ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра экономических и информационных систем

Р.Р. Халимов, Е.И. Горожанина

ПРОЕКТНЫЙ ПРАКТИКУМ

Часть 2

УЧЕБНОЕ ПОСОБИЕ

Самара
2017

УДК 004.89
ББК 30
Х172

Рекомендовано к изданию методическим советом, протокол №61 от 20.04.2017

Рецензент
доцент кафедры «Экономические и информационные системы» ПГУТИ, к.т.н.
Богданова Е.А.

Халимов, Р.Р.
Х Проектный практикум. Часть 2. Учебное пособие. [Текст] / Р.Р.Халимов, Е.И. Горожанина. – Самара. ФГБОУ ВО ПГУТИ, 2017. – 84 с.

Учебное пособие имеет целью ознакомить учащихся с тестированием и требованиями к программному обеспечению, вариантами реализации программного обеспечения, некоторыми языками программирования, программной реализацией базы данных проекта, этапами составления заявки на изобретение.

Учебное пособие подготовлено на кафедре "Экономические и информационные системы" ФГБОУ ВО ПГУТИ, предназначены для студентов всех форм обучения специальности 09.04.03 (Прикладная информатика). Могут быть полезны студентам смежных специальностей.

Содержание

Введение

Курс «Проектный практикум» изучается студентами специальности «Прикладная информатика» на протяжении 6 и 7 семестров.

В 6 семестре студенты изучают

- вопросы построения бизнес-процессов, используя различные методологии, такие как ARIS, IDEF0, IDEF3, UML,
- вопросы построения логической схемы базы данных, используя методологию IDEF1X,
- разработку технического задания;
- разработку технико-экономического обоснования проекта
- и др.

Подготовленное учебное пособие призвано помочь студентам в процессе освоения дисциплины «Проектный практикум» в 7 семестре, а именно при изучении разделов:

- тестирование программного обеспечения;
- разработка требований к программному обеспечению;
- варианты реализации программного обеспечения;
- языки программирования;
- программная реализация базы данных проекта;
- этапы составления заявки на изобретение.

Целями освоения дисциплины являются формирование у студентов теоретических знаний в области управления проектами и развития практических навыков и профессиональных компетенций в части выполнения проектных работ по автоматизации прикладных процессов.

Раздел 1 Разработка программных алгоритмов

Программный алгоритм является переходом между абстрактной моделью программы (её проектом, реализованным в виде текстового описания, UML-схем, ER-схем) и непосредственно программным кодом, который будет выполняться на каком-либо вычислительном устройстве (например, компьютере, сервере, мобильном устройстве и т.п.).

Программный алгоритм не является программным кодом на каком-либо языке программирования, он описывает только логику работы программы или её отдельных компонент (программных модулей или функций разрабатываемой программы), последовательность операций, выполняемых в процессе работы будущей программы, в виде графической блок-схемы.

Схема алгоритма — это набор взаимосвязанных блоков различного назначения, которые следуют друг за другом в определенной последовательности, описывая логику работы разрабатываемой программы и те вычислительные операции, которые программа должна выполнять.

Опытные программисты зачастую не используют данный этап проектирования, так как реализуют программные алгоритмы сразу в виде программного кода. Однако, такой подход не всегда оправдан, так как в чрезмерно сложных проектах (например, в программных системах уровня крупного предприятия) невозможно охватить и спланировать сразу все аспекты работы программы и реализация её сразу в виде программного кода может привести к серьезным логическим ошибкам, которые могут обнаружиться далеко не сразу и привести к серьезным последствиям для организации, использующей данное программное обеспечение.

Гораздо эффективнее в таких случаях перед написанием программного кода описать логику работы программы и её компонентов в виде графической, легкочитаемой всеми участниками проекта, схемы. Преимущества такого подхода

заключается в том, что при наличии каких-либо логических ошибок или отклонений от технического задания их сможет обнаружить любой из участников проекта (а не только программист), и сделает это ещё до того, как будут написаны тысячи строк программного кода. Таким образом, это является ещё одним – последним – этапом контроля выполняемых работ со стороны как заказчика, так и исполнителя проекта.

Как правило, сначала составляется обобщенная (укрупненная) блок-схема алгоритма, отражающая логику работы программы на высоком уровне абстрагирования. Такая блок-схема содержит блоки, соответствующие отдельным функциям или вычислительным операциям программы. Затем некоторые блоки (операции) обобщенного алгоритма могут детализироваться несколькими подблоками (подоперацями), после чего такие подблоки могут быть также детализированы на несколько подблоков и так далее, до тех пор, пока логика разрабатываемого алгоритма не будет достаточно понятной для реализации данного алгоритма в виде программного кода в выбранной среде программирования.

Стоит отметить, что значительная детализация алгоритма не требуется, так как то, в какой последовательности выполнять операции внутри блока, какими методами и программными модулями это делать решает программист уже на этапе написания программного кода. Алгоритм же описывает только логику работы программы, то есть то, как «она себя ведет» с пользователем и то, какие действия и с какими данными позволяет ему совершать.

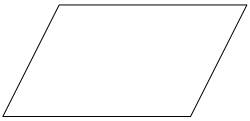
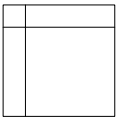
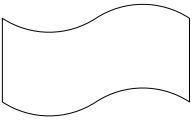
Схему моделирующего алгоритма целесообразно оформлять в соответствии с ГОСТ, например, следующими:

- ГОСТ 19.003-80. ЕСПД. Схемы алгоритмов и программ. Обозначения условные графические
- ГОСТ 19.701-90 (ИСО 5807-85). ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.

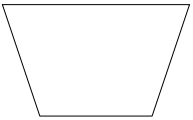
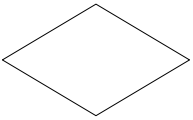
В данных регламентирующих документах определены основные фигуры схем алгоритмов (табл. 1), использование которых при построении схем алгоритмов обеспечит понимание таких схем различными специалистами, заказчиком проекта, а также разработчиками программного кода, участвующими в реализации программы.

Таблица 1

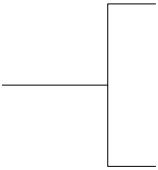
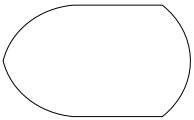
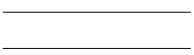
Основные фигуры схем алгоритмов

N	Символ	Значение символа
1		данные (например, те, которые программа получает на этапе инициализации своих переменных, то есть на этапе получения значений исходных данных, которые известны до запуска программы и будут использоваться далее в процессе её работы)
2		данные, которые хранятся на каком-либо носителе данных (на жестком диске данного компьютера, на сервере, в облачном хранилище и т.д.)
3		данные, хранящиеся в оперативной памяти устройства (в ОЗУ), на котором выполняется программа
4		бумажный носитель (документ), пригодный для компьютерной обработки (для сканирования, передачи факсом и т.д.)
5		процесс, то есть выполнение программой отдельной операции

Продолжение табл. 1

N	Символ	Значение символа
6		составной процесс, то есть процесс, включающий в себя несколько операций (подпроцессов). Как правило, такие блоки указываются на обобщенной схема алгоритма и “раскрываются” (заменяются набором взаимосвязанных подпроцессов)
7		ручная операция, выполняемая человеком. Его целесообразно использовать в тех случаях, когда определенные этапы выполнения программы требуют действий от пользователя – например, для продолжения работы программы нужно ввести какие-то данные: логин или пароль или выбрать способ отображения результатов программы. То есть данный символ можно использовать везде, где есть взаимодействие с пользователем, требующим от него выполнения каких-либо действий
8		условный оператор и результатом его выполнения является то или иное решение, в зависимости от выполненных в нём условий. Верхний угол фигуры является входом (в него входит стрелка от предыдущего блока), оставшиеся три угла могут быть использованы и как входы и как выходы.

Продолжение табл. 1

N	Символ	Значение символа
9		«соединитель» связывает последовательные блоки схемы алгоритма на разных страницах или на одной странице, если связать блоки обычным способом проблематично
10		переход из внешней по отношению к алгоритму среды или во внешнюю среду. Он записывается вначале и в конце алгоритма, а также там, где алгоритм завершается
11		комментарий. Связывается с блоком. Текст комментария записывается справа в квадратной скобке
12		вывод какого-либо сообщения для пользователя на экран или на монитор компьютера. Также он может быть использован для обозначения вывода запросов программы или промежуточных результатов её работы
13		распараллеливание операций. Его можно записывать после любого блока. Выходом данного символа являются несколько стрелок, ведущих к блокам операций, выполняемых параллельно

При построении схем алгоритмов следует учитывать также следующие правила:

– все блоки связываются между собой стрелками;

– в блоки операций стрелки входят сверху и выходят снизу;

– линии всех стрелок на схеме алгоритма должны быть либо горизонтальными, либо вертикальными и, в случае их пересечения, пересекаться под прямым углом;

– стрелки на схеме алгоритма не должны пересекать блоки;

– все блоки схемы алгоритма нумируются в последовательности их выполнения натуральными числами, начиная с единицы;

– номер блока должен располагаться вверху справа от блока.

На рис. 1.1 показан один из вариантов применения символа распараллеливания операций.

Операции C, D и E выполняются параллельно, сразу после выполнения операции в блоке A. Но операция F не может начаться, пока все предшествующие ей параллельные операции – B, C и D – не будут завершены. Даже если одна из таких операций, например, операция C, завершится раньше, то переход в блок с операцией F произойдет только после того, как завершатся и операция B и операция D.

Рассмотрим пример программного алгоритма работы банкомата коммерческого банка (рис. 1.2).

При этом учтём, что данный пример является учебным и потому упрощенным – в действительности логика работы банкомата является гораздо более сложной.

Работа банкомата начинается с чтения устройством (банкоматом) вставленной банковской карты (блок 1).

Если карта вставлена правильно и ее удалось идентифицировать (блок 2), т.е. определить, что это именно банковская карта, то (условие Блока 3) запрашивается PIN-код карты (блок 6).

Если же карта была вставлена не той стороной или это была не банковская карта (условие Блока 3), то карта

возвращается владельцу (блок 4) и на экран выводится сообщение об ошибке чтения карты (блок 5).

В ответ на запрос PIN-кода карты (блок 6) пользователь вводит такой код (блок 7). Код карты проверяется системой банкомата (блоки 8, 9 и 10) и, если он верен, то пользователю предоставляется выбор необходимого ему сервиса (проверка баланса, снятие наличных, оплата сотовой связи и т.д.) - блок 11. Как только пользователь осуществит выбор такой операции, система банкомата осуществляет такую операцию (блок 12), сообщает об этом пользователю (блок 13), возвращает карту (блок 14) и завершает работу.

