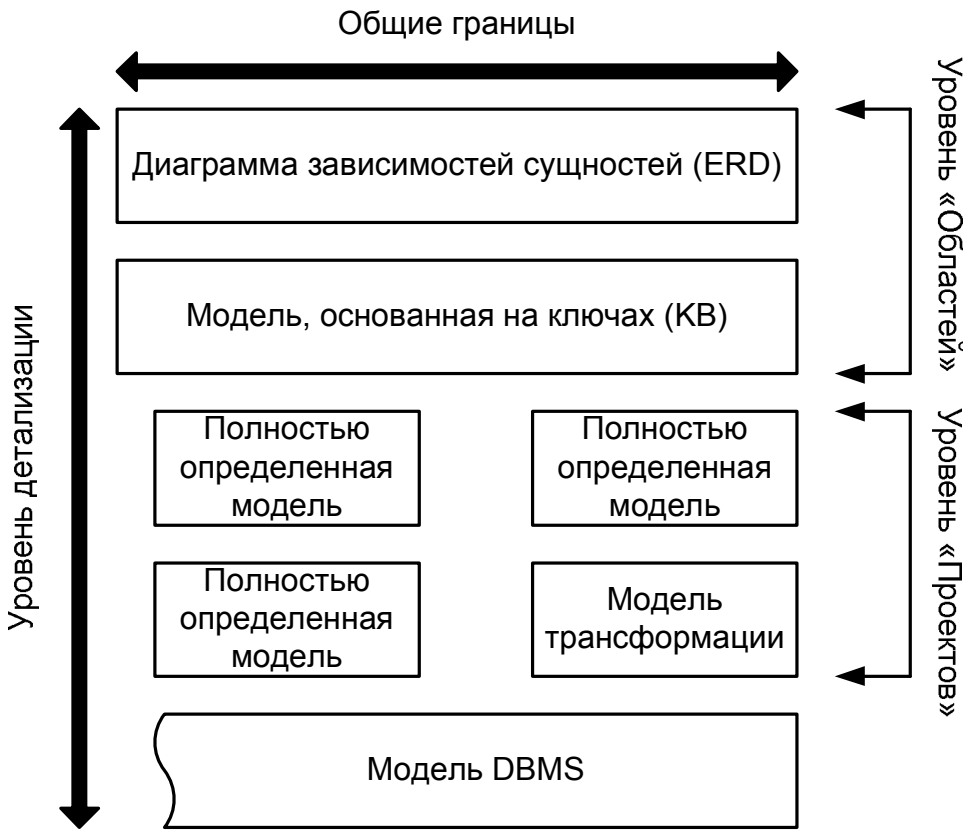

Рисунок 2.9. Уровни проектирования базы данных IDEF1X

В соответствии с методологией стандарта IDEF1X **построение концептуальной модели** осуществляется поэтапно путем последовательного выполнения следующих фаз проектирования:

1. Определение сущностей.
2. Построение модели уровня сущностей.
3. Построение модели уровня ключей.
4. Построение полноатрибутной модели.

Рассмотрим процесс построения концептуальной модели по методологии стандарта IDEF1X на примере построения базы данных «Учет автовладельцев».

Фаза 1. *Определение сущностей*

Первым шагом проектирования является составление предварительного пула (списка) сущностей.

Сущность в контексте IDF1X модели представляет собой множество реально существующих или абстрактных индивидуальных объектов, которые обладают одинаковым набором свойств (атрибутов). Механизм определения сущностей следующий. Сначала на основе анализа предметной области определяются так называемые кандидаты, в сущности. Для этого выявляют экземпляры объектов предметной области с одинаковыми характеристиками и объединяют их в одну сущность-кандидат.

Для того чтобы выбрать среди объектов, являющихся кандидатами, собственно сущности, разработчик модели должен относительно каждого кандидата ответить на следующие вопросы:

- можно ли описать этот объект, т.е. можно ли получить о нем информацию?
- имеет ли этот объект характеризующие его свойства?
- можно ли получить информацию об этих свойствах?
- можно ли выделить несколько образцов этого объекта, т.е. набор экземпляров этого объекта с одинаковыми свойствами?
- можно ли отличить один образец этого объекта от другого, т.е. имеется ли у объекта свойство (группа свойств), определяющее уникальность каждого образца этого объекта?
- является ли этот объект характеристикой чего-либо, т.е. описывает ли этот объект некоторый другой объект?

Если схема ответов на все эти вопросы такова: ДА, ДА, ДА, ДА, ДА, НЕТ, то этот объект обычно включают в пул сущностей. В противном случае, данный объект, скорее всего, нужно рассматривать как атрибут некоторой другой сущности.

При включении сущности в пул необходимо задать ей уникальное имя, которое является обязательно существительным, номер (например, Автовладелец, номер 1), сформулировать определение объекта реального мира, соответствующего этой сущности, и привести описание механизма

включения нового экземпляра в данную сущность. Результат работы на данной фазе проектирования оформляется в виде таблицы (табл. 2.4).

Здесь требуется отметить, что не все сущности из списка, составленного на первой фазе проектирования, останутся в пуле сущностей к концу фазы 4. Кроме того, на следующих фазах проектирования в пул могут быть добавлены новые сущности. В связи с этим, в качестве одного из обязательных документов, отражающих результат проектных работ на каждой из последующих фаз проектирования, должен быть скорректированный пул сущностей.

Таблица 2.4.

Пул сущностей

Номер множества сущностей	Имя множества сущностей	Определение множества сущностей	Описание множества сущностей
1.	Автовладелец	Лицо, купившее машину и зарегистрированное в УВД	
2.	Регион	Область, в которой зарегистрирован автовладелец	
3.	Автомобиль	Транспорт, который купил автовладелец	Автомобиль зарегистрирован по месту проживания ее владельца

Фаза 2. *Построение модели уровня сущностей*

Модель уровня сущностей состоит из уточненного пула сущностей, матрицы связей, описания связей между сущностями, диаграммы ER-типа.

Проектирование на данной фазе начинается с определения всех возможных бинарных (связей между двумя сущностями) связей между выявленными сущностями, на основании чего строится матрица связей. *Матрица связей* (табл. 2.2) представляется в виде двумерной таблицы, заголовками столбцов и строк которой являются сущности из списка. Если между

некоторыми сущностями выявлена бинарная связь, то в точках пересечения соответствующих строк и столбцов таблицы помещается некоторый символ (например, «X»). Пример. В пуле имеются сущности с номерами Автовладелец/1, Регион/2, Автомобиль/3. Пусть между сущностями Автовладелец/1 и Регион/2, Автовладелец/1 и Автомобиль/3, определены бинарные связи, построим матрицу связей (табл. 2.5).

Таблица 2.5.

Матрица связей

	Автовладелец/1	Регион/2	Автомобиль/3
Автовладелец/1		X	X
Регион/2	X		
Автомобиль/3	X		

Каждой выявленной связи назначаются имя, которое является глаголом или глагольной фразой, и уникальный номер (например, зарегистрирован/1). Матрица связей отражает только факт наличия бинарных связей между сущностями, но не отражает их природу. Поэтому далее исследуется каждая выявленная бинарная связь в обоих направлениях. Схема действий при этом следующая.

Сначала определяется мощность (кардинальное число) для каждого конца связи. Для этого необходимо предположить существование экземпляра одной из связанных сущностей и, исходя из семантики предметной области, определить, сколько конкретных экземпляров второй сущности может быть связано с первым. Затем такой же анализ проводится относительно другой сущности этой связи (в обратном направлении связи).

Далее определяется тип связи (определенная/неопределенная), ее обязательность. На данном этапе не всегда можно определить, является связь идентифицирующей или не идентифицирующей. Это решение откладывается на последующие этапы проектирования.

Кроме того, для каждой из сущностей участниц связи необходимо определить является ли сущность родительской, или дочерней в заданной связи. Для связей 1:N, сущность, которая имеет единичное участие в связи, определяется как родительская, множественное – как дочерняя. Для связей 1:1 сущность, которая имеет частичное участие в связи, определяется как родительская, сущность, которая тотально участвует в связи, как дочерняя. Для неопределенных связей (N: N) родительские и дочерние сущности не определяются.

Выбор родительских и дочерних сущностей влияет на определение имен связей. В случаях определенных связей, имя связи является глаголом или глагольной фразой связывающей имена сущностей, участниц связей в направлении от родительской к дочерней. В случае неопределенной связи, имя связи задается двумя глаголами, которые связывают имена сущностей в обоих направления связи.

Результат этой работы отражается в таблице описания связей (табл. 2.6). В процессе формирования описания связей может корректироваться пул сущностей.

В графе «Тип связи» указывается тип связи (например, обязательная определенная, необязательная определенная, неопределенная, связь типа категория, иерархическая рекурсия и т.п.).

Таблица 2.6.

Таблица описания связей

Номер связи	Номер 1-й сущности	Номер 2-й сущности	Имя связи	Тип связи	Мощность связи
1.	1-дочерняя	2-родительская	зарегистрирован	определенная	M:1
2.	1-родительская	3-дочерняя	принадлежит	определенная	1:M

В графе «Мощность связи» указываются кардинальные числа обоих концов связи для неопределенной связи или одного конца для определенной связи.

Необходимо отметить, что таблица описания связей может корректироваться на следующих фазах проектирования.

Далее на основе пула сущностей и таблицы описания связей строится ER-диаграмма. В моделях этого уровня разрешается отображение в диаграмме всех типов бинарных связей. Атрибуты в ER-диаграммах не отображаются. Так как на данном этапе не всегда можно сделать заключение о том, является связь идентифицирующей или не идентифицирующей, на диаграмме все сущности, как правило, изображаются прямоугольниками с прямыми углами (рис. 2.10).

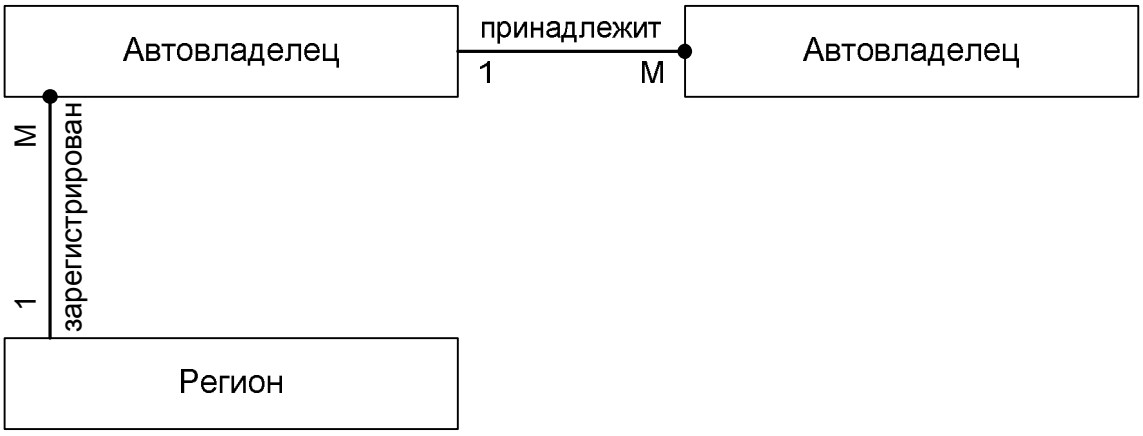

Рисунок 2.10. Пример ER-диаграммы зависимостей сущностей

Фаза 3. *Построение модели уровня ключей*

Модель уровня ключей является дальнейшим развитием модели уровня сущностей. Эта модель включает в себя описание доменов, описание атрибутов, пул сущностей, описание связей, KB-диаграмму.

На фазе 3 проектирования выполняются следующие основные действия:

– для каждой сущности определяются первичный и альтернативные ключи;

– реализуются все связи между сущностями;

– разрешаются все неопределенные связи.

Проектные работы на этой фазе начинаются с описания используемых доменов. Результат оформляется в виде таблицы (табл. 2.7). На следующей фазе проектирования это описание может уточняться.

Таблица 2.7.

Таблица описания доменов

Имя домена	Имя общего домена	Признак обязательности
Номера	Numeric	NOT NULL
Адреса	Character	NOT NULL
…	….	….

В стандарте IDF1X определено три исходных домена:

– Character – множество всех символьных строк;

– Numeric – множество всех чисел;

– Boolean – значения истина/ложь.

Все остальные домены являются производными от перечисленных выше доменов.

В графе «Признак обязательности» указывается: NULL, если допускается, что атрибут домена может не иметь значений; NOT NULL, если атрибут домена должен обязательно иметь значения. Этот признак обязательно относится ко всем первичным ключам.

Затем производится описание атрибутов, которые выделены в качестве первичного и альтернативных ключей. Результат оформляется в виде таблицы (табл. 2.8).

Далее приступают к реализации связей между сущностями.

Определенные связи между сущностями реализуются посредством миграции первичного ключа родительской сущности в дочернюю (т.е. в дочерней сущности появляется атрибут – внешний ключ). Если связь является идентифицирующей, то первичный ключ мигрирует в область пер-

вичных ключей дочерней сущности. Если связь является не идентифицирующей (обязательной или необязательной), то первичный ключ мигрирует в область атрибутов дочерней сущности (рис. 2.11–2.12).

Таблица 2.8.

Таблица описания атрибутов в инфологической[6] модели

Множество сущностей	Имя атрибута	Имя домена	Примечание
Автовладелец	Код автовладельца	Номера_В	Первичный ключ
Регион	Код региона	Номера_Р	Первичный ключ
Автомобиль	Номер машины	Номера_А	Первичный ключ
Автовладелец	Код региона	Номера_Р	Вторичный ключ
Автовладелец	Код автовладельца	Номера_В	Вторичный ключ

Если связь является иерархической рекурсией, то первичный ключ этой сущности мигрирует в область атрибутов этой же сущности. При этом внешнему ключу должно быть обязательно назначено имя выполняемой роли (рис. 2.13).

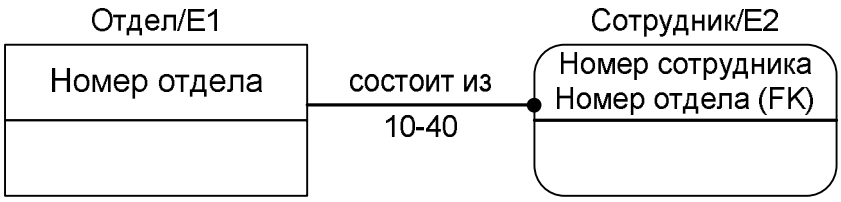

Рисунок 2.11. Отображение идентифицирующей определенной связи в КВ-диаграмме

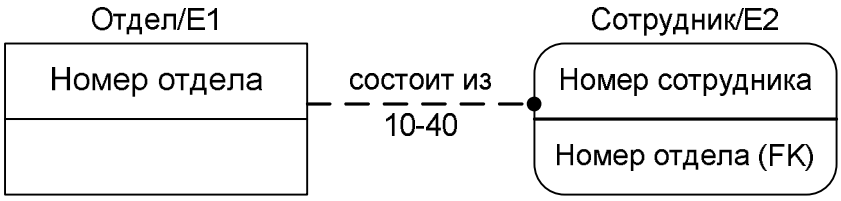

Рисунок 2.12. Отображение обязательной неидентифицирующей определенной связи в КВ-диаграмме

[6] Информационно-логическая модель ≡ Инфологическая модель

Рисунок 2.13. Отображение иерархической рекурсивной связи в КВ-диаграмме

Для реализации связи типа Категория (полная или неполная) первичный ключ родовой сущности мигрирует в область первичных ключей всех сущностей типа категория, участвующих в этой связи (рис. 2.14). При этом первичный ключ сущности типа Категория не имеет других атрибутов, кроме мигрировавших.

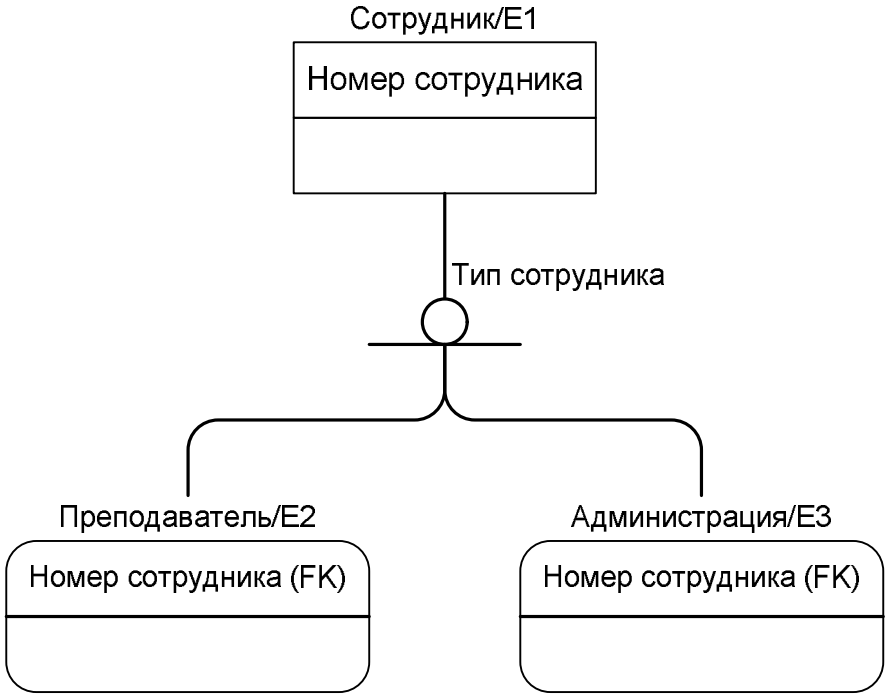

Рисунок 2.14. Пример отображения связи типа Категория в КВ-диаграмме

В КВ-диаграмме примера «Учет автовладельцев» связи между сущностями Автовладелец/1 и Регион/2, Автовладелец/1 и Автомобиль/3 являются обязательными неидентифицирующими (рис. 2.15).

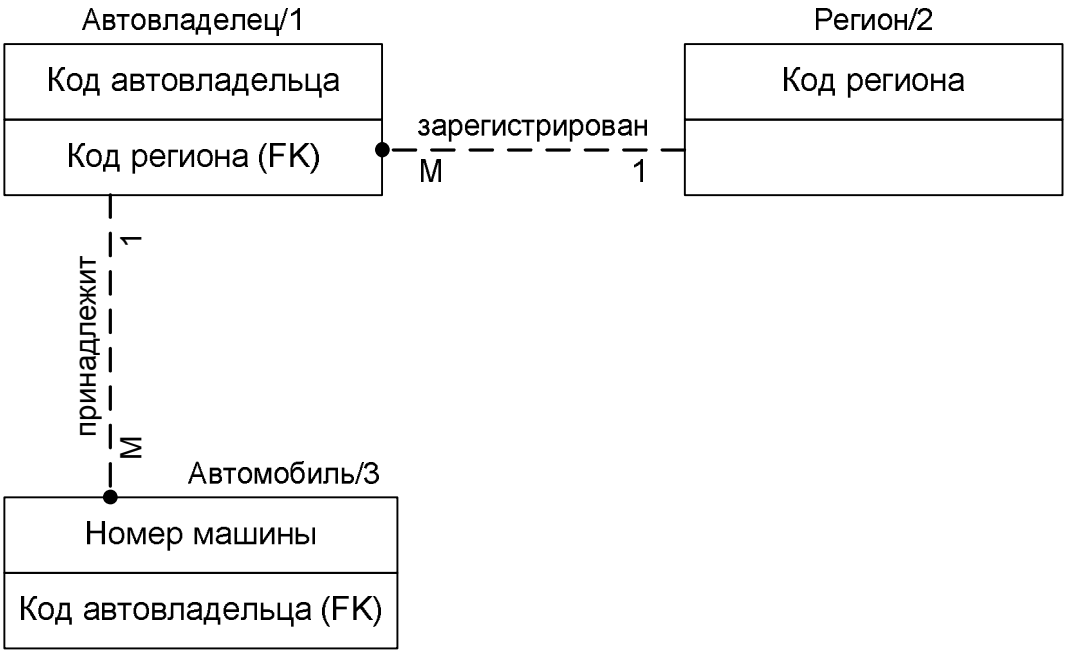

Рисунок 2.15. КВ-диаграмма для базы данных «Учет автовладельцев»

Механизм преобразования неопределенных связей в определенные заключается в следующем:

- каждая неопределенная связь преобразуется в одну новую сущность, которая является дочерней по отношению к связываемым сущностям, и две определенные связи;
- новой сущности присваиваются уникальные имя и номер;
- между новой сущностью и каждой из связываемых сущностей устанавливается идентифицирующая определенная связь; каждой из этих связей назначаются имя и уникальный номер;
- первичные ключи обеих родительских сущностей мигрируют в область первичного ключа новой сущности. Новая сущность является зависимой по идентификации.

Фаза 4. *Построение полноатрибутной модели*

Полноатрибутная модель является дальнейшим развитием модели уровня ключей и включает в себя следующие элементы: пул сущностей, описание доменов, описание атрибутов, описание связей, FA-диаграмму (рис. 2.16).

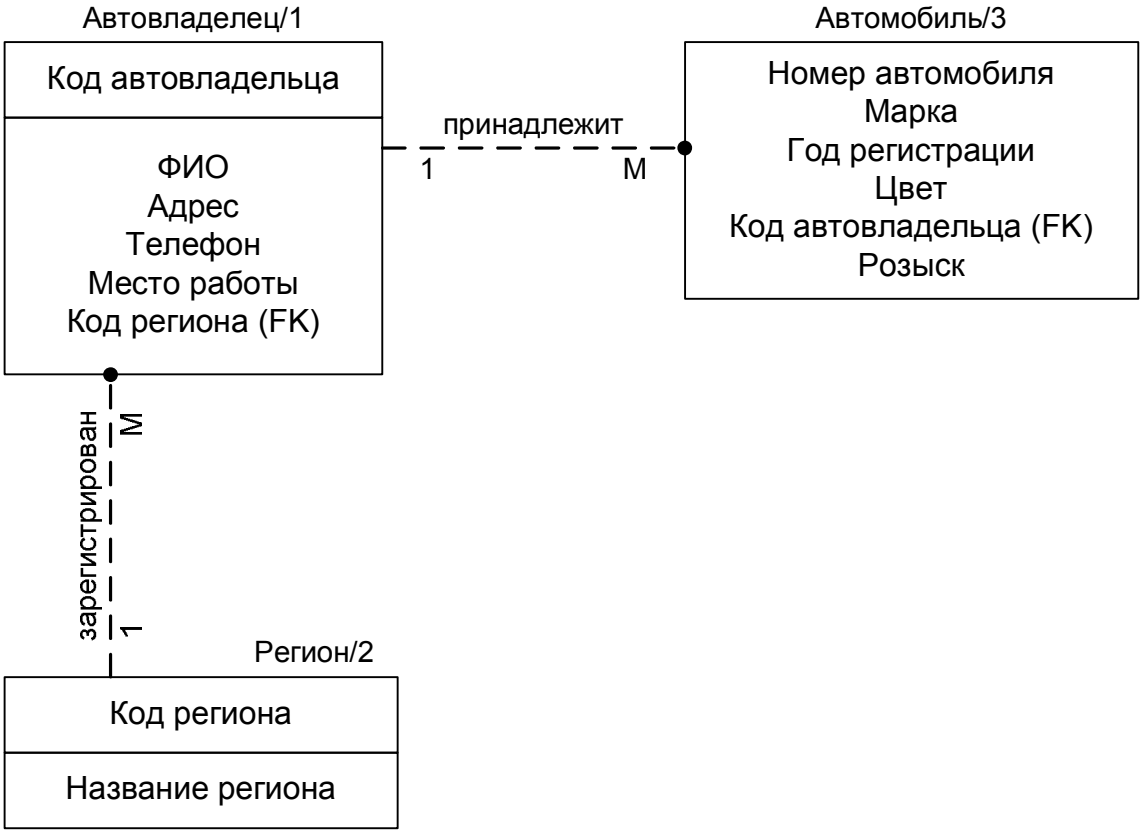

Рисунок 2.16. FA-диаграмма для базы данных «Учет автовладельцев»

На данном этапе проектирования определяются все атрибуты модели, уточняется описание используемых доменов и строится FA-диаграмма. Данная информация сохраняется в соответствующих таблицах, структура которых совпадает со структурой ранее описанных таблиц для хранения информации о ключевых атрибутах (табл. 2.9).

Существует целый ряд программных продуктов, предназначенных для моделирования данных, в том числе для построения диаграмм модели IDEFIX. Наиболее распространенным является программа ERWin.

Таблица 2.9.

Описание атрибутов

Имя атрибута	Тип	Определение	Сущность	Домен
Код автовладельца	Счетчик	Уникальный номер автовладельца	Автовладелец	Номера
Фамилия, Имя, Отчество	Текстовый	Фамилия, Имя, Отчество автовладельца	Автовладелец	Имена
…	….	….	….	….

2.2.1.2. Моделирование данных. Диаграммы потоков данных

Диаграммы потоков данных (DFD) лежат в основе методологии моделирования потоков данных, при котором модель системы строится как иерархия диаграмм потоков данных, описывающих процесс преобразования от ее входа до выдачи пользователю.

Диаграммы верхних уровней иерархии, или контекстные диаграммы, определяют основные процессы или подсистемы ИС с внешними входами и выходами. Их декомпозиция выполняется с помощью диаграмм более низкого уровня, вплоть до элементарных процессов.

Основными компонентами диаграмм потоков данных являются:

- *внешние сущности* – источники или потребители информации, порождающие или принимающие информационные потоки (потоки данных);
- *системы/подсистемы*, преобразующие получаемую информацию и порождающие новые потоки;
- *процессы*, представляющие преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом;
- *накопители данных*, представляющие собой абстрактное устройство для хранения информации, которую можно поместить в накопитель и через некоторое время извлечь;
- *потоки данных*, определяющие информацию, передаваемую через некоторое соединение от источника к приемнику.

В CASE-средствах и системах обычно используют следующий типовой набор графических блоков для обозначения компонентов DFD (в конкретных CASE-средствах и системах этот набор может иметь некоторые отличия):

1. *Внешняя сущность* обозначается прямоугольником с тенью (рис. 2.17).

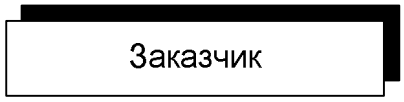

Рисунок 2.17. Внешняя сущность

2. *Система и подсистема* изображаются в форме прямоугольника с полями: номер, имя с определениями и дополнениями и имя проектировщика (рис. 2.18).

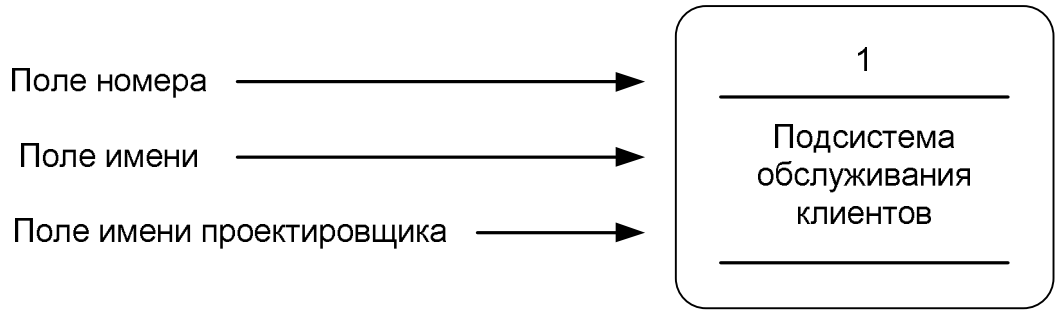

Рисунок 2.18. Подсистема

3. *Процесс* изображается в форме прямоугольника с полями: номер, имя (содержит наименование процесса в виде предложения сделать что-либо) и физической реализации (указывает, какое подразделение, программа или устройство выполняет процесс) (рис. 2.19).

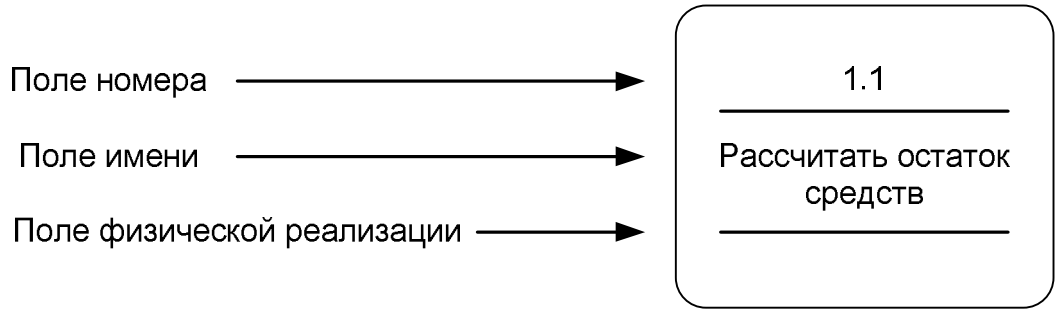

Рисунок 2.19. Процесс

4. *Накопитель данных* изображается в форме прямоугольника без правой (или правой и левой) линии границы: идентификатор (буква D с числом) и имя (указывает на хранимые данные) (рис. 2.20).

Рисунок 2.20. Накопитель данных

5. *Поток данных* изображается линией со стрелкой, показывающей направление потока, и именем, отражающим его содержание (рис. 2.21).

Рисунок 2.21. Поток данных

Построение иерархии диаграмм потоков данных начинается с построения контекстной диаграммы. В случае простой ИС можно ограничиться одной контекстной диаграммой. Применительно к сложной информационной системе требуется построение иерархии контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит набор подсистем, соединенных потоками данных.

Правила построения модели:

1. Все потоки данных должны начинаться или заканчиваться процессом. Данные не могут протекать непосредственно от источника до потребителя или между источником/потребителем и хранилищем данных, если они не проходят через промежуточный процесс.

2. Многочисленные потоки данных между двумя компонентами можно показывать двумя линиями потока данных или двунаправленной стрелкой.

3. Название процесса состоит из глагола, следующего за существительным. В соответствии с соглашением, названия источников, получате-

лей и хранилищ данных использует заглавные буквы, в то время как названиям процесса и потоки данных показываются произвольно.

4. Процессы в уровне 1 диаграмма потока данных перечисляется 1, 2, 3 и так далее. Подпроцессам в декомпозированной диаграмме потока данных назначают номера, начинающиеся с номера родительского процесса.

5. Символы могут быть повторены для облегчения чтения диаграммы.