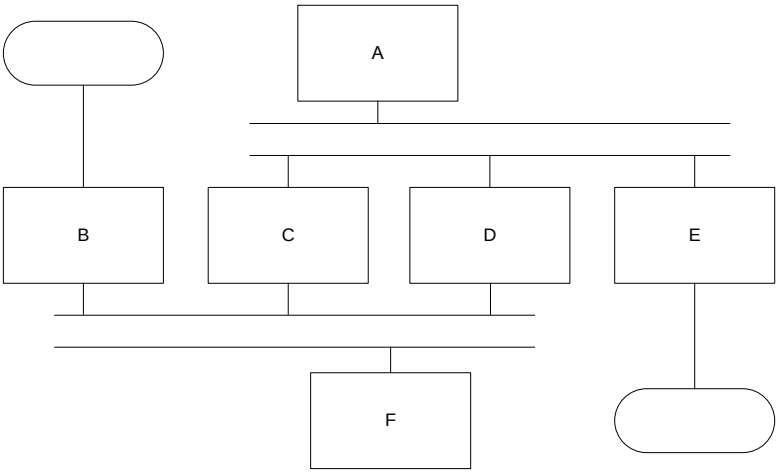


Рис. 1.1 – Применение символа распараллеливания операций

Обратите внимание, блоки 11 и 12 являются составными (операции в них не вполне очевидны из названия самих блоков) и детализируются – то есть они должны быть “разбиты” на подблоки (подоперации), для того чтобы описать логику производимых в них действий более подробно и однозначно определить производимые в них операции.

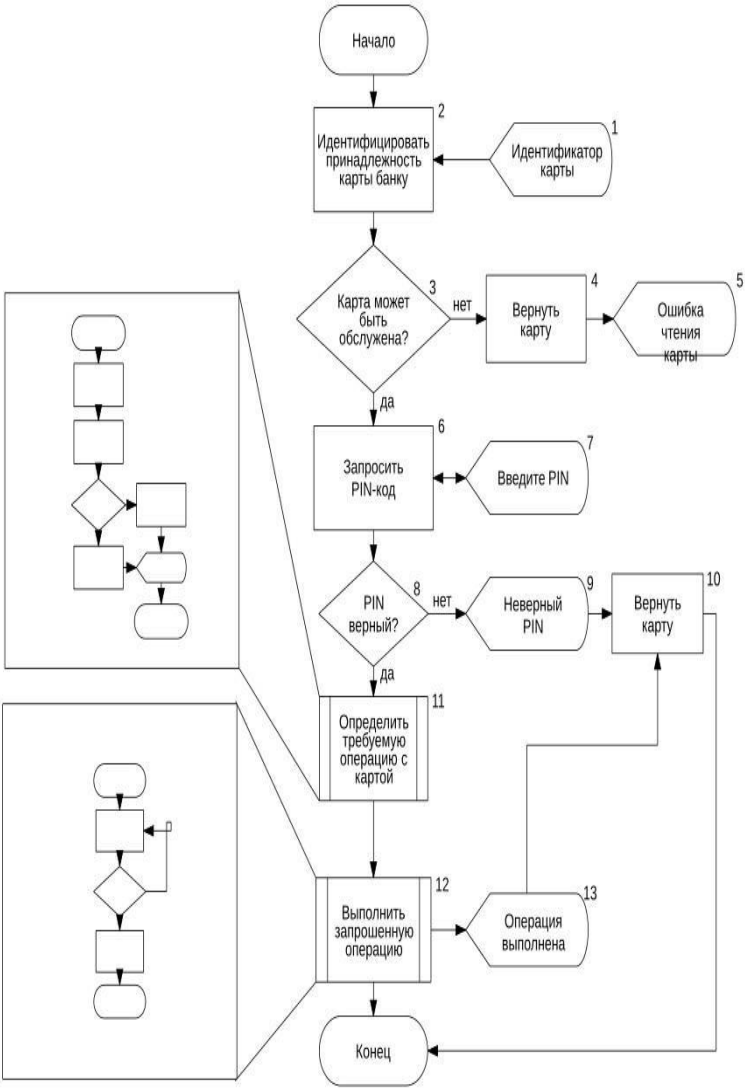


Рис. 1.2 - Программный алгоритм работы банкомата

Представленная схема является обобщенной схемой алгоритма. Но если вместо блоков 11 и 12 вставить их детализацию (набор блоков на рисунке слева), то получится

детализация схемы алгоритма первого уровня. Если, в свою очередь, детализировать один или несколько блоков из детализации блоков 11 или 12 (из детализации первого уровня), то получится детализация второго уровня.

“Погружаясь” таким образом в алгоритм можно получить настолько детализированный его вариант, который будет достаточен для его программной реализации. При этом детализация алгоритма производится до тех пор, пока всё его блоки (операции) не будут определены подробно и однозначно, исключая вариации в их толковании разными участниками проекта и, главное - программистами.

Выводы

Существует много способов описания алгоритмов, отличающихся компактностью, наглядностью, простотой реализации, степенью формализации, ориентацией на машинную реализацию и др. Наибольшее распространение получили способы записи: словесный и графический блок-схемный.

Граф-схемный способ компактен, формализован, имеет хорошую наглядность используемых структур и допускает различные степени детализации алгоритма.

Раздел 2 Требования к программному обеспечению

Стандарт IEEE Std (1990) определяет требования как:

1. Условия или возможности, необходимые пользователю для решения проблем или достижения целей.
2. Условия или возможности, которыми должна обладать система или системные компоненты, чтобы выполнить контракт или удовлетворять стандартам, спецификациям или другим формальным документам.
3. Документированное представление условий или возможностей для первых двух пунктов.

Это определение охватывает требования как пользователей (внешнее поведение системы), так и разработчиков (некоторые скрытые параметры).

Основной закон – требования должны быть документированы.

2.1 Уровни требований

Можно выделить три уровня требований к программному обеспечению (ПО).

1. *Бизнес-требования.* Содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансирует проект, покупатели системы. В этом документе объясняется, почему организации нужна такая система, то есть описаны цели, которые организация намерена достичь с ее помощью. Менеджеры, выступающие заказчиками в рамках проекта, определяют бизнес-требования для ПО, которые помогут их компании работать эффективнее (для информационных систем) или успешно конкурировать на рынке (для коммерческих продуктов). Каждое требование пользователю должно быть сопоставлено бизнес-требованию.
2. *Требования пользователей.* Описывают цели и задачи, которые пользователям позволит решить система. К отличным способам представления этого вида требований относятся варианты использования, сценарии и таблицы «событие-отклик». То есть здесь указано, что клиенты смогут

делать с помощью системы. На основе требований пользователя аналитики определяют функции, которые разрешат пользователям выполнять их задачи.

Пример варианта использования на UML-диаграмме. «Сделать заказ» для аренды автомобиля или заказа гостиницы через интернет.

3. *Функциональные требования.* Определяют функциональность ПО, которую разработчики должны построить, чтобы пользователи смогли выполнить свои задачи в рамках бизнес-требований. Иногда их называют требованиями поведения, они содержат положения с пограничными словами «должен/должна».

Функциональные требования документируются в спецификации требований к ПО, где описывается ожидаемое поведение системы.

Спецификация требований к ПО используется при разработке, тестировании, гарантии качества продукта, управлении проектом и связанном с проектом функциях.

Спецификация требований не содержит деталей дизайна или реализации (кроме известных ограничений), данных о планировании проекта или сведений о тестировании. Для проекта, как правило, создаются требования других типов: документ, где описана среда разработки, ограничения бюджета, руководство пользователя или требования для выпуска продукта. Все это относится к требованиям к проекту, но не к продукту.

Пример. «Система должна по электронной почте отправлять пользователю подтверждение о заказе».

Также еще каждая система имеет свои нефункциональные требования: бизнес-правила, атрибуты качества, внешний интерфейс, ограничения.

Бизнес-правила включают корпоративные политики, правительственные постановления, промышленные стандарты и вычислительные алгоритмы. Бизнес-правила не

являются требованиями к ПО, т.к. находятся снаружи границ любой системы ПО. Однако они часто налагают ограничения, определяя, кто может выполнять конкретные варианты использования, или диктовать, какими функциями должна обладать система, подчиняющаяся соответствующим правилам. Иногда бизнес-правила становятся источником атрибутов качества, которые реализуются в функциональности.

Атрибуты качества представляют собой дополнительное описание функций продукта, выраженное через описание его характеристик, важных для пользователей или разработчиков. К таким характеристикам относятся легкость и простота использования, легкость перемещения, целостность, эффективность и устойчивость к сбоям.

Другие нефункциональные требования описывают внешние взаимодействия между системой и внешним миром, а также ограничения дизайна и реализации. Ограничения касаются выбора возможности разработки внешнего вида и структуры продукта.

Разработчикам необходимы функциональные и нефункциональные требования, чтобы создавать решения с желаемой функциональностью, определенным качеством и требуемыми рабочими характеристиками, не выходя за рамки налагаемых ограничений.

2.2 Разработка и управление требованиями

Область разработки технических условий можно разделить на разработку требований и управление требованиями.

В свою очередь, этап разработки требований включает в себя подэтапы: извлечение, анализ, документирование, проверка (утверждение).

В эти подэтапы входят все действия, включающие сбор, оценку и документирование требований для ПО (ПО-содержащих продуктов), в том числе:

1. Идентификация классов пользователей для данного продукта.

2. Выяснение потребностей тех, кто представляет каждый класс пользователей, определение целей и задач пользователей.

3. Анализ информации, полученной от пользователей, чтобы отделить задачи от функциональных и нефункциональных требований, бизнес-правил, предполагаемых решений и поступающих извне данных.

4. Установление относительной важности атрибутов качества.

5. Документирование собранной информации и построение моделей.

6. Просмотр спецификации требований, который позволяет удостовериться в том, что запросы пользователей всеми понимаются одинаково, и устранение возникших проблем до передачи документа разработчикам.

Этап управления требованиями определяется как «выработка и поддержание взаимного согласия с заказчиками по поводу требований к разрабатываемому ПО». Это соглашение воплощается в спецификации (в письменной форме) и моделях.

К действиям по управлению требованиями относятся:

1. Определение основной версии требований (моментальный срез) требований для конкретной версии продукта.

2. Согласование плана проекта с требованиями.

3. Обсуждение новых обязательств, основанных на оцененном влиянии изменения требований.

4. Отслеживание отдельных требований до их дизайна, исходного кода и вариантов тестирования.

5. Отслеживание статуса требований и действий по изменению на протяжении всего проекта.

Выводы

Разработчики программного обеспечения прикладывают значительные усилия для сбора и документирования требований к ПО, а также управления ими [2].

Недостаточный объем информации, поступающей от пользователей, требования, сформулированные не полностью, их кардинальное изменение – вот основные причины, из-за которых зачастую командам, работающим в области информационных технологий, не удается вовремя и в рамках бюджета предоставить клиентам всю запланированную функциональность. Многие разработчики не умеют профессионально собирать требования пользователей к ПО.

Из всех возможных способов совершенствования процесса разработки ПО наибольшее преимущество за формулированием требований. Потратив на это достаточно времени, вы можете быть уверены, что свели к минимуму риск переделки продукта, создания плохого ПО или срыва сроков сдачи проекта.