Основные принципы построения модели:

- *принцип сохранения данных*. Любые данные, которые входят в процесс, должны использоваться или воспроизводиться этим процессом. Любые выходные данные процесса должны быть введены или созданы алгоритмом в пределах процесса. Любые данные, используемые алгоритмом в пределах процесса, должны быть сначала введены в процесс. Любые данные, созданные алгоритмом, должны или использоваться другим алгоритмом в пределах того же самого процесса или выведены процессом.
- *принцип итераций*. Процессы высокого уровня декомпозируются в процессы низшего уровня. На самом низком уровне – примитивные процессы, которые исполняют единственную функцию (или алгоритм).

Контекстная диаграмма (уровень 0) определяет границы системы, выдвигая на первый план источники и получатели. Выделение границы системы при изображении контекстной диаграммы помогает аналитику, пользователю и ответственным менеджерам рассматривать альтернативные логические проекты системы высокого уровня.

Уровень 1 диаграммы потока данных показывает важнейшие процессы системы, хранилища данных, источники и получатели, связанные потоками данных. Процесс уровня 1 является сложным и может включать про-

граммы, руководства, ручные процедуры, аппаратные средства ЭВМ, процедуры и другие действия. Каждый процесс уровень 1 состоит из нескольких подпроцессов, которые внесены в список описаний процессов. Чтобы разбить диаграмму потока данных, аналитик создает независимый уровень 2 диаграммы потока данных для каждого процесса уровня 1.

Функциональный примитив – процесс, который не требует никакого дальнейшего разложения. Отдельные физические компоненты системы находятся на один шаг ниже функциональных примитивов.

Документирование. Элементы данных документируются в словаре данных. Каждый процесс определен в описании процесса, которое обращает внимания на вход и элементы данных выхода и кратко описывает задачи или действия, которые он выполняет. Описания процессов иногда документируются в словаре данных.

Проверка модели включает следующие этапы: 1) проверка синтаксиса; 2) проверка элементов данных; 3) взаимные ссылки; 4) проверка целей.

2.2.1.3. Моделирование данных. Методология функционального моделирования SADT

Методология SADT[7] (технология структурного анализа и проектирования) изначально создавалась для проектирования систем более общего назначения по сравнению с другими структурными методами, выросшими из проектирования программного обеспечения. SADT – одна из самых известных и широко используемых методик проектирования. Новое название методики, принятое в качестве стандарта – IDEF0 (Icam DEFinition) – часть программы ICAM (Integrated Computer-Aided Manufacturing – интегрированная компьютеризация производства), проводимой по инициативе ВВС США. Методология функционального моделирования (SADT) служит для

[7] Structured Analyses and Design Technique

построения функциональной модели структуры объекта какой-либо предметной области: выполняемые действия объекта и связи между объектами.

В IDEF0 система представляется как совокупность взаимодействующих работ (или функций). Связи между работами определяют технологический процесс или структуру взаимосвязи внутри организации. Модель SADT представляет собой серию диаграмм, разбивающих сложный объект на составные части.

Каждый блок IDEF0-диаграммы может быть представлен несколькими блоками, соединенными интерфейсными дугами, на диаграмме следующего уровня. Эти блоки представляют подфункции (подмодули) исходной функции. Каждый из подмодулей может быть декомпозирован аналогичным образом. Число уровней не ограничивается, зато рекомендуется на одной диаграмме использовать не менее 3 и не более 6 блоков.

Работы (activity) обозначают поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты (изображается в виде прямоугольников). Каждая из работ на диаграмме может быть впоследствии декомпозирована (рис. 2.22).

Как отмечалось выше, в терминах IDEF0 система представляется в виде комбинации блоков и дуг. Блоки представляют функции системы, дуги представляют множество объектов (физические объекты, информация или действия, которые образуют связи между функциональными блоками). Взаимодействие работ с внешним миром и между собой описывается в виде стрелок или дуг. С дугами связываются метки на естественном языке, описывающие данные, которые они представляют. Дуги показывают, как функции системы связаны между собой, как они обмениваются данными и осуществляют управление друг другом. Дуги могут разветвляться и соединяться. Ветвление означает множественность (идентичные копии одного объекта) или расщепление (различные части одного объекта). Соединение

означает объединение или слияние объектов. Пять типов стрелок допускаются в диаграммах:

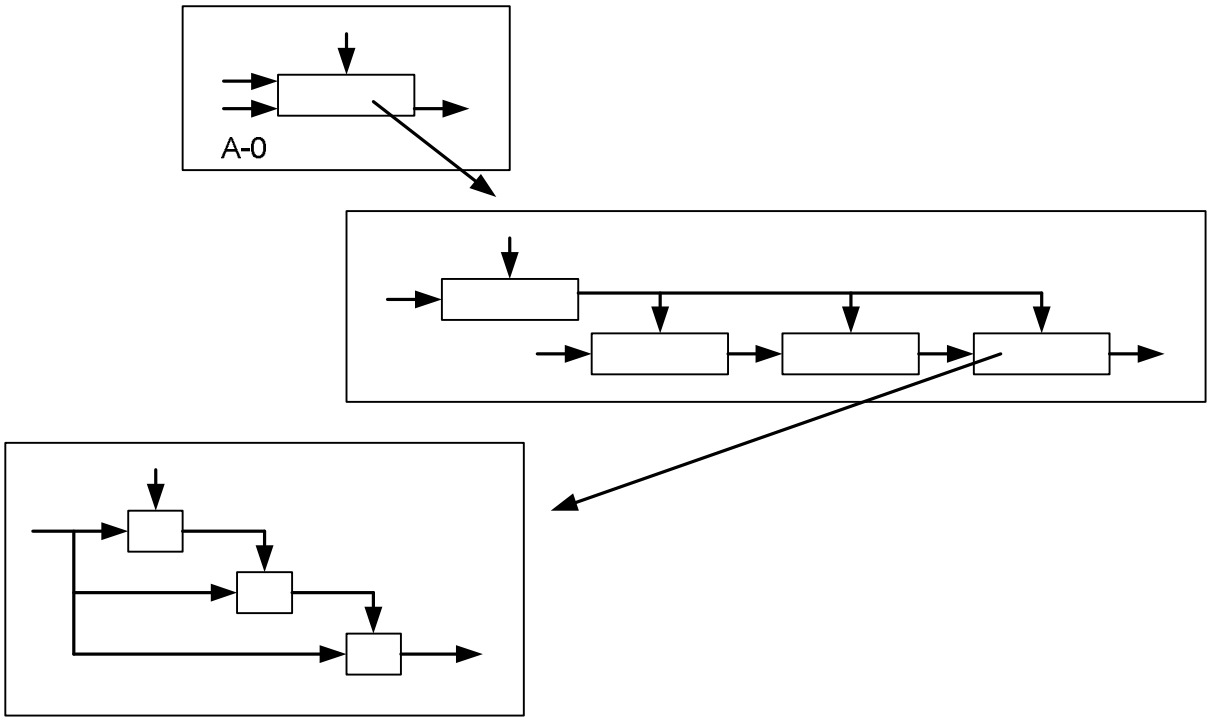

Рисунок 2.22. Блоки IDEF0-диаграммы

Вход (*Input*) – материал или информация, которые используются работой для получения результата (стрелка, входящая в левую грань).

Управление (*Control*) – правила, стратегии, стандарты, которыми руководствуется работа (стрелка, входящая в верхнюю грань). В отличии от входной информации управление не подлежит изменению.

Выход (*Output*) – материал или информация, которые производятся работой (стрелка, исходящая из правой грани). Каждая работа должна иметь хотя бы одну стрелку выхода, т.к. работа без результата не имеет смысла и не должна моделироваться.

Механизм (*Mechanism*) – ресурсы, которые выполняют работу (персонал, станки, устройства – стрелка, входящая в нижнюю грань).

Вызов (*Call*) – стрелка, указывающая на другую модель работы (стрелка, исходящая из нижней грани).

Различают в IDEF0 пять типов связей работ.

Связь по входу (*input–output*), когда выход вышестоящей работы направляется на вход следующей работы (рис. 2.23).

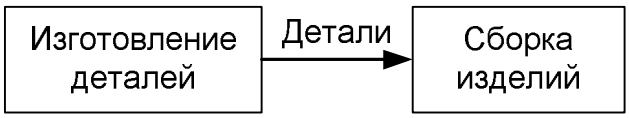

Рисунок 2.23. Связь по входу

Связь по управлению (*output–control*), когда выход вышестоящей работы направляется на управление следующей работы. Связь показывает доминирование вышестоящей работы (рис. 2.24).

Рисунок 2.24. Связь по управлению

Обратная связь по входу (*output–input feedback*), когда выход нижестоящей работы направляется на вход вышестоящей. Используется для описания циклов (рис. 2.25).

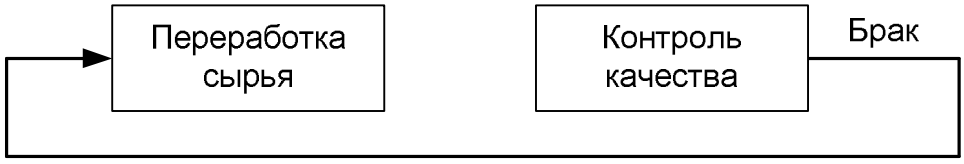

Рисунок 2.25. Обратная связь по входу

Обратная связь по управлению (*output–control feedback*), когда выход нижестоящей работы направляется на управление вышестоящей. Является показателем эффективности бизнес-процесса (рис. 2.26).

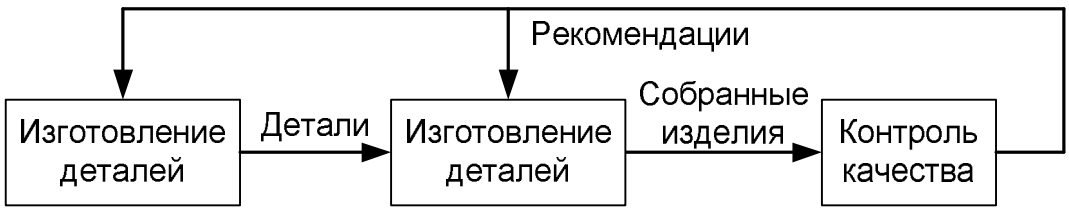

Рисунок 2.26. Обратная связь по управлению

Связь выход–механизм (*output–mechanism*), когда выход одной работы направляется на механизм другой и показывает, что работа подготавливает ресурсы для проведения другой работы (рис. 2.27).

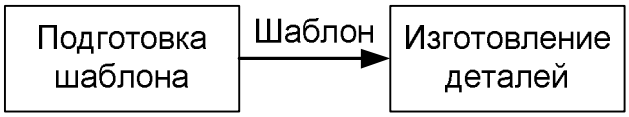

Рисунок 2.27. Связь выход-механизм

Из перечисленных блоков строится диаграмма (рис. 2.28):

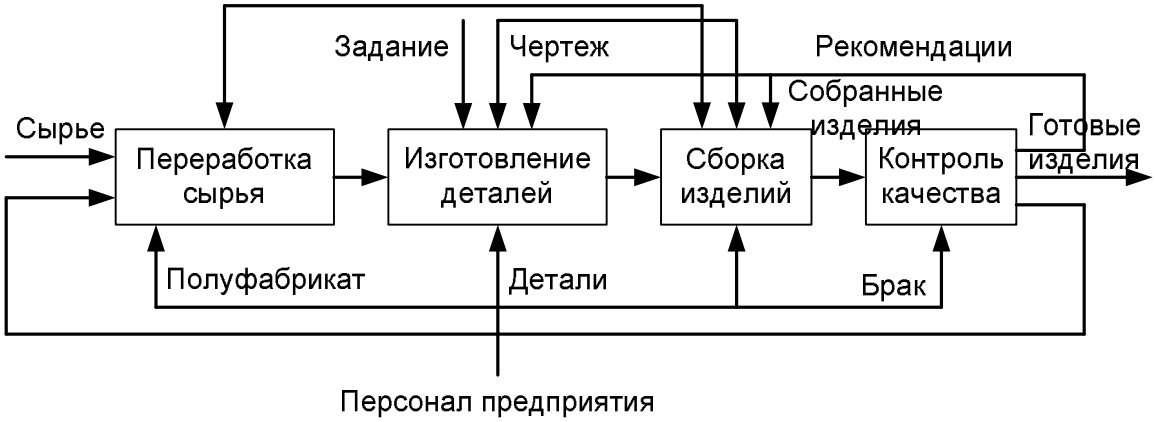

Рисунок 2.28. Пример SADT-диаграммы

2.2.2. CASE-средства проектирования информационных систем

Для автоматизации процессов проектирования и разработки информационных систем в 70–80-е года широко применялась структурная методология, означающая использование формализованных методов описания разрабатываемой системы и принимаемых технических решений. При этом использовались графические средства описания различных моделей информационных систем с помощью схем и диаграмм. Это явилось одной из причин появления программно-технологических средств, получивших название CASE-средств и реализующих их CASE-технологий создания и сопровождения информационных систем.

Термин CASE (Computer Aided Software/System Engineering) используется в весьма широком смысле. Первоначальное значение термина CASE

ограничивалось лишь вопросами автоматизации разработки программного обеспечения. В настоящее время этот термин получил более широкий смысл, означающий *автоматизацию разработки информационных систем*.

CASE-*средства* представляют собой программные средства, поддерживающие процессы создания и/или сопровождения информационных систем, такие как: анализ и формулировка требований, проектирование баз данных и приложений, генерация кода, тестирование, обеспечение качества, управление конфигурацией и проектом.

CASE-*систему* можно определить как набор CASE-средств, имеющих определенное функциональное предназначение и выполненных в рамках единого программного продукта.

CASE-*технология* представляет собой совокупность методологий анализа, проектирования, разработки и сопровождения сложных систем и поддерживается комплексом взаимосвязанных средств автоматизации.

CASE-*индустрия* объединяет сотни фирм и компаний различной направленности деятельности. Практически все серьезные зарубежные программные проекты осуществляются с использованием CASE-средств, а общее число распространяемых пакетов превышает 500 наименований.

Основная цель CASE-*систем и средств* состоит в том, чтобы отделить проектирование программного обеспечения от его кодирования и последующих этапов разработки (тестирование, документирование и пр.), а также автоматизировать весь процесс создания программных систем, или *инжиниринг* (от англ. engineering – разработка).

Современные CASE-средства поддерживают разнообразные технологии проектирования информационных систем: от простых средств анализа и документирования до полномасштабных средств автоматизации, охватывающих весь жизненный цикл программного обеспечения.

Наиболее трудоемкими этапами разработки ИС являются этапы анализа и проектирования, в процессе которых CASE-средства обеспечивают качество принимаемых технических решений и подготовку проектной документации. При этом важную роль играют методы визуального представления информации. Это предполагает построение структурных или иных диаграмм в реальном масштабе времени, использование многообразной цветовой палитры, сквозную проверку синтаксических правил. Графические средства моделирования предметной области позволяют разработчикам в наглядном виде изучать существующую информационную систему, перестраивать ее в соответствии с поставленными целями и имеющимися ограничениями.

CASE-средства составляют основу проекта любой ИС. Методология реализуется через конкретные технологии и поддерживающие их стандарты, методики и инструментальные средства, которые обеспечивают выполнение процессов жизненного цикла информационных систем.

Характерные особенности CASE-средств:

– *Единый графический язык*. CASE-технологии обеспечивают всех участников проекта, включая заказчиков, единым строгим, наглядным и интуитивно понятным графическим языком, позволяющим получать обозримые компоненты с простой и ясной структурой. При этом программы представляются двумерными схемами (более простыми в использовании, чем многостраничные описания), позволяющими заказчику участвовать в процессе разработки, а разработчикам общаться с экспертами предметной области, разделять деятельность системных аналитиков, проектировщиков и программистов, облегчая им защиту проекта перед руководством, а также обеспечивая простоту сопровождения и внесения изменений в систему.

– *Единая база данных проекта*. Основа CASE-технологии – использование базы данных проекта (репозитария) для хранения всей информации о проекте, которая может совместно использоваться разработчиками в соответствии с их правами доступа. Содержимое репозитария включает не только информационные объекты различных типов, но и отношения между их компонентами, а также правила применения или обработки этих компонентов. Репозитарий[8] может хранить объекты различных типов: структурные диаграммы, определения экранов и меню, проекты отчетов, описания данных и логики их обработки, а также модели данных, организации и обработки, исходные коды, элементы данных и т.п.
– *интеграция средств*. На основе репозитария осуществляются интеграция CASE-средств и разделение системной информации между разработчиками. При этом возможности репозитария обеспечивают несколько уровней интеграции: общий пользовательский интерфейс по всем средствам, передачу данных между средствами, интеграцию этапов разработки через единую систему представления фаз жизненного цикла, передачу данных и средств между различными платформами.
– *Поддержка коллективной разработки и управления проектом*. CASE-технология поддерживает групповую разработку проекта, обеспечивая возможность работы в сети, экспорт-импорт любых фрагментов проекта для их развития и/или модификации, а также планирование, контроль, руководство и взаимодействие, то есть функции, необходимые в процессе разработки и сопровождения проектов. Эти функции также реализуются на основе репозита-

[8] В области проектирования ИС для обозначения понятия «хранилище» используют термин «репозитарий» (от английского reposition), в отличие от остальных видов человеческой деятельности, где принят термин «депозитарий» (от латинского depositium).

рия. В частности, через репозитарий могут осуществляться контроль безопасности (ограничения и привилегии доступа), контроль версий и изменений и т.п.

- *Макетирование*. CASE-технология дает возможность быстро строить макеты (прототипы) будущей системы, что позволяет заказчику на ранних этапах разработки оценить, насколько она его устраивает и насколько она приемлема для будущих пользователей.
- *Генерация документации*. Вся документация по проекту генерируется автоматически на базе репозитария (как правило, в соответствии с требованиями действующих стандартов). Несомненное достоинство CASE-технологии заключается в том, что документация всегда отвечает текущему состоянию дел, поскольку любые изменения в проекте автоматически отражаются в репозитарий (известно, что при традиционных подходах к разработке программного обеспечения документация в лучшем случае запаздывает, а ряд модификаций вообще не находит в ней отражения).
- *Верификация проекта*. CASE-технология обеспечивает автоматическую верификацию и контроль проекта на полноту и состоятельность на ранних этапах разработки, что влияет на успех разработки в целом.
- *Автоматическая генерация программного кода*. Генерация программного кода осуществляется на основе репозитария и позволяет автоматически построить до 85–90% текстов на языках высокого уровня.
- *Сопровождение и реинжиниринг*. Сопровождение системы в рамках CASE-технологии характеризуется сопровождением проекта, а не программных кодов. Средства реинжиниринга позволяют создавать модель системы из ее кодов и интегрировать полученные

модели в проект, автоматически обновлять документацию при изменении кодов, автоматически изменять спецификации при редактировании кодов и т.п.

Разработка программ начинается с некоторого предварительного варианта системы. В качестве такого варианта может выступать специально для этого разработанный прототип, либо устаревшая система. В последнем случае для восстановления знаний о программной системе с целью последующего их использования применяют повторную разработку – реинжиниринг.

Повторная разработка сводится к построению исходной модели программной системы путем исследования ее программных кодов. Имея модель, можно ее усовершенствовать, а затем вновь перейти к разработке. Одним из наиболее известных принципов такого типа является принцип возвратного проектирования (Round Trip Engineering (RTE)).

Современные CASE-системы обеспечивают и первичную, и повторную разработку, что существенно ускоряет разработку приложений и повышает их качество.

В настоящее время среди прочих требований к CASE-средствам предъявляются следующие:

- наличие возможностей определения основной модели прикладной задачи (бизнес-модели, обычно объектно-ориентированной) и правил ее поведения (бизнес-правил);
- поддержка процесса проектирования с помощью библиотек, оснащенных средствами хранения, поиска и выбора элементов проектирования (объектов и правил);
- наличие средств для создания пользовательского интерфейса и поддержания распространенных программных интерфейсов (поддержка стандартов OLE, OpenDoc, доступ к библиотекам HTML/Java и т.п.);

– наличие возможностей для создания различных распределенных клиент-серверных приложений.

2.2.2.1. Классификация CASE-средств

При классификации CASE-средств используют следующие признаки:

– ориентация на этапы жизненного цикла;
– функциональная полнота;
– тип используемой модели;
– степень независимости от СУБД;
– допустимые платформы.

По **ориентации на этапы жизненного цикла** выделяют следующие основные типы CASE-средств:

– *средства анализа*, предназначенные для построения и анализа моделей предметной области, например: Design/IDEF (Meta Software) и BPwin (Logic Works);
– *средства анализа и проектирования*, обеспечивающие создание проектных спецификаций, например: Vantage Team Builder (Cayenne), Silverrun (Silverrun Technologies), PRO-IV (McDonnell Douglas) и CASE.Аналитик (МакроПроджект);
– *средства проектирования баз данных*, обеспечивающие моделирование данных и разработку схем баз данных для основных СУБД, например: ERwin (Logic Works), S-Designor (SPD), DataBase Designer (ORACLE);
– *средства разработки приложений*, например: Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer/2000 (ORACLE), New Era (Informix), SQL Windows (Centura) и Delphi (Borland).

По **функциональной полноте** CASE-системы и средства можно условно разделить на следующие типы:

- системы, предназначенные для решения частных задач на одном или нескольких этапах жизненного цикла, например, ERwin (Logic Works), S-Designor (SPD), CASE.Аналитик (МакроПроджект) и Silverrun (Silverrun Technologies);
- интегрированные системы, поддерживающие весь жизненный цикл ИС и связанные с общим репозиторием, например система Vantage Team Builder (Cayenne) и система Designer/2000 с системой разработки приложений Developer/2000 (ORACLE).

По **типу используемых моделей** CASE-системы условно можно разделить на три основные разновидности: структурные, объектно-ориентированные и комбинированные.

Исторически первые *структурные* CASE-системы основаны на методах структурного и модульного программирования, структурного анализа и синтеза, например, система Vantage Team Builder (Cayenne).

Объектно-ориентированные методы и CASE-системы получили массовое использование с начала 90-х годов. Они позволяют сократить сроки разработки, а также повысить надежность и эффективность функционирования ИС. Примерами объектно-ориентированных CASE-систем являются Rational Rose (Rational Software) и Object Team (Cayenne).

Комбинированные инструментальные средства поддерживают одновременно структурные и объектно-ориентированные методы, например: Designer/ 2000 (ORACLE).

По **степени независимости от СУБД** CASE-системы можно разделить на две группы:

- *независимые* системы CASE-системы поставляются в виде автономных систем, не входящих в состав конкретной СУБД. Обычно они поддерживают несколько форматов баз данных через интер-

фейс ODBC. К числу независимых CASE-систем относятся S-Designor (SDP, Powersoft), ERwin (LogicWorks) и Silverrmi (Computer Systems Advisers Inc.).

– *встроенные* в СУБД CASE-системы обычно поддерживают главным образом формат баз данных СУБД, в состав которой они входят. При этом возможна поддержка и других форматов баз данных. Примером встроенной системы является Designer/2000, входящая в состав СУБД ORACLE.

Кратко остановимся на некоторых из них.

S-Designor[9] представляет собой графический инструмент, позволяющий в определенной степени автоматизировать процесс проектирования реляционных БД. При разработке структуры БД с помощью S-Designor формируется концептуальная модель данных, которая впоследствии преобразуется в физическую модель данных.

Для описания концептуальной модели данных предоставляются удобные средства графического интерфейса в стиле MS Windows; можно задать правила контроля ограничений, накладываемых на столбец таблицы (минимальное и максимальное значения, умалчиваемое значение и список допустимых значений).

Построение физической модели данных проводится на основе концептуальной модели и означает создание таблиц и описаний структур БД для некоторой СУБД или построение готового приложения в специальной среде разработки, например PowerBuilder.

При генерации физической модели данных каждой сущности ставится в соответствие таблица, атрибуты сущностей преобразуются в колонки таблиц, а идентификаторы сущностей становятся ключами.

---

[9] Начиная с версии S-Designor 6.0, продукт выпускается под названием PowerDesigner 6.0

Система позволяет создавать базы данных путем подключения к работающему серверу СУБД через интерфейс ODBC или готовить текстовые файлы (пакеты) SQL-операторов по созданию структуры БД. Файлы SQL-операторов после этого обрабатываются некоторой СУБД, в результате чего создаются нужные БД.

S-Designor имеет интерфейсы со многими СУБД, включая Oracle, Ingress Informix, Sybase, SQL Server, Access и Paradox.

Система S-Designor работает в среде Windows и обеспечивает возможность использования других инструментальных средств разработки программ, таких как PowerBuilder, Team Windows и Progress. Для инструментальной системы PowerBuilder пакет S-Designor позволяет выполнить автоматическую генерацию приложения.

Создание таблиц БД сопровождается синтезом средств обеспечения поддержки ссылочной целостности данных в соответствии с типом конкретной СУБД.

Наряду с синтезируемыми программными объектами, система S-Designor поддерживает генерацию отчетов о концептуальной и физической моделях данных. Отчеты можно готовить в виде ASCII-текстов или в формате RTF для текстовых процессоров, например MS Word.

В системе S-Designor имеются средства администрирования групповой работы с парольной защитой.

ERwin представляет собой систему концептуального моделирования баз данных. Система ERwin реализует проектирование схемы БД, генерацию ее описания на языке целевой СУБД (Oracle, Sybase, MS SQL Server и др.) и реинжиниринг баз данных. Для ряда систем быстрой разработки приложений (PowerBuilder, SQL Windows, Delphi, Visual Basic) обеспечивается генерация форм и прототипов приложений.

ERwin – это графический инструментарий для моделирования данных, основной целью которого является помощь аналитику в использова-

нии бизнес-правил и требований к информации при создании логических и физических моделей данных. ERwin позволяет проектировать, документировать и сопровождать базы данных, хранилища данных и витрины данных (data marts). Создав наглядную модель базы данных, вы сможете оптимизировать структуру БД и добиться её полного соответствия требованиям и задачам организации. Визуальное моделирование повышает качество создаваемой базы данных, продуктивность и скорость её разработки.

По функциональным возможностям ERwin близок к S-Designor. В ERwin связь с СУБД организуется напрямую, в отличие от S-Designor, в которой связь с СУБД осуществляется через ODBC-интерфейс с использованием внешних файлов. Отсюда следует, что ERwin менее универсальна и поддерживает меньшее число СУБД.

Silverrun представляет собой открытую систему, используемую совместно с продуктами других различных фирм. Она имеет интерфейсы с СУБД, в число которых входят следующие: DB2, Informix, Ingress, Oracle, Progress, SQLBase, SQLServer. Система Silverrun имеет интерфейсы с системами программирования для языков четвертого поколения, включая системы PowerBuilder, Progress, SQLWindows, Uniface.

Область применения системы Silverrun – инструментальная поддержка структурных методологий информационных систем бизнес-класса. Эта система ориентирована на начальные стадии проектирования и может быть использована специалистами по анализу и моделированию деятельности организаций, разработчиками информационных систем, а также администраторами БД.

Она позволяет независимо строить модели двух видов: функциональные и информационные. Функциональные модели в виде диаграмм потоков данных DFD ориентированы на пользователей-заказчиков для обоснования требований и постановки задач. Информационные модели в

виде диаграмм СУЩНОСТЬ-СВЯЗЬ служат для генерации схем баз данных.

Система Silverrun состоит из трех основных подсистем: модуля построения диаграмм потоков данных и двух модулей построения диаграмм типа сущность-связь: модуля концептуальных моделей ERX (Entity Relationship eXpert) и модуля реляционных моделей RDM (Relational Data Modeler).

В целом система Silverrun по своим возможностям близка к системам S-Designor и ERwin. Система поддерживает коллективную разработку в разнородной среде и может функционировать на платформах Windows, OS/2, Macintosh и Solaris.

Встроенная CASE-система **Designer/2000** фирмы ORACLE является встроенной и используется в СУБД Oracle. Основу CASE-технологии, реализованной в продуктах фирмы ORACLE, составляют:

- методология структурного нисходящего проектирования;
- поддержка всех этапов жизненного цикла прикладной системы;
- ориентация на технологию «клиент-сервер»;
- наличие централизованной базы данных (репозитория) для хранения всей информации в ходе проектирования;
- возможность одновременной работы с репозиторием многих пользователей;
- автоматизация последовательного перехода между этапами разработки;
- автоматизация проектирования и создания приложения (создание документации, проверка спецификаций, автоматическая генерация программ и т.д.).

Система Designer/2000 поддерживает следующие этапы разработки прикладных систем: моделирование и анализ деятельности организации,

разработку концептуальных моделей предметной области, проектирование приложения и синтез программ.

Средства поддержки этапа моделирования и анализа позволяют строить наглядные модели технологических и организационных процессов и структур организации для изучения и совершенствования. При этом широко применяются средства мультимедиа, включая звуковое сопровождение, видео и анимацию.

Модель деятельности организации представляется в виде совокупности диаграмм, описывающих отдельные процессы. Диаграммы строятся из стандартных элементов, основными из которых являются: *базовый процесс, шаг процесса, хранилище, поток, организационные единицы и события*.

На этапе концептуального моделирования предметной области строятся модели, описывающие особенности предметной области, характер решаемых задач, информационные потребности и ресурсы, технологические ограничения и т, д. Используются модели двух видов: информационные (отражают существующие информационные структуры и взаимосвязи между ними) и функциональные (отражают технологию и способы обработки информации).

Основой информационных моделей является специальный вид модели Чена, близкий к бинарной модели типа <сущность-связь>.

Функциональное описание предметной области производится с помощью диаграмм иерархии функций и моделей потоков данных. Первый вид моделей предполагает декомпозицию общей функции на подфункции, каждая из которых, в свою очередь, раскладывается на более мелкие функции и т. д. При необходимости можно описать события, вызывающие выполнение определенной функции. Диаграммы потоков данных позволяют описать движение данных в процессе работы организационных структур.

Концептуальное моделирование в системе Designer/2000 поддерживается совокупностью графических редакторов: ER-диаграмм, иерархии функций и диаграмм потоков данных. Кроме представления моделей, редакторы позволяют вводить дополнительную информацию об элементах диаграмм, выполнять семантические проверки диаграмм на полноту и корректность, получать отчеты и документы по концептуальному моделированию.

На этапе проектирования из полученных концептуальных моделей вырабатываются технические спецификации на прикладную систему, описывающие структуру и состав БД, а также набор программных модулей. Создаваемые спецификации разделяются на информационные и функциональные, как и исходные концептуальные модели.

Функциональное описание будущего приложения предполагает определение: структуры меню пользовательского интерфейса, экранных форм, отчетов, процедурных модулей и прочее.

Этап проектирования реализуется с помощью трех редакторов: схем программ, диаграмм взаимосвязей модулей и схем модуля. Перечисленные редакторы, кроме построения диаграмм, позволяют вводить дополнительную информацию об отдельных элементах диаграмм.

На этапе создания программ используются генераторы программного кода, которые позволяют автоматизировать этот этап, существенно сократить время разработки, повысить качество и надежность получаемого продукта. Имеющиеся в системе генераторы делятся на две группы: генератор серверной части и генераторы клиентских частей.

Объектно-ориентированные CASE-системы. Областью применения объектно-ориентированных инструментальных систем являются сложные проекты, такие как: создание операционных систем, средств разработки приложений и систем реального времени.

В рамках объектно-ориентированного подхода существует множество моделей описания (нотаций) и методов разработки программных систем.

Современные объектно-ориентированные CASE-системы можно разделить на две основные группы: CASE-средства, поддерживающие несколько объектно-ориентированных моделей, и средства, ориентированные только на один вид моделей. В системах первого типа обычно имеется возможность перехода от одной модели к другой. Иногда в этих системах предоставляется возможность создавать собственные нотации.

Rational Rose представляет собой семейство объектно-ориентированных CASE-систем фирмы Rational Software Corporation, служащее для автоматизации анализа и проектирования ПО, генерации кодов на различных языках и подготовки проектной документации. Кроме того, в его составе имеются средства реинжиниринга программ, обеспечивающие повторное использование программных компонентов в новых проектах. В этой системе используется синтез-методология объектно-ориентированного анализа проектирования Г. Буча, Д. Рамбо и И. Джекобсона, их унифицированный язык моделирования UML.

Конкретный вариант системы определяется языком, на котором выполняется генерация кодов программ (C++, Smalltalk, PowerBuilder, Ada, SqlWindows и OhjectPro). Основным вариантом системы является Rational Rose/C++, позволяющий генерировать программные коды на C++, подготавливать проектную документацию в виде диаграмм и спецификаций.

В процессе работы с помощью Rational Rose выполняется построение диаграмм и спецификаций, определяющих логическую и физическую структуру модели, ее статические и динамические свойства. В их состав входят следующие диаграммы: *классов, состояний, сценариев, модулей и процессов*. Основными компонентами системы являются следующие:

- репозиторий, представляющий объектно-ориентированную БД;

- графический интерфейс пользователя;
- средства просмотра проекта, обеспечивающие перемещение по элементам проекта, в том числе по иерархиям классов и подсистем, переключение между видами диаграмм;
- средства контроля проекта, позволяющие находить и устранять ошибки;
- средства сбора статистики;
- генератор документов, позволяющий формировать тексты выходных документов на основе информации из репозитория.

Кроме того, для каждого языка программирования добавляется свой генератор кода и анализатор для C++, обеспечивающий восстановление модели проекта по исходным текстам программ (реинжиниринг). Средства автоматической генерации кодов программ на C++ на основе логической и физической моделей проекта формируют заголовочные файлы и файлы описаний классов и объектов. Полученный таким образом скелет программы можно дополнить путем непосредственного программирования на C++.

Анализатор кодов C++ позволяет создавать модули проектов по информации, содержащейся в определяемых пользователем исходных текстах программ. Анализатор осуществляет контроль правильности исходных текстов и диагностику ошибок. Полученная в результате модель может использоваться в нескольких проектах.

2.2.2.2. Рекомендации по применению CASE-систем

Различные CASE-системы могут взаимодействовать друг с другом напрямую (ERWin и BPWin), либо с помощью дополнительных модулей (Model Mart – средство коллективной разработки, ERWin Translation Wizard – модуль импорта в ERWin моделей, созданных в Rational Rose).

Кратко обобщим вышеизложенную информацию по использованию CASE-средств при проектировании ИС.

1. CASE-системы позволяют ускорить и облегчить разработку, повысить качество создаваемых программ и информационных систем. Многие из CASE-систем имеют средства управления коллективной работой над проектом.

2. CASE-системы особенно полезными оказываются на начальных этапах разработки. Они являются необязательной частью инструментария разработчика и пока не могут подменить средства проектирования и разработки в составе СУБД. Одной из основных причин этого является разнообразие средств разработки приложений, программно-аппаратных платформ и методологий проектирования.

3. Предоставляемая многими CASE-системами возможность перехода от концептуальной модели БД к физической и обратно полезна для решения задач анализа, совершенствования и переноса приложений из среды одной СУБД в другую.

4. Большинство современных CASE-систем являются структурными, но благодаря некоторым преимуществам объектно-ориентированных систем последние приобретают все большую популярность, особенно при реализации сложных проектов.

5. Современные CASE-системы ориентированы на квалифицированного пользователя, поскольку для их использования требуется знание теории проектирования баз данных. Так, например, для разработки структуры БД с помощью системы S-Designor информацию о проектируемой информационной системе нужно представить в виде ER-модели.

2.2.3. Объектно-ориентированные модели

Популярность объектно-ориентированных технологий привела к сближению большинства известных моделей. Многообразие моделей порождает трудности проектировщиков по выбору модели и по обмену информацией при работе над разными проектами. В этой связи известные

специалисты Г. Буч, Д. Рамбо и И. Джекобсон при поддержке фирмы Rational Software Corporation провели работу над унифицированной моделью и методом, получившим название UML (Unified Modeling Language – унифицированный язык моделирования).

2.2.3.1. Общая характеристика унифицированного языка моделирования UML

UML представляет собой единый язык моделирования, предназначенный для спецификации, визуализации, конструирования и документирования материалов программных систем, а также для моделирования бизнеса и других непрограммных систем.

UML можно определить также как промышленный объектно-ориентированный стандарт моделирования. Он включает в себя в унифицированном виде лучшие методы визуального (графического) моделирования. В настоящее время имеется целый ряд инструментальных средств, производители которых заявляют о поддержке UML, среди них можно выделить: Rational Rose, Select Enterprise, Platinum и Visual Modeler. На заключительной стадии разработки, унификации и принятия UML в качестве стандарта большой вклад внес консорциум OMG (Object Management Group – группа управления объектом).

Типы диаграмм UML. Создаваемый с помощью UML проект ИС может включать в себя следующие 8 видов диаграмм (diagrams):

- прецедентов использования (use case);
- классов (class);
- состояний (statechart);
- активности (activity);
- следования (sequence);
- сотрудничества (collaboration);
- компонентов (component);

– размещения (deployment).

Диаграммы *состояний*, *активности*, *следования* и *сотрудничества* образуют набор диаграмм, служащих для описания поведения разрабатываемой ИС. Причем, последние две обеспечивают описание взаимодействия объектов ИС. Диаграммы *компонентов* и *размещения* описывают физическую реализацию ИС.

2.2.3.2. Проектирование ИС с использованием UML

Порядок работы.

1. Разработка модели бизнес-прецедентов.
2. Разработка модели бизнес-объектов.
3. Разработка концептуальной модели данных.
4. Разработка требований к системе.
5. Анализ требований и предварительное проектирование системы.
6. Разработка моделей базы данных и приложений.
7. Проектирование физической реализации системы.

На этапе создания **концептуальной модели** для описания бизнес-деятельности используются *модели бизнес-прецедентов* и *диаграммы видов деятельности*, для описания бизнес-объектов – *модели бизнес-объектов* и *диаграммы последовательностей*.

На этапе создания **логической модели** ИС описание требований к системе задается в виде *модели и описания системных прецедентов*, а предварительное проектирование осуществляется с использованием *диаграмм классов, диаграмм последовательностей и диаграмм состояний*.

На этапе создания **физической модели** детальное проектирование выполняется с использованием *диаграмм классов, диаграмм компонентов, диаграмм развертывания*.

На рис. 2.29 показаны отношения между различными видами диаграмм UML. Указатели стрелок можно интерпретировать как отношение

«является источником входных данных для...» (например, диаграмма прецедентов является источником данных для диаграмм видов деятельности и последовательности). Приведенная схема является наглядной иллюстрацией итеративного характера разработки моделей с использованием UML.

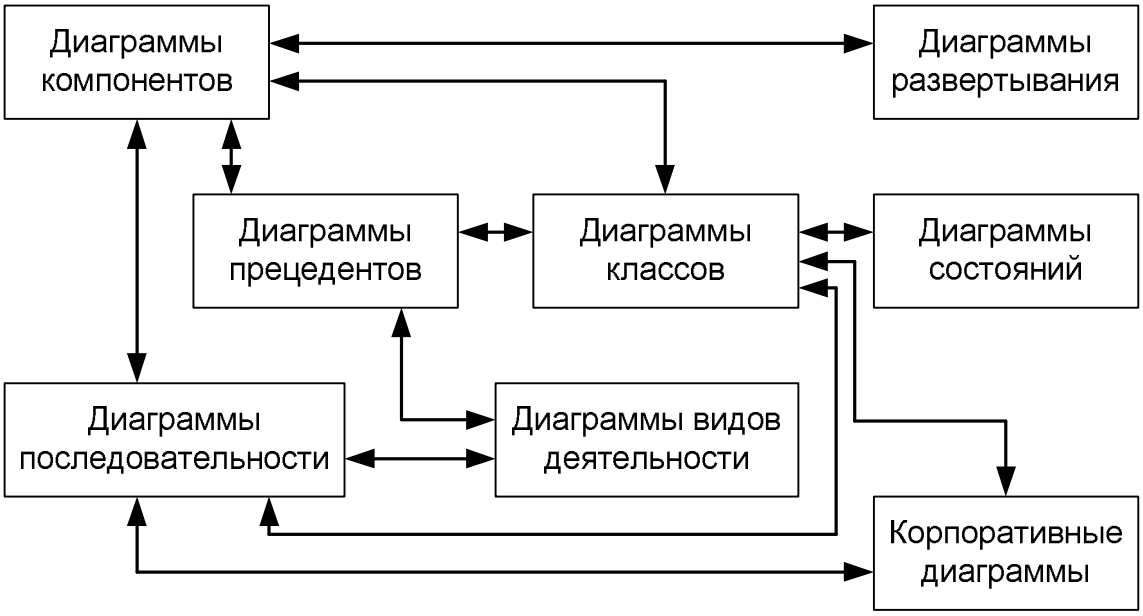

Рисунок 2.29. Отношения между различными видами диаграмм UML

Этап 1. *Разработка модели бизнес-прецедентов*

Модель бизнес-прецедентов описывает бизнес-процессы с точки зрения внешнего пользователя, т.е. отражает взгляд на деятельность организации извне. Проектирование системы начинается с изучения и моделирования бизнес-деятельности организации. На этом этапе вводится и отображается в модели ряд понятий, свойственных объектно-ориентированному подходу.

Исполнитель (действующее лицо, Actor) – личность, организация или система, взаимодействующая с ИС; различают внешнего исполнителя (который использует или используется системой, т.е. порождает прецеденты деятельности) и внутреннего исполнителя (который обеспечивает реализацию прецедентов деятельности внутри системы). На диаграмме исполнитель представляется стилизованной фигуркой человека.

Прецедент – законченная последовательность действий, инициированная внешним объектом (личностью или системой), которая взаимодействует с ИС и получает в результате некоторое сообщение от ИС. На диаграмме представляется овалом с надписью, отражающей содержание действия.

Класс – описание совокупности однородных объектов с их атрибутами, операциями, отношениями и семантикой. На диаграмме представляется прямоугольником, содержащим описания атрибутов и операций класса.

Ассоциация – связь между двумя элементами модели. На диаграмме представляется линией.

Обобщение – связь между двумя элементами модели, когда один элемент (подкласс) является частным случаем другого элемента (суперкласса). На диаграмме представляется стрелкой.

Агрегация – отношение между элементами модели, когда один элемент является частью другого элемента (агрегата). На диаграмме представляется стрелкой с ромбовидным концом.

Для иллюстрации этапов разработки проекта использованы адаптированные материалы проекта ИС учебного заведения[10]. Назначение ИС – автоматизация ведения и использования данных об успеваемости студентов. В настоящее время эта работа производится вручную персоналом ВУЗа. На рис. 2.30 представлена общая модель деятельности ВУЗа в виде диаграммы прецедентов. Прецедент «Обучение» реализуется через множество других, более ограниченных прецедентов (рис. 2.30), отражающих детализацию представления функционирования ВУЗа.

---

[10] Марков Н.Г. Базы данных. Учебное пособие / – Томск: Томский политехнический университет, 2002. –115 с.

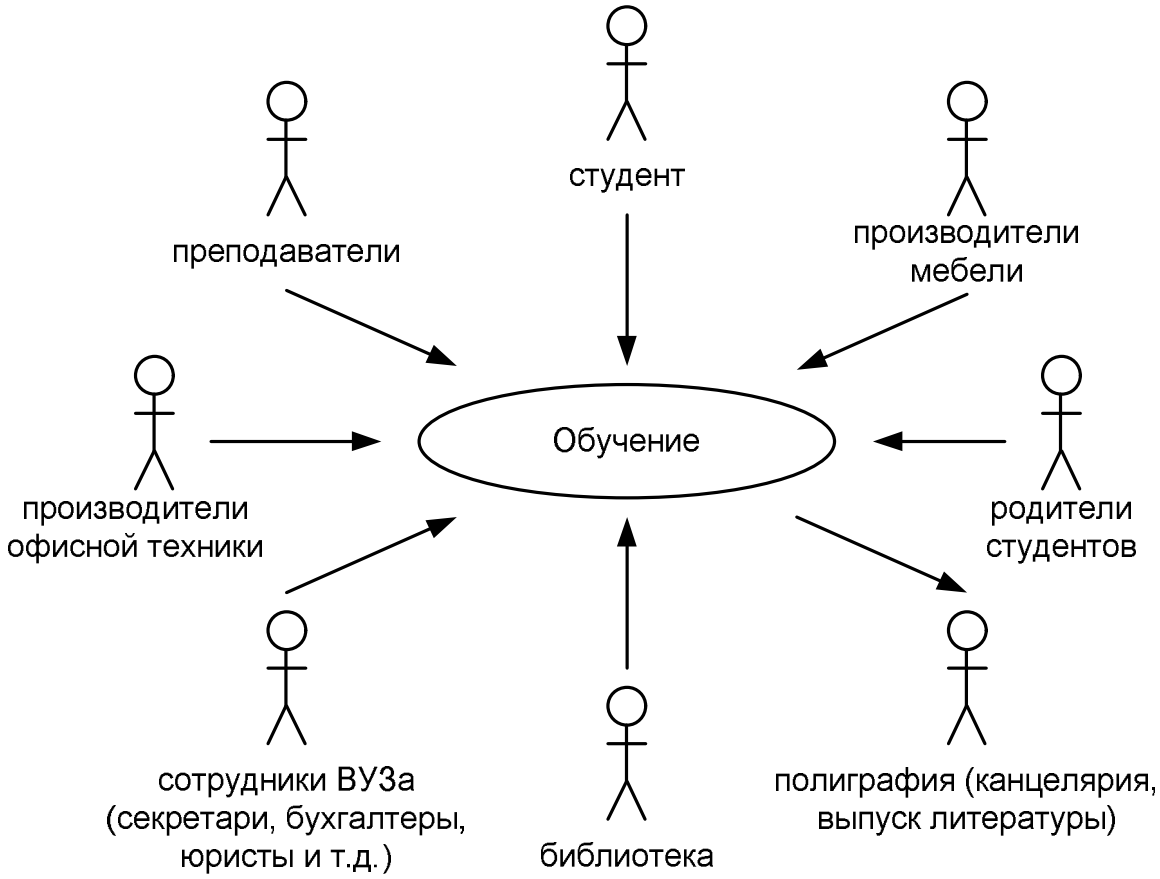

Рисунок 2.30. Общая диаграмма деятельности ВУЗа по обучению студентов

Для включения в диаграмму выбранные прецеденты должны удовлетворять следующим критериям:

- прецедент должен описывать, ЧТО нужно делать, а не КАК;
- прецедент должен описывать действия с точки зрения ИСПОЛНИТЕЛЯ;
- прецедент должен возвращать исполнителю некоторое СООБЩЕНИЕ;
- последовательность действий внутри прецедента должна представлять собой одну НЕДЕЛИМУЮ цепочку.

Исходя из цели создания системы, для дальнейшего исследования и моделирования отбираются только те бизнес-прецеденты, которые связаны с использованием записей об успеваемости студентов.

Выполнение прецедента описывается с помощью диаграмм видов деятельности, которые отображают исполнителей и последовательность выполнения соответствующих бизнес-процессов (рис. 2.31).

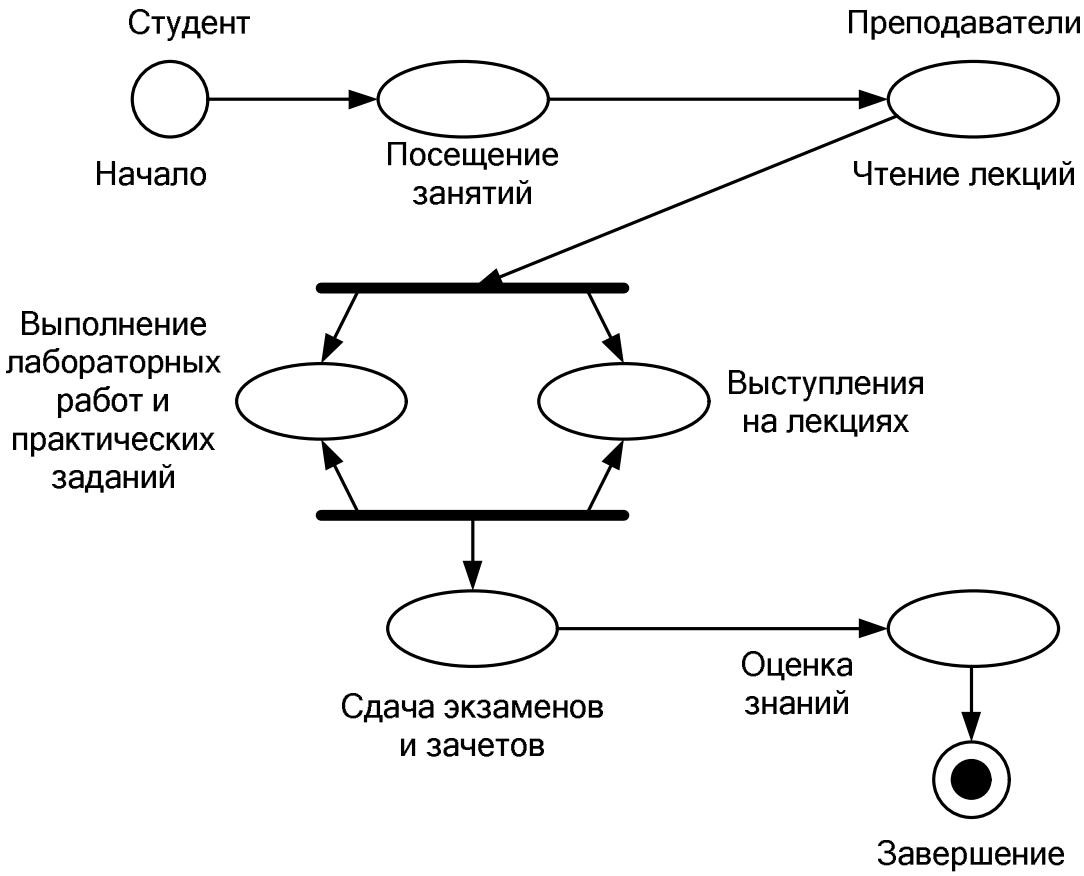

Рисунок 2.31. Модель бизнес - прецедентов, составляющих процесс обучения

Несмотря на то, что проведение учебной деятельности предусматривает множество разнообразных действий исполнителей, для нашей задачи существенными являются только процессы обмена информацией между этими исполнителями, и именно они отображаются в создаваемых моделях. Поэтому на диаграмме (рис. 2.32) отражен процесс оценки действий студента на основании выполненных им заданий в процессе обучения, а также присутствия и выступления на лекциях.

Общее поле диаграммы деятельности делится на несколько «плавательных дорожек», каждая из которых содержит описание действий одного

из исполнителей. Основными элементами диаграмм видов деятельности являются обозначения состояния («начало», «конец»), действия (овал) и момента синхронизации действий (линейка синхронизации, на которой сходятся или разветвляются несколько стрелок).

Этап завершается после разработки диаграмм видов деятельности для всех выделенных в модели бизнес-прецедентов. Естественно, на последующих этапах анализа и проектирования будут выявлены какие-то важные подробности в описании деятельности объекта автоматизации. Поэтому разработанные на данном этапе модели будут еще неоднократно корректироваться.

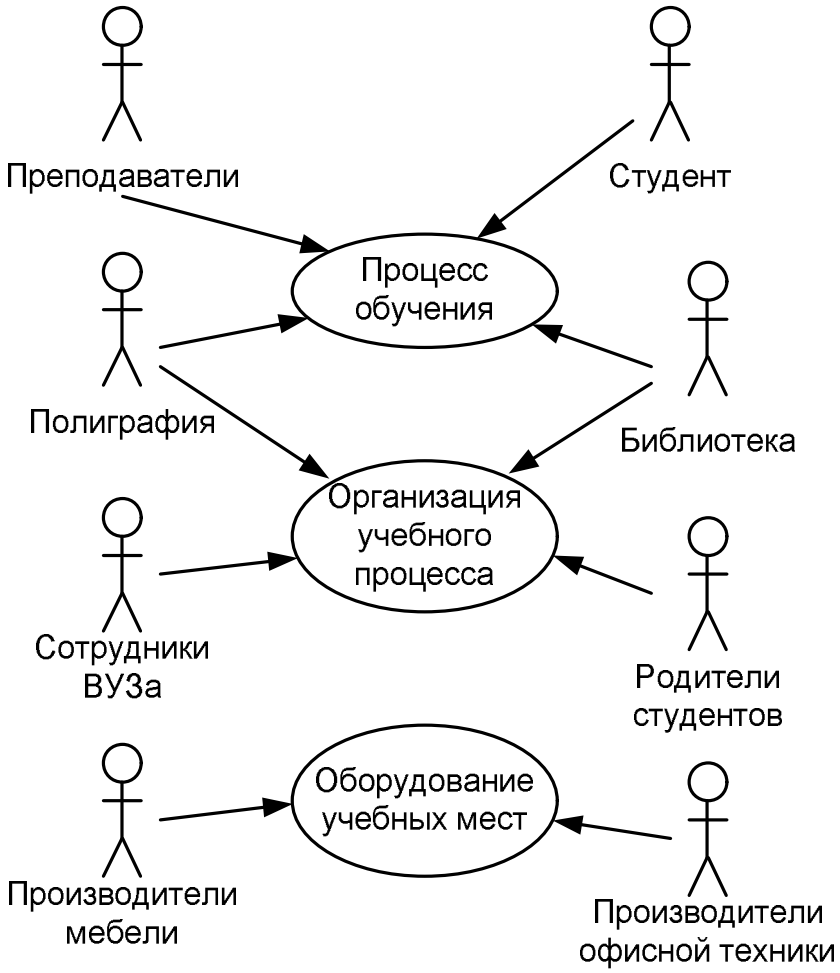

Рисунок 2.32. Диаграмма видов деятельности для прецедента «Процесс обучения»

Этап 2. *Разработка модели бизнес-объектов*

Данный этап показывает выполнение бизнес-процессов организации ее исполнителями. Основными компонентами моделей бизнес-объектов являются исполнители, а также бизнес-сущности, отображающие все, что используют исполнители для реализации бизнес-процессов. Пример модели бизнес-объектов для прецедента «Ответ на запрос» приведен на рис. 2.33.

В этой диаграмме появилось новое действующее лицо – отправитель запроса об успеваемости. На самом деле с запросом об успеваемости студента могут обращаться в систему многие из действующих лиц: преподаватели, родители студента, а также сам студент. Таким образом, понятие «Отправитель запроса об успеваемости» служит для обобщенного представления всех этих действующих лиц при описании прецедента «Ответ на запрос» (рис. 2.34). «Отправитель запроса об успеваемости» становится суперклассом по отношению к обобщаемым понятиям (подклассам).

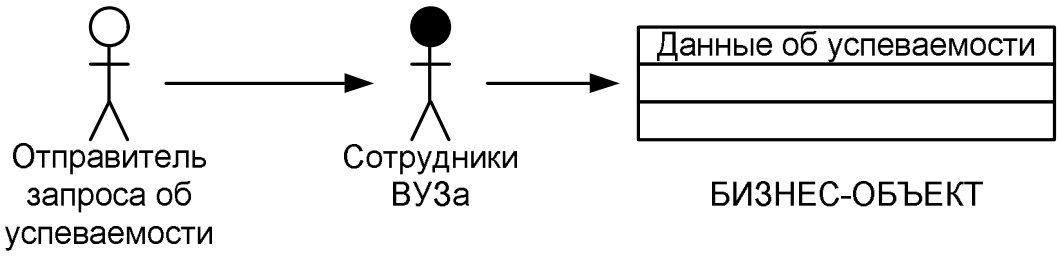

Рисунок 2.33. Модель бизнес-объектов прецедента «Ответ на запрос»

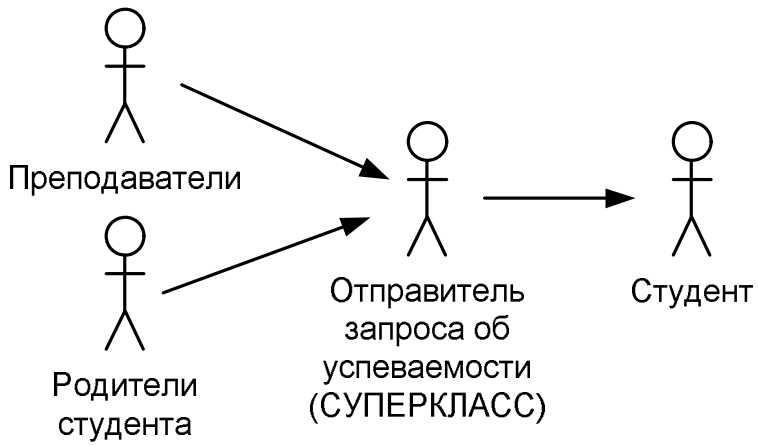

Рисунок 2.34. Обобщение классов

Для детального описания выполнения бизнес-процессов обычно используются диаграммы последовательностей (рис. 2.35).

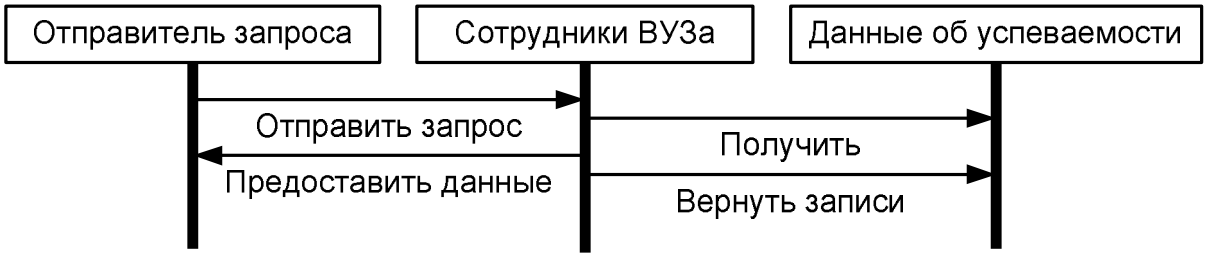

Рисунок 2.35. Диаграмма последовательностей для прецедента «Ответ на запрос»

Основными элементами диаграммы последовательностей являются обозначения объектов (прямоугольники), вертикальные линии, отображающие течение времени при деятельности объекта, и стрелки, показывающие выполнение действий объектами.

Результатом этого этапа являются согласованные с заказчиком и достаточно подробные описания действий специалистов организации, внедряющей ИС, необходимые для обеспечения исполнения ее функций.

Этап 3. *Разработка концептуальной модели данных*

Разработка концептуальной модели данных выполняется на основе информации, выявленной на этапах бизнес-моделирования. На рис. 2.36 представлена в виде диаграммы классов модель данных для объекта «Данные об успеваемости».

Модель показывает, что данные об успеваемости включают (агрегируют) ряд блоков. При этом «данные о студенте» и «данные о начальном образовании» могут быть включены в каждую запись об успеваемости в единственном экземпляре, а блоки «сведения о зачетах», «оценки за экзамены», «выполнение лр и пр», «участия в лекциях», «посещение» могут повторяться неограниченное число раз.

Архив состоит из множества записей об успеваемости студентов (агрегирует такие записи), но может быть и пустым.

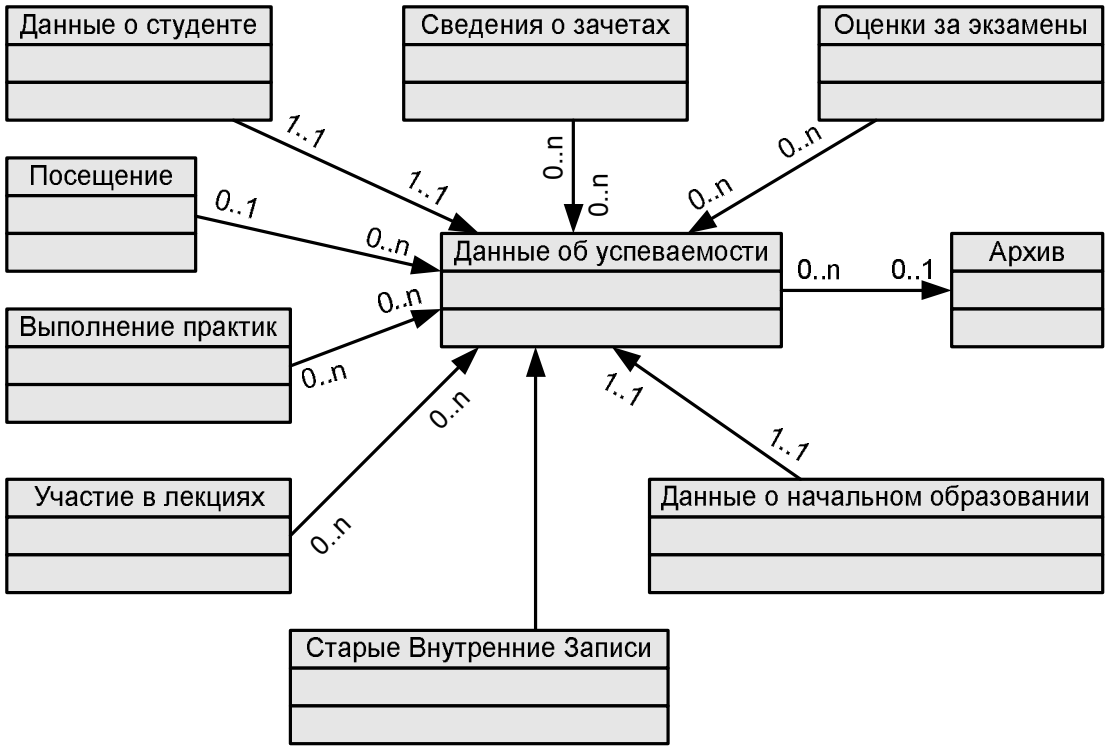

Рисунок 2.36. Концептуальная модель данных

Поскольку студент может учиться уже не первый семестр, появляется дополнительная разновидность (подкласс) записей об успеваемости: старые внутренние записи.