Чтобы управлять границами требований, для начала уточните бизнес-цели проекта, основы его образа, рамки, ограничения, критерии успеха и ожидаемую пользу. Оцените, как предполагаемые характеристики или изменения требований отразятся на связанной с ними структуре. Эффективный процесс модификации заставляет интенсивнее работать аналитиков, которые помогут всем заинтересованным лицам принять обдуманные бизнес-решения относительно того, какие изменения следует принять, и увязать их стоимость со временем, ресурсами.

Раздел 3 Варианты реализации программного обеспечения проекта

В данной главе будут рассмотрены три варианта реализации программного обеспечения проекта: в виде desktop-приложения, в виде web-приложения, в виде мобильного приложения.

Любое приложение (любая программа, не зависимо от того является ли она desktop, мобильным или веб-приложением) представляет собой файл или набор файлов, содержащих программный код (последовательность действий), понятный операционной системе, для которой данное приложение разрабатывалось.

Операционная система является “посредником” между программами, которые запускает пользователь, и оборудованием, на котором он это делает (процессор, оперативная память, дисковое пространство компьютера или мобильного устройства). Поэтому когда пользователь запускает приложение, операционная система открывает файлы этого приложения и “читает” из них инструкции, то есть действия, которые она должна выполнить и выполняет их.

Пример. Когда запускается браузер, то система «отрисовывает» окно пользовательского интерфейса данного браузера (вид которого может отличаться, в зависимости от производителя и версии браузера), посылая при этом инструкции процессору и видеокарте компьютера через общую шину данных. Также система делает HTTP-запрос к тому сайту, который должен открыться в браузере по умолчанию (если он был задан), посылая при этом инструкции сетевой карте компьютера и т.д.

Таким образом, запущенная программа посредством своих инструкций операционной системе (системных вызовов) выполняет тот функционал, который в ней заложен её разработчиками, обеспечивая этим функционалом пользователя. Соответственно, если программа запускается

на операционной системе, для которой она не предназначена (например, программа написана для Linux, а запускается в Windows), то операционная система (ОС), прочитав инструкции из файлов данной программы, просто их не “поймет” и сгенерирует ошибку, при этом программа не будет запущена.

Тем не менее, независимо от операционной системы (т.е. программной платформы) и сложности приложения, оно всегда представлено в ОС в виде файлов с программным кодом. Если приложение простое, то это может быть один файл. Если оно сложное, то оно может состоять из исполняемого файла (того, который запускает пользователь) и дополнительных модулей.

Пример. В ОС Windows исполняемые файлы имеют расширение *.exe, дополнительные модули - это файлы с расширением *.dll).

Структура приложения определяется её разработчиками на этапе проектирования приложения и зависит от множества факторов: от ОС, в которой будет работать приложение; от того, где физически будут расположены файлы приложения (на одном дисковом устройстве или на разных); от характеристик того устройства, на котором будет работать приложение; от того, насколько часто требуется модификация программного кода приложения и др.

Если для проектирования приложения используется язык моделирования UML, то структура приложения определяется диаграммой компонентов, на которой отображаются основные модули (компоненты и/или файлы) будущей программы.

Рассмотрим особенности различных реализаций программного обеспечения проекта, сравнивая их по нескольким критериям: варианты размещения, достоинства, недостатки, безопасность, области применения реализации, пример приложения.

1. Desktop-приложение представляет собой программу, установленную на жестком диске персонального

компьютера, работающего под управлением определенной ОС, имеющую оконный (или многооконный) пользовательский интерфейс и используемую пользователем данного компьютера монопольно.

1.1 Размещение приложения

Desktop-приложение размещается на одном устройстве (компьютере). Даже если приложение является корпоративным, то есть используется на множестве компьютеров одной и той же организации для решения одних и тех же задач, но разными пользователями, оно устанавливается на каждый из таких компьютеров и настраивается отдельно для конкретного пользователя (владельца данного компьютера).

Приложение условно разделено на три компонента – пользовательский интерфейс (который может быть графическим или консольным), исполняемые файлы приложения и хранилище данных (база данных). Несмотря на то, что приложение имеет несколько компонент, все они размещены и функционируют на одном и том же устройстве (компьютере).

1.2 Достоинства

Достоинствами desktop-варианта реализации приложения являются следующие:

– высокая скорость работы и отклика на действия пользователя;

– высокая безопасность обрабатываемой приложением информации;

– простота разграничения доступа;

– сравнительно простая интеграция с другими приложениями, работающими на том же устройстве;

– более простая интеграция с периферийными устройствами (принтерами, сканерами и т.п.);

– зачастую не требует наличия подключения к сети передачи данных.

1.3 Недостатки

К недостаткам desktop-варианта реализации приложения можно отнести следующие:

– сложность установки и обновления (необходимо устанавливать/обновлять на каждом компьютере);

– отсутствие возможности удаленной работы – если нет доступа к компьютеру, то приложение и все его данные будут недоступны;

– возможные проблемы с резервным копированием данных, так как либо они должны храниться на том же компьютере (при поломке которого они могут быть утеряны), либо переноситься в удаленное (внешнее) хранилище данных (что требует больше ресурсов и времени);

– накладные расходы (время, ресурсы, большая вероятность ошибок) в рамках синхронизации данных, если приложение имеет клиент-серверную архитектуру;

– сложности тестирования приложения при разработке, так как можно привлечь для тестирования только небольшое число пользователей;

– зависимо от платформы (например, приложение, разработанное для Windows, не будет корректно работать в Linux);

– под каждую платформу зачастую необходимо писать свою версию приложения;

– сложно отслеживать ошибки.

1.4 Безопасность

На сегодняшний день desktop-приложения являются наиболее безопасными, так как функционируют на конкретном устройстве, под управлением, чаще всего, какой-нибудь популярной ОС (Windows, Mac OS), имеющей встроенные средства защиты от проникновения из Сети или от взлома учетной записи пользователя на компьютере. Также серьезная антивирусная защита в виде различного рода коммерческих и бесплатных антивирусов существует для персонального компьютера в гораздо большем количестве, нежели для других платформ.

1.5 Области применения

Desktop-приложения чаще всего используют в своей работе корпоративные или промышленные предприятия. Такие приложения работают в защищенной от внешних угроз корпоративной сети предприятия, обеспечивая взаимодействия пользователей и надежную защиту данных.

1.6 Пример приложения

Приложение (программа), рассматриваемое в данном примере, разработано на языке Delphi для ОС семейства Windows и является приложением для работы с базой данных (БД) абитуриентов. Приложение позволяет осуществлять добавление и редактирование данных об абитуриентах, выставление оценок (сохранение оценок в БД), экспорт отчетов в файлы MS Excel. Для работы с БД данное приложение использует систему управления БД (СУБД) Paradox.

Проект данного приложения имеет следующую структуру файлов (рис. 3.1).

Файл с расширением .dpr (Project1.dpr) является главным файлом проекта в среде разработки Delphi. Файлы с расширением .cfg и .res являются служебными и содержат служебные данные, необходимые для «сборки» данного проекта в готовое приложение – для компиляции приложения. Файлы с расширением .dfm и .pas являются файлами программных модулей приложения, то есть содержат разные части (компоненты) кода приложения, реализующего соответствующие им функции приложения. Файлы MS Excel (с расширением .xls) необходимы для выгрузки в них отчетов при работе программы.

Папка dbase имеет следующее содержание (рис. 3.2).

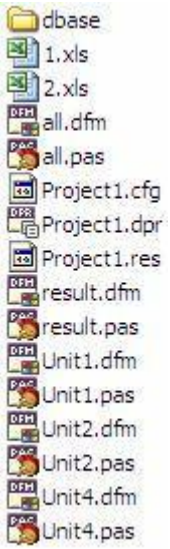


Рис. 3.1

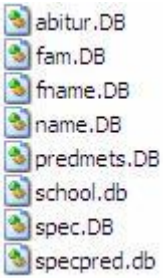


Рис. 3.2

Данные файлы являются файлами (таблицами) БД, созданной в СУБД Paradox специально для данного приложения. По сути, это и есть БД, в которую помещаются данные об абитуриентах и их оценках и из которой эти данные затем извлекаются по запросу пользователей приложения (например, по запросу преподавателей).

Язык программирования Delphi, на котором написано данное приложение является высокоуровневым, то есть при написании программы не требуется описывать программным кодом все компоненты приложения – окна, кнопки, поля для ввода, модули и т.д. Всё это среда разработки делает автоматически. То есть когда мы создаем графическую форму, например Unit1.dfm, среда разработки автоматически создает для неё соответствующий программный модуль (файл, содержащий программный код) – Unit1.pas.

При добавлении на форму разных графических компонентов – кнопок, окон, картинок и т.п. путем перетаскивания их с панели инструментов среды разработки, описывающий их программный код (код, который «говорит»

компьютеру, что в данной части данной графической формы должна быть кнопка стандартного размера, стандартного цвета и т.д.) генерируется средой разработки и автоматически добавляется в модуль Unit1.pas. Когда меняются свойства кнопки с помощью инструментов среды разработки (инспектора объектов), например, размер данной кнопки, то эти изменения также автоматически заносятся в программный модуль формы (Unit1.pas), что избавляет разработчика от необходимости писать код вручную.

Каждый модуль программы (Unit1.pas, Unit2.pas и т.д.) является классом приложения и на UML-диаграмме классов представлял бы сущность (прямоугольную фигуру - класс).

Стоит также сказать, что рассмотренные файлы являются только проектом приложения, то есть содержат программный код и сами по себе приложением не являются. Для того чтобы получить из данного проекта работающее приложение, его необходимо скомпилировать. Разные среды разработки и языки программирования используют разные (собственные) компиляторы. Среда разработки Delphi также имеет свой собственный компилятор, который преобразует программный код из файлов приложения в исполняемый код (в код, понятный ОС).

После компиляции данного проекта из всех перечисленных файлов проекта (модулей приложения) генерируется один единственный файл с расширением .exe, который возможно запускать как обычную программу в операционной системе Windows. При этом программные модули для работы данной программы понадобятся только в том случае, если будет необходимо что-нибудь изменить в программе (ее придется компилировать снова, получая новый исполняемый .exe файл).

2. Web-приложение представляет собой программу, доступную через один или несколько протоколов сети Интернет или локальной сети предприятия, работающую на удаленном компьютере (например, сервере), имеющую многопользовательский интерфейс, представленный в форме

веб-страницы (или множества веб-страниц) для отображения в веб-браузере компьютера.

2.1 Размещение приложения

Компоненты одного и того же веб-приложения могут быть размещены на различных устройствах, работающих под управлением разных ОС и находящихся на значительном удалении друг от друга. Причем какая-то часть веб-приложения выполняется на программном сервере (например, исполняемые файлы – скрипты), какая-то часть на сервере данных (например, компоненты обработки запросов и доступа к данным – СУБД), какая-то на устройстве пользователя (непример, файлы и скрипты пользовательского интерфейса – HTML, CSS, Javascript).

Но также все компоненты веб-приложения могут быть размещены полностью и на одном устройстве.