Этот этап завершает процедуры бизнес-моделирования и позволяет представить команде проектировщиков в едином формате ту информацию, которая будет необходима для создания системы.

Разработанные диаграммы являются отправной точкой в процессах проектирования баз данных и приложений системы, обеспечивают согласованность действий бизнес-аналитиков и разработчиков в процессе дальнейшей работы над системой. Эти диаграммы, конечно же, будут претерпевать изменения в процессе последующего проектирования, однако эти изменения будут фиксироваться в формате, уже привычном для всей команды разработчиков, и будут автоматически отражаться в последующих моделях.

Этап 4. *Разработка требований к системе*

На этапе формирования требований, прежде всего, необходимо определить область действия разрабатываемой системы и получить точное представление о желаемых возможностях системы.

Основой разработки требований является модель системных прецедентов, отражающая выполнение конкретных обязанностей исполнителями с использованием ИС.

Источником данных для создания модели системных прецедентов являются разработанные на предыдущем этапе бизнес-модели. Однако при создании модели полезно предварительно составить детальные описания прецедентов, содержащие определения используемых данных и точную последовательность их выполнения. Описание осуществляется в соответствии с принятым в организации шаблоном, который обычно включает следующие разделы:

- заголовок (название прецедента, ответственный за исполнение, дата создания шаблона/внесения изменений);
- краткое описание прецедента;
- ограничения;
- предусловия (необходимое состояние системы или условия, при которых должен выполняться прецедент);
- постусловия (возможные состояния системы после выполнения прецедента);
- предположения;
- основная последовательность действий;
- альтернативные последовательности действий и условия, их инициирующие;
- точки расширения и включения прецедентов.

В процессе создания модели системных прецедентов осуществляется преобразование и перенос компонентов бизнес - моделей на новые диаграммы. Типовые преобразования по технологии Rational Unified Process приведены в табл. 2.7.

Таблица 2.7.

Элементы бизнес-модели	Элементы модели системных прецедентов
Бизнес-прецеденты	Подсистемы
Внутренние исполнители	Исполнители или прецеденты
Процессы, выполняемые исполнителями	Прецеденты

На рис. 2.37 представлена модель системных прецедентов для бизнес-прецедента «Процесс обучения». Исходя из цели создания системы, в модели системных прецедентов отражены только те действия исполнителей, которые связаны с предоставлением доступа и обновлением записей об успеваемости.

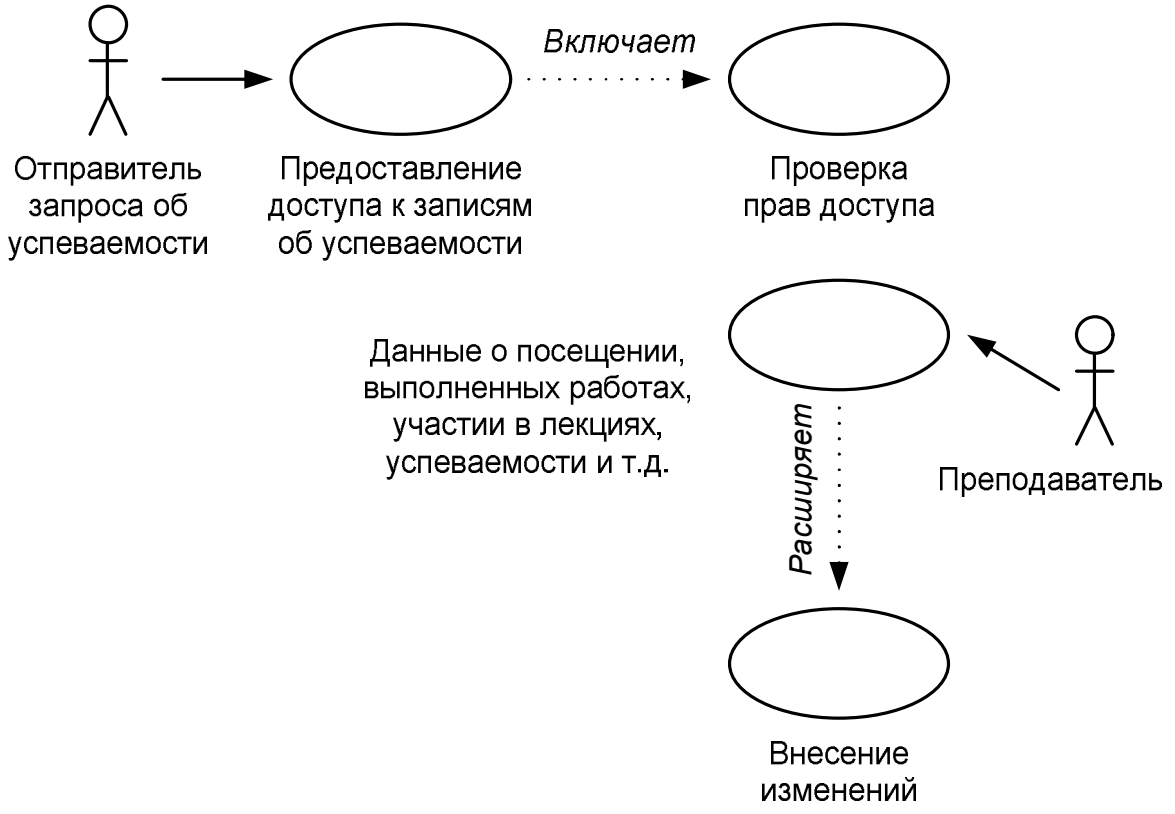

Рисунок 2.37. Модель системных прецедентов

Описываемые моделью функции характерны только для одного вида деятельности – непосредственно процесса обучения, и в основном не используются в других видах деятельности ВУЗа. Это позволяет объединить выделенные функции в некую единую подсистему проектируемой ИС.

Исполнитель «Сотрудники ВУЗа» и выполняемый ими ручной процесс преобразован в системный прецедент «Предоставление доступа к записям об успеваемости».

В модели отражены два специальных типа связи между прецедентами:

- «включает» – один прецедент в процессе своего исполнения обязательно выполняет некий блок действий, составляющих другой прецедент;
- «расширяет» – когда прецеденты подобны по своим действиям, но один несет несколько большую функциональную нагрузку.

Прецедент «Проверка прав доступа» впервые появился на диаграммах и реализуется средствами разрабатываемой ИС. Поэтому для него приходится разрабатывать диаграмму последовательностей, описывающую его исполнение (рис. 2.38). В результате в проектируемой ИС появляются два новых объекта – программный модуль «Менеджер защиты» и информационный блок «Набор прав».

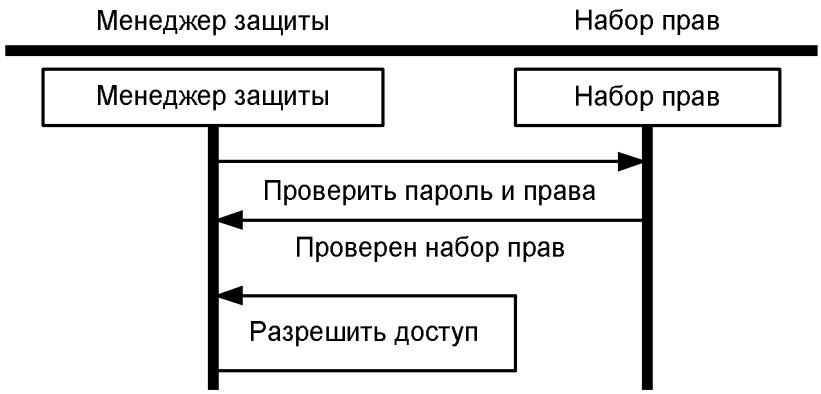

Рисунок 2.38. Диаграмма последовательностей для прецедента «Проверка прав»

Этап 5. *Анализ требований и предварительное проектирование системы*

Основные задачи этапа:

- определить проект системы, который будет отвечать всем бизнес - требованиям;
- разработать общий предварительный проект для всех команд разработчиков (проектировщиков баз данных, разработчиков приложений, системных архитекторов и пр.)

Основным инструментом на данном этапе являются диаграммы классов системы, которые строятся на основе разработанной модели системных прецедентов. Одновременно на этом этапе уточняются диаграммы последовательностей выполнения отдельных прецедентов, что приводит к изменениям в составе объектов и диаграммах классов. Это естественное отражение средствами UML итеративного процесса разработки системы.

Диаграммы классов системы заполняются объектами из модели системных прецедентов. Они содержат описание этих объектов в виде классов и описание взаимодействия между классами. Диаграмма классов, описывающая процедуры защиты доступа к данным, приведена на рис. 2.39.

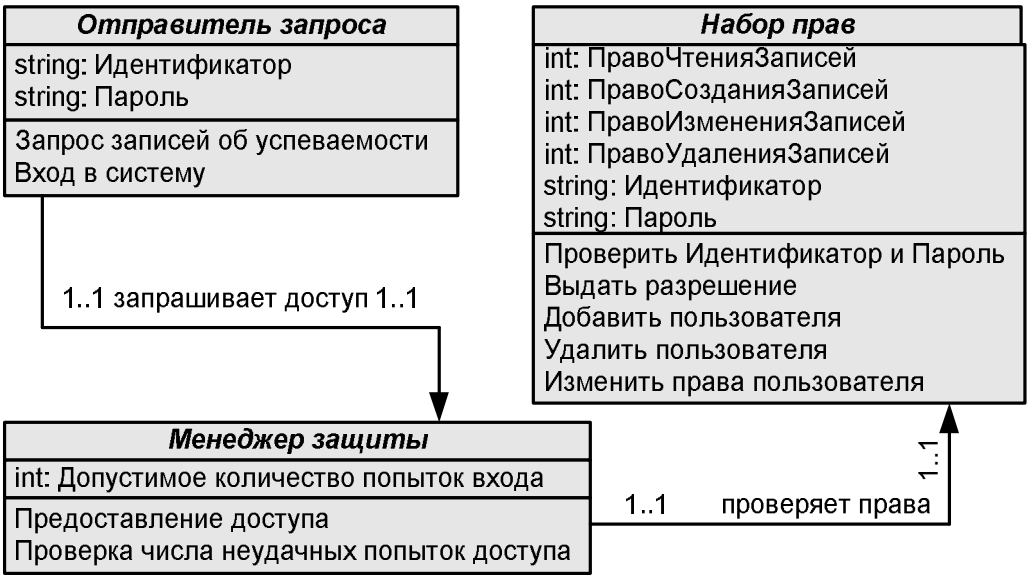

Рисунок 2.39. Диаграмма классов «Защита доступа»

Таким образом, в результате этого этапа проектирования появляется достаточно подробное описание состава и функций проектируемой системы, а также информации, которую необходимо использовать в базе данных и в приложениях.

Поскольку диаграммы классов строятся на основе разработанных ранее бизнес-моделей, появляется уверенность в том, что разрабатываемая система будет действительно удовлетворять исходным требованиям заказчика.

Этап 6. *Разработка модели базы данных и приложения*

На этом этапе осуществляется отображение элементов полученных ранее моделей классов в элементы моделей базы данных и приложений:

- классы отображаются в таблицы;
- атрибуты – в столбцы;
- типы – в типы данных используемой СУБД;
- ассоциации – в связи между таблицами («один – к одному», «многие – ко многим» и «один – ко многим»);
- приложения – в отдельные классы с окончательно определенными и связанными с данными в базе методами и атрибутами.

Для каждого простого класса в модели базы данных формируется отдельная таблица, включающая столбцы, соответствующие атрибутам класса. Отображение классов подтипов в таблицы осуществляется одним из стандартных способов:

- одна таблица отображается на класс;
- одна таблица отображается на суперкласс;
- одна таблица отображается на иерархию.

В первом случае для каждого из классов создается отдельная таблица, между которыми затем устанавливаются необходимые связи. Во втором случае создается таблица для суперкласса, а затем в каждую таблицу подклассов включаются столбцы для каждого из атрибутов суперкласса. В третьем – создается единая таблица, содержащая атрибуты как суперкласса, так и всех подклассов (рис. 2.40). При этом для выделения исходных таблиц подклассов в результирующую таблицу добавляется один или более дополнительных столбцов (на рисунке показан курсивом).

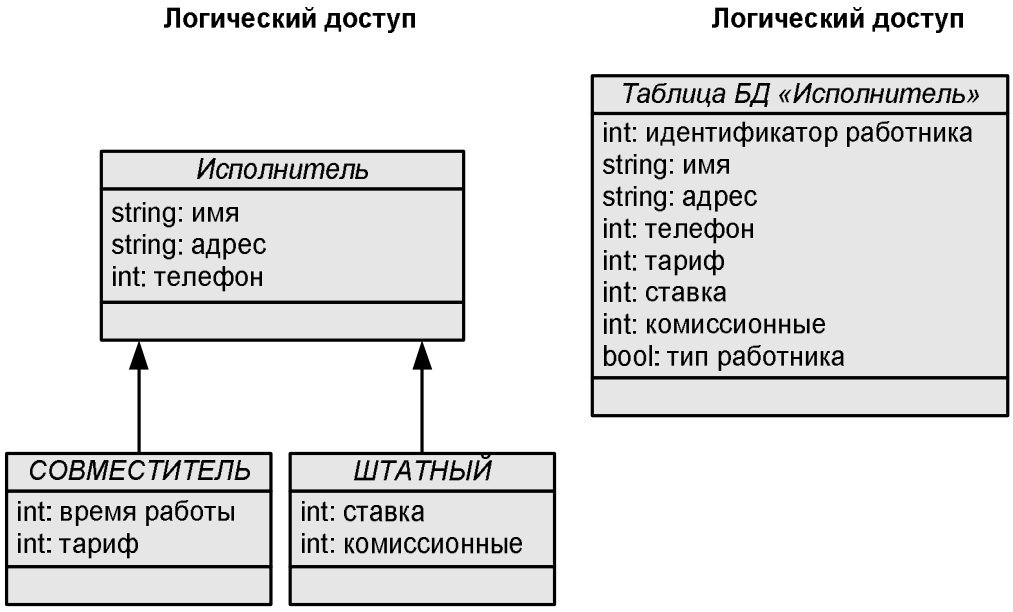

Рисунок 2.40. Преобразование иерархии в таблицу

Разработка проекта базы данных осуществляется с использованием специального UML-профиля (Profile for Database Design), который включает следующие основные компоненты диаграмм:

- таблица – набор записей базы данных по определенному объекту;
- столбец – элемент таблицы, содержащий значения одного из атрибутов таблицы;
- первичный ключ (PK) – атрибут, однозначно идентифицирующий строку таблицы;
- внешний ключ (FK) – один или группа атрибутов одной таблицы, которые могут использоваться как первичный ключ другой таблицы;
- обязательная связь – связь между двумя таблицами, при которой дочерняя таблица существует только вместе с родительской;
- необязательная связь – связь между таблицами, при которой каждая из таблиц может существовать независимо от другой;
- представление – виртуальная таблица, которая обладает всеми свойствами обычной таблицы, но не хранится постоянно в базе данных;
- хранимая процедура – функция обработки данных, выполняемая на сервере;
- домен – множество допустимых значений для столбца таблицы.

На рис. 2.41 представлен фрагмент модели базы данных – две таблицы, соответствующие классам «Студент» и «Данные о студенте». Связь между ними обязательная, поскольку «Данные о студенте» не может существовать без «Студента».

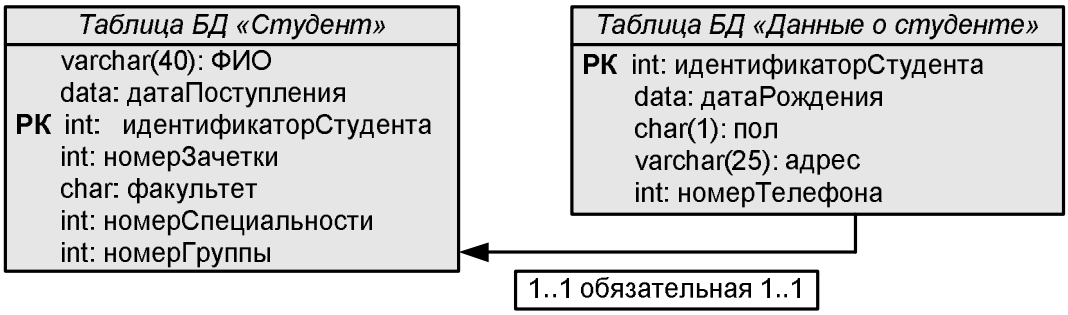

Рисунок 2.41. Фрагмент модели базы данных

Результатом этапа является детальное описание проекта базы данных и приложений системы.

Этап 7. *Проектирование физической реализации системы*

На этом этапе проектирования модели баз данных и приложений дополняются обозначениями их размещения на технических средствах разрабатываемой системы. На рис. 2.42 приведено изображение разделения таблицы «Студент» на три экстента (<<Tablespace>>) в соответствии с первой буквой фамилии студента.

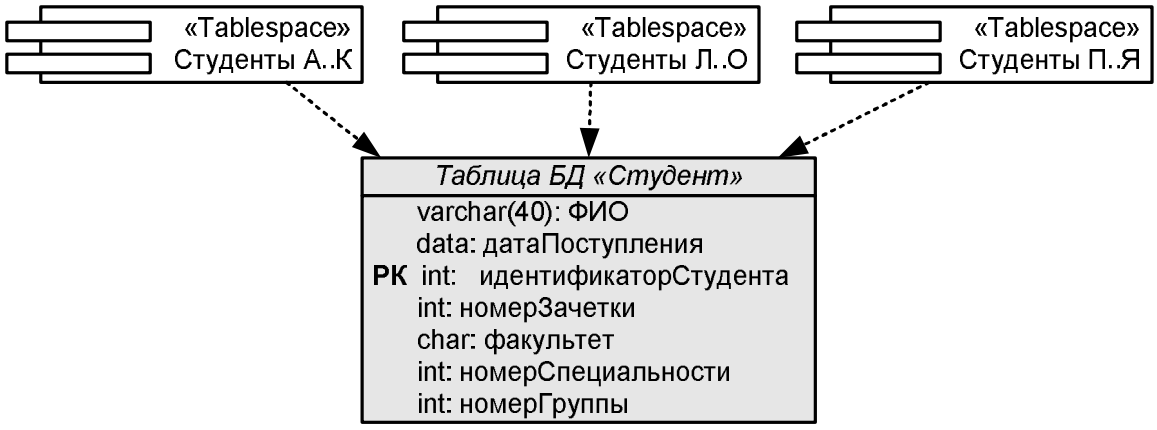

Рисунок 2.42. Экстенты таблицы «Студент»

Основными понятиями UML, которые используются на данном этапе, являются следующие:

– компонент – самостоятельный физический модуль системы;
– зависимость – связь между двумя элементами, при которой изменения в одном элементе вызывают изменения другого элемента;

– устройство – узел, не обрабатывающий данные;

– процессор – узел, выполняющий обработку данных;

– соединение – связь между устройствами и процессорами.

Диаграммы развертывания позволяют отобразить на единой схеме различные компоненты системы (программные и информационные) и их распределение по комплексу технических средств (рис. 2.43).

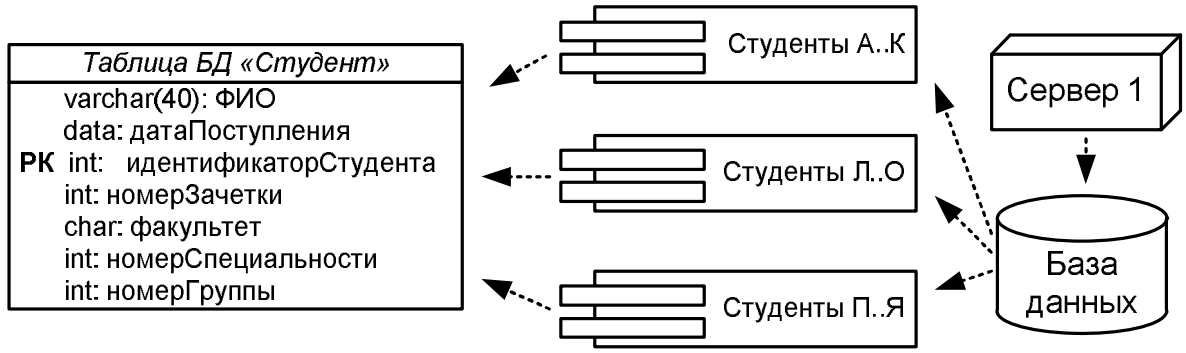

Рисунок 2.43. Фрагмент диаграммы развертывания ИС

В результате получаем готовую ИС, созданную при помощи UML. Визуализированные средства UML модели ИС позволяют наладить плодотворное взаимодействие между заказчиками, пользователями и командой разработчиков. Они обеспечивают ясность представления выбранных архитектурных решений и позволяют понять разрабатываемую систему во всей ее полноте.

Выводы. Таким образом, при проектировании сложной ИС она разделяется на части, и каждая из них затем исследуется и создается отдельно. В настоящее время используются (и рассмотрены в данном разделе) два различных способа такого разбиения ИС на подсистемы: структурное (или функциональное) разбиение и объектная (компонентная) декомпозиция.

С позиций проектирования ИС суть функционального разбиения может быть выражена известной формулой: «Программа = Данные + Алгоритмы». При функциональной декомпозиции программной системы ее структура описывается блок-схемами, узлы которых

представляют собой «обрабатывающие центры» (функции), а связи между узлами описывают движение данных.

При объектном разбиении в системе выделяются «активные сущности» – объекты (или компоненты), которые взаимодействуют друг с другом, обмениваясь сообщениями и выполняя соответствующие функции (методы) объекта.

Если при проектировании ИС разбивается на объекты, то для ее визуального моделирования следует использовать UML. Если в основу проектирования положена функциональная декомпозиция ИС, то UML не нужен и следует использовать рассмотренные ранее структурные нотации.

В то же время, при выборе подхода к разработке ИС следует учитывать, что визуальные модели все более широко используются в существующих технологиях управления проектированием систем, сложность, масштабы и функциональность которых постоянно возрастают. Они хорошо приспособлены для решения таких часто возникающих при создании систем задач как: физическое перераспределение вычислений и данных, обеспечение параллелизма вычислений, репликация БД, обеспечение безопасности доступа к ИС, оптимизация балансировки нагрузки ИС, устойчивость к сбоям и т.п. Визуализированные средствами UML модели ИС позволяют наладить плодотворное взаимодействие между заказчиками, пользователями и командой разработчиков. Они обеспечивают ясность представления выбранных архитектурных решений и позволяют понять разрабатываемую систему во всей ее полноте.

2.2.4. Методология RAD

Методология создания ИС, основанная на использовании средств быстрой разработки приложений, получила в последнее время широкое распространение и приобрела название методологии быстрой разработки приложений (Rapid Application Development, RAD). Данная методология

охватывает все этапы жизненного цикла современных информационных систем.

Методология RAD – это комплекс специальных инструментальных средств, позволяющих оперировать с определенным набором графических объектов, функционально отображающих отдельные информационные компоненты приложений. Основные принципы методологии RAD можно свести к следующим:

- использование итерационная (спиральная) модель разработки;
- полное завершение работ на каждом из этапов жизненного цикла не обязательно;
- обеспечение тесного взаимодействия между заказчиком и будущими пользователями в процессе разработки ИС;
- применение CASE-средств и средств быстрой разработки приложений;
- применение средств управления конфигурацией, облегчающих внесение изменений в проект и сопровождение готовой системы;
- использование прототипов приложений, позволяющих полнее выяснить и реализовать потребности конечного пользователя;
- осуществление тестирования и развитие проекта одновременно с разработкой;
- обеспечение грамотного руководства разработкой системы, четкого планирования и контроль выполнения работ, выполняемых немногочисленной и хорошо управляемой командой профессионалов.

Фазы жизненного цикла в рамках методологии RAD. При использовании методологии быстрой разработки приложений жизненный цикл ИС состоит из четырех фаз:

- анализ и планирование требований;

- процесс проектирования ИС;
- процесс построения ИС;
- фаза внедрения ИС.

Ограничения методологии RAD. Применение методологии RAD наиболее эффективно при создании сравнительно небольших систем, разрабатываемых для конкретного заказчика.

Методология RAD не подходит для создания:

- типовых систем, не являющихся законченным продуктом, а представляющих собой совокупность типовых элементов ИС;
- сложных расчетных программ, операционных систем и программ управления сложными инженерно-техническими объектами, то есть программ, требующих написания большого объема уникального кода;
- для разработки приложений, в которых интерфейс пользователя является вторичным, то есть отсутствует наглядное определение логики работы системы. Примерами таких приложений могут служить приложения реального времени.
- для разработки систем, от которых зависит безопасность людей, например систем управления транспортом или атомными электростанциями. Это обусловлено тем, что итеративный подход, являющийся одной из основ RAD, предполагает, что первые версии системы не будут полностью работоспособными, что в данном случае может привести к серьезнейшим катастрофам.

2.2.5. Разработка интерфейса ИС

Еще один класс задач, решаемых при проектировании информационных систем, относится к созданию удобного и соответствующего целям ИС пользовательского интерфейса. Задача эргономичности интерфейса не формализуется, но, в то же время, она является очень существенной. Поль-

зователи обычно судят о качестве системы в целом, исходя из качества ее интерфейса. Более того, от качества интерфейса зависит эффективность системы. В последние годы появились так называемые средства визуальной разработки приложений, в значительной мере упростившие задачу разработки графического интерфейса пользователя. Их можно условно разделить на два класса.

Специализированные средства ориентированы исключительно на создание приложений для вполне определенной СУБД и не предназначены для разработки обычных приложений, не использующих базы данных. Примером средств такого рода может служить система Power Builder фирмы Sybase.

Универсальные средства могут использоваться как для разработки информационных приложений, взаимодействующих с базами данных, так и для разработки любых других приложений, не использующих базы данных. Из таких средств наибольшей известностью пользуются системы Delphi фирмы Borland, Visual Basic, Visual C фирмы Microsoft.

С выходом платформы Microsoft.NET достоинства и недостатки языков программирования стали сглаживаться, появилась возможность межъязыковой интеграции. Создавать программное обеспечение для NET можно с помощью восьмой версии Delphi.

Подавляющее большинство информационных систем работает в режиме диалога с пользователем. В наиболее общем случае типовые программные компоненты, входящие в состав ИС, реализуют:

- диалоговый ввод-вывод;
- логику диалога;
- прикладную логику обработки данных;
- логику управления данными;
- операции манипулирования файлами и (или) базами данных.

Интерфейсы конечного пользователя – это то, что заказчик критикует в наибольшей степени, в силу того, что именно эти части информационной системы он может более или менее квалифицированно оценить (обычно только их он и видит). Это означает, что интерфейсы являются наиболее часто изменяемым элементом информационной системы именно на этапе реализации.

Часто изменяемый компонент (компоненты) ИС следует изолировать от редко изменяемых компонентов, чтобы одни изменения не влекли за собой другие. Один из приемов подобной изоляции – изоляция запросов к данным от интерфейса следующим образом:

- каждый из запросов кодируется идентификатором или «закрывается» определенной системной функцией;
- разработчик интерфейса не знает о запросе к данным ничего, кроме параметров атрибутов выборки – их типа и, возможно, количества строк в выборке;
- обработка ошибок в запросах данных представляет собой отдельный модуль;
- обработка ошибок в интерпретации результата запроса также представляет собой отдельный модуль.

При обработке результатов запросов данных следует также особое внимание уделить вопросам соответствия типов включающего языка и СУБД, в том числе вопросам точности числовых типов, так как представление их у разных СУБД существенно различается. Кроме того, следует обратить внимание на запросы к данным, которые используют функции, зависящие от операционной системы, например функции работы с байтами и словами значения атрибута.

Если информационная система использует данные из нескольких баз данных под управлением разных СУБД, то неявных преобразований типов лучше избегать.

Следует также установить достаточно жесткие правила для внешнего вида интерфейсов пользователя. Должно создаваться впечатление единого стиля для всех компонентов информационной системы.

ЛИТЕРАТУРА

ОСНОВНАЯ

1. Гайдамакин Н.А. Автоматизированные информационные системы, банки и базы данных. Вводный курс: Учебное пособие. – М.: Гелиос АРБ, 2002. – 368 с.

2. Вендеров А.М. Проектирование программного обеспечения экономических информационных систем: учебник – М.: Финансы и статистика, 2000.

3. Избачков Ю.С., Петров Ю.Н. Информационные системы: Учебник для вузов. 2-е изд. – СПб.: Питер, 2005. – 656 с.

4. Петров В.Н. Информационные системы. – СПб.: Питер, 2002. – 688 с.

5. Проектирование экономических информационных систем: методология и современные технологии: Учебное пособие / В.П. Романов, Н.З. Емельянова, Т.Л. Партыка. – М.: Издательство «Экзамен», 2005. – 256 с. (Серия «Учебник Плехановской академии»).

6. ГОСТ Р ИСО / МЭК ТО 10000-1-99. Информационная технология. Основы и технология функциональных стандартов. Часть 1. Основные положения и основы документирования.

7. ГОСТ Р ИСО / МЭК ТО 10000-2-99. Информационная технология. Основы и таксономия функциональных стандартов. Часть 2. Принципы и таксономия профилей ВОС.

ДОПОЛНИТЕЛЬНАЯ

8. Автоматизированные информационные технологии в экономике. Учебник/ М.И. Семенов, И.Т. Трубилин, В.И. Лейко – М.: Финансы и статистика, 2003.

9. Волков А.А. Тесты TPC// Системы Управления Базами Данных, №2, 1995. С. 70–78.

10. Базы данных: достижения и перспективы на пороге 21-го столетия / Под ред. Ави Зильбершатца, Майкла Стоунбрейкера и Джеффа Ульмана // Системы Управления Базами Данных, №3, 1996. С. 103–117.

11. Орфали Р., Харки Д.., Эдвардс Д. Основы CORBA: Пер. с англ. – М.: МАЛИП, Горячая линия – Телеком, 1999.

12. Дэвитт Т., Грей Д. Параллельные системы баз данных: будущее высокоэффективных систем баз данных // Системы Управления Базами Данных, №2,1995. С. 8–31.

13. Роберт Сигнор, Михаэль О. Стегмаи. Использование ODBC для доступа к базам данных: Пер. с англ. М.: БИНОМ.

14. Меллинг В.П. Корпоративные информационные архитектуры: и все-таки они меняются // Системы Управления Базами Данных, №2, 1995. С. 45–59.