2.2 Достоинства

К достоинствам веб-приложений можно отнести следующие:

– доступность 24 часа 7 дней в неделю;

– постоянная репликация (резервное копирование) данных;

– проще реализуется кросплатформенность (не зависит от операционной системы, только от браузера);

– простота обновления (обновление программы требуется только на сервере);

– доступно с любого устройства, имеющего доступ в Интернет;

– более тщательное тестирование за счет привлечения большого числа различных пользователей с различными устройствами;

– возможность отслеживать логику действий пользователей, системы, сохранять отчеты и немедленно сообщать о сбоях разработчикам;

– множество готовых наработок (готовых модулей и решений) других разработчиков в свободном доступе,

которые можно использовать для разработки собственного приложения.

2.3 Недостатки

– менее защищено от внешних угроз;
– сложнее в реализации;
– требует доступа в сеть передачи данных;
– сложнее организовать разграничение доступа;
– сложнее интегрировать с периферийными (внешними) устройствами и с локальными приложениями на устройстве пользователя.

2.4 Безопасность

Веб-приложения подвержены множеству угроз, как внешних (из сети Интернет), так и внутренних (с устройства пользователя). Примерами таких угроз могут быть вирусы, которые проникают как на сервер, на котором хранятся файлы приложения, так и на устройство пользователя (например, через его браузер); различного рода сетевые атаки как на само приложение (например: XSS-атаки, SQL-injection, man-in-the-middle и т.п.), так и на сервер, на котором оно функционирует (например: DDoS-атака, переполнение буфера на сервере, IP-спуфинг и т.п.). Поэтому при проектировании и разработке веб-приложения вопросам безопасности следует уделять особое внимание.

2.5 Области применения

Веб-приложение в современном мире используются практически повсеместно. В настоящий момент почти не осталось сервисов, для которых нет варианта его реализации в виде веб-приложения. Также современной тенденцией являются так называемые SaaS-приложения (System as a Service), функционирующие в “облаке” (например, Google Docs, Google Drive, Dropbox и т.п.). Они в последние годы получают все большее развитие и распространение.

2.6 Пример приложения

В качестве примера веб-приложения рассмотрим приложение для построения в браузере иерархического дерева узлов и сохранения его в БД, написанное на скриптовом языке Perl, с использованием языка разметки HTML, языка стилей CSS и языка веб-сценариев Javascript. Для реализации БД в данном приложении используется СУБД MySQL.

Состав файлов приложения выглядит следующим образом (рис. 3.3). Папка lib содержит следующие модули программы, то есть основной программный код веб-приложения (рис. 3.4).

Модуль Node.pm содержит код для осуществления действий с узлами дерева: добавление, удаление узлов дерева. Модуль TreeManager.pm отвечает за роутинг, то есть он сопоставляет действия пользователя с деревом на веб-странице в браузере с программным кодом, который эти действия выполняет. Папка TreeManager содержит папку Controller, в которой находится модуль Tree.pm. Данный файл (Tree.pm) является главным модулем проекта (файлом-контроллером) – именно он определяет логику работы программы в зависимости от действий пользователя на веб-странице в браузере.

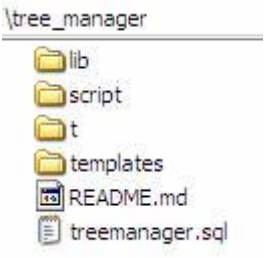


Рис. 3.3

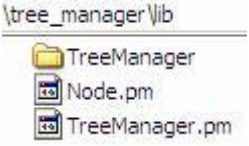


Рис. 3.4

Папка script содержит файл tree_manager, который является «точкой входа» приложения – именно он запускает приложение и указывает системе на файл TreeManager.pm,

осуществляющий связь действий пользователя с функциями веб-приложения.

Папка templates содержит шаблоны веб-страниц, которые пользователь приложения видит в веб-браузере. В ней находится две папки: папка layouts, содержащая файл HTML-шаблон index.html.ep; и папка tree, содержащая файл HTML-шаблон build_tree.html.ep. Из данных файлов в процессе работы приложения генерируются обычные веб-страницы (HTML-страницы), отображаемые пользователю в браузере.

При открытии приложения в браузере веб-сервер, осуществляющий связь браузера с программами на сервере обращается к файлу /script/tree_manager, который, по аналогии с desktop-приложением, можно назвать исполняемым файлом приложения.

Если построить UML-диаграмму классов для данного приложения, то в роли классов в данном случае выступали бы модули приложения, например: Tree.pm, Node.pm и т.д. При этом файл Tree.pm изображался бы на диаграмме классов фигурой контроллера.

3. Мобильное приложение чаще всего представляет собой, по сути, web-приложение, работающее на мобильном устройстве под управлением определенной ОС и использующее для своей работы мобильные сети связи (Wi-Fi, 3G, LTE и т.п.). Также бывают мобильные приложения, не требующие для своей работы связи с другими устройствами и работающие автономно, подобно desktop-приложениям, на конкретном мобильном устройстве для конкретного пользователя (например, некоторые мобильные игры).

3.1 Размещение приложения

Размещение мобильного приложения очень похоже на размещение веб-приложения, так как мобильное приложение является подклассом последнего. Единственным отличием является бо́льшая по сравнению с классическим веб-приложением мобильность за счет использования мобильных устройств в качестве программной платформы и беспроводных сетей передачи данных (Wi-Fi, 3G, LTE).

3.2 Достоинства

Аналогичны достоинствам веб-приложений.

3.3 Недостатки

Аналогичны недостаткам веб-приложений.

3.4 Безопасность

Мобильные приложения являются более уязвимыми по сравнению с классическими веб-приложениями, так как пользователи мобильных устройств практически не пользуются антивирусами, а мобильные операционные системы не имеют таких же мощных встроенных средств защиты (firewall'ов), как их desktop-аналоги.

Также при получении доступа к мобильному приложению злоумышленник может использовать его для взлома других приложений мобильного устройства и кражи личных данных (номеров телефонов, фото, фалов с важными данными, реквизитов доступа к другим приложениям устройства – мобильному банку, мобильному кошельку и т.п.).

3.5 Области применения

Мобильные приложения имеют сегодня гораздо большее распространение и популярность у обычных пользователей, чем desktop и веб-приложения, и используют их, в основном, в личных целях. Организации и предприятия используют в своей деятельности мобильные варианты доступа к корпоративным данным с большей осторожностью, предпочитая им более классические варианты – desktop и веб-приложения.

3.6 Пример приложения

Рассмотрим пример мобильного варианта приложения для мобильных устройств, работающих под управлением ОС Android. В качестве примера разберем вариант реализации японской игры «Судоку».

Главный каталог приложения содержит папки gen, res, src и файл AndroidManifest.xml.

Папка gen, в свою очередь, содержит папку org, в которой находится папка example. Последняя папка содержит папку sudoku, в которой находится автоматически созданный средой программирования, в которой разрабатывалась данное приложение (Eclipse), файл R.java.

В папке res главного каталога приложения находятся следующие папки: anim, drawable, layout, layout-land и др.

Папка src содержит папку org, содержащую папку example, в которой находится папка sudoku, содержащая в себе файлы основного программного кода приложения – About.java, Game.java, Keypad.java, Music.java, Options.java, Prefs.java, PuzzleView.java и Sudoku.java.

Игровое меню приложения описано в файле /res/layout/main.xml.

Описание пользовательского интерфейса можно менять путем редактирования XML кода в указанном файле или с помощью среды разработки (в данном случае - Eclipse).

За игровую логику отвечает класс /src/org/example/sudoku/Game.java. Именно программный код этого файла (класса) и загружает задания и проверяет условия выигрыша.

Выводы

Очевидно, что приложения могут работать под разными платформами (операционными системами), для которых они разрабатывались. Причем одни файлы могут выполняться на сервере (как, например, модули рассмотренного веб-приложения), другие на пользовательском компьютере (как, например, рассмотренное desktop- и мобильное приложение, а также и отдельные файлы шаблонов (html, css, javascript) веб-приложения). Однако, чаще всего, приложение представляет собой файл или несколько файлов, содержащих программный код на каком-либо языке программирования, понятный ОС, на которой она выполняется. При этом состав файлов и их тип чаще всего зависят от того, для какой ОС разрабатывалось приложение, на каком языке программирования оно было разработано, а также (иногда) от того, в какой среде разработки был написан код приложения.

Раздел 4 Обзор языков программирования

Чаще всего при проектировании информационных программных систем требуется заранее определить язык разработки. Это требуется для того чтобы можно было определить состав исполнителей проекта (программистов), сроки разработки, затраты и т.д. Иногда язык программирования (ЯП), на котором будет реализован проект или отдельные его части, определяет заказчик, исходя из своих предпочтений или ограничений. Иногда это делает исполнитель проекта, исходя из своего опыта или ограничений. Тем не менее, ЯП должен быть определен до программной реализации проекта, и данное решение должно быть согласовано между исполнителем и заказчиком и утверждено обеими сторонами.

Удобнее и правильнее всего делать это на этапе разработки технического задания проекта. Техническое задание обычно разрабатывают одновременно несколько специалистов – как со стороны исполнителя, так и со стороны заказчика. Поэтому проектировщику также важно знать и понимать отличия наиболее популярных ЯП и области их применения.

В настоящий момент существует большое множество различного рода ЯП, созданных для решения как общих, так и более конкретных практических задач. Существует несколько категорий ЯП, среди которых можно выделить:

- ЯП, относящиеся к процедурному программированию,
- объектно-ориентированные ЯП,
- мультипарадигмальные ЯП,
- ЯП для работы с базами данных.

Процедурное программирование является подходом к написанию программного кода, при котором программный код представляется в виде подпрограмм, вызываемых другими программами или подпрограммами с помощью встроенных механизмов самого ЯП.

В *объектно-ориентированных* ЯП используется понятие классов и объектов, то есть программный код представляется

в виде описания классов (объектов – структур данных), их свойств и методов (операций над ними). Особенностью такого подхода является то, что без создания объекта с помощью специального метода такого объекта доступ к его свойствам и методам невозможен.

Мультипарадигмальные ЯП позволяют использовать сразу несколько «неродственных» подходов к написанию программ. Например, использовать одновременно процедуры и объекты.

ЯП для работы с базами данных – это специализированная категория ЯП, предназначенных исключительно для создания и манипуляции с данными в базах данных.

Рассмотрим более подробно несколько наиболее популярных ЯП.

*1. Си (**C**)*

Язык программирования Си относится к процедурным ЯП, был разработан в 60-70 годах 20 века сотрудником компании Bell Labs Денисом Ритчи для реализации операционной системы UNIX.

С момента разработки данный ЯП получил широкое распространение и оказал большое влияние на другие ЯП: на Си написаны операционные системы семейства UNIX; также на нём или его аналогах разрабатывают программы для микроконтроллеров, используемых практически в любом бытовом или промышленном электроприборе или устройстве. Множество других ЯП перед тем как выполнять свой программный код кодируют его в код Си, либо используют встроенные в ОС низкоуровневые библиотеки, написанные на Си.