15. Михайлов М. СУБД нового поколения // КомпьютерПресс, №11, 1990. С. 25–30, 79.

16. Системы управления базами данных и знаний: Справ, изд. / Наумов А. Н., Вен-дров А. М., Иванов В. К. и др.; Под ред. А. Н. Наумова. М.: Финансы и статистика, 1991.

17. Шнитман В.З., Кузнецов С.Д. Серверы корпоративных баз данных. Информационно-аналитические материалы Центра Информационных Технологий. http:\\www.citforum.ru.

ГЛАВА 3. ОБУЧАЮЩИЕ И ТЕСТИРУЮЩИЕ СИСТЕМЫ

В настоящее время во многих образовательных организациях возникла проблема систематизации, хранения и предоставления многоуровневого доступа к большому объему накопленных данных, представленных как в традиционной печатной, так и в электронной формах. При этом доля электронных материалов по сравнению с печатными - невелика. Тем не менее, анализ ситуации с электронными ресурсами в системе образования показывает, что в образовательных учреждениях России накоплен значительный опыт по созданию электронных средств учебного назначения, по разработке информационных и образовательных сайтов, по внедрению информационных технологий в учебный процесс. Однако эта деятельность носит разрозненный и фрагментарный характер и требует организации систематической работы по сбору, классификации и размещению в едином информационном пространстве информационных, учебно-методических и других материалов учебного назначения.

В связи с этим создание в регионах и вузах специализированной службы разработки и поддержки ресурсов учебного назначения, в том числе для системы дистанционного обучения является актуальной задачей, решаемой в рамках создаваемой системы ресурсных центров системы образования. Кроме этого, очень важно при создании такой службы учесть региональную специфику развития информационно-сетевой инфраструктуры системы образования. Каждая образовательная и научная структура региона может выступать как в качестве информационного донора, так и в качестве потребителя информации.

В настоящее время в сфере образования России происходит формирование единой информационно-образовательной среды. В Концепции создания и развития системы дистанционного образования в России под информационно-образовательной средой понимается «системно-

организованная совокупность средств передачи данных, информационных ресурсов, протоколов взаимодействия, аппаратно-программного и организационно-методического обеспечения, ориентированная на удовлетворение образовательных потребностей пользователей».

В последнее время в основном используется менее технократичное определение, рассматривающее информационно-образовательную среду с точки зрения двух основных ее функций: информационной и образовательной, т.е. как образовательную среду, базирующуюся на широком использовании информационных технологий. Под информационной образовательной средой учебного заведения понимается сложная система, аккумулирующая интеллектуальные, культурные, программно-методические, организационные и технические ресурсы, обеспечивающие образовательный процесс в целом и процесс обучения в частности.

Одним из главных элементов информационно-образовательной среды являются образовательные ресурсы.

Если рассмотреть пять основных направлений информатизации системы образования, предложенных рабочей группой Минобразования, то в рамках этих направлений можно выделить следующие приоритетные виды деятельности:

1. Содержание образовательных программ, образовательные модели и методики, кадровое обеспечение информатизации образования.
2. Педагогика информационных и коммуникационных технологий (ИКТ). ИКТ в модернизации образования (содержание и методы обучения).
3. Обеспечение развития научно-образовательной инфраструктуры информатизации образования.
4. Индустрия информационных услуг.
5. Информационно-образовательные технологии, технологические стандарты и нормативно-правовое обеспечение системы образования.

6. Обеспечение контроля качества информационно-образовательных продуктов и технологий.

7. Региональная политика в области информатизации образования.

8. Информационное обеспечение российского образования в международном научно-образовательном пространстве.

9. Создание Интегрированной автоматизированной информационной системы сферы образования, включая разработку среднесрочной стратегии развития автоматизированных информационных систем образовательных учреждений.

10. Создание отраслевого депозитария электронных библиотечных ресурсов.

11. Развитие высокопроизводительных вычислений.

12. Обеспечение информационной безопасности единой образовательной информационной среды.

Таким образом, как можно видеть, как минимум пять из них, так или иначе, имеют отношение к созданию, распространению и каталогизации электронных образовательных ресурсов.

Под образовательными ресурсами понимается учебная, методическая, справочная, нормативная, организационная и другая информация, необходимая для эффективной организации прохождения учебного процесса с гарантированным уровнем качества.

Вводится также понятие электронного образовательного ресурса (ЭОР): *Электронный образовательный ресурс* – это любой электронный ресурс, содержащий информацию образовательного характера.

Общепринятой классификации образовательных ресурсов не существует, что создает определенные проблемы при их каталогизации.

3.1. Терминология, принятая в данной области

3.1.1. История развития процесса создания терминологии и основные проблемы

Компьютер с момента своего рождения сразу же стал проблемой образования. Во-первых, он стал объектом изучения. Но уже через короткий отрезок времени в компьютере «обнаружились» возможности, которые можно было использовать и для изучения предметов непосредственно не связанных с компьютерной техникой. Так уже в 1945 году Bush предложил новый подход к организации документов. При хранении информационных данных в машине он предложил устанавливать ассоциативные связи между отдельными документами и их фрагментами. В 1963 г. T. Nelson использовал изобретенный им термин «гипертекст» для обозначения нового понятия – комбинации текста на естественном языке со способностью компьютера осуществлять интерактивный выбор следующей порции информации или динамического воспроизведения нелинейного текста, который не может быть напечатан обычным способом на листе бумаги. Как известно в настоящее время эта технология стала базисом для построения новых классов обучающих систем.

Идеи использования компьютеров в учебном процессе нашли свое выражение в 60-е годы в форме концепции программированного обучения. Начинается создание специализированных пакетов программ, ориентированных на создание и сопровождение прикладных обучающих программ - автоматизированных учебных курсов (АУК). Примером могут служить, PLATO IV, СПОК, Наставник, Садко. Все эти и многие другие АОС были системами селективного типа. Педагогические основы такого рода систем составляли следующие психологические модели: линейная модель подкрепляемого научения американского психолога Б.Ф. Скиннера, модифи-

цированная линейная модель С. Пресси и разветвленная модель Н. Краудера.

70-е годы стали свидетелями рождения нового поколения компьютерных средств поддержки процесса обучения – интеллектуальных (продуцирующих, экспертных) обучающих систем (ИОС). С появления первой интеллектуальной обучающей системы прошло уже ровно более 30 лет. В 1970 году J.R. Carbonell представил свою систему SCHOLAR, на примере которой была продемонстрирована эффективность использования методов искусственного интеллекта в такой области как обучение. Если началом исследований в области искусство интеллекта принято считать 1955 год, когда Ньюэлл и Саймон приступили к исследованиям «сложных процессов обработки информации» в Технологическом институте Карнеги, то 1970 год можно смело считать годом рождения нового научного направления, появившегося на стыке программированного обучения и искусственного интеллекта. Неудовлетворенность практиков возможностями программированного обучения стала причиной появления ИОС, где обучающие воздействия выбираются не педагогом, а «зашиты» в соответствующую систему и выбираются или генерируются в зависимости от целей обучения и с учетом текущего состояния знаний обучаемого. Для этого в обучающей системе представлены знания о том, чему обучать, как обучать и знания о самом обучаемом плюс имеются некоторые умения, позволяющие вести диалог с обучаемым. Такие системы позволяют адаптивно выдавать учебные воздействия, сопровождать решение задач, производить глубокую диагностику знаний обучаемого, что подразумевает реализацию еще целого ряда «интеллектуальных» возможностей.

В конце 80-х гг. стало ясно, что интеллектуализация обучающих систем, в первую очередь, связана с практическим использованием при их разработке и реализации методов и средств, созданных в рамках исследований по экспертным системам. Это, в свою очередь, вызвало к жизни се-

рьезные исследования по моделям объяснения в АОС, с одной стороны, и интеллектуальным технологиям формирования моделей предметной области, стратегий обучения и оценки знаний обучаемых на основе более сложных моделей самих обучаемых, с другой стороны. В этот же период были предложены классификации АОС, включая интеллектуальные АОС, и сделаны первые шаги в направлении разработки технологии создания обучающих систем различных классов.

С начала 90-х годов тенденцией является объединение с общих позиций компьютерной технологии обучения традиционных информационных, контролирующих, игровых и обучающих систем с диалоговыми системами для автоматизированного решения задач, средствами искусственного интеллекта, экспертными системами и технологии гипермедиа. Изменение технической базы, появление новых технологий программирования и завершение «жизненного цикла» ранних АОС послужили толчком создания новых программ, базирующихся на старых психолого-педагогических моделях. Всплеск появления таких систем наблюдается последние 10–15 лет.

Попытки классификации компьютерных образовательных технологий предпринимаются регулярно, но критерии классификации и терминология отличаются своим непостоянством. Кроме того, появляются новые системы, которые не «вписываются» в предлагаемые схемы. В связи можно сформулировать следующие вопросы:

- можно ли в настоящее время построить единую классификацию;
- имеет ли значение при проведении классификации, с чьей позиции она производится (студента, педагога, менеджера, разработчика);
- возможна ли выработка единого терминологического словаря рассматриваемой области.

3.1.2. Рекомендованные основные понятия

Согласно Межгосударственному стандарту ГОСТ 7.83–2001 следует различать:

- *электронный документ*: документ на машиночитаемом носителе, для использования которого необходимы средства вычислительной техники;
- *электронное издание*: электронный документ (или группа электронных документов), прошедший редакционно-издательскую обработку, предназначенный для распространения в неизменном виде, имеющий выходные сведения.

Можно предложить следующую классификацию электронных образовательных ресурсов.

Электронный образовательный ресурс, может иметь следующие виды:

- электронные данные;
- электронные программы или их сочетание в одном ресурсе.

По знаковой природе информации:

- электронные данные делятся на текстовые, числовые, звуковые, графические, шрифтовые и демонстрационные;
- электронные программы делятся на системные, прикладные и сервисные; сочетание электронных данных и программ – на интерактивные мультимедиа и онлайновые службы.

По наличию печатного эквивалента:

- электронный аналог печатного издания: Электронное издание, в основном воспроизводящее соответствующее печатное издание (расположение текста на страницах, иллюстрации, ссылки, примечания и т.п.);

– самостоятельное электронное издание: Электронное издание, не имеющее печатных аналогов.

По целевому назначению учебно-методические ресурсы можно разделить:

– *официальные электронные ресурсы*: электронные ресурсы, публикуемые от имени государственных органов, учреждений, ведомств или общественных организаций, содержащие материалы нормативного или директивного характера в области образования;
– *научные электронные ресурсы*: электронные ресурсы, содержащие сведения о теоретических и (или) экспериментальных исследованиях, конкретного вуза или межвузовских научных объединений;
– *научно-популярные электронные ресурсы*: электронные ресурсы, содержащие сведения о теоретических и (или) экспериментальных исследованиях в области науки, культуры и техники, изложенные в форме, доступной читателю-неспециалисту;
– *учебные электронные ресурсы*: электронные ресурсы, содержащие систематизированные сведения научного или прикладного характера, изложенные в форме, удобной для изучения и преподавания, и рассчитанные на учащихся разного возраста и степени обучения;
– *учебно-методические электронные ресурсы*: электронные ресурсы, содержащие сведения методического характера в помощь изучению конкретного курса или дисциплины;
– *справочные электронные ресурсы*: электронные ресурсы, содержащее краткие сведения научного и прикладного характера, расположенные в порядке, удобном для их быстрого отыскания, не предназначенное для сплошного чтения.

По технологии распространения:

- *локальный электронный ресурс*: электронный ресурс, предназначенный для локального использования и выпускающийся в виде определенного количества идентичных экземпляров (тиража) на переносимых машиночитаемых носителях;
- *сетевой электронный ресурс*: электронный ресурс, доступный потенциально неограниченному кругу пользователей через телекоммуникационные сети;
- *электронный ресурс комбинированного распространения*: электронный ресурс, который может использоваться как в качестве локального, так и в качестве сетевого.

По характеру взаимодействия пользователя и электронного ресурса:

- *детерминированный электронный ресурс*: электронный ресурс, параметры, содержание и способ взаимодействия с которым определены создателем и не могут быть изменяемы пользователем;
- *недетерминированный (интерактивный) электронный ресурс*: электронный ресурс, параметры, содержание и способ взаимодействия с которым прямо или косвенно устанавливаются пользователем в соответствии с его интересами, целями, уровнем подготовки и т.п. на основе информации и с помощью алгоритмов, определенных создателем.

По периодичности:

- *непериодический электронный ресурс*: электронный ресурс, выходящий однократно, не имеющий продолжения;
- *сериальный электронный ресурс*: электронный ресурс, выходящий в течение времени, продолжительность которого заранее не установлена, как правило, нумерованными и (или) датированными выпусками (томами), имеющими одинаковое заглавие;

– *периодический электронный ресурс*: сериальный электронный ресурс, выходящее через определенные промежутки времени, постоянным для каждого года числом номеров (выпусков), неповторяющимися по содержанию, однотипно оформленными нумерованными и (или) датированными выпусками, имеющими одинаковое заглавие;
– *продолжающийся электронный ресурс*: сериальный электронный ресурс, выходящий через неопределенные промежутки времени, по мере накопления материала, не повторяющимися по содержанию, однотипнооформленными нумерованными и (или) датированными выпусками, имеющими общее заглавие;
– *обновляемый электронный ресурс*: электронный ресурс, выходящий через определенные или неопределенные промежутки времени в виде нумерованных или датированных выпусков, имеющих одинаковое заглавие и частично повторяющееся содержание; каждый следующий выпуск содержит в себе всю оставшуюся актуальную информацию и полностью заменяет предыдущий.

По структуре:

– *однотомный электронный ресурс*: непериодический электронный ресурс, выпущенный на одном машиночитаемом носителе;
– *многотомный электронный ресурс*: непериодический электронный ресурс, состоящий из двух или более пронумерованных частей, каждая из которых представлена на самостоятельном машиночитаемом носителе, представляющий собой единое целое по содержанию и оформлению;
– *электронная серия*: сериальный электронный ресурс, включающий совокупность томов, объединенных общностью замысла, тематики, целевым назначением, выходящих в однотипном оформлении.

Сочетания электронных данных и электронных программ по характеру подразделяются на интерактивные мультимедиа и онлайновые службы.

Интерактивная мультимедиа-система, позволяющая одновременное использование различных средств отображения и передачи информации.

Онлайновая служба – системно-ориентированная деятельность, которая поддерживает доступ к информации и ее использование по Интернет.

По жанру следует различать:

- *образовательные порталы*;
- *сайты вузов*, факультетов, кафедр, лабораторий, сайты НИИ;
- *научно-популярные журналы*;
- *сайты научно-образовательных проектов*;
- *электронные пособия, учебники, хрестоматии*;
- *электронные курсы* (образовательные объекты);
- *архивы образовательных ресурсов*;
- *базы данных* (полнотекстовые, библиографические, справочные, адресные и т.д.);
- *лаборатории и вычислительный и имитационный эксперимент*, виртуальные тренажеры;
- *дистанционное управление экспериментальными комплексами*;
- *электронные библиотеки*;
- *электронное представление ресурсов* обычных библиотек;
- *персональные страницы преподавателей вузов*, организаторов образования;
- *отдельные статьи* или страницы;
- *информационно-поисковые системы*.

При выполнении проекта по созданию Федерального ресурсного центра по Сибирскому федеральному округу (в рамках ФЦП РЕОИС) группой разработчиков из Томского государственного университета по итогам проведения работ по подготовке, систематизации и классификации различных электронных ресурсов для дошкольного и всех уровней школьного образования была предложена следующая классификация образовательных ресурсов по функциональному признаку, определяющему их значение и место в учебном процессе:

- *программно-методические электронные ресурсы* (учебные планы образовательных учреждений всех уровней, рабочие программы учебных дисциплин в соответствии с учебными планами);
- *учебно-методические электронные ресурсы* (методические указания, методические пособия, методические рекомендации для изучения отдельного курса, руководства по выполнению проектных работ, тематические планы проведения отдельных уроков, изучения отдельных тем, сценарии организации образовательных мероприятий);
- *обучающие электронные ресурсы* (сетевые учебники и учебные пособия, мультимедийные учебники, электронные текстовые учебники, электронные учебные пособия);
- *вспомогательные электронные ресурсы* (сборники документов и материалов, хрестоматии, книги для чтения, энциклопедии, справочники, аннотированные указатели научной и учебной литературы, научные публикации педагогов, материалы конференций, сценарии развлекательных и воспитательных мероприятий);
- *контролирующие электронные ресурсы* (тестирующие программы, банки контрольных вопросов и заданий по учебным дисциплинам, банки тем рефератов, проектных работ).

Дополнительно были выделены еще две группы электронных ресурсов:

- *электронные ресурсы, созданные детьми* (оцифрованные фотографии детских рисунков и поделок, Интернет-проекты и компьютерные программы, созданные школьниками);
- *информационные электронные ресурсы* (общие информативные материалы об образовательных учреждениях всех уровней, информация об образовательных проектах, реализуемых в регионе).

По характеру информации электронного образовательного ресурса там же предложено использовать ГОСТ 7.60–90 «Издания. Основные виды. Термины и определения»:

- монография;
- автореферат диссертации;
- препринт;
- тезисы докладов (сообщений) научной конференции (семинара и т.д.);
- материалы конференции (симпозиума и т.д.);
- сборник научных трудов;
- уставное издание;
- инструкция;
- стандарт;
- прейскурант;
- пособие;
- наглядное пособие;
- практическое пособие;
- практическое руководство;
- учебник;
- учебное пособие;

- хрестоматия;
- учебное наглядное пособие;
- учебно-методическое пособие;
- учебная программа;
- практикум;
- словарь;
- энциклопедия;
- энциклопедический словарь;
- языковый словарь;
- толковый словарь;
- терминологический словарь;
- разговорник;
- справочник;
- биографический справочник;
- библиографический справочник;
- путеводитель;
- проспект;
- каталог;
- издательский каталог;
- каталог выставки;
- альбом;
- атлас;
- карта;
- документально-художественное издание;
- научно-художественное издание;
- альманах;
- антология;
- газета;

– журнал;
– бюллетень;
– календарь;
– реферативный сборник;
– реферат.

Очевидно, классификация по функциональному признаку, определяющему их значение и место в учебном процессе, является оптимальной с позиций структурирования целей и задач разработки и поддержки ресурсов учебного назначения.

3.2. Характеристики электронного издания

Реформа современного образования может состояться лишь при условии создания таких компьютерных пакетов (электронных учебников, пособий, тренажеров, тестеров и проч.), наличие которых обеспечит одну и ту же компьютерную среду в специализированной аудитории на практических занятиях, в компьютерном классе учебного заведения или общежитии, оборудованном для самостоятельной работы учащихся, а также дома на персональном компьютере.

Основываясь на официальных определениях электронного издания (ЭИ), учебного электронного издания (УЭИ) и электронного учебника (ЭУ), необходимо расширить и конкретизировать понятие ЭУ.

3.2.1. Электронный учебник – новый жанр учебной литературы

Электронный учебник (даже самый лучший) не может и не должен заменять книгу. Так же как экранизация литературного произведения принадлежит к иному жанру, так и электронный учебник принадлежит к совершенно новому жанру произведений учебного назначения. И так же как просмотр фильма не заменяет чтения книги, по которой он был поставлен, так и наличие электронного учебника не только не должно заменять чте-

ния и изучения обычного учебника (во всех случаях мы подразумеваем лучшие образцы любого жанра), а напротив, побуждать учащегося взяться за книгу.

Именно поэтому для создания электронного учебника недостаточно взять хороший учебник, снабдить его навигацией (создать гипертексты) и богатым иллюстративным материалом (включая мультимедийные средства) и воплотить на экране компьютера. Электронный учебник не должен превращаться ни в текст с картинками, ни в справочник, так как его функция принципиально иная.

Электронный учебник должен максимально облегчить понимание и запоминание (причем активное, а не пассивное) наиболее существенных понятий, утверждений и примеров, вовлекая в процесс обучения иные, нежели обычный учебник, возможности человеческого мозга, в частности, слуховую и эмоциональную память, а также используя компьютерные объяснения.

Текстовая составляющая должна быть ограничена – ведь остаются обычный учебник, бумага и ручка для углубленного изучения уже освоенного на компьютере материала.

3.2.2. Некоторые принципы, которыми следует руководствоваться при создании электронного учебника

1. Принцип квантования: разбиение материала на разделы, состоящие из модулей, минимальных по объему, но замкнутых по содержанию.

2. Принцип полноты: каждый модуль должен иметь следующие компоненты (рис. 3.1):

- теоретическое ядро;
- контрольные вопросы по теории;
- примеры;
- задачи и упражнения для самостоятельного решения;

– контрольные вопросы по всему модулю с ответами;
– контрольная работа;
– контекстная справка (Help);
– исторический комментарий.

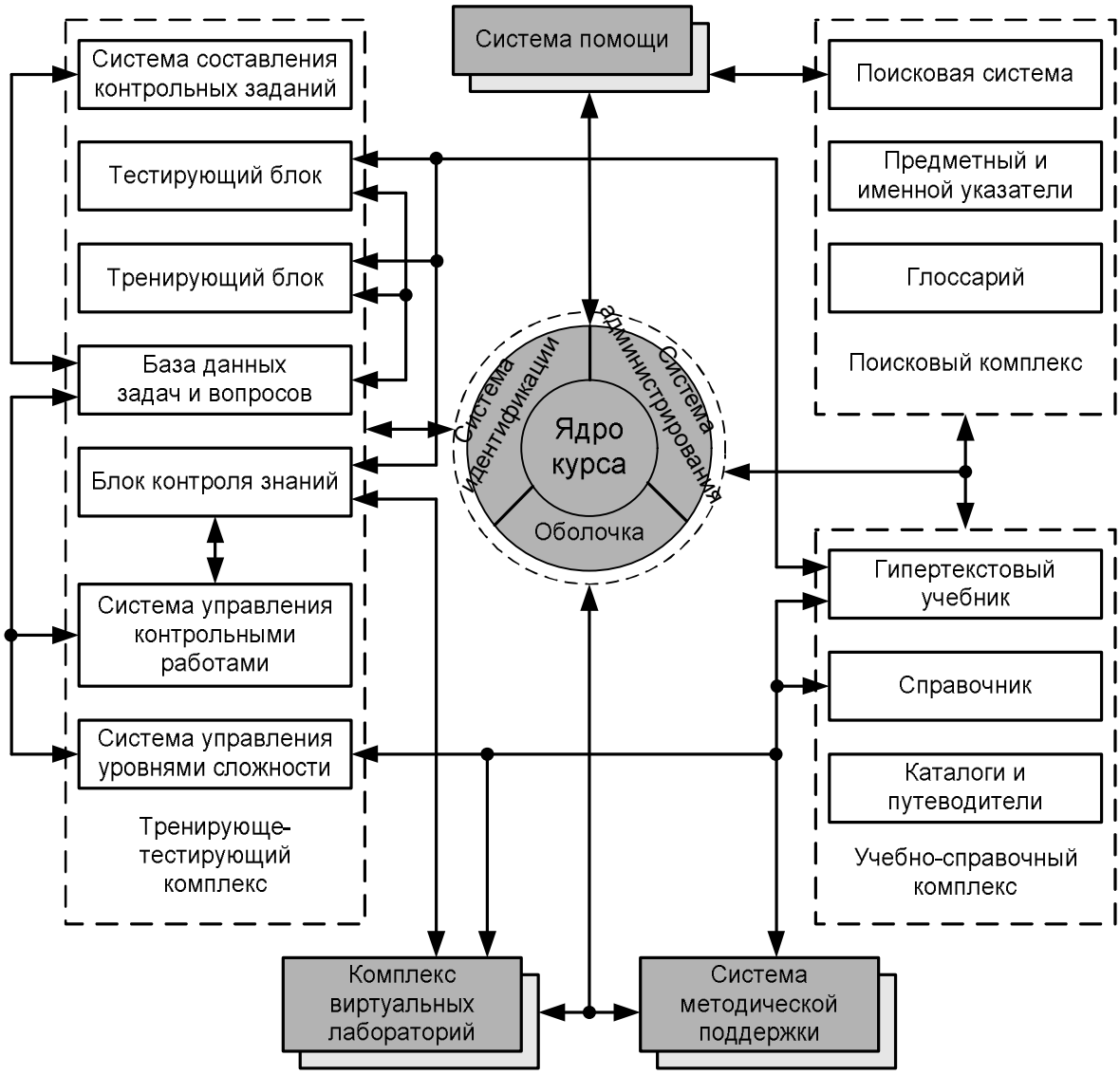

Рисунок 3.1. Графическое представление принципа полноты курса

3. Принцип наглядности: каждый модуль должен состоять из коллекции кадров с минимумом текста и визуализацией, облегчающей понимание и запоминание новых понятий, утверждений и методов.

4. Принцип ветвления: каждый модуль должен быть связан гипертекстными ссылками с другими модулями так, чтобы у пользователя был

выбор перехода в любой другой модуль. Принцип ветвления не исключает, а даже предполагает наличие рекомендуемых переходов, реализующих последовательное изучение предмета.

5. Принцип регулирования: учащийся самостоятельно управляет сменой кадров, имеет возможность вызвать на экран любое количество примеров (понятие «пример» имеет широкий смысл: это и примеры, иллюстрирующие изучаемые понятия и утверждения, и примеры решения конкретных задач, а также контрпримеры), решить необходимое ему количество задач, задаваемого им самим или определяемого преподавателем уровня сложности, а также проверить себя, ответив на контрольные вопросы и выполнив контрольную работу, заданного уровня сложности.

6. Принцип адаптивности: электронный учебник должен допускать адаптацию к нуждам конкретного пользователя в процессе учебы, позволять варьировать глубину и сложность изучаемого материала и его прикладную направленность в зависимости от будущей специальности учащегося, применительно к нуждам пользователя генерировать дополнительный иллюстративный материал, предоставлять графические и геометрические интерпретации изучаемых понятий и полученных учащимся решений задач.

7. Принцип компьютерной поддержки: в любой момент работы учащийся может получить компьютерную поддержку, освобождающую его от рутинной работы и позволяющую сосредоточиться на сути изучаемого в данный момент материала, рассмотреть большее количество примеров и решить больше задач. Причем компьютер не только выполняет громоздкие преобразования, разнообразные вычисления и графические построения, но и совершает математические операции любого уровня сложности, если они уже изучены ранее, а также проверяет полученные результаты на любом этапе, а не только на уровне ответа.

8. Принцип собираемости: электронный учебник (и другие учебные пакеты) должны быть выполнены в форматах, позволяющих компоновать их в единые электронные комплексы, расширять и дополнять их новыми разделами и темами, а также формировать электронные библиотеки по отдельным дисциплинам (например, для кафедральных компьютерных классов) или личные электронные библиотеки студента (в соответствии со специальностью и курсом, на котором он учится), преподавателя или исследователя.

3.2.3. Необходим ли электронный учебник?

Электронный учебник необходим для самостоятельной работы учащихся при очном и, особенно, дистанционном обучении потому, что он:

- облегчает понимание изучаемого материала за счет иных, нежели в печатной учебной литературе, способов подачи материала: индуктивный подход, воздействие на слуховую и эмоциональную память и т.п.;
- допускает адаптацию в соответствии с потребностями учащегося, уровнем его подготовки, интеллектуальными возможностями и амбициями;
- освобождает от громоздких вычислений и преобразований, позволяя сосредоточиться на сути предмета, рассмотреть большее количество примеров и решить больше задач;
- предоставляет широчайшие возможности для самопроверки на всех этапах работы;
- дает возможность красиво и аккуратно оформить работу и сдать ее преподавателю в виде файла или распечатки;
- выполняет роль бесконечно терпеливого наставника, предоставляя практически неограниченное количество разъяснений, повторений, подсказок и проч.

Учебник необходим студенту, поскольку без него он не может получить прочные и всесторонние знания и умения по данному предмету.

Электронный учебник полезен на практических занятиях в специализированных аудиториях потому, что он:

- позволяет использовать компьютерную поддержку для решения большего количества задач, освобождает время для анализа полученных решений и их графической интерпретации;
- позволяет преподавателю проводить занятие в форме самостоятельной работы за компьютерами, оставляя за собой роль руководителя и консультанта;
- позволяет преподавателю с помощью компьютера быстро и эффективно контролировать знания учащихся, задавать содержание и уровень сложности контрольного мероприятия.

Электронный учебник удобен для преподавателя потому, что он:

- позволяет выносить на лекции и практические занятия материл по собственному усмотрению, возможно, меньший по объему, но наиболее существенный по содержанию, оставляя для самостоятельной работы с ЭУ то, что оказалось вне рамок аудиторных занятий;
- освобождает от утомительной проверки домашних заданий, типовых расчетов и контрольных работ, передоверяя эту работу компьютеру;
- позволяет оптимизировать соотношение количества и содержания примеров и задач, рассматриваемых в аудитории и задаваемых на дом;
- позволяет индивидуализировать работу со студентами, особенно в части, касающейся домашних заданий и контрольных мероприятий.

3.2.4. Методическое обеспечение электронного учебника

Если создание электронного учебника не будет сопровождаться разработкой надлежащих методических материалов, затраченные силы и средства пропадут даром, поскольку тогда электронный учебник не будет воспринят системой образования. Поэтому методическое обеспечение ЭИ имеет принципиальное значение для успеха проекта в целом. Исходя из этого, мы уделяем данному вопросу особое место.

Реформа образования требует создания таких УЭИ, наличие которых обеспечит одну и ту же компьютерную среду для учащихся и преподавателей, в аудитории и дома. Здесь уместно провести параллель с реформой европейского образования, связанной с изобретением книгопечатания (Гутенберг, 1440 г.)

Средневековые школяры полностью зависели от своего наставника, ибо только он владел информацией. Изобретение Гутенбергом книгопечатания сделало источник информации (книгу) одинаково доступным для всех, что принципиально изменило систему образования. Книга, перо и бумага – всем этим стал владеть и преподаватель, и учащийся, причем и в аудитории, и дома.

Аналогично, для успешной реформы современного образования необходимо сделать новые источники информации (в частности, УЭИ) одинаково доступными для всех. Однако в данном случае именно преподаватели зачастую оказываются в худшем положении по сравнению со студентами, так как они по ряду причин объективного и субъективного характера меньше привыкли к работе с компьютером и меньше готовы к восприятию новых технологий в образовании.

Очевидно, что с появлением и совершенствованием различных УЭИ должны принципиально измениться учебные программы и планы лекций и практических занятий, а также роль преподавателя в учебном процессе.

3.2.5. Роль методического обеспечения

Важно понять, что если ЭУ и ЭУП будут разработаны в соответствии с принципами, изложенными выше, то можно будет считать компьютеризацию математического образования состоявшейся. По разным причинам, и не только материального характера, никогда не будет так, чтобы компьютеры были в каждом доме, в каждой аудитории и в каждой комнате общежития. Это не только невозможно, но и, как будет показано ниже, не нужно.