Основным плюсом языка Си является его низкоуровневость, то есть возможность работать напрямую с памятью компьютера, с системными вызовами ОС и т.д. Также элементы кода языка Си можно использовать в других ЯП.

Основным минусом данного ЯП является его относительная сложность и трудоёмкость написания на нём программ.

Также следует отметить, что Си является компилируемым ЯП. То есть, чтобы написанная на нём программа работала, её нужно сначала скомпилировать («собрать») для работы под определенной платформой (например, Windows или Linux), под определенным процессором (32-битным, 64-битным) и т.д. И при изменении какой-то части кода программы (даже незначительном), всю программу необходимо перекомпилировать снова. Но данный аспект даёт возможность максимально повысить эффективность работы программы на конкретной платформе и конкретном процессоре путем использования кода и алгоритмов, которые работают на них лучше всего.

2. *Си плюс-плюс (**C++**)*

Язык C++ является усовершенствованным Си, к которому добавлена возможность объектно-ориентированного программирования. Таким образом, C++ является мультипарадигмальным ЯП (совмещает несколько парадигм, подходов к программированию).

C++ позволяет разрабатывать любое программное обеспечение для любых desktop-платформ. Также он позволяет разрабатывать как консольные приложения, так и приложения, имеющие графический пользовательский интерфейс. Помимо этого, некоторые разработчики используют C++ для написания кода для веб-приложений (например, сайтов).

Основными недостатками C++ является сложность его освоения и серьезность последствий при неправильном его использовании. ЯП C++ предоставляет разработчику широкие возможности, но, вместе с тем, требует большей внимательности и ответственности при написании программного кода. Например, невысвобожденная своевременно разработчиком память компьютера может привести к тому, что компьютер зависнет и прекратит выполнение запущенных на нём приложений. Если

компьютером является сервер, обслуживающий сотни тысяч клиентов, то такой сбой может принести серьезные убытки предприятию.

3. Java

Java - объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённый компанией Oracle). Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине вне зависимости от компьютерной архитектуры.

Достоинством данного ЯП является то, что на нем можно писать платформонезависимые (для любых платформ) приложения, которые будут выполняться в ОС, имеющей установленную виртуальную машину Java (JVM).

Недостатком данного ЯП является меньшая по сравнению с Си или C++ скорость выполнения программ в связи с наличием дополнительных операций конвертации программного кода в байт-код, затем из байт-кода – в код ОС, на которой программа выполняется.

*4. Си-шарп (**C#**)*

C# является мультипарадигмальным ЯП и основан на языках Си и Java. Данный ЯП предназначен для разработки приложений для виртуальной платформы Microsoft .NET Framework, в которой такие приложения и выполняются.

C# наследует достоинства и недостатки ЯП, на которых он основан.

Данный ЯП используется как для разработки desktop, так и для разработки веб-приложений.

Пример. Запрос к БД на языке C#

```
using System;
using System.Data.SqlClient;
using System.Data;
namespace Students
{
class Program
```

```
{
static void Main(string[] args)
{
//Объявляем строковую переменную и записываем в нее строку подключения.
Data Source - имя сервера, по стандарту (local)\SQLEXPRESS
Initial Catalog - имя БД
Integrated Security=параметры безопасности//
string connStr = @"Data Source=(local)\SQLEXPRESS;
Initial Catalog=Test;
Integrated Security=True";
//Здесь указано имя БД для того, чтобы проверить, может БД уже создана. Создаем экземпляр класса SqlConnection по имени conn и передаем конструктору этого класса строку подключения//
SqlConnection conn = new SqlConnection(connStr);
try
{
conn.Open(); //пробуем подключиться//
}
catch (SqlException se)
{
Console.WriteLine("Ошибка подключения:{0}",se.Message);
return;
}
Console.WriteLine("Соединение установлено");
//Создаем экземпляр класса SqlCommand по имени cmdCreateTable и передаем конструктору этого класса, запрос на добавление строки в таблицу Students и объект типа SqlConnection//
SqlCommand cmd = new SqlCommand("Insert into Students" +
"(ID,FIO,Grupa) Values (@ID,@FIO,@Grupa)", conn);
Console.WriteLine("Добавляем запись");
try
{
cmd.ExecuteNonQuery();
}
catch
{
Console.WriteLine("Ошибка при добавлении записи");
return;
}
```

```
cmd = new SqlCommand("Select Teacher.surname_pr, Teacher.name_pr, Teacher.middleName_pr, Students.id_student, Students.id_group, Schedule.disciplina FROM ‘Schedule’ INNER JOIN ‘Students’ ON Students.id_group = Schedule.id_group INNER JOIN Teacher.id_pr = Schedule.id_pr", conn);
```

//Выводим название предмета, ФИО преподавателя и ФИО студентов, занимающихся у данного преподавателя на указанном предмете//

```
conn.Close(); //закрываем соединение//
conn.Dispose();
Console.WriteLine();
```

5. Delphi

Язык Delphi является мультипарадигмальным диалектом разработанного фирмой Apple объектно-ориентированного языка Object Pascal. Данный ЯП используется для разработки desktop-приложений для ОС семейства Windows.

Язык Object Pascal, в свою очередь, основан на языке Pascal, который благодаря своей простоте широко используется в образовательных целях – для обучения основам программирования школьников и студентов.

Достоинством данного ЯП является его относительная простота – ЯП имеет достаточно простой синтаксис, а существующие для него среды разработки предоставляют встроенные возможности для простого создания графического пользовательского интерфейса из готовых графических компонент (окон, кнопок, меню и т.д.), при этом программный код для графических элементов генерируется средой программирования автоматически.

Недостатком данного ЯП является то, что он является некроссплатформенным и недостаточно гибок, по сравнению, например, с языками Си и С++.

6. PHP, Perl и Ruby

Данные ЯП являются мультипарадигмальными скриптовыми (интерпретируемыми) ЯП и используются для разработки веб-приложений (сайтов, программ для серверов и т.д.).

Достоинством данных ЯП является то, что они могут выполняться на любой платформе (являются кроссплатформенными), где установлен соответствующий интерпретатор, генерирующий команды для процессора и ОС из программного кода, написанного на данных ЯП.

Недостатком данных ЯП является сложность в их изучении, «непрозрачность» некоторых процедур, невозможность работать напрямую с памятью компьютера.

Пример. Создание таблицы на языке PHP

```
mysql_connect("mysql.myhost.com", "user", "sesame") or die(mysql_error());
// начинаем с соединения с MySQL-сервером//
mysql_select_db("people") or die(mysql_error());
//используем функцию mysql_select_db для выбора БД "people". Далее создаём таблицу "persons" из 5 столбцов//
mysql_query("CREATE TABLE persons (
id INT AUTO_INCREMENT,
//используем INT для специфицирования того, что столбец содержит числа, а затем добавляем AUTO_INCREMENT для автоматического инкремента этих чисел и гарантирования того, что для каждого ряда будет сгенерирован уникальный ID//
FirstName CHAR,
LastName CHAR,
Phone INT,
BirthDate DATE
PRIMARY KEY(id)
//используем PRIMARY KEY для установки столбца "id" как первичного ключа//
)") Or die(mysql_error());
mysql_close ();
```

7. *Python*

Данный ЯП является кроссплатформенным компилируемым ЯП и в настоящее время широко используется в научных кругах для написания программ по обработке больших массивов данных, их анализа (Data Mining) и т.п.

Достоинством данного ЯП является простота его синтаксиса (похож на обычные предложения на английском языке).

Недостатком является относительно низкое быстродействие.

Пример. Запрос к БД на языке Python

```
import sqlite as db
c = db.connect(database="tvprogram")
cu = c.cursor()
cu.execute("SELECT weekday, wdname FROM wd ORDER BY weekday;")
for i, n in cu.fetchall():
print i, n
```

В результате выполнения запроса получится:

0 Воскресенье
1 Понедельник
2 Вторник
3 Среда
4 Четверг
5 Пятница
6 Суббота
7 Воскресенье

8. *JavaScript*

Является скриптовым ЯП для написания веб-сценариев для браузеров. Является кроссплатформенным, выполняется на стороне клиента (пользователя сайта). Файл с кодом скачивается браузером на компьютер пользователя при открытии веб-страницы и выполняется на его компьютере.

Достоинством являются широкие возможности манипулирования объектами на веб-странице, написанными на HTML и CSS.

Недостатком является сложность в освоении, так как он схож по строгости синтаксиса с языком Си.

Пример. Создание таблицы на языке *JavaScript*

```
conn.query("CREATE TABLE t1 (alpha INTEGER, beta VARCHAR(255), pi FLOAT);");
```

9. *Objective-C и Swift*

Данные ЯП предназначены исключительно для разработки мобильных приложений для платформы iOS.

Недостатком данных ЯП является отсутствие кроссплатформенности.

10. SQL, PL/SQL, Transact-SQL

Данные ЯП являются специализированными и используются исключительно для работы с БД. Позволяют создавать, редактировать БД, манипулировать хранящимися в них данными.

Достоинствами является то, что данные ЯП предоставляют интерфейсы (возможности взаимодействия) для практически любых ЯП, используемых для разработки приложений; как правило имеют хорошо оптимизированные интерпретаторы.

Пример программной реализации БД проекта на ЯП SQL будет рассмотрен далее.

Кроме перечисленных ЯП существуют также специализированные ЯП для математических расчетов, моделирования и т.п., также отличающиеся платформой, скоростью работы, сложностью синтаксиса.

Выводы

Теоретически, любую задачу можно решить на любом из существующих языков программирования. Другое дело, что такая реализация будет иметь особенности и ограничения, присущие выбранному для неё ЯП. Какие-то ЯП лучше справляются с одним типом задач, какие-то ЯП позволяют проще решать другие задачи.

Поэтому для каждой конкретной задачи необходимо выбирать определенный ЯП (или несколько ЯП, если задача разбивается на несколько относительно независимых подзадач), который справится с ней наилучшим образом.

Раздел 5 Программная реализация базы данных проекта в СУБД MySQL

В рамках данного раздела будет рассмотрен пример программной реализации базы данных проекта из ER-модели. В качестве предметной области проекта выбран аэропорт.

Фрагмент ER-модели базы данных аэропорта представлен на рис. 5.1.

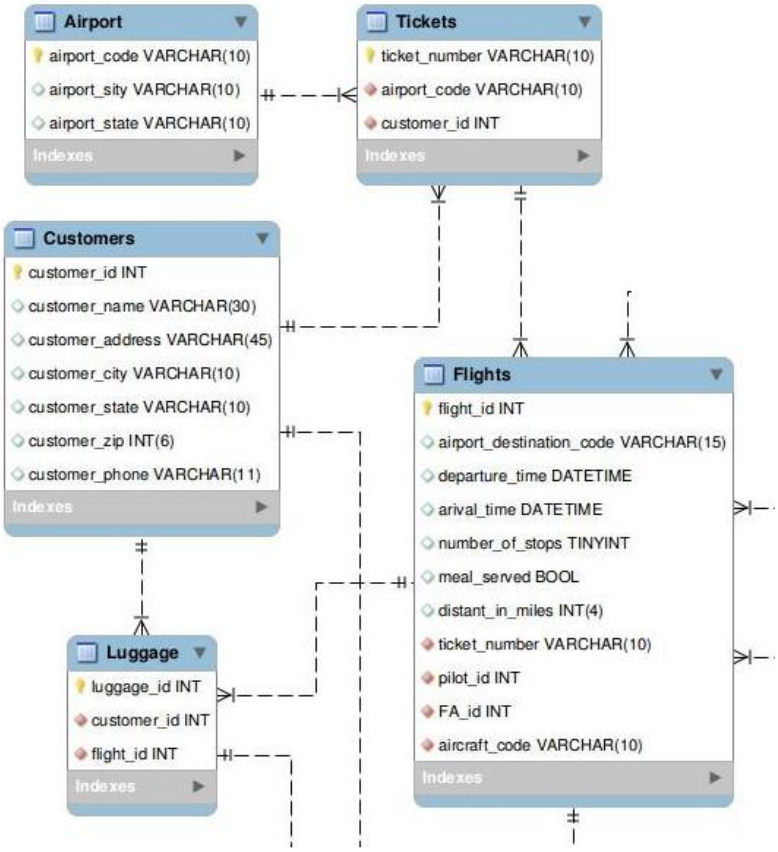


Рис. 5.1 - Фрагмент ER-модели базы данных аэропорта

Данная модель включает следующие сущности (таблицы БД), отражающие объекты реального мира и их свойства:

– таблица рейсов (Flights), содержащая данные о всех рейсах аэропорта;

– таблица воздушных судов (Aircraft) содержащая данные об имеющихся в данном аэропорте самолетах;

– таблица пилотов (Pilots), содержащая сведения о пилотах аэропорта;

– таблица членов экипажа (Flight_Attendents), содержащая сведения о бортпроводниках рейсов;

– таблица билетов (Tickets), содержащая информацию о купленных пассажирами билетах;

– таблица аэропортов (Airport), содержащая информацию об аэропортах, с которыми осуществляется воздушное сообщение;

– таблица клиентов (Customers), хранящая данные о клиентах аэропорта;

– таблицы сведений о багаже (Laggage) и данных, необходимых для обеспечения безопасности в аэропорте, являющиеся связующими таблицами в связях многие-ко-многим между другими таблицами базы данных.

Таблицы рассматриваемой БД связаны между собой связями один-ко-многим и многие-ко-многим (данный тип связи также реализуется через две связи один-ко-многим).

Для программной реализации данной ER-модели БД необходимо написать SQL-запросы, позволяющие создать указанные в модели сущности (таблицы БД), их атрибуты (поля или столбцы таблиц БД) и связи между сущностями. В качестве языка реализации был выбран диалект SQL – язык СУБД MySQL.

Весь программный код (набор SQL-запросов), необходимый для программной реализации рассматриваемой БД, выглядит следующим образом.

```
1| CREATE SCHEMA IF NOT EXISTS `mydb`
DEFAULT CHARACTER SET utf8
2| COLLATE utf8_general_ci ;
```

```
3| SHOW WARNINGS;
4| USE `mydb` ;
5|
```

В строках 1-2 проверяется существует ли БД с именем 'mydb' и, если нет, то такая база создается, при этом для базы выбирается кодировка utf8 и, в часности, диалект (COLLATE) utf8_general_ci данной кодировки. Заданная кодировка указывает обработчику MySQL, обрабатывающему данные в БД и отображающему их пользователям БД по их запросам, в какой кодировке хранить и представлять текст в таблицах БД.

Если, например, для БД задать кодировку utf8, но при этом поместить в какую-нибудь таблицу этой БД текст в кодировке Windows-1251, то такой текст будет нечитаемый при выводе его пользователю, т.е. будет содержать нечитаемые символы. Поэтому в большинстве приложений, например, в веб-сайтах, работающих с MySQL базой данных, все вводимые пользователем на сайте символы – логин, пароль, ФИО и т.д. сразу автоматически кодируют в нужную кодировку для дальнейшего корректного сохранения текста в базе данных.}

```
6| -- -----------------------------------------------------
7|  -- Table `mydb`.`Aircraft`
8|  -- -----------------------------------------------------
```

В строках 6-8 программного кода записан комментарий для разработчиков, которые, возможно, будут сопровождать данную БД и её программный код. Комментарий говорит о том, что далее идет SQL-запрос, создающий таблицу 'Aircraft' в базе данных 'mydb'.}

```
9| DROP TABLE IF EXISTS `mydb`.`Aircraft` ;
10|
11| SHOW WARNINGS;
12| CREATE TABLE IF NOT EXISTS `mydb`.`Aircraft` (
13| `aircraft_code` VARCHAR(10) NOT NULL,
14| `aircraft_description` VARCHAR(45) NULL,
```

15| `first_class_seats` *VARCHAR(45) NULL*,
16| `coach_seats` *VARCHAR(45) NULL*,
17| PRIMARY KEY (`aircraft_code`))
18| ENGINE = InnoDB;
19|

В строках 9-19 происходит непосредственно создание таблицы '*Aircraft*'. При этом сначала проверяется, не существует ли уже данной таблицы (строка 9). Если таблица с таким именем существует, то она удаляется; включаются предупреждения (строка 11), которые будут выведены программисту, если при создании таблицы что-то пойдет не так или таблица не создастся. Непосредственно создание таблицы происходит кодом в строках 12-18. Здесь задается имя создаваемой таблицы (строка 12), имена, типы данных и свойства, длина полей (столбцов) таблицы (строки 13-16), столбец, который будет использоваться в качестве первичного уникального ключа (строка 17) и тип таблицы (строка 18).

В качестве типа данных для столбцов указан *VARCHAR* , т.е. текстовое поле переменной длины с минимальным значением для поля, указанным в скобках (может быть разным для каждого поля). В качестве свойства для всех столбцов, кроме первичного ключа, указано *NULL*. Это означает, что поля могут быть пустыми (не содержать никакого значения), в то время как столбец, который является первичным ключом таблицы, не может иметь пустого значения. Для первичного ключа указано свойство *NOT NULL*, запрещающее ввод и хранение в нем пустого значения. В качестве типа таблицы выбран '*InnoDB*' – данный тип таблиц позволяет в MySQL создавать связи между ними.

20| SHOW WARNINGS; 21|

В строке 20 снова указан оператор, сигнализирующий программисту в том случае, если предыдущий запрос на создание таблицы завершился некорректно.

```
22| -- -----------------------------------------------------
23| -- Table `mydb`.`Airport`
24| -- -----------------------------------------------------
25| DROP TABLE IF EXISTS `mydb`.`Airport` ;
26|
27| SHOW WARNINGS;
28| CREATE TABLE IF NOT EXISTS `mydb`.`Airport` (
29| `airport_code` VARCHAR(10) NOT NULL,
30| `airport_sity` VARCHAR(10) NULL,
31| `airport_state` VARCHAR(10) NULL,
32| PRIMARY KEY (`airport_code`))
33| ENGINE = InnoDB;
34|
35| SHOW WARNINGS;
36|
37| -- -----------------------------------------------------
38| -- Table `mydb`.`Customers`
39| -- -----------------------------------------------------
40| DROP TABLE IF EXISTS `mydb`.`Customers` ;
41|
42| SHOW WARNINGS;
43| CREATE TABLE IF NOT EXISTS `mydb`.`Customers` (
44| `customer_id` INT NOT NULL,
45| `customer_name` VARCHAR(30) NULL,
46| `customer_address` VARCHAR(45) NULL,
47| `customer_city` VARCHAR(10) NULL,
48| `customer_state` VARCHAR(10) NULL,
49| `customer_zip` INT(6) NULL,
50| `customer_phone` VARCHAR(11) NULL,
51| PRIMARY KEY (`customer_id`))
52| ENGINE = InnoDB; 53|
54| SHOW WARNINGS; 55|
```

Аналогичным образом в строках 22-54 указаны SQL-запросы, создающие в базе '*mydb*' таблицы '*Airport*' и '*Customers*'. Единственным отличием является тип данных в поле `customer_zip` таблицы '*Customers*', в котором хранится почтовый индекс клиента. Для данного поля указан тип

INT(6), то есть натуральное число, содержащее максимум 6 знаков, так как почтовые индексы РФ содержат не более 6 цифр.

Стоит отметить, что если аэропорт планируется использовать для международных рейсов, которыми смогут пользоваться граждане и других стран, то данное поле лучше сделать текстовым полем переменной длины (*VARCHAR*), так как почтовые индексы других стран могут содержать другое количество цифр и буквы. Данный момент важно продумать непосредственно на этапе проектирования базы данных (при создании ER-модели), так как впоследствии, когда таблица уже будет заполнена данными, преобразовать тип этого поля из числового в текстовый может быть невозможным.

```
56| -- -----------------------------------------------------
57| -- Table `mydb`.`Tickets`
58| -- -----------------------------------------------------
59| DROP TABLE IF EXISTS `mydb`.`Tickets` ;
60|
61| SHOW WARNINGS;
62| CREATE TABLE IF NOT EXISTS `mydb`.`Tickets` (
63| `ticket_number` VARCHAR(10) NOT NULL,
64| `airport_code` VARCHAR(10) NOT NULL,
65| `customer_id` INT NOT NULL,
66|  PRIMARY KEY (`ticket_number`),
67|  INDEX `fk_Tickets_Airport_idx` (`airport_code` ASC),
68| INDEX `fk_Tickets_Customers1_idx` (`customer_id` ASC),
69| CONSTRAINT `fk_Tickets_Airport`
70|     FOREIGN KEY (`airport_code`)
71| REFERENCES `mydb`.`Airport` (`airport_code`)
72| ON DELETE NO ACTION
73|     ON UPDATE NO ACTION,
74| CONSTRAINT `fk_Tickets_Customers1`
75| FOREIGN KEY (`customer_id`)
76| REFERENCES `mydb`.`Customers` (`customer_id`)
78| ON DELETE NO ACTION
79| ON UPDATE NO ACTION)
80| ENGINE = InnoDB;
```

```
81|
82| SHOW WARNINGS;
```

В строках 56-82 записан SQL-запрос, создающий таблицу '*Tickets*'. Он аналогичен уже рассмотренным запросам, с тем лишь отличием, что содержит операторы, создающие для таблицы индексы (строки 67-68), служащие для ускорения поиска данных в таблице при выполнении запросов, и внешние ключи (строки 69-79), служащие для связи данной таблицы с двумя другими таблицами рассматриваемой базы данных – таблицами '*Airport*' и '*Customers*'.