Однако даже самые лучшие электронные средства обучения осядут мертвым грузом на компьютерах, если их использование не будет методически обеспечено, если не будет создано компьютерное учебно-информационное пространство, единое для преподавателей и учащихся.

Успешная компьютеризация образования зависит не от количества компьютеров, а от качества средств обучения и методического обеспечения их использования (здесь уместно вспомнить термин «внедрение»).

На наш взгляд, отсутствие полного комплекса методических материалов, а также удобных и эффективных форм повышения квалификации, оперативной и полной информации о появлении и содержании новых компьютерных учебных пакетов, вынуждают преподавателя не только не использовать в своей профессиональной деятельности достижений компьютеризации, но иногда даже запрещать студентам использовать компьютер при выполнении домашних заданий и типовых расчетов.

Сейчас уже трудно убедить студентов в том, что они не только должны овладеть техникой вычисления производных, интегралов и т.п., но и в дальнейшем, при изучении других разделов математики должны решать вручную от начала до конца любую задачу, не имея времени сосредоточиться на ее сути и не понимая, что же именно они изучают в данный момент.

В то же время многие принципиальные вопросы остаются неисследованными из-за недостатка времени у преподавателя в аудитории и у студентов дома. Например, при решении дифференциальных уравнений после вычислений (иногда довольно громоздких) всех интегралов студент совершенно не представляет, что ему делать с найденным решением (построить график, исследовать поведение при t, рассмотреть вопросы устойчивости, ...). Список таких примеров можно продолжить. Кроме того, многие важные разделы современной математики (качественная теория дифференциальных уравнений, элементы функционального анализа, случайные процессы, прикладная математическая статистика, теория принятия решений и т.д.) не изучаются вовсе или изучаются «галопом по Европам», часто только на лекциях без поддержки на практических занятиях и без домашних заданий, и следовательно, быстро стираются из памяти учащегося и не могут быть использованы при изучении других дисциплин, как естественнонаучных и общетехнических, так и профилирующих, не говоря уже о профессиональной деятельности будущего выпускника (ср. с «выживаемостью» школьных знаний). В результате недостаточности и короткой «выживаемости» математических знаний программы специальных дисциплин пестрят доморощенными «методами» решения стандартных математических задач.

Все сказанное вовсе не означает, что преподавателей надо немедленно усадить за компьютеры, а занятия перенести в компьютерные классы. Это не только невозможно, но и вредно (хотя такие попытки систематически предпринимаются).

3.2.6. Требования к современному методическому обеспечению

Прежде чем перейти к описанию содержательной части методического обеспечения математического образования, попытаемся сформулировать некоторые положения, без выполнения которых принципиально

изменить содержание и форму математического образования и превратить его в образование 21-го (а не 19-го!) века, на наш взгляд, невозможно:

1. Нельзя проводить занятия в компьютерном классе в течение всего семестра, но студенты всех групп должны иметь равные возможности получить 3–4 занятия в компьютерном классе в соответствии с сеткой расписания.

2. Каждое занятие в соответствии со стандартными программами должно быть оснащено методической разработкой, не зависящей от того проходит ли занятие в компьютерном классе или в обычной аудитории (изменится лишь соотношение вопросов и задач, рассмотренных в аудитории, и заданных на дом). *Это возможно, если учесть, что компьютерная среда в компьютерном классе и на домашнем компьютере одна и та же*.

3. Преподаватель за компьютер не садится – он ведет занятие по математике, а компьютеры служат лишь подспорьем, позволяющем сэкономить время и сделать работу более эффективной: решить большее количество задач (и уменьшить домашнее задание), проанализировать результаты, воспользоваться графическими возможностями компьютера.

4. При чтении лекций и проведении занятий в обычной аудитории преподаватель учитывает наличие у всех студентов электронного учебника и других компьютерных пособий (на домашнем компьютере или в специальных аудиториях, оборудованных для самостоятельной работы студентов) и, следовательно, имеет возможность ограничиться наиболее существенными вопросами, а остальное передать студентам для самостоятельного изучения (для обеспечения полноценной самостоятельной работы студентов в компьютерных классах, может быть, целесообразно вернуть в расписание т.н. «День самостоятельных занятий» или в некоторые дни проводить занятия лишь до обеда.)

5. В компьютерных классах очень удобно проводить контрольные работы. Учитывая экономию времени, которое студенты тратят на реше-

ние задач с помощью компьютера, можно контрольную работу провести за половину занятия, разделив группу пополам и проводя параллельно занятие (с одной частью группы) и контрольную работу (с другой частью), причем компьютер выдает результаты контрольной немедленно. Очень важно, что преподаватель сам вызывает нужную ему контрольную работу в необходимом количестве вариантов и выбирает уровень ее сложности (группу также можно разделить по уровню подготовки).

6. Компьютерная поддержка позволяет индивидуализировать работу со студентами, особенно в части, касающейся домашних заданий и контрольных мероприятий, таким образом, чтобы каждый студент ощущал, что задания ему по силам и он продвигается от успеха к успеху. Это стимулирует интерес к предмету и делает учебу осмысленной и эффективной. Нравственное и воспитательное значение индивидуализации заданий трудно переоценить.

3.2.7. Содержание методического комплекса

Самые скромные требования к содержательной части методического обеспечения преподавания математики предполагают наличие основных элементов:

1. Новые планы лекций и практических занятий, разработанные с учетом компьютерной поддержки.

2. Методические пособия (печатные и электронные), содержащие подробные рекомендации по каждому занятию.

3. Подробная информация о наличии, содержании и возможностях компьютерных пакетов учебного назначения вместе с методическими рекомендациями по их использованию в аудитории, при выдаче домашних заданий и проведении контрольных мероприятий.

3.3. Некоторые вопросы стандартизации, оценки качества и сертификации учебных электронных ресурсов

Вопрос стандартизации форматов и принципов разработки образовательных ресурсов является чрезвычайно важным для сферы образования России. В большинстве вузов страны разрабатываются электронные пособия и курсы и оболочки для их выполнения. Спецификации для этих разработок заново создаются в каждом университете самостоятельно. Результатом этого является создание большого числа единиц учебного материала и программных систем, разработанных на основе различных и несовместимых между собой стандартов.

Конечная цель стандартизации информационно-образовательного пространства – достижение мобильности, интероперабельности ресурсов, стабильности и эффективности учебного процесса. Это также должно быть необходимым условием реализации учебных программ дистанционного и электронного образования.

В настоящее различными международными ассоциациями и консорциумами разрабатываются различные технологические средства и стандарты в области дистанционного образования, описывающие достаточно большое количество образовательных элементов. В связи с этим происходит изменение принципов разработки учебных материалов. Современным образовательным средам, характеризующимся указанными изменениями, свойственны высокий уровень адаптивности и интерактивности с обучаемым. Образовательные системы, создаваемые на основе международных спецификаций, обладают свойствами масштабируемости, мобильности, интероперабельности и дружественности к пользователю.

Одной из интегрированных форм учебных материалов в традиционных формах обучения является учебно-методический комплекс (УМК), объединяющий большинство из названных материалов. При ДО аналогом УМК становится электронный учебник (ЭУ). Однако степень интеграции в

ЭУ может быть различной, в связи с чем обычно используется классификация ЭУ на несколько уровней (классов). Одна из так классификаций введена в международном стандарте AECMA 1000D, посвященном разработке интерактивных электронных технических руководств (ИЭТР) для авиационных отраслей промышленности.

В соответствии с этой классификацией учебные материалы класса 0 относятся к обычным документам, переведенным в электронный вид и предназначенным для архивации. Класс 1 относится к документам, части которого индексированы и доступны по ссылкам из оглавления. Документы класса 2 – файлы в коде ASCII, внутри которых применена разметка с помощью тегов, что позволяет осуществлять навигацию внутри пособия. Документы класса 3 отличаются тем, что в них применена разметка с помощью языка SGML.

Документы классов 0–3 являются линейными в том смысле, что в них, как и в обычных бумажных пособиях, материал излагается последовательно страница за страницей. В отличие от них документы класса 4 имеют не линейную, а иерархическую структуру, и предназначены для интерактивных презентаций. Развитие класса 4 в направлении увеличения степени интеллектуализации приводит к классу 5, в котором имеются средства формирования версий пособий, адаптированных к запросам и уровню подготовленности пользователя.

В технологиях ИЭТР используется также ряд других стандартов. Это стандарт ISO 8879, посвященный языку разметки SGML, стандарт ISO 10744 (HyTime – Hypermedia / Time-based Document Structuring Language), а также спецификации министерства обороны США MIL-87268…87270. Так, документ MIL-M-87268 (Interactive Electronic Technical Manual Content) определяет общие требования к содержанию, стилю, формату и средствам диалогового общения пользователя с интерактивными электронными техническими руководствами. В спецификации MIL-D-87269

содержатся требования к базам данных для интерактивных электронных технических руководств и справочников, описаны методы представления структуры и состава промышленного изделия и его компонент на языке SGML, даны шаблоны документов на составные части технической документации, перечислены типовые элементы документов.

3.3.1. Стандартизация в области образовательных технологий

В настоящее время продолжают разрабатываться методики создания электронных учебников, в том числе по федеральным научно-техническим программам. В них справедливо уделяется внимание вопросам широкого использования мультимедийных технологий, повышения эффективности тестирующих систем, учета психологических факторов при обучении и др. Необходимо в число требований к создаваемым средствам компьютерного обучения включать требования интероперабельности учебников, компиляции версий учебных материалов, адаптированных к индивидуальным особенностям обучаемых, целесообразно уделять большее внимание снижению временных и материальных затрат на создание версий учебников. Базой для реализации этих требований должны стать международные стандарты в области информационных технологий обучения и их творческое развитие в отечественных образовательных организациях.

3.3.2. Причины появления и назначение стандартов в области информационных технологий обучения

Индустрия компьютерных средств обучения развивается на протяжении уже более двадцати пяти лет. На первых порах в учебном процессе использовались различные программно-методические комплексы для освоения студентами элементов информационных технологий. Примерами таких комплексов могут служить учебно-исследовательские САПР, создававшиеся в ряде вузов страны. Одновременно получили развитие компью-

терные средства контроля знаний студентов. В конце 80-х годов стали создаваться компьютерные обучающие системы (КОС) на базе электронных учебников по различным дисциплинам с текстовыми и графическими фрагментами.

Появление Web-технологий в первой половине 90-х годов стало очевидным стимулом для развития информационных технологий в обучении. Во второй половине 90-х годов началось становление дистанционного обучения, в том числе обучения на базе Internet. Появилась концепция открытого образования, как системы предоставления образовательных услуг с помощью средств, имеющихся в распределенной информационно-образовательной среде, выбираемых пользователем и адаптированных под его конкретные запросы.

Однако существовавшие к тому времени КОС не были приспособлены к реализации идей дистанционного обучения и открытого образования в силу своей уникальности, несовместимости форматов данных, структур электронных обучающих средств и т.п. Электронный учебник, созданный с помощью авторской подсистемы в одной КОС, не мог быть воспроизведен и использован в рамках другой КОС. Существующие электронные учебники не отличались гибкостью, отсутствовали технологии адаптации содержания электронных курсов к запросам конкретных обучаемых, что не позволяло в нужной степени удовлетворить требования индивидуализации обучения. Нерешенной оставалась проблема легкости сопровождения учебников, своевременного отражения в них современного состояния науки и техники.

Со всей очевидностью возникла проблема унификации архитектур обучающих систем, структур и форматов данных для представления учебных материалов, моделей обучаемых, средств управления учебным процессом и компиляции индивидуализированных версий учебных пособий, отражающих последние научно-технические достижения.

Для решения этой проблемы было создано несколько международных и национальных организаций, поставивших перед собой цель стандартизации компьютерных средств обучения на основе современных информационных технологий. Среди этих организаций выделяются:

- IMS Global Learning Consortium – международный образовательный консорциум, развивающий концепцию, технологии и стандарты обучения на базе системы управления обучением IMS (Instructional Management System);
- IEEE LTSC - IEEE Learning Technology Standards Committee - комитет стандартизации в области технологий обучения, созданный в IEEE (Institute of Electrical and Electronics Engineers);
- AICC – Aviation Industry CBT Committee – комитет компьютерного обучения в авиационной промышленности;
- ADL – Advanced Distributed Learning Initiative Network – организация распределенного обучения, основанная департаментом политики в области науки и технологий в администрации президента США (OSTP – White House Office of Science and Technology Policy) и министерством обороны США (DoD), как сеть распределенного обучения, обеспечивающая широкомасштабный доступ к образовательным ресурсам многих пользователей.

3.3.2.1. Спецификации IMS

Консорциум IMS создан в 1997 г. ведущими промышленными компаниями в области информационных технологий, университетами и правительственными органами нескольких стран.

Система IMS включает спецификации:

- IMS Content Packaging Specification – компоновка содержания учебников и учебных пособий;

- IMS Learner Information Package Specification – описание данных об обучаемом;
- IMS Metadata Specification – описание метаданных учебных материалов;
- IMS Digital Repositories Interoperability – описание связей разных репозиториев;
- IMS Question and Test Specification – описание типичных вопросов и средств тестирования;
- IMS Digital Repositories – описание хранилищ цифровых данных и ряд других.

Эти спецификации предназначены для обеспечения распределенного процесса обучения, открытости средств обучения, интероперабельности обучающих систем, обмена данными о студентах между электронными деканатами в системах открытого образования. Распространение IMS спецификаций должно способствовать созданию единой информационно-образовательной среды, развитию баз учебных материалов, в том числе благодаря объединению усилий многих авторов при создании электронных учебников и энциклопедий.

Спецификация IMS Content Packaging Specification разработана в конце 2000 г. Совместимость учебных средств и систем обеспечивается применением специального формата (IMS Content Packaging XML format), основанного на языке разметки XML. Спецификация определяет функции описания и комплексирования учебных материалов, в том числе отдельных курсов и наборов пособий, в пакеты для сети КОС, поддерживающих концепции IMS. Пакеты (дистрибутивы) снабжаются сведениями, называемыми манифестом, о структуре содержимого, типах фрагментов, размещении учебных материалов. Манифест представляет собой иерархическое описание структуры со ссылками на файлы учебного материала. Каждый учебный компонент, который может использоваться самостоятельно, имеет

свой манифест. Из манифестов компонентов образуются манифесты интегрированных курсов.

Спецификация IMS Learner Information Package посвящена созданию модели обучаемого, включающей его идентификационные (биографические) данные, сведения, характеризующие уровень образования индивида, цели, жизненные интересы, предысторию обучения, владение языками, предпочтения в использовании компьютерных платформ, пароли доступа к средствам обучения и т.п.. Эти сведения используются для определения средств и методики обучения, учитывающие индивидуальные особенности обучаемого. Они могут быть представлены в виде таблицы, иерархического дерева, объектной модели. Возможно использование рекомендаций этой спецификации для представления данных об авторах учебных материалов и преподавателях, что может быть полезно использовано в системах управления образовательным учреждением.

Назначение спецификации IMS Digital Repositories Interoperability – унифицировать интерфейс между различными наборами ресурсов – базами учебных материалов (репозиториями), используемыми в разных обучающих системах. Обращаться к репозиториям могут разработчики курсов, обучаемые, администраторы репозиториев, программные агенты. В спецификации оговорены основные функции обращений к репозиториям, инвариантные относительно структуры наборов. Это функции помещения учебного ресурса в базу, поиска материала по запросам пользователя, компиляции учебного пособия. Система управления репозиторием при этом осуществляет запоминание вводимых данных, доставку и экспозицию запрошенного материала соответственно. Репозитории могут быть ориентированы на форматы SQL, XML, Z39.50. Формат Z39.50 используют для поиска библиотечной информации, формат XQuery (XML Query) – для поиска XML-метаданных, а протокол SOAP – для передачи сообщений. До-

ступ к репозиториям может быть непосредственным или через промежуточный модуль.

Определены сценарии действий пользователей при записи нового материала в репозиторий, при корректировке имеющихся материалов, поиске метаданных как в одном, так и сразу во многих репозиториях и в случае посылки запроса по найденным метаданным непосредственно пользователем или программным агентом, заказе извещений на изменения в метаданных.

Описание метаданных в документе IMS Learning Resource Meta-Data Information Model базируется на соответствующем документе, разработанном в IEEE LTSC (P1484.12). Спецификация определяет элементы метаданных и их иерархическую соподчиненность. В их число входят различные элементы, характеризующие и идентифицирующие данный учебный материал. Всего в спецификации выделено 89 элементов (полей), причем ни одно из полей не является обязательным. Примерами элементов метаданных могут служить идентификатор и название материала, язык, аннотация, ключевые слова, история создания и сопровождения материала, участники (авторы и спонсоры) создания или публикации продукта, его структура, уровень агрегации, версия, технические данные – формат, размер, размещение, педагогические особенности, тип интерактивного режима, требуемые ресурсы, ориентировочное время на изучение, цена, связь с другими ресурсами, место в таксономической классификации и др. Каждый элемент описывается такими параметрами, как имя, определение, размер, упорядоченность, возможно указание типа данных, диапазона значений, пояснение с помощью примера.

Метаданные используются для правильного отбора и поиска единиц учебного материала, обмена учебными модулями между разными системами, автоматической компиляции индивидуальных учебных пособий для конкретных обучаемых.

В документе IMS Question and Test Specification описана иерархическая структура тестирующей информации (с уровнями пункт, секция, тест, банк) и даны способы представления заданий (вопросов), списка ответов, разъяснений и т.п. В спецификации приведены классификация форм заданий, рекомендации по сценариям тестирования и обработке полученных результатов.

3.3.2.2. Спецификации IEEE LTSC

В комитете по стандартизации образовательных технологий Learning Technology Standards Committee (LTSC) в IEEE создан ряд рабочих групп с дифференциацией направлений работ. Эти группы занимаются разработкой и развитием следующих документов:

- P1484.1 – модель архитектуры образовательной системы (Architecture and Reference Model);
- P1484.3 – терминологический словарь (Glossary);
- P1484.11 – управление обучением (Computer Managed Instruction);
- P1484.12 – метаданные обучающих средств (Learning Objects Metadata);
- P1484.14 – семантика и замены (Semantics and Exchange Bindings);
- P1484.15 – протоколы обмена данными (Data Interchange Protocols);
- P1484.18 – профили платформ и сред (Platform and Media Profiles);
- P1484.20 – определение компетенции (Competency Definitions).

3.3.2.3. Модель SCORM

SCORM (Shareable Content Object Reference Model) – промышленный стандарт для обмена учебными материалами на базе адаптированных спецификаций ADL, IEEE, IMS, Dublin Core, and vCard. Цели создания SCORM: обеспечение многократного использования учебных модулей, ин-

тероперабельности учебных курсов (их использования в средах разных КОС), легкого сопровождения и адаптации курсов, ассемблирования контента отдельных модулей в учебные пособия в соответствии с индивидуальными запросами пользователей. В SCORM достигается независимость контента от программ управления.

Первая версия объектной модели разделяемых образовательных ресурсов SCORM [10] была представлена организацией ADL Initiative в начале 2000 г. Модель SCORM стала результатом обобщения многих проводившихся работ в области стандартизации обучающих средств для Internet. Версия 1.2 появилась в октябре 2001 г.

Основой модели SCORM является модульное построение учебников и учебных пособий, близкое к концепции модульных учебников, использованной в свое время при создании отечественной обучающей системы CTS [11] и изложенной в [12]. Модули (learning objects или instructional objects) учебного материала в SCORM называются разделяемыми объектами контента (SCO – Shareable Content Objects). Как и модули в [12], SCO – автономная единица учебного материала, имеющая метаданные и содержательную часть. Совокупность модулей определенной предметной области называется в [12] прикладной энциклопедией или в SCORM библиотекой знаний (Web-репозиторием). Модули (SCO) могут в различных сочетаниях объединяться друг с другом в составе учебников и учебных пособий, для компиляции которых создается система управления модульным учебником (сервер управления контентом), наиболее часто используемое ее название – Learning Management System (LMS).

Несмотря на общность основных идей концепций [12] и SCORM, между ними имеются и определенные различия. Так, для [12] характерно наличие онтологии приложения и поддержка соответствующего тезауруса, на их базе развита система компиляции версий электронного учебника. В

SCORM рекомендуется максимально возможная автономность содержания SCO, что не всегда соответствует характеру излагаемого материала.

В SCORM используется язык XML для представления содержимого модулей, определяются связи с программной средой и API, даны спецификации создания метаданных.

SCORM включает три части:

1. Введение (общая часть), в котором описываются основы концепции SCORM и перспективы ее развития.

2. Модель агрегирования модулей CAM (Content Aggregation Model) в законченные учебные пособия.

3. Описание среды исполнения (Run Time Environment), представляющей собой интерфейс между содержательной и управляющей частями и использующей Web-технологии и язык JavaScript. Эта часть опирается на модель данных и концепцию API, разработанную в AICC.

CAM включает:

Метаданные (Metadata Dictionary) с описанием назначения и типа содержимого модуля, сведениями об авторах, цене, требованиями к технической платформе и др.; эта часть CAM заимствована из спецификаций IEEE.

XML-данные (Content Structure) о структуре контента. Язык XML в SCORM используется в виде версии CSF (Course Structure Format). С помощью CSF представляется структура учебного курса, определяются все элементы и внешние ссылки, необходимые для интероперабельности в рамках концепций IMS, IEEE и AICC. CSF основан на модели AICC Content Model.

Данные (Content Packaging) о способах объединения модулей в пособия на базе спецификации IMS Content Packaging specification. При этом каждый элемент автоматически получает уникальный идентификатор.

Система управления LMS состоит из нескольких компонентов, выполняющих одноименные функции:

- управление контентом (Content Management Service);
- визуализация (Delivery Service);
- упорядочение материала (Sequencing Service);
- администрирование курсов (Course Administration Service);
- тестирование (Testing/Assessment Service);
- моделирование обучаемых (Leaner Profile Service);
- определение траектории обучения (Tracking Service);
- коммуникация с системной средой (API Adapter).

Предусмотрено тестирование SCORM материалов, заключающееся в проверке адекватности представления материала с помощью CSF.

Благодаря модульной структуре, многократному использованию модулей в разных версиях учебных пособий и адаптации пособий к особенностям обучаемых достигается уменьшение стоимости обучения на 30–60%, времени обучения на 20–40%, повышается степень усвоения материала.

Следует отметить еще одну версию XML, используемую в КОС. Это созданная компанией Saba Software версия Universal Learning Format (ULF). Она также основана на концепциях IMS, ADL, IEEE. Ее назначение – реализация обменов учебными материалами между различными приложениями. На ULF разработаны каталоги метаданных, профили обучаемых, библиотеки классов и т.п.

Спецификация AECMA 1000D – технология представления технической документации, признанная в авиационной промышленности (AECMA – European Association of Aerospace Constructors). В основе AECMA 1000D, как и в старших классах ИЭТР, лежит декомпозиция представляемого материала на модули. Модули включают идентификационную и содержательную секции, записанные на языках SGML или HyTime с иллюстрациями в форматах CGM или JPEG, и хранятся в специальной БД – Common

Source Data Base (CSDB). Предусмотрена автоматическая простановка гиперссылок (для этого имеются соответствующие программные средства).

Нужно иметь в виду, что SCORM пока еще окончательно не утвердился как стандарт, и что процедура независимого сертифицирования для него еще даже не начата. Поэтому по отношению к SCORM правомерно употреблять терминологическое выражение «претендующий на соответствие стандарту». Тем не менее, для эффективной работы в системах обучения, использующих ресурсы Интернета, соблюдение требований SCORM необходимо.

Для отслеживания успехов и достигнутого уровня компетенции учащихся, а также для разработки определенного маршрута продвижения учащегося по материалам курса требуется соблюдение спецификаций SCORM «Среда выполнения программ» и «Последовательность подачи материала».

Для экспортирования учебных материалов («содержания») в другие виртуальные среды обучения, соответствующие требованиям SCORM, необходимо соблюдать формат обмена данными под названием «упаковка содержания», описанный в «Модели интеграции содержания» в рамках SCORM.

Для того чтобы учебный материал был удобен для поиска и мог использоваться в определенных контекстах, нужно маркировать его содержание с помощью одного спецификации «Метаданные» – одного из компонентов «Модели интеграции содержания».

Согласно требованиям SCORM, учебные программы должны содержать три основных компонента:

1. Язык взаимодействия программ (run-time communications) – иными словами, стандартный язык, на котором обучающая программа «общается» с системой организации обучения (LMS) или с виртуальной средой обучения (VLE). Наличие такого языка важно прежде всего потому, что он

позволяет запустить и завершить программу обучения, находясь в LMS или VLE. Кроме того, этот язык делает возможной передачу данных об оценках из учебной программы в LMS.

2. Файл-манифест / пакет содержания (Content package). Этот файл содержит полное описание курса обучения и его составляющих.

3. Метаданные о курсе. Каждый фрагмент курса – изображение, страница HTML или видеоклип – ассоциируется с определенным файлом метаданных, в котором содержатся указания на то, что этот фрагмент собой представляет и где находится.

3.4. Метаданные

3.4.1. Определение метаданных

Метаданные – это данные о данных, информация об информации, описание контента. Хранение и доставка информации в электронном виде порождает много проблем. Пользователи должны иметь возможность найти нужную информацию, получить доступ к ней в приемлемой для них форме. Создатели информации должны быть уверены, что их права на интеллектуальную собственность будут защищены, а администраторы и иные специалисты должны иметь возможности по сопровождению электронной информации, например, обеспечение ее сохранности в течение длительного времени. Метаданные являются ключевым компонентом для решения этих проблем. Учитывая, что значительная часть служебных задач может решаться и реально решается без участия человека, метаданных подразделяют на предназначенные для использования приложениями и для использования человеком. В английском языке этому подразделению соответствуют термины machine-readable и human-readable.

Метаданные иногда рассматривают как разновидность давно определенной практики библиотечной каталогизации, но это не совсем так. Они отличаются друг от друга областью применения, используемыми подхода-

ми и пр. Кроме того, метаданные уже несколько десятилетий используются отнюдь не только в библиотечном сообществе. Например, можно упомянуть о т.н. «словарях-справочниках данных» (DD/D), которые еще в 70-е годы прошлого века использовались в технологиях баз данных. Давно известны и до сих пор еще используются т.н. «дескрипторные информационно-поисковые системы», в которых в качестве метаданных используются дескрипторы, содержащие произвольные термины, взятые из контента данного информационного ресурса.

Интерес к метаданным существенно возрос в связи с интенсивным развитием сетевых технологий, которые предполагают формирование и существование многочисленных сообществ, в которых взаимодействуют друг с другом люди с различными уровнями знаний и интересов, а границы между традиционными ролями (например, издатель, автор, библиотека) размыты.

Существуют различные классификации метаданных, отличающиеся между собой, главным образом, степенью детализации. Для целей данной работы мы считаем достаточным разделить их на две большие группы:

1. Метаданные описания контента.
2. Административные метаданные.

Контентные метаданные охватывают описание всех аспектов данного информационного объекта, как отдельной сущности. Иногда их дополнительно подразделяют на структурные и описательные.

Административные метаданные объединяют различные группы и отличаются большим разнообразием. Например, они позволяют владельцу ресурса проводить четкую и гибкую политику в отношении информационного объекта, включая авторизацию, аутентификацию, управление авторскими правами, доступом, а также служат для идентификации и категоризации объектов в рамках специальной коллекции или организации. Метаданные для архивирования могут включать в себя не только метаданные,

необходимые для нахождения ресурсов, возможные правила и условия доступа и т.д., но и периоды времени для классифицированной информации, информацию об открытом или закрытом хранении, данные об использовании, историю миграции с одной программно-аппаратной платформы на другую и т.д. Другая группа административных метаданных может использоваться для позиционирования данного информационного ресурса в контексте группы подобных документов, информационно-поисковой системы, предметной области и т.д. Существует группа административных метаданных, которые можно назвать «техническими». В качестве примера можно назвать схемы хранения данных в базах данных, схемы распределенных баз данных и др. Наконец, метаданные можно использовать для «кодирования» содержательной информации о том, для каких групп пользователей предназначен ресурс, для ориентирования пользователей относительно его философского, мировоззренческого смысла (т.е. метаданные будут содержать сравнительную и оценочную компоненты, призванные помочь пользователю «встроить» данную информацию в структуру его миропонимания).

Метаданные состоят из элементов, объединенных в наборы. Широко известным примером набора элементов метаданных является т.н. Дублинское ядро (Dublin Core, DC). Такие наборы разрабатываются с различными целями (например, для описания различных информационных объектов) различными организациями, которые предпринимают в случае целесообразности усилия по распространению и стандартизации своих разработок. В том случае, если набор элементов метаданных рассматривается и принимается соответствующей уполномоченной организацией (например, International Standart Organization, ISO), он становится официальным стандартом метаданных.

Необходимо подчеркнуть, что реальные наборы метаданных обычно содержат элементы как контентных, так и административных метаданных.

Т.е. необходимо понимать, что вышеприведенное разделение вполне условное, хотя есть несколько специализированных наборов именно для целей администрирования.

Поскольку могут существовать и реально существуют различные наборы метаданных, возникает потребность в специальных форматах обмена метаданными между различными информационными системами.

3.4.2. Роль метаданных

Метаданные - понятие исключительно широкое и емкое. Данный обзор ориентирован прежде всего на пользователей и создателей электронных информационных ресурсов. Применительно к этой области применения, роль метаданных об электронных ресурсах, прежде всего, состоит в:

- предоставлении возможностей более быстрого, точного и полного обнаружения необходимых ресурсов;
- обеспечении гибких и разнообразных механизмов отбора в соответствии с требованиями пользователя (поисковым запросом);
- предоставлении информации о необходимых требованиях к возможностям использования (требуемое прикладное программное обеспечение, свободное дисковое пространство и т.п.);
- управлении жизненным циклом информационных ресурсов (процессами создания, использования и хранения цифровых документов).

Метаданные способны ускорить процесс международного доступа к информации, т.к. могут быть представлены на языках, отличных от языка самого объекта.

Возможности использования метаданных исключительно широки и еще до конца не осознанны. Например, метаданные можно использовать для объединения и оценки электронных объектов в рамках обучающих ситуаций. Можно предположить, что роль метаданных для учебных материа-

лов, а также метаданных для документов, которые, по мнению пользователя, могут быть интегрированы в образовательный процесс, будет постоянно возрастать. Рабочая группа по разработке образовательных элементов набора метаданных ДЯ занимается определением дополнительных элементов и квалификаторов в конкретных областях.