Оператор создания внешнего ключа содержит следующие директивы: *FOREIGN KEY* (для того, чтобы задать название внешнему ключу); *REFERENCES* (для того, чтобы указать название другой таблицы, с которой осуществляется связь); *ON DELETE* (для того, чтобы задать действие, которое выполняется автоматически над данными в таблице (в таблице '*Tickets*') при удалении связанных с ними данными в связуемой (главной) таблице (в данном случае в таблице '*Airport*')); *ON UPDATE* (для того, чтобы задать действие, которое выполняется автоматически над данными в данной таблице (в таблице '*Tickets*') при обновлении связанных с ними данными в связуемой (главной) таблице (в данном случае в таблице '*Airport*')). В данном случае в обеих директивах указано '*NO ACTION*', что означает что никаких действий с данными в рассматриваемой таблице '*Tickets*' при обновлении/удалении данных в таблице '*Airport*' производить не нужно.

Остальные таблицы и связи между ними, указанные в рассматриваемой ER-модели создаются аналогичным образом.

Выводы

В настоящее время на рынке существует большое количество как баз данных, так и СУБД. СУБД MySQL является достаточно популярным средством для управления БД, что, соответственно, позволяет сделать вывод о большом числе пользователей этой системы. В связи с различным уровнем профессиональных навыков и квалификации пользователей возникает необходимость создания определенных программных ограничений, поддерживающих непротиворечивость данных, хранимых в базе.

Также следует отметить, что MySQL - система с открытым исходным кодом: любой желающий имеет возможность использовать и модифицировать это программное обеспечение по своему усмотрению. Каждый пользователь имеет право получить данное программное обеспечение посредством сети Интернет бесплатно.

Немаловажным является тот факт, что СУБД MySQL является клиент-серверной системой, включающей много поточный SQL-сервер, поддерживающий различные платформы, несколько клиентских программ и библиотек, инструменты администрирования и широкий диапазон программных интерфейсов приложений (API-интерфейсов).

Раздел 6 Программная реализация приложения

В данном разделе рассматривается пример программной реализации простейшего приложения для работы с базой данных (наподобие той, что описана в предыдущем разделе), реализованного на ЯП Delphi в среде разработки Lazarus.

Создайте пустой проект в Lazarus. В папку проекта необходимо скопировать библиотеку DLL для работы с MySQL. Скачать ее можно по следующей ссылке: http://www.freepascal.ru/download/libmysql.zip.

Поместите на главную форму программы компоненты TMySQL50Connection, TSQLTransaction, TSQLQuery из панели SQLbd; и компонент TDatasource из панели Data Access (рис. 6.1).

Рис. 6.1 – Компоненты на главной форме

Настройте связи между этими компонентами следующим образом:

В свойстве Transaction объекта MySQL50Connection1 выберите SQLTransaction1.

В свойстве Database объекта SQLQuery1 выберите MySQL50Connection1

В свойстве Transaction объекта SQLQuery1 выберите SQLTransaction1

В свойстве DataSet объекта Datasource1 выберите SQLQuery1

Для подключения к базе данных необходимо для объекта MySQL50Connection1 указать адрес сервера (свойство HostName), название базы данных (свойство DatabaseName), имя пользователя (свойство UserName) и пароль для доступа к базе данных (свойство Password). Соответственно, в базе

данных должен быть предварительно создан пользователь с именем UserName и паролем Password, имеющий доступ на редактирование к базе данных и таблице exampleTable.

Объекты класса TSQLQuery представляют собой наборы данных. В него загружается копия таблицы с сервера БД, с которой потом и осуществляется вся работа.

TSQLQuery поддерживает два принципиально разных способа доступа к данными: навигационный, который заключается в обработке каждой отдельной (текущей) записи (строки) таблицы; и реляционный, основанный на обработке сразу группы записей, посредствам SQL-запросов.

1. Реляционный способ доступа к данным

SQL-запросы можно разделить на две группы: возвращающие и не возвращающие результат. Например, запрос

Select * from Tickets

подразумевает запись в набор данных копии таблицы Tickets. Для его выполнения нужно использовать следующий код:

```
SQLQuery1.Close;
SQLQuery1.SQL.Clear;
SQLQuery1.SQL.Add(Select * from exampleTable;');
SQLQuery1.Open;
```

При таком подходе сначала закрывается набор данных SQLQuery1 (на случай, если на момент запроса он был открыт), вызывая его метод Close (аналогичного результата можно добиться, установив свойство Active набора SQLQuery1 в false в инспекторе объектов). Если набор закрыт, его связь с базой данных разорвана. Затем необходимо очистить свойство SQL с помощью метода Clear. Далее, с помощью метода Add в набор записывается текст нового запроса. Исполняется запрос при выполнении команды Open, которая переводит набор данных SQLQuery1 в открытый режим и записывает в него результаты выполнения SQL-запроса.

Если SQL-запрос не подразумевает возврата таблиц данных (например, как в запросах на изменение данных:

INSERT и UPDATE), то необходимо воспользоваться альтернативным способом его выполнения:

```
SQLQuery1.Close;
SQLQuery1.SQL.Clear;
SQLQuery1.SQL.Add('
    INSERT INTO exampleTable (text, description , keywords)
    VALUES('+#39+memo3.text+#39+',
'+#39+memo2.text+#39+', '+#39+memo1.text+#39+');
');
SQLQuery1.ExecSQL;
```

В этом случае, при попытке перевести набор данных в открытый режим (выполнить SQLQuery1.Open;) возникнет ошибка. Для ее устранения нужно либо записать в свойство SQL новый запрос, возвращающий таблицу, как показано ниже:

```
SQLQuery1.Close;
SQLQuery1.SQL.Clear;
SQLQuery1.SQL.Add('
    INSERT INTO exampleTable (text, description , keywords)
    VALUES('+#39+memo3.text+#39+',
'+#39+memo2.text+#39+', '+#39+memo1.text+#39+');
');
SQLQuery1.ExecSQL;
SQLQuery1.SQL.Text:='SELECT * from exampleTable;';
SQLQuery1.Open;
```

Либо создать и использовать для таких SQL-запросов отдельный набор данных (SQLQuery2), который никогда не переводить в открытый режим.

2. Подключение и отключение от базы данных в Lazarus

Будем подключаться к удаленной базе данных не сразу, а после получения явной команды от пользователя (щелчка по соответствующей кнопке). Установим в инспекторе объектов свойство Connected компонента TMySQL50Connection и

свойства Active компонентов TSQLTransaction и TSQLQuery в false.

Создадим на форме кнопку «Подключиться». Создадим обработчик нажатия кнопки и напишем в нем следующее:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  isAllOk := true;
  try
    MySQL50Connection1.Connected := true;
  except
    ShowMessage('Ошибка: невозможно подключиться к базе данных');
  exit;
  end;
  try
    SQLTransaction1.Active := true;
  except
    ShowMessage('Ошибка: Создание транзакции невозможно.');
  exit;
  end;
  try
    SQLQuery1.Active := false;
    SQLQuery1.SQL.Clear;
    SQLQuery1.sql.add('
      SET character_set_client='+#39+'utf8'+#39+',
      character_set_connection='+#39+'cp1251'+#39+
      ',character_set_results='+#39+'utf8'+#39+';');
    SQLQuery1.ExecSQL;
    SQLQuery1.SQL.Clear;
    SQLQuery1.sql.add('SELECT * from  exampleTable;');
    SQLQuery1.Open;
  except
    ShowMessage('Ошибка при выполнении SQL запроса.');
  exit;
  end;
```

Таким образом, сначала производится попытка подключения к MySQL базе данных, если она прошла

успешно, создается новая транзакция, после чего при активации объекта SQLQuery1 выполняется SQL-запрос к базе данных, выбирающей все строки из таблицы. Перед тем, как получить таблицу с сервера, необходимо соответствующим образом настроить кодировки. В рассматриваемом примере данные хранятся в кодировке cp1251, поэтому для получения данных в корректной кодировке необходимо выполнить следующий запрос:

```
SET character_set_client="utf8", character_set_connection="cp1251", character_set_results="utf8"
```

При отключении от базы данных необходимо действовать в обратном порядке. Задайте код отключения в обработчике события OnDestroy формы вашего приложения:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
  SQLQuery1.Active := false;
  SQLTransaction1.Commit;
  MySQL50Connection1.Connected := false;
end;
```

3. Навигационный способ доступа к данным

Навигационный способ подразумевает последовательную работу с записями (строками) таблицы, содержащейся в наборе данных. На навигационном способе доступа основана работа визуальных компонент из вкладки «Data Controls».

Разместите на форме компоненты TDBGrid, TDBNavigator и три компонента TDBMemo (рис. 6.2). Установите в их свойствах DataSource ссылку на DataSource1. В свойства DataField объектов DBMemo1, DBMemo2, DBMemo3 запишите description, text и keywords соответственно. (description, text и keywords – это название полей (колонок) в таблице exampleTable) Поскольку набор данных первоначально находится в закрытом состоянии (Action = false), Lazarus будет выдавать предупреждение при попытке ввода названий полей.

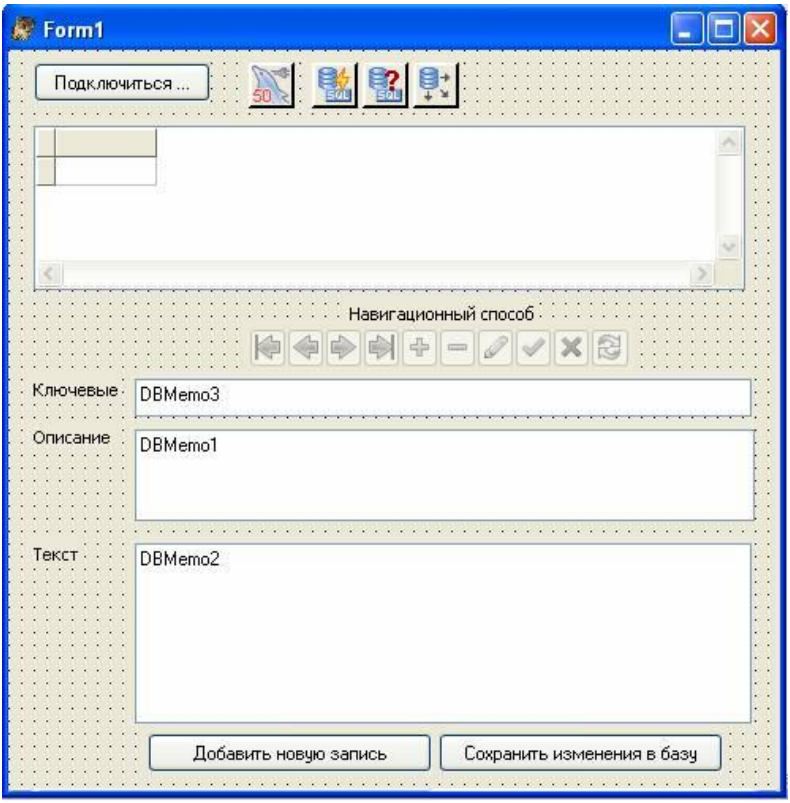


Рис. 6.2 - Навигационный способ доступа к данным

Запустите программу и нажмите кнопу «Подключиться». После подключения к базе данных полученная с сервера таблица будет отображаться в объекте DBGrid1. Стрелкой слева обозначена текущая запись таблицы (курсор). С помощью кнопок панели DBNavigator1 можно перемещаться по строкам таблицы. Вы также можете редактировать записи, однако после закрытия программы все сделанные изменения сбросятся. Для их сохранения в базу необходимо вызвать метод ApplyUpdates:

SQLQuery1.ApplyUpdates;

Ниже представлены методы класса TSQLQuery, предназначенные для навигации по базе:

TSQLQuery.Next – перейти на запись вперед

TSQLQuery.Prior – перейти на запись назад

TSQLQuery.First – перейти на первую запись

TSQLQuery.Last – перейти на последнюю запись

TSQLQuery.Insert – добавить новую запись

TSQLQuery.Delete – удалить запись

TSQLQuery.Edit – перевести набор данных в режим редактирования

TSQLQuery.Post – сохранить сделанные изменения в набор данных

TSQLQuery.Refresh – обновить таблицу.