3.5. Технология создания локальных и сетевых электронных образовательных ресурсов – HTML

3.5.1. Введение

Современные информационные технологии характеризуются стремительным ростом сети Internet. Технологии Internet дают возможность организовать рекламу и продажу самых разнообразных товаров, а также разместить любую другую информацию, которая сразу будет доступна сотням миллионов людей в разных странах.

Беспрецедентное увеличение интереса к размещению собственной информации в сети Internet привело к тому, что на рынке появилось очень много простых и удобных средств, с помощью которых можно создавать Web-серверы и документы HTML (Hyper Text Markup Language – язык разметки гипертекстовой информации). Такие документы и составляют основу содержимого Web-сервера.

Современные операционные системы содержат встроенные средства для работы в сети Internet. Каждый пользователь может создать собственный сервер Web, FTP и Gopher, который будет доступен миллионам пользователей глобальной сети Internet.

Web-страницы описываются на специальном языке, называемом HTML, ставшем основным языком описания документов в Internet. HTML является простым подмножеством универсального языка разметки документов SGML (Standard Generalized Markup Language – стандартный язык разметки документов), являющегося стандартом для обмена документами

между различными платформами. Точнее, весь синтаксис HTML полностью описывается с помощью SGML DTD (Document Type Definition). По этой причине почти все программы, совместимые с SGML, могут быть использованы при подготовке HTML-документов.

За сравнительно короткое время разработчики Web-страниц прошли путь от простого перевода текстовых документов на язык HTML до создания красочных, искусно оформленных интерактивных страниц, с умело используемой графикой и различными стилями размещения текста на странице. Появилась профессия под названием «Web-дизайнер», то есть человек, специализирующийся на создании Web-страниц высшего качества.

Некоторые современные Web-страницы можно со всей ответственностью назвать произведениями искусства.

Интересно отметить некоторые особенности, отличающие верстку информации для Web и верстку для «обычной», то есть бумажной, технологии передачи документов. В отличие от языков описания печатных документов, вроде известного языка PostScript, упор делается на переносимость информационного наполнения страниц, а не их внешнего оформления.

Поясним сказанное на примере: при переносе документа на языке PostScript между двумя компьютерами гарантируется сохранение его внешнего вида, то есть размеров, шрифтового оформления; тогда как для HTML-документов гарантируется лишь сохранение логической структуры.

Это происходит потому, что никто не гарантирует, что устройство, на котором пользователь будет просматривать Web-страницу, не окажется черно-белым алфавитно-цифровым терминалом 1970-го года выпуска! Или же что программа просмотра, используемая пользователем, способна корректно отобразить графические вставки в различных форматах. И поэтому

Web-дизайнер несет особую ответственность за представление информации на своих страницах.

Очевидно, что язык HTML сохранит свои позиции и в будущем как основное средство для разметки WWW-документов. Поэтому его необходимо изучать во всех тонкостях. Хотя современные средства создания Web-страниц позволяют осуществлять и визуальное проектирование, превращая эту работу в простое форматирование текста или таблиц, но в дальнейшем, при усложнении оформления страниц, возможности многих редакторов исчерпываются. Кроме того, большинство редакторов создают страницы, которые содержат много избыточной информации, и задача оптимизирования кода ложится на специалиста HTML.

Более того, одинаковые страницы по-разному отображаются в браузерах различных производителей, поэтому для создания полноценного Интернет-ресурса необходимо учитывать и эти особенности. Например, элементы типа «бегущая строка», которые корректно отображаются в Microsoft Explorer, в браузере Netscape Navigator выглядят простым текстом.

Если сайт создается для пользования внутри организации, то можно использовать любые технологии, поскольку заранее известно, каким будет программное обеспечение клиента, но если дело касается сети Интернет, то нужно либо создавать несколько версий одного ресурса, либо использовать устоявшиеся стандарты.

Кроме того, в поисках большей функциональности и привлекательности наиболее распространенная схема разметки время от времени меняется. В этом случае необходимо ориентироваться на лидеров, уловить тенденции и в то же время найти свой собственный стиль. Вообще, понятие стиля очень важно в Web-дизайне. Для того чтобы все ваши документы были узнаваемыми, необходимо соединить их единым дизайном.

В настоящее время доступно множество статей по Web-дизайну, однако стоит отметить, что многие из них предлагают путь исследователя, обучающегося на ошибках.

Если вы создаете корпоративный сайт, который будет визитной карточкой вашей организации в Сети, то стилю следует с самого начало уделить максимум внимания, привлекая, если это необходимо, для консультации профессиональных дизайнеров, имеющих художественное образование и опыт работы.

Для того чтобы опубликовать информацию для глобального распространения, необходим универсальный язык, который потенциально могли бы воспринять все компьютеры. В Мировой паутине таким языком является HTML, который, так или иначе, используется при создании любого сайта.

HTML был разработан Тимом Бернерс-Ли, когда он работал в организации CERN в Женеве. Первоначально HTML был предназначен для публикации электронных документов и описания их структуры, т.е. с его помощью можно было задать, что в документе является заголовком, где начинается новый параграф, где находятся иллюстрации, куда ведут ссылки и т.д. Знание структуры документов облегчало их автоматическую обработку, например, зная, какие заголовки есть в тексте, можно легко сделать автоматическую генерацию оглавления.

Благодаря росту Мировой паутины HTML быстро набрал популярность. Многие хотели использовать его для красочного оформления сайтов, однако в этом плане HTML оказался очень ограничен. Он был больше ориентирован на описание структуры документа, чем на его визуальное оформление. HTML стал совершенствоваться. В нём появлялось больше средств для управления внешним видом Web-страницы и для придания Web-страницам динамичности.

Сегодня HTML предоставляет следующие средства:

- публикации электронных документов с заголовками, текстом, таблицами, списками, фотографиями и т.д.;
- оперативного получения информации лёгким нажатием кнопки благодаря гипертекстовым ссылкам;
- создания формы для проведения операций с различными сервисами: поисковыми системами, электронными магазинами и т.д.
- вставки электронных таблиц, видеоклипов, звуковых эффектов и многого другого прямо в документы.

Для написания страниц с разными стилями и сценариями используются языки расширения – CSS (Cascading Style Sheets) и JSSS (JavaScript Sheets).

3.5.2. Что такое гипертекстовый документ

Сеть Интернет представляет собой гигантский набор информации, основанной на гипертекстовых документах. Любой гипертекстовый документ представляет собой обычный текстовый файл в формате ANSI ASCII, содержащий собственно текст и специальные теги для его разметки, а также ссылки на другие подобные документы, графические изображения и любые иные файлы. Когда браузер – программа просмотра гипертекста – загружает подобный файл, все теги обрабатываются и содержимое файла форматируется для отображения. Теги задаются в файле и обрабатываются браузером в соответствии с правилами специального языка – HTML (Hyper Text Markup Language – Язык Разметки Гипертекста).

Следует отметить, что HTML, вопреки распространенному заблуждению, ни в коей мере не является языком программирования. HTML – это типичный язык разметки, т.е. с его помощью можно оформлять документы, создавать ссылки, но никак не писать программы. Даже специальные эффекты, которые можно увидеть на страницах Web, создаются не с помощью HTML, а с использованием дополнительных средств, например

встроенных в документ программ на языке JavaScript или с использованием Java-апплетов.

Путешествуя по сети, вы, вероятно, уже не раз задавались вопросом: а как созданы все эти страницы? Любой браузер позволяет просмотреть исходный текст HTML-документа. Для IE это команда «просмотр HTML-кода» из меню «Вид», а для Netscape (Mozilla) – «Page Source» из меню «View».. Таким образом, если вам приглянулась та или иная страница, вы всегда можете просмотреть, как она сделана.

Сразу необходимо оговориться, что если вы видите в заголовке страницы что-либо вроде «Microsoft FrontPage 3.0» или «Netscape Composer», смело закрывайте окно просмотра источника: скорее всего, ничего полезного извлечь все равно не удается. Автоматические генераторы HTML составляют код страниц так, как это им удобно, и никакой осмысленной структуры документа вы не увидите.

Из доступности исходных текстов страниц следует сделать два вывода: во-первых, вы всегда можете воспользоваться чужим опытом; во-вторых, все ваше творчество также будет доступно в исходном виде для всех посетителей ваших страниц.

Следует отметить, что подходы к отображению исходного текста у браузеров MSIE и Netscape 4 разные – если первый показывает исходный вид страницы, то второй это делает после ее первичной обработки интерпретатором. Таким образом, в MSIE вы увидите, например, исходные тексты функций JavaScript, предназначенных для генерации тела документа, а в Netscape –результаты их работы.

Возможно, если вы посмотрите на исходный текст сейчас, то ничего не поймете, но, уже прочитав только первую часть книги, вам будет совершенно ясно, что и для чего используется, а прочитав книгу до конца, вы сами сможете создавать практически сколь угодно сложные и интересные страницы.

3.5.3. Действительные документы HTML

Возможно, вы будете удивлены, если узнаете, что большинство документов, находящихся в сети, не являются действительными документами HTML 4.0. Это объясняется тем, что браузеры весьма снисходительно относятся к неточностям в разметке документов. Кроме того, большинство из существующих редакторов по-прежнему используют устаревшие и отмененные в HTML 4.0 атрибуты тегов для форматирования документов.

Такая ситуация в значительной мере вызвана проблемами совместимости различных версий браузеров с теми или иными стандартами HTML. Например, Netscape Navigator вплоть до версии 3.0 не поддерживает таблицы стилей, а Internet Explorer 3.0 поддерживает их крайне ограниченно, допуская ряд ошибок при обработке CSS. Впрочем, к настоящему моменту доля пользователей, все еще использующих столь старые программы, стремительно приближается к нулю. Поэтому, разрабатывая современные страницы для Интернет, следует придерживаться существующих стандартов. Для документов WWW таковым в настоящий момент является HTML 4.01. Он достаточно хорошо поддерживается браузерами Internet Explorer и Netscape Navigator (начиная с 4-х версий), и от редакции к редакции программ поддержка стандарта только улучшается. Что касается Internet Explorer 5.x и, особенно, Netscape Navigator 6.0, то в них поддержка HTML 4 обеспечена почти на все 100%. Очень корректно HTML 4 обрабатывает и 3-я по популярности программа просмотра WWW – Opera 4.

Поэтому нерешительность некоторых авторов, которые игнорируют ряд новых и не очень достижений (таблицы стилей, использование фреймов, встраиваемых объектов и т.д.), объясняется зачастую не тем, что они «хотят поддерживать максимальное количество платформ», а элементарным нежеланием переучиваться.

Вместе с тем стандарт – стандартом, а реальные пользователи используют реальные программы просмотра. Поэтому не следует всецело

полагаться на описанные стандарты – необходимо также регулярно просматривать результаты работы хотя бы двумя наиболее популярными на текущий момент браузерами. На ближайшее время можно порекомендовать Netscape 6, Internet Explorer 6.0, а также Mozilla и Opera 7. Следование данным правилам сделает ваши страницы более привлекательными для посетителей.

Пользователям компьютеров с процессором класса ниже, чем Pentium 166, возможно, следует использовать MSIE 4.01, поскольку интерфейс пятой версии слишком медлителен, а различия между MSIE 4.01, 5.0 и 5.5 не столь кардинальны, чтобы жертвовать удобством работы.

Проверить любой документ на соответствие действующему стандарту можно при помощи специального сервиса, предоставляемого всем желающим консорциумом по стандартизации Интернет – W3C. Если вы зайдете на их сервер, расположенный по адресу «http://www.w3.org», то можете найти ссылку на страницу с «HTML-Validator». Указав URL проверяемой страницы и подождав несколько секунд, вы увидите подробный список всех ошибок и несоответствий. Если таковых не имеется, то получите сообщение вроде «This document is HTML 4.0 Strict».

Вообще, данный ресурс (www.w3.org) является неисчерпаемым источником информации по всему, что связано с сетью. На нем вы найдете специальные документы – RFC, в которых описываются все принятые и разрабатываемые стандарты по языку HTML, сетевым протоколам, и всему, что имеет отношение к Интернет. Единственное, но, возможно, критичное ограничение для российских пользователей – это то, что вся информация приводится на английском языке.

3.5.4. HTML- редакторы

Теперь перейдем непосредственно к творчеству. Для создания любого файла нужна какая-нибудь программа. Если, как в нашем случае, речь

идет о текстовом файле, то нам нужен текстовый редактор. Теперь ознакомимся с тем, что предлагают разработчики на рынке редакторов HTML.

Прежде всего, следует отметить, что все HTML-редакторы подразделяются на два типа – редакторы тегов и WYSWYG- редакторы (What You See Is What You Get – что видишь, то и получаешь). Первые предполагают, что пользователь знаком с языком разметки (HTML), и предоставляют пользователю возможность самому вводить теги, а вторые подобны обычному текстовому процессору, например программе Microsoft Word. Следует отметить, что сам Word 97 (или 2000), по существу, тоже является WYSWYG-редактором, поскольку позволяет сохранять отформатированный документ в виде HTML-файла.

Рассмотрим подробнее вторую группу. Прежде всего, если вы используете Netscape Communicator, то у вас имеется программа для редактирования – Netscape Composer. Microsoft, в свою очередь, предлагает программу Front Page Express, входящую в поставку Windows 98. Существуют, конечно, и другие, но эти две – самые распространенные. К достоинствам Composer следует отнести более удобный интерфейс, а Front Page – средства поддержки проекта (сервера). Среди недостатков у Composer – это менее эффективный и читабельный код на выходе, а у Front Page, как правило, – безликие и серые страницы. Последнее обусловлено тем, что масса шаблонов и мастеров создания документов, на самом деле, ограничивают фантазию разработчика и навязывают ему «штампованный» дизайн.

В любом случае, работая с подобными редакторами, очень трудно создать что-нибудь оригинальное и эффектно смотрящееся, поскольку ваша фантазия всегда будет ограничена рамками поддерживаемых конкретной программой средств. Более того, практически все WYSWYG-редакторы «оптимизированы» под конкретный браузер и создают страни-

цы, которые могут неадекватно восприниматься конкурирующими программами просмотра.

В противоположность им редакторы тегов не ставят никаких ограничений. Автор страниц может свободно использовать любые теги, применяя самые новые возможности HTML, не дожидаясь, пока выйдет новая версия редактора, которая их поддерживает. Как правило, подобные программы создаются небольшими фирмами или пишутся отдельными программистами. Среди них есть совсем простые, например SNK HTMLPad. Такие программы позволяют быстро создавать заготовку документа, вставлять наиболее распространенные теги по нажатию сочетания горячих клавиш и вызывать браузер для просмотра редактируемого документа подобным образом.

Некоторые, например SNK LightPad, также поддерживают подсветку синтаксиса. Есть и более мощные, например HomeSite фирмы Allaire или отечественный Visual HTML Workshop. При помощи этих программ можно не только использовать специальные мастера для создания наиболее сложных элементов (например, таблиц), но и работать над проектами – набором взаимосвязанных файлов, образующих сайт. Так, HTML Workshop позволяет автоматически конвертировать все документы в нужную кодировку при отправке на сервер, производить поиск и замену сразу по всем файлам проекта, создавать ссылки между файлами одним движением мышки, следить за изменениями сайта, загружать файлы по FTP и т.д.

Между тем в качестве редактора тегов можно использовать и простейшую программу, входящую в поставку Windows. Имя ей – Notepad, или блокнот. К сожалению, отсутствие горячих клавиш, ограничения на размер файла и т.д. не добавляют этому редактору достоинств, но, тем не менее, его использование вполне может быть оправдано, когда надо быстро подправить 1–2 строчки в небольшом файле. В то же время некоторые пользователи устанавливают вместо блокнота альтернативные программы,

например можно инсталлировать тот же HTMLPad себе на компьютер, переименовать его исполняемый файл в notepad.exe и перезаписать поверх имеющегося. Тем самым во всех случаях, когда должен загрузиться блокнот Windows, будет загружаться выбранный вами редактор. Конечно, использовать для таких целей Workshop, HomeSite или иной достаточно громоздкий редактор не рекомендуется, поскольку такие программы требуют довольно много ресурсов и не слишком бистро загружаются.

3.5.5. Первый документ HTML

Определившись с редактором, рассмотрим структуру документа более внимательно. Если у вас установлен HTMLPad, то нажмите «Ctrl+N», а если нет, то наберите в блокноте (Notepad) следующий текст:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
<HEAD>
<TITLE> </TITLE>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Это – заготовка для любого HTML-документа. Любой документ HTML состоит из трех частей:

- строки, содержащей информацию о версии HTML;
- блока заголовка документа;
- тела документа, либо содержащего непосредственно отображаемую информацию, либо определяющего набор фреймов.

Если вы сохраните этот текст в файл с расширением «htm» или «html», например «hello.html», а затем откроете в браузере, то увидите пу-

стое окно, что не совсем интересно. Поэтому мы немного модифицируем этот файл. Между <TITLE> и </TITLE> введите фразу: «Мой первый документ HTML», а между <BODY> и </BODY> напишите «Всем привет!». Если вы сохраните этот файл и откроете его в браузере, то увидите примерно то, что изображено на рис. 3.5.

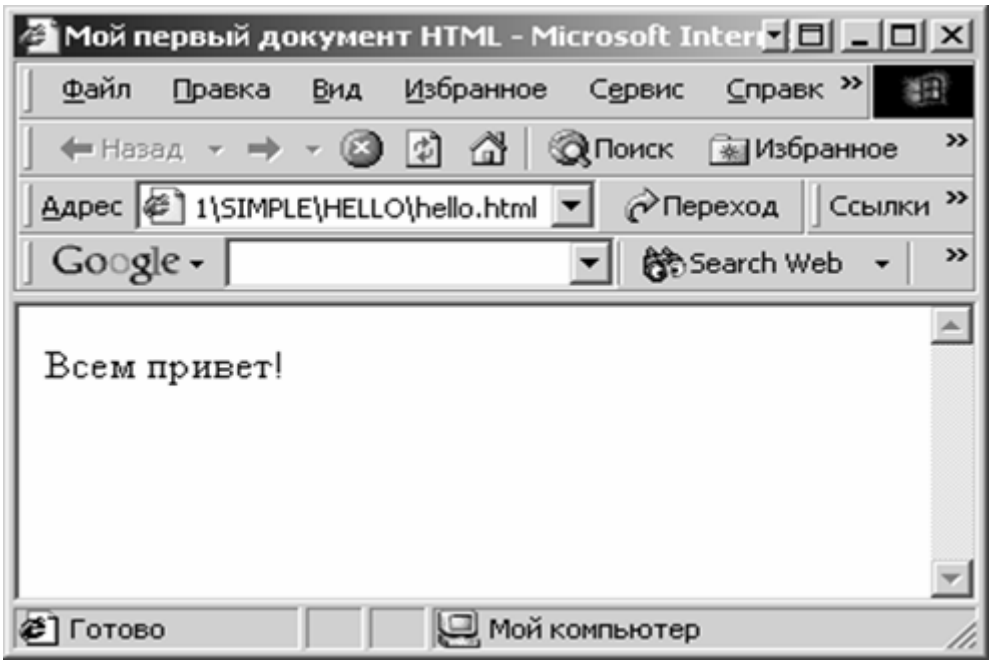

Рисунок 3.5. Простейший HTML-документ в MSIE

Если вы работаете с HTMLPad (или LightPad), то сохранять файл перед просмотром необязательно. Это делается автоматически, когда вы нажимаете кнопку просмотра. При этом оригинальное содержимое файла (после последнего сохранения) заносится в специальный буфер, что дает возможность, в случае необходимости, отказаться ото всех внесенных изменений.

Мы видим, что текст, набранный нами между тегами <TITLE> и </TITLE>, отобразился в заголовке окна, а то, что написано внутри <BODY> – в самом окне браузера. Сам элемент TITLE помещен внутри контейнера HEAD, который является заголовком документа. Помимо TITLE, в заголовке могут располагаться и другие элементы, например META, однако на текущем этапе мы не будем их рассматривать.

А пока что можно себя поздравить – своими руками мы только что создали свой первый документ HTML.

3.5.6. Гиперссылки

Основа гипертекста – это ссылки, как на отдельные части документа, так и между различными документами. Подобные ссылки называются гиперссылками. В языке HTML предусмотрен элемент A, при помощи которого и создаются гиперссылки. Давайте создадим еще один документ и назовем его index.html. Вот его содержание:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
<HEAD>
<TITLE> Главная страница </TITLE>
</HEAD>
<BODY>
Приветствую вас на своей страничке!
Вот мое  <A HREF="hello.html">приветствие</A>.
</BODY>
</HTML>
```

При просмотре его в браузере мы увидим примерно то, что изображено на рис. 3.6. Обратите внимание, что перенос строки не воспринят браузером. Дело в том, что весь текст HTML-файла воспринимается программой просмотра как единая строка, при этом все лишние пробелы, символы конца строки и табуляции просто отбрасываются.

Если вы сохранили файл в той же директории (папке), что и предыдущий, то при щелчке по ссылке на слове «приветствие» откроется предыдущий документ. Так работают гиперссылки между документами. Для того

чтобы создать ссылку на часть внутри документа, применяется следующая конструкция:

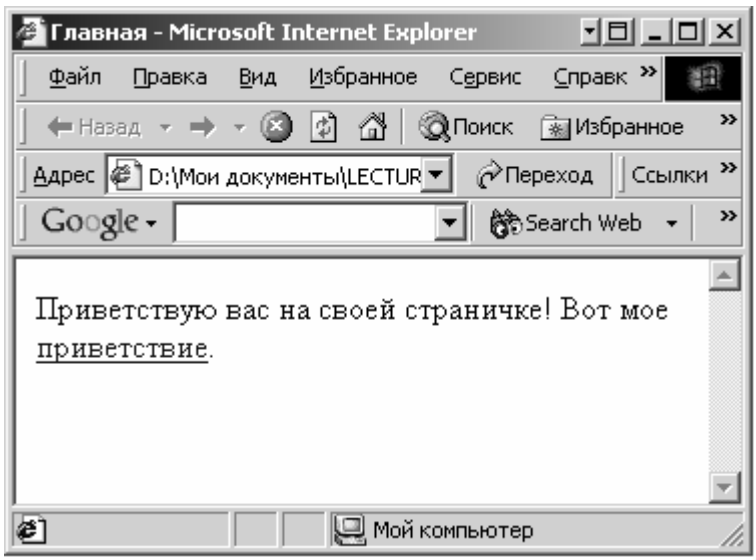

Рисунок 3.6. Документ с гиперссылкой

```
...
<A HREF="#ANCHOR1"> Ссылка-1 </A>
… часть документа …
<A NAME="ANCHOR1">
Текст, на который ссылается ссылка-1.
…
```

Как видно, тоже ничего сложного. Значок # указывает браузеру, что объект ссылки расположен внутри текущего файла. Параметр NAME задает имя метки. Подобные метки называются якорями (Anchors) и могут состоять из произвольной комбинации латинских букв и (или) цифр. В HTML 4 вместо NAME рекомендуется использовать атрибут ID, что дает возможность ссылаться не только через элементы A, однако лишь MSIE 5.x и Netscape 6 или версий выше могут распознать ID как адрес для ссылки.

Ссылки внутри файла и на внешний файл можно комбинировать, например:

```
<A HREF="seconfile.html#ANCHOR1"> Ссылка-2 </A>
```

создает ссылку на якорь ANCOR1 в файле «secondfile.html». Главное правило в использовании якорей – «не злоупотребляй», поскольку увлечение ими может существенно затруднить навигацию по сайту.

Как уже отмечалось ранее, ссылки можно создавать не только на HTML-документы, но и на любые другие файлы. Делается это точно так же:

```
<A HREF="my_file1.zip">Мой любимый файл</A>.
```

Во всех предыдущих примерах подразумевалось, что файл, на который указывает ссылка, и файл с самой ссылкой, расположены в одной и той же директории. Если это не так, то создается относительная ссылка, в которой прописывается часть пути между директорией с исходным документом и файлом-назначением. Для создания такой ссылки сделайте следующее: создайте папку «hello» внутри папки с документом «index.html» и переместите в нее файл «hello.html». Затем модифицируйте index.html следующим образом: поменяйте строку

```
Вот мое  <A HREF="hello.html">приветствие</A>
```

на

```
Вот мое <A
HREF="hello/hello.html">приветствие</A>.
```

В данном случае мы создали ссылку на файл во вложенной директории. Чтобы создать ссылку на файл во внешнем каталоге, т.е. расположенном уровнем выше, используется двоеточие. Чтобы проверить все это на практике, откройте файл «hello.html» и добавьте следующий текст:

```
<A HREF="../index.html">вернуться к главной стра-
нице </A>
```

Сохраните оба файла и проверьте работоспособность ссылок. Если все сделано правильно, то обе ссылки будут работать. Когда требуется создать ссылку на файл на два уровня выше, то двоеточие используется дважды, например:

```
<A HREF="../../index.html1">Вернуться на два ка-
талога вверх</A>
```

Все рассмотренные выше варианты верны для случая, когда создаются ссылки внутри одного сайта. Если же надо сделать ссылку на внешний сервер или на какой-либо другой ресурс, то используется строка вида:

```
[протокол][адрес][порт][путь][имя файла]
```

Такой адрес называется URI – Universal Resource Identifier. В качестве протокола, как правило, используется HTTP. Типичный URI выглядит примерно так: «http://snkey.net/center/index.html». Если используется другой протокол, например FTP (для передачи файлов), то и URI должен быть соответствующим. Например, файл «cc32e47.exe», находящийся в каталоге «pub» на FTP-сервере Netscape, будет иметь следующий URI: «ftp://ftp.netscape.com/pub/cc32e47.exe». Сама ссылка будет выглядеть так:

```
<A HREF="ftp://ftp.netscape.com/pub/cc32e47.exe">
Скачать NC 4.7</A>
```

Параметр «порт», как правило, явно не указывается, т.к. используются значения по умолчанию – 80 для HTTP и 21 – для FTP. Если же порты используются, то адрес будет выглядеть так:

```
http://www.anyhost.ru:8080/hiddendir/foo.html
```

Очень часто URI путают с URL. Следует отметить, что URL является подмножеством от более общей схемы – URI.

Если вы хотите создать ссылку на адрес электронной почты, то она будет выглядеть примерно следующим образом:

```
<A HREF="mailto:i.am@myhost.ru">пишите пись-
ма!</A>
```

Здесь «mailto:» указывает браузеру, что далее следует адрес электронной почты, и при щелчке по ссылке необходимо вызвать почтовую программу. Если вы хотите передать еще и тему сообщения, то после адреса электронной почты укажите «subject». Например, чтобы создать сообщение со строкой "Привет" в качестве темы (subject), надо написать так:

```
<A HREF="mailto:i.am@myhost.ru?subject = При-
вет!"> пишите!</A>
```

Подобным образом создаются и ссылки на news-конференции, только вместо «mailto:» пишется «news:», а вместо адреса электронной почты – имя группы новостей. Иные параметры элемента A, в частности TARGET, который определяет то, где должна "сработать" ссылка, будут рассмотрены в дальнейшем.

3.5.7. Форматирование документа

В предыдущих разделах мы вкратце ознакомились со структурой документа, узнали о том, что такое ссылки и как их делать. Теперь рассмотрим, как выполняется форматирование документов. Следует отметить, что форматирование и оформление – не одно и то же. Форматирование задает структуру, а оформление – внешний вид. И если до недавних пор все это было перемешано, то теперь для оформления в HTML 4 используют каскадные таблицы стилей, или CSS. О том, что это такое и как их использовать, вы узнаете уже через главу, а к рассмотрению структуры приступим прямо сейчас.

К элементам структурной разбивки в первую очередь относятся абзацы и заголовки. В HTML для обозначения абзаца используется элемент P, а для заголовков – H1, H2, H3, H4, H5 и H6. Кроме того, существуют так

называемые элементы уровня блока, например DIV, предназначенный для форматирования части или блока страницы, например группы абзацев.

Рассмотрим применение этих элементов на практике. Откройте файл «index.html» и измените его следующим образом:

```
<!DOCTYPE HTML PUBLIC  "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML>
<HEAD>
<TITLE> Главная страница <TITLE>
</HEAD>
<BODY>
<H!>Приветствую вас на своей страничке!</H1>
<H2>Это – главная страница. </H2>
 А вот мое <A HREF="hello/hello.html">приветствие
</A>.
Больше пока ничего нет.
</BODY>
</HTML>
```

Абзац, формируемый элементом P, также заставляет браузер создать отступ величиной приблизительно в одну строку до и после текста. Однако если поставить несколько тегов P подряд, то эффект будет таким же, как и от одного. Поэтому для задания более значительного отступа применяют несколько элементов BR. Он заставляет браузер сделать то, чего вы добиваетесь, нажимая на клавиатуре клавишу «Enter» при редактировании текста в обычном текстовом редакторе.

3.5.8. Синтаксис гипертекстовой разметки

Прежде чем продолжить знакомство с технологиями создания страниц, следует сделать отступление насчет синтаксиса в HTML-документах.

Прежде всего это касается правил использования кавычек для строковых ресурсов. Они сводятся к следующему:

1. Вы можете использовать как одинарные – ('), так и двойные – (") кавычки, но при этом строка должна быть заключена в кавычки одного типа:

```
<A HREF= "http://www.snkey.net" > – верно
<A HREF= 'http://www.snkey.net' > – тоже верно
<A HREF= "http://www.snkey.net'> – неверно!
```

2. В самом языке HTML рекомендуется все значения атрибутов элементов заключать в кавычки, даже если они не содержат пробелов:

```
<IMG SRC= «5.gif» ALT= «Рисунок N 5» > – верно;
<IMG SRC=5.gif ALT='Рисунок N 5' > – тоже верно;
<IMG SRC=5.gif ALT=Рисунок N 5 > – неверно!
```

Атрибуту ALT будет присвоено значение «Рисунок», а «N» и «5» будут нераспознанными атрибутами.

3. Если требуется указать кавычки внутри строки, то используйте кавычки разного типа:

```
<IMG SRC="5.gif" ALT="Рис. 5 – 'Цветочки' " > –
верно;
<IMG SRC=5.gif ALT='Рис. 5 – "Цветочки" ' > –
тоже верно;
<IMG SRC=5.gif ALT="Рис. 5 – "Цветочки" " > – не-
верно!
```

Атрибуту ALT будет присвоено значение «Рис. 5 – ».

4. Все, что сказано в 1-м и 3-м пунктах, верно и для JavaScript:

```
var string1 = "Некоторая строка"
var string2 = 'Другая строка'
```

```
var strings = 'Строка с "Кавычками" '
```

5. Кроме того, в JavaScript для того чтобы вставить в строку кавычки или иные символы, можно применять обратную косую черту:

```
var string1 = "Некоторая строка с \"Кавычками\" "
var string2 = 'Другая строка с \'Кавычками\' '
```

Будьте предельно внимательны при расстановке кавычек, поскольку их отсутствие или неправильное использование может привести к непредсказуемым результатам. Например, если вы не закроете кавычку, как в приведенном ниже примере, то почти весь текст HTML-документа будет воспринят как содержание атрибута CONTENT элемента META:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
 Transitional//EN" >
<HTML>
<HEAD>
<TITLE>Заголовок</TITLE>
<META NAME=Description CONTENT="Мой документ >
</HEAD>
<BODY>
```

Содержимое тела документа, которое никто не увидит.

```
<A HREF="foo.htm">Ссылка, которая не будет рабо-
тать </A>
</BODY>
</HTML>
```

Когда интерпретатор дойдет до CONTENT=«, он все последующие символы, вплоть до первой встретившийся кавычки, присвоит атрибуту CONTENT. Далее foo.htm» (именно так, вместе с примыкающей кавычкой) будет принят за нераспознаваемый атрибут элемента META. Затем угловая

скобка – > будет распознана как завершение элемента META, и все дальнейшее будет показано как тело документа. В результате окно браузера будет выглядеть, как показано на рис. 3.7.

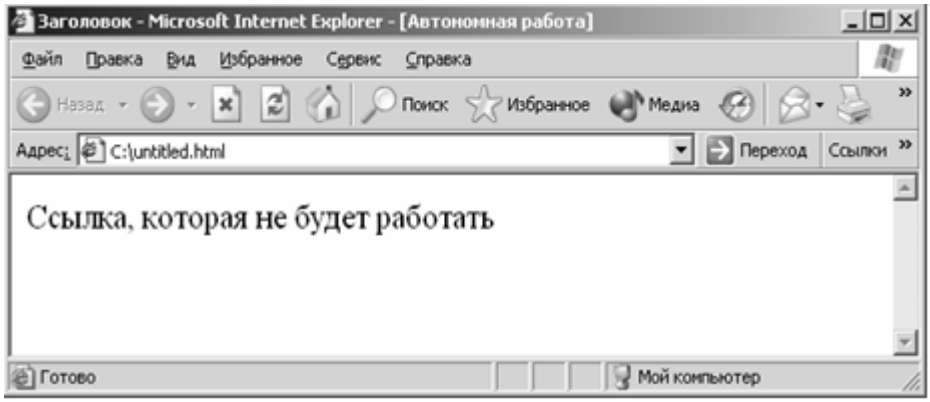

Рисунок 3.7. Ошибки при расстановке кавычек

Если же закрывающей кавычки так и не найдется, то все содержимое HTML-документа, начиная с «проблемного» элемента будет выведено в окно браузера. При этом, конечно, содержимое будет показано, а ссылка будет работать, но это будет слабым утешением. Окно браузера будет выглядеть, как показано на рис. 3.8.

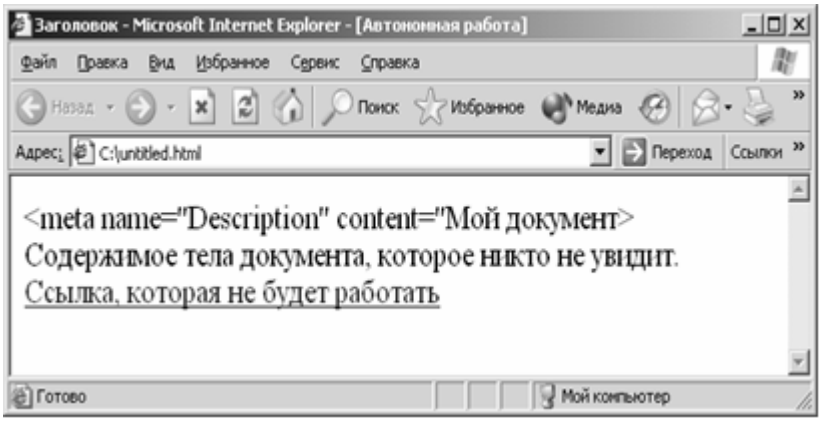

Рисунок 3.8. Другая ошибка

Еще одна часто встречающаяся ошибка – это использование обратного слеша – «\» – вместо обычного – «/». Если для работы в DOS и Windows используется обратный слеш, то в UNIX и в Интернет вообще – только обычный.

3.5.9. Каскадные таблицы стилей

Вернемся к предыдущему примеру. Как правило, заголовки лучше выглядят, когда их выравнивают по центру. Это можно сделать двумя способами: либо указать для каждого из них выравнивание по центру, либо поместить оба в единый блок. Для первого случая добавим в начало документа «index.html» (в часть HEAD) следующий текст:

```
<STYLE TYPE="text/css">
H1,H2 {text-align: center;}
</ STYLE >
```

Если просмотреть этот документ снова, то можно увидеть, что заголовки теперь выровнены по центру. Чтобы применить выравнивание сразу на несколько блоков, определим выравнивание при помощи элемента DIV. Для этого, вместо определения стилей для HI и H2, определим стиль для DIV:

```
DIV {text-align: center;}
```

Затем вложим оба заголовка в контейнер DIV. В результате должен получиться примерно такой документ:

```
<HTML>
<HEAD>
<TITLE> Главная страница </TITLE>
<STYLE TYPE="text/css">
 DIV {text-align: center;}
</ STYLE >
</HEAD>
<BODY>
<DIV>
<H1>Приветствую вас на своей страничке!</H1>
<H2>Это – главная страница. </H2>
```

```
А вот мое <A HREF="hello/hello.html">приветствие
</A>. Больше пока ничего нет.
</BODY>
</HTML>
```

В браузере это будет выглядеть точно так же, как и в предыдущем варианте. В обоих случаях мы занимались уже не форматированием, а оформлением документа, для чего использовали таблицы стилей. Вообще, таблицы стилей позволяют очень гибко оформлять документ – выравнивать заголовки и абзацы, задавать цветовое и шрифтовое оформление, создавать рамки вокруг объектов и многое другое. Причем действие таблиц стилей может распространяться как на конкретный элемент, так и целиком на страницу или даже на множество страниц сразу. Именно поэтому они и называются каскадными. Для того, чтобы не возникали конфликты между таблицами различных уровней, определены приоритеты области воздействия. Суть их сводится к тому, что чем ближе к элементу находится описание стиля, тем больший приоритет он имеет.

Рассмотрим данный аспект более внимательно. Для этого возьмем последний вариант файла «index.html» и добавим к открывающему тегу <H2> атрибут STYLE, в котором укажем, что выравнивание элемента надо проводить по правому краю:

```
<H2 STYLE="text-align:right">Это – главная стра-
ница </H2>
```

Теперь, если открыть этот документ в браузере, то строка «Это – главная страница» будет выровнена не по центру, а по правому краю. Таким образом, мы убедились, что встроенные при помощи атрибута STYLE таблицы стилей имеют наибольший приоритет, ведь получается, что стиль определен внутри тега – ближе некуда!

Ранее было отмечено, что действие каскадных таблиц стилей может распространяться на несколько документов сразу. Для этого их выносят в отдельный CSS-файл, а затем ссылаются на него из других документов. Например, создадим файл «mystyle1.css» такого содержания:

```
H1,H2 {color: red;}
```

Затем добавим в часть HEAD файла «index.html» строку:

```
<LINK REL=STYLESHEET HREF="mystyle1.CSS"
TYPE="text/CSS">
```

Теперь заголовки стали красного цвета. И любой документ, имеющий ссылку на «mystyle1.css», будет иметь заголовки HI и H2 красного цвета, если только в самом документе значение цвета для этих тегов не будет переопределено. Т.е. если в файле «index.html» мы в определении STYLE добавим

```
H1 {color: blue;}
```

то строка «Приветствую Вас на своей страничке!» будет написана синим цветом.

Помимо применения общего для всех одноименных тегов определения стиля, можно создавать классы элементов. Например, если мы хотим, чтобы в одном и том же документе одни абзацы выравнивались по центру, а другие – по обоим краям (по ширине), то можно создать соответствующие классы для элемента P. Делается это так:

```
<STYLE TYPE="text/CSS">
P.Center  {text-align: center;}
P.Both  {text-align: justify;}
</STYLE>
. . .
```

```
<P CLASS="Center"> ...некоторый текст, который
надо разместить по центру... </P>
<P CLASS="Both"> ...некоторый текст, который дол-
жен быть выровнен по обоим краям... </P>
```

Здесь в определении стиля мы создали два класса для тега P: Center и Both. Для первого указано, что текст форматировать надо по центру, а для второго – по обоим краям (или, как это еще называют, по ширине окна). Для того чтобы указать, к какому именно классу принадлежит конкретный тег, используется атрибут CLASS. Если атрибут CLASS не указан, то будет применено форматирование, принятое по умолчанию (в случае с абзацем – выравнивание по левому краю).

Кроме задания классов элементов, можно ссылаться на конкретный элемент, заданный уникальным идентификатором – ID. Это делается при помощи определения стиля, явно не привязанного ни к одному из элементов:

```
# somename1 {color: green}
. . .
<P ID="somename1"> этот текст должен быть зеле-
ным! </P>
```

Стиль, определенный как «somename1», будет применен к одноименному элементу, вне зависимости от того, чем этот элемент является. Важно только знать, что идентификатор (ID) элемента должен быть уникальным, т.е. на странице не должно быть более одного элемента с ID, равным, например «somename1».

Примеры с таблицами стилей к этой главе расположены в каталоге «Samples/Part_l/w_styles».

Следует учитывать, что классы и идентификаторы могут быть как общими (как в примере с ID), так и привязанными к типу элемента (как в

примере с классами для P). Например, если задать класс, определенный для P, иному элементу (например, H1), то такая запись не сработает. Поэтому если вам надо создать класс, который может использоваться с любыми элементами, то используйте следующий синтаксис:

```
.Center  {text-align:   center;}
```

Более подробно правила определения классов и идентификаторов CSS рассмотрены ниже в главе «Таблицы стилей».

Когда требуется указать числовое значение, необходимо также указать и его тип – проценты, пиксели, сантиметры и т.д. В таблице 3.1 приведены типы значений, которые можно указывать в CSS1 и CSS2.

Таблица 3.1

Типы значений CSS1 и CSS2

Обозначение	Тип	Описание
em	высота/ширина	Размер относительно высоты текущего шрифта
px	высота/ширина	Размер в пикселях относительно текущего устройства
in	высота/ширина	Абсолютный размер в дюймах (1in=2,54cm)
cm	высота/ширина	Абсолютный размер в сантиметрах
mm	высота/ширина	Абсолютный размер в миллиметрах
pt	высота/ширина	Абсолютный размер в поинтах (1pt=1/72in)
pc	высота/ширина	Абсолютный размер в пиках (1pc=12pt)
%	высота/ширина	Размер в процентах относительно текущего
deg	углы	Угол в градусах
rad	углы	Угол в радианах
s, ms	время	Секунды и миллисекунды, для звуковых устройств
Hz	частоты	Частота звука в герцах, для звуковых устройств
kHz	частоты	Частота в килогерцах, для звуковых устройств

В HTML 4.0 атрибуты STYLE, CLASS и ID предусмотрены практически для всех элементов. Единственное исключение – тег BR, для которого не существует атрибутов CLASS и STYLE, и действие таблиц стилей на него не распространяется.

3.5.10. Типы представления документов

Знакомство с таблицами стилей будет неполным, если не отметить еще одно важнейшее их свойство – тип представления. CSS уровня 2 определяет следующие типы представления:

- ALL (общее) – задает оформление для всех видов устройств доступа к Интернет;
- aural (звуковое) – для синтезаторов речи;
- braille (Брайля) – для интерактивных осязательных устройств;
- embossed (рельефное) – для печатающих устройств на основе таблиц Брайля;
- handheld (блокнотное) – для карманных компьютеров, монохромных дисплеев и подобных устройств;
- print (печатное) – для обычных страничных принтеров;
- projection (проекционное) – для проекторов и печати на пленках;
- screen (экранное) – для обычных компьютеров с цветными дисплеями;
- try (строчное) – для устройств вроде телетайпа или пейджера;
- tv (телевизионное) – для отображения на цветных устройствах с низким разрешением и звуковым сопровождением.

При всем изобилии вариантов особенно важно выделить screen и print, так как именно для просмотра на компьютерах и печати на принтерах создается, по крайней мере, сейчас, большинство Web-страниц. Например, если ваши страницы выглядят на экране как светлый текст на темном фоне, то имеет смысл задать обратное представление для печати. Для этого

предусмотрен атрибут MEDIA элемента STYLE. Также можно задать для монохромных устройств – принтеров и микроноутбуков (принтер вполне можно рассматривать как монохромное устройство) один вариант оформления, а для просмотра на мониторе – другой:

```
<LINK REL="stylesheet" TYPE="text/css" ME-
DIA="print,  handheld" HREF="printstyle1.css">
<LINK REL="stylesheet" TYPE="text/css" ME-
DIA="screen"
HREF="coolstyle2.css">
```

Если атрибут MEDIA не указан, то стили будут задействованы для любого случая. Несмотря на определенную экзотичность, пренебрегать указанием типа представления не следует. В типичном случае бывает полезно ограничить действие таблиц стилей компьютером, т.е. указав MEDIA="screen".

Помимо этого, указанные выше свойства сгруппированы на три группы: визуальную (screen, tty, print, projection, handheld), звуковую (aural) и осязательную (braille, emboss). Телевизионное представление относится сразу к двум группам – визуальной и звуковой. Таким образом, в качестве значения MEDIA можно указывать не только конкретный тип, но и группу:

```
<LINK REL="stylesheet" TYPE="text/css" ME-
DIA="visual" HREF="coolstyle3.ess">
```

В дальнейшем таблицы стилей будут рассмотрены более подробно.

3.5.11. Правила оформления документа

Создание HTML-документов требует от автора предельной внимательности и богатого воображения. Дело в том, что иногда даже трудно представить, в какой конфигурации окажется компьютер посетителя. Это относится как к аппаратному обеспечению, так и к программному. Напри-

мер, можно предположить, что наиболее распространенное разрешение экрана – 800×600 точек, глубина цвета – 16 и более бит, операционная система –Windows, а браузер – MSIE 5.0 или выше. Однако много ли посетителей вашего сайта имеют именно такой набор? Не более 25%, могу вас уверить! Так что, если вы «оптимизируете» свой сайт для вроде как самой распространенной конфигурации, три четверти посетителей останутся недовольны, а вернее, не увидят того, что вы ожидали.

Для фиксированного размещения объектов на Web-страницах многие дизайнеры используют таблицы, указывая размеры (в частности, ширину) непосредственно в пикселях. Поскольку минимальное из встречающихся на практике разрешений экрана – 640×480, то стараются ограничить ширину таблицы шестьюстами пикселями (не следует забывать, что существует еще и полоса прокрутки, а также небольшой отступ от границы экрана до самой таблицы). Конечно, разрешение 640×480 встречается уже крайне редко, поэтому можно смело выбирать ширину рабочей области в 750 пикселей.

При таком подходе все содержимое страницы помещается внутрь таблицы, и можно быть уверенным, что при изменении размеров окна дизайн не «рассыплется». Однако есть и недостатки: дело в том, что пока таблица не прогрузится до конца, большинство браузеров не сможет показать ее содержимое. Кроме того, структура страницы, созданная при помощи таблицы, если только последняя не представляет собой одну большую ячейку, трудна для восприятия в исходном (HTML) виде.

В начале 1997 года Netscape для решения этой и ряда других проблем ввела элемент LAYER, при помощи которого можно не только указывать непосредственные размеры, но и управлять наложением одних частей страницы на другие. Фактически, если вы работали с мощным графическим редактором, то представляете себе, что такое слои. Сразу после выхода Communicator 4.0 на главной странице Netscape можно было видеть

применение слоев в динамике, когда навигация по серверу была выполнена в стиле кнопки «Пуск» – с раскрывающимися подменю и прочими атрибутами.

К сожалению, технология слоев в таком виде не нашла поддержки у других производителей, в частности у Microsoft. Но не следует сильно огорчаться: решение, и, причем достаточно универсальное, все же есть. Сводится оно к использованию каскадных таблиц стилей. Вы можете указывать фиксированные размеры и расположение для любого блочного элемента. Как правило, для этих целей используют элемент DIV:

```
DIV {
margin-left: 10%;
width: 550px;
}
```

В этом примере мы задали ширину блока в 550 пикселей с отступом слева величиной в 10% от ширины экрана. Таким образом, все, размещенное внутри блока, охваченного тегом DIV, будет усечено по ширине в 550 пикселей и будет иметь отступ в 64 или более пикселей от левой границы окна браузера в зависимости от разрешения (вплоть до 180, если установлен режим 1800×1440).

Теперь несколько слов о цвете. Прежде всего, если возможно, старайтесь не использовать светлый шрифт на темном фоне. Если текста на странице много, то читать его будет тяжело. В крайнем случае, указывайте достаточно крупный размер шрифта. Кроме того, следует следить за контрастностью между фоном и цветом текста: чем она выше, тем лучше. Чтобы это проверить, снимите копию экрана, нажав «Print Screen», вставьте полученное изображение из буфера обмена в графический редактор и конвертируйте его в полутоновое (grayscale) изображение. Если текст

останется легко читаемым, значит, палитра имеет достаточную контрастность, если нет – попробуйте подобрать другое сочетание цветов.

Отдельно следует рассмотреть использование рисунка в качестве фонового изображения. Прежде всего, рисунок (так же, как и фон вообще) не должен быть ярким и контрастным, иначе он будет «забивать» само содержимое страницы. Для фона хорошо подходят неяркие, немного расплывчатые изображения. Они могут быть как темными, так и светлыми. Гигантские, иногда на десятках компакт-дисков, коллекции текстур и фоновых изображений предлагают некоторые крупные фирмы, разрабатывающие графические пакеты. Объемностью своих коллекций славится канадская корпорация Corel. Есть из чего выбрать и у других поставщиков. Как правило, такие коллекции стоят недорого – в пределах 10 долларов за диск, хотя найти в них что-нибудь подходящее бывает довольно трудно. Разумеется, множество архивов с коллекциями графических файлов можно найти непосредственно в сети Интернет.

Выбрав фоновое изображение, не забудьте подобрать близкий по тону цвет фона страницы. Это делается для случая, когда загрузка графики у пользователя отключена. Кроме того, фон может прогрузиться не сразу, и если вы используете белый текст на темном фоне, то до тех пор, пока фон не прогрузится (если вообще прогрузится!), посетитель вашей странички ничего не сможет прочитать, поскольку в большинстве систем по умолчанию используется белый или светло-серый фон.

Определившись с цветом, выберите подходящий шрифт. На самом деле выбор невелик: до недавних пор было возможно использовать только две гарнитуры: с фиксированным и с пропорциональным шагом. Для большинства систем это соответственно Courier и Times.

Разработчики шестой версии Netscape пошли немного дальше и предлагают выбрать не два, а три типа шрифтов: фиксированный, пропор-

циональный с засечками (например, Times) и пропорциональный без засечек (например, Arial).

Использование таблиц стилей позволяет указывать любой шрифт, однако следует учесть, что применение экзотических гарнитур приведет к тому, что большинство пользователей не увидят того, что вы ожидали показать. Это связано с тем, что на разных компьютерах установлены разные шрифты. Единственный распространенный шрифт (помимо двух указанных выше) – это Arial (или Helvetica). Только его можно использовать без особого риска.

Выход, однако, есть. Спецификация CSS2 явно определяет использование «загружаемых» шрифтов. Иными словами, вместе со страницей на компьютер посетителя будет загружен и файл с требуемым шрифтом. Следует, правда, оговориться, что формат подобных файлов отличается от применяемых в операционных системах типа Windows. В связи с этим разработчики браузеров создают такие шрифты сами. На страницах сервера Microsoft (www.microsoft.com) можно найти несколько подобных гарнитур. К сожалению, символы кириллицы в них пока что отсутствуют.

Ну, и еще одно замечание, на этот раз по ссылкам. Нет ничего хуже, чем попытка «замаскировать» ссылки. Постарайтесь сделать так, чтобы ссылки отличались от остального текста – либо подчеркиванием, либо цветом, а лучше и тем, и другим. В противном случае, посетителю будет трудно ориентироваться на вашей странице.

3.5.12. Чего надо стараться избегать

Прежде чем заняться творчеством для WWW, следует раз и навсегда запомнить, что максимальный размер типичной Web-страницы не должен превышать 70–100 Кб, включая как сам HTML-файл, так и все внедренные в него графические и иные объекты. Некоторые редакторы, например Visual HTML Workshop, имеют в своем распоряжении специальную функцию,

вычисляющую суммарный размер и предполагаемое время загрузки документа. Дело в том, что скорость загрузки страницы в браузер по сети на несколько порядков ниже, чем с жесткого диска.

Кроме того, если страница будет действительно огромной, то ее вообще никто не увидит, поскольку время соединения стоит денег и маловероятно, что найдется человек, готовый ждать 5, а то и 15 минут пока прогрузится ваше полумегабайтное творение, не зная даже наверняка, что он увидит в результате!

Из этого следует сделать такие выводы:

- осторожно подходите к выбору графики для оформления страницы;
- не забывайте, что фоновое изображение – тоже рисунок;
- каждый баннер – это, в среднем, еще 10 килобайт объема;
- не помещайте крупные рисунки и «скриншоты» на главной странице.

Если вы, например, нарисовали в 3D MAX что-нибудь замечательное, которое интересно смотреть только в полноэкранном режиме и в 24-битном цвете, ни в коем случае не помещайте такой рисунок на главную страницу. Лучше создайте ссылку на этот файл, не забыв указать его размер. Можете не сомневаться: кому интересно, тот обязательно посмотрит. Кроме того, во многих случаях целесообразно создать миниатюрные, размером в 3–4 килобайта копии для того, чтобы посетитель мог разобраться, стоит ли ему щелкать по ссылке и дожидаться загрузки полноразмерной картинки.

Еще одно ограничение – это неумеренное использование апплетов Java, анимированных рисунков и прочих двигающихся и мигающих объектов. Конечно, автора, впервые создавшего какой-нибудь «потрясающий» апплет или нарисовавшего мигающе-переливающийся GIF, понять можно. Только вот посетители страницы могут не разделять таких восторгов.

Кроме того, подобные творения, как правило, довольно долго грузятся. А когда их много, то восприятие страницы вообще становится проблематичным. Поэтому прежде чем наполнять страницу столь специфическим содержанием, подумайте, действительно ли оно того стоит?

Помимо оформления не следует забывать о главном, то есть о содержании, поскольку именно содержание определяет суть сайта. О чем именно рассказать на сайте – дело автора, надо лишь учесть тот факт, что то, о чем вы рассказываете, должно быть интересно не только вам. Когда речь идет о корпоративном (или любом ином коммерческом) сервере, повышенное внимание следует уделять стилю изложения информации. Если текст будет трудным для восприятия, то не много найдется желающих его прочитать. А если будут встречаться еще и орфографические ошибки, то ваша репутация может сильно пострадать.

В любом случае, плохо сделанный, да еще, не дай бог, содержащий ошибки корпоративный сайт, вместо того, чтобы привлечь клиентов, с большой долей вероятности будет их отпугивать.

Персональные странички, конечно, не предъявляют столь жестких требований к стилю изложения, поскольку ясно, что никакой менеджер по паблик рилейшенс над содержимым не трудился. Тем не менее, требования к русскому языку остаются в силе. Если вы не уверены, что можете свободно писать без ошибок, установите резидентную программу для проверки орфографии, например «ОРФО», либо прогоняйте весь текст через текстовый процессор (например, Microsoft Word).

Иногда можно встретить странички, русское содержимое которых продублировано на английском языке. В 90% случаев английский дубль содержит столь чудовищное количество ошибок, что его полное отсутствие было бы лучшим вариантом. В среднем один раз в месяц можно встретить неправильно написанное само слово «English». Варианты бывают разными, например «Inglish» или «Englesh». Поэтому если вы не владе-

ете свободно английским, то без крайней надобности не делайте английскую копию: сэкономленную энергию можно направить и на более полезные занятия.

Если сайт представляет собой не одну, а несколько страничек (или несколько сотен), то необходимо следить за единством стиля. Единство стиля подразумевает под собой общее шрифтовое и цветовое оформление, сходную компоновку страниц, выполненную в одном стиле графику и т.д. Конечно, не обязательно все страницы должны быть одного цвета, иногда разные разделы сайта имеют различное цветовое решение. Но в таком случае цвет выполняет как бы поясняющую роль, т.е. показывает, в каком разделе находится посетитель. И уж если вы используете градиентный фон (скажем, переходящий из красного в белый) для какого-либо раздела, то потрудитесь сделать фон такого же типа для всех остальных страниц сайта. Пусть они будут желто-белыми, сине-голубыми или даже изумрудно-салатовыми, но выполненные в таком же стиле – с плавным переходом от темного к светлому.

Самый лучший цвет фона – белый. На нем без проблем можно разместить любые иллюстрации. Кроме того, данные, полученные из различных источников, всегда без особых проблем можно перенести на белый фон и быстро привести к единому стилю.

На самом деле, можно еще долго обсуждать, что следует делать, а что – нет, но суть, как мне кажется, раскрыта достаточно полно. Качество проделанной работы сполна смогут оценить ваши будущие посетители. Но было бы полезно, прежде чем выкладывать работу на всеобщее обозрение, показать ее нескольким знакомым, сотрудникам, родственникам, в конце концов. Требуйте от них критического восприятия вашего творчества и задумывайтесь над тем, что вам скажут.

3.5.13. Публикация

Вот мы и подошли к заключительной стадии работы – публикации готовых страниц в сети. Подробно об этом рассказывается в следующей части. Сейчас ограничимся только общими советами.

1. Не забывайте загружать в Интернет все связанные файлы. Очень часто начинающие авторы, отправив сам HTML-документ, забывают загрузить связанные с ним картинки.

2. Помните о регистре! В Интернет файлы «abc20.gif» и «abc20.GIF» – не одно и то же.

3. Всегда проверяйте результаты работы. То, что прекрасно работало, находясь у вас на компьютере, может перестать работать, оказавшись в сети. В первую очередь это касается рисунков и ссылок.

Большинство HTML-редакторов, в частности уже упоминавшиеся Composer, Front Page, HomeSite и Visual HTML Workshop, имеют собственные средства для загрузки страниц. Если вы используете программу, располагающую такими средствами, то лучше всего их и использовать, поскольку они выполнят загрузку не только новых и измененных HTML-страниц, но и связанных с ними файлов автоматически.

Если же вы предпочитаете использовать отдельный FTP-клиент, то будьте внимательны. Рассмотрим работу широко распространенного в России файлового менеджера, имеющего также функции FTP-клиента – FAR. Его важным достоинством является то, что он бесплатен для жителей постсоветского пространства. Кроме того, внешне он очень похож на любимый многими Norton Commander.

Для использования функций FTP запустите программу FAR, нажмите Alt+F2, чтобы открыть список дисков для правой панели. В нижней части диска имеется пункт FTP. Выберите этот пункт. Для того чтобы создать новое соединение с FTP-сервером, нажмите клавиши "Shift+F4". В появившемся окне, необходимо ввести строку следующего формата:

```
FTP://username:password@host:port/dirname
```

Здесь «username» обозначает ваш логин для доступа к FTP-серверу, «password» –пароль, «host» – адрес FTP-сервера, «port» – его порт (как правило, 21), а «dirmane» – опциональный параметр, указывающий на директорию, которая должна быть открыта в начале сеанса. Например, если ваш сайт находится по адресу – «ftp.newmail.ru», ваше имя пользователя – «user1232», а пароль – «qwerty», то настройки будут такими:

```
ftp://user123:qwerty@ftp.newmail.ru
```

После того, как введены настройки, можно подключиться к удаленному серверу (при этом, разумеется, должно быть установлено соединение с Интернет). Дополнительные настройки сеанса FTP могут быть такими:

- ask password directly before connecting – запрашивать пароль при каждом подключении;
- ASCII mode – передавать файлы не в двоичном, а в текстовом формате;
- passive mode и Use firewall – иногда эти настройки требуются при работе через шлюз.

Еще одна популярная программа – CuteFTP (www.cuteftp.eom). Главное окно программы разделено на четыре части:

- верхнее окно, в котором отображаются передаваемые серверу команды;
- левое центральное окно, в котором отображается содержимое вашего винчестера;
- правое центральное окно – показывает файлы на сервере;
- нижнее окно – информирует об отложенных заданиях.

В самом верху расположена панель инструментов, на которую вынесены наиболее часто применяемые функции. Пожалуй, самой важной является первая – «Site Manager». Нажав на нее, вы попадете в FTP Site Man-

ager – менеджер FTP-сайтов. С его помощью вы можете создавать новые подключения, редактировать имеющиеся и, собственно, подключаться к FTP. Для создания нового подключения нажмите «Add site», после чего появится окно. Поле «Site label» указывает на название сайта, «Host Address» – адрес ftp-сервера, «User ID» и «Password» – ваши логин (имя пользователя) и пароль для этого сайта. В целом программа достаточно проста в использовании, и, при желании, вы без труда сможете с ней освоиться.

А вот о чем следует сказать отдельно – так это о формате передачи файлов. Как правило, лучше использовать двоичный режим передачи файлов. Он просто необходим при загрузке и выгрузке двоичных файлов, таких, как графика, архивы и программы. Он же подходит и для HTML-файлов.

Однако если вы загружаете cgi-скрипты, то вам необходимо установить текстовый режим, поскольку в UNIX-системах используются иные правила для переноса строк. При загрузке в текстовом режиме файлы приводятся к нужному виду автоматически, если же вы будете закачивать файлы скриптов в двоичном режиме, то вам придется сначала обработать концы строк при помощи какой-либо специальной утилиты. Как правило, вам надо самостоятельно устанавливать тип передачи файлов, но некоторые FTP-клиенты автоматически переходят в текстовый режим при передаче cgi- и pl-файлов.

Еще один важный момент для публикации файлов скриптов на UNIX-серверах – это права доступа к файлу. Как правило, требуется разрешить чтение и выполнение для всех групп и дополнительно – права на запись для владельца (т.н. 755). Если щелкнуть по файлу (или каталогу) правой кнопкой мышки и выбрать пункт «UNIX-доступ», то появится диалоговое окно, в котором вы сможете указать права доступа.

ЛИТЕРАТУРА

1. Международное стандартное библиографическое описание для электронных ресурсов – ISBD (ER).

2. Агеев В.Н. Электронная книга: Новое средство соц. коммуникации. М.: 1997.

3. Гречихин А.А., Древс Ю.Г. Вузовская учебная книга: Типология, стандартизация, компьютеризация. М.: Логос, 2000.

4. Мильчин А.Э. Издательский словарь-справочник. М.: Юристъ, 1998.

5. Субботин М.М. Новая информационная технология: Создание и обработка гипертекстов. М., 1992.