Для того чтобы получить значения какого-нибудь поля записи, можно воспользоваться методом FieldByName(fn:string), где параметр fn – это название поля (колонки) таблицы. Этот же метод можно использовать для записи новых данных в таблицу. Формат чтения/записи указывается с помощью свойств AsString, AsInteger, AsFloat, AsDataTime и т.д. Например, чтобы записать текущее значение поля id в Label, можно воспользоваться следующим кодом:

```
Label1.caption:=SQLQuery1.FieldByName('id').AsString;
```

Для редактирования текущей строки таблицы используется следующий подход:

```
SQLQuery1.Edit;
SQLQuery1.FieldByName('keywords').AsString:='
Используем Lazarus';
SQLQuery1.FieldByName('text').AsString:='Новый текст';
SQLQuery1.Post;
SQLQuery1.ApplyUpdates;
```

Стоит отметить, что не каждый набор данных можно редактировать навигационным способом. Чтобы получить в результате SQL-запроса редактируемый набор (то есть такой, что внесенные изменения можно было сохранить в базу данных), необходимо выполнение следующих условий:

- данные отбираются только из одной таблицы;
- таблица допускает модификацию;

- в запросе не используется оператор DISTINCT и статические функции;
- в запросе не применяются соединения таблиц;
- в запросе отсутствуют подзапросы и вложенные запросы;
- не используется группирование данных;
- сортировка применяется только к индексным полям;

4. Использование транзакций и TSQLTransaction в Lazarus

Механизм транзакции необходим для сохранения целостности базы данных. Допустим, для внесения актуальных данных в базу вам необходимо записать в разные таблицы большое количество информации. Если в процессе записи произойдет какой-либо сбой (например, выключится электричество), то часть таблиц могут содержать актуальную информацию, а часть неактуальную. В результате чего база данных окажется испорченной. Избежать этого позволяет механизм транзакций. Если все операции записи завершились успешно, транзакция считается успешной и все изменения в таблицах базы считаются подтвержденными и сохраняются. Если же хотя бы одна операция записи не выполнена, транзакция считается неудачной и таблицы базы данных возвращаются к состоянию, которое они имели до начала транзакции.

Для работы с транзакциями в Lazarus добавлен компонент TSQLTransaction. Однако, в настоящее время он не работает с MySQL базами данных. TSQLTransaction автоматически переходит в активное состояние при открытии набора данных (TSQLQuery) и закрывается при его закрытии. Для явного подтверждения успешности транзакции можно использовать метод TSQLTransaction.Commit. Вызывать его следует сразу после метода TSQLQuerty.ApplyUpdates. Для отката транзакции можно использовать метод TSQLTransaction.Rollback.

Раздел 7 Тестирование программного обеспечения, разработанного в рамках проекта

Тестирование (software testing) – деятельность, выполняемая для оценки и улучшения качества программного обеспечения (ПО). Эта деятельность, в общем случае, базируется на обнаружении дефектов и проблем в программных системах. Тестирование программных систем состоит из динамической верификации поведения программ на конечном (ограниченном) наборе тестов (set of test cases), выбранных соответствующим образом из обычно выполняемых действий прикладной области и обеспечивающих проверку соответствия ожидаемому поведению системы.

В соответствии с IEEE Std 829-1983 тестирование – это процесс анализа ПО, направленный на выявление отличий между его реально существующими и требуемыми свойствами и на оценку свойств ПО.

Определение тестирования по SWEBOK (стандарт IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004*)* звучит следующим образом: тестирование – это проверка соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранных определенных образом.

По ГОСТ Р ИСО МЭК 12207-99 в жизненном цикле ПО определены среди прочих вспомогательные процессы верификации, валидации, аттестации, совместного анализа и аудита.

Процесс верификации является процессом определения того, что программные продукты функционируют в полном соответствии с требованиями или условиями, реализованными в предшествующих этапах. Данный процесс может включать анализ, проверку и испытание (тестирование).

Валидация – определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя.

Процесс аттестации является процессом определения полноты соответствия установленных требований, созданной системы или программного продукта их функциональному назначению.

Процесс совместного анализа является процессом оценки состояний и, при необходимости, результатов работ (продуктов) по проекту.

Процесс аудита является процессом определения соответствия требованиям, планам и условиям договора.

В сумме эти процессы и составляют то, что обычно называют тестированием.

Тестирование основывается на тестовых процедурах с конкретными входными данными, начальными условиями и ожидаемым результатом, разработанными для определенной цели (например, проверка отдельной программы или верификация соответствия на определенное требование). Тестовые процедуры могут проверять различные аспекты функционирования программы – от правильной работы отдельной функции до адекватного выполнения бизнес-требований.

Тестирование обычно проводится циклами, каждый из которых имеет конкретный список задач и целей. Цикл тестирования может совпадать с итерацией или соответствовать ее определенной части. Как правило, цикл тестирования проводится для конкретной сборки системы.

7.1 Уровни тестирования

Тестирование обычно производится на протяжении всей разработки и сопровождения на разных уровнях. Уровень тестирования определяет «над чем» производятся тесты: над отдельным модулем, группой модулей или системой в целом. При этом ни один из уровней тестирования не может считаться приоритетным. Важны все уровни тестирования, вне зависимости от используемых моделей и методологий. Требования к системе также тестируют и согласовывают.

Выделяют следующие уровни тестирования:

– модульное тестирование (Unit testing). Этот уровень тестирования позволяет проверить функционирование

отдельно взятого элемента системы. Что считать элементом (модулем) системы определяется контекстом. Наиболее полно данный вид тестов описан в стандарте IEEE 1008-87 «Standard for Software Unit Testing», задающем интегрированную концепцию систематического и документированного подхода к модульному тестированию.

– Интеграционное тестирование (Integration testing). Данный уровень тестирования является процессом проверки взаимодействия между программными компонентами/модулями.

– Системное тестирование (System testing). Системное тестирование охватывает целиком всю систему.

Большинство функциональных сбоев должно быть идентифицировано еще на уровне модульных и интеграционных тестов. В свою очередь, системное тестирование, обычно фокусируется на нефункциональных требованиях – безопасности, производительности, точности, надежности и т.п. На этом уровне также тестируются интерфейсы к внешним приложениям, аппаратному обеспечению, операционной среде и т.д.

7.2 Цели тестирования

Тестирование проводится в соответствии с определенными целями (которые могут быть заданы явно или неявно) и различным уровнем точности. Определение цели точным образом, выражаемым количественно, позволяет обеспечить контроль результатов тестирования. Тестовые сценарии могут разрабатываться как для проверки функциональных требований (известны как функциональные тесты), так и для оценки нефункциональных требований.

При этом, существуют такие тесты, когда количественные параметры и результаты тестов могут лишь опосредованно говорить об удовлетворении целям тестирования (например, «usability» – легкость, простота использования, в большинстве случаев, не может быть явно описана количественными характеристиками).

Можно выделить следующие, наиболее распространенные и обоснованные цели (виды) тестирования:

– установочное тестирование (Installation testing). Из названия следует, что данные тесты проводятся с целью проверки процедуры инсталляции системы в целевом окружении.

– Альфа- и бета-тестирование (Alpha and beta testing). Перед тем, как выпускается ПО, как минимум, оно должно проходить стадии альфа (внутреннее пробное использование) и бета (пробное использование с привлечением отобранных внешних пользователей) версий. Отчеты об ошибках, поступающие от пользователей этих версий продукта, обрабатываются в соответствии с определенными процедурами, включающими подтверждающие тесты (любого уровня), проводимые специалистами группы разработки. Данный вид тестирования не может быть заранее спланирован.

– Функциональные тесты/тесты соответствия (Conformance testing/Functional testing/Correctness testing). Эти тесты могут называться по-разному, однако, их суть заключается в проверке соответствия системы предъявляемым к ней требованиям, описанным на уровне спецификации поведенческих характеристик.

– Достижение и оценка надежности (Reliability achievement and evaluation). Помогая идентифицировать причины сбоев, тестирование подразумевает и повышение надежности программных систем. Случайно генерируемые сценарии тестирования могут применяться для статистической оценки надежности. Обе цели (повышение и оценка надежности) могут достигаться при использовании моделей повышения надежности.

– Регрессионное тестирование (Regression testing). Определение успешности регрессионных тестов (IEEE 610-90 «Standard Glossary of Software Engineering Terminology») гласит: «повторное выборочное тестирование системы или компонент для проверки сделанных модификаций не должно приводить к непредусмотренным эффектам». На практике

это означает, что если система успешно проходила тесты до внесения модификаций, она должна их проходить и после внесения таковых.

– Тестирование производительности (Performance testing). Специализированные тесты проверки удовлетворения специфических требований, предъявляемых к параметрам производительности. Существует особый подвид таких тестов, когда делается попытка достижения количественных пределов, обусловленных характеристиками самой системы и ее операционного окружения.

– Нагрузочное тестирование (Stress testing). Необходимо понимать отличия между рассмотренным выше тестированием производительности с целью достижения ее реальных (достижимых) возможностей производительности и выполнением программной системы с повышением нагрузки, вплоть до достижения запланированных характеристик и далее, с отслеживанием поведения на всем протяжении повышения загрузки системы.

– Сравнительное тестирование (Back-to-back testing). Единичный набор тестов, позволяющих сравнить две версии системы.

– Восстановительные тесты (Recovery testing). Цель – проверка возможностей рестарта системы в случае непредусмотренной катастрофы, влияющей на функционирование операционной среды, в которой выполняется система.

– Конфигурационное тестирование (Configuration testing). В случаях, если ПО создается для использования различными пользователями, данный вид тестирования направлен на проверку поведения и работоспособности системы в различных конфигурациях.

– Тестирование удобства и простоты использования (Usability testing). Цель – проверить, насколько легко конечный пользователь системы может ее освоить, включая не только функциональную составляющую (саму систему), но и ее документацию; насколько эффективно пользователь может выполнять задачи, автоматизация которых

осуществляется с использованием данной системы; насколько хорошо система застрахована (с точки зрения потенциальных сбоев) от ошибок пользователя.

– Разработка, управляемая тестированием (Test-driven development). По-сути, это не столько техника тестирования, сколько стиль организации процесса разработки, жизненного цикла, когда тесты являются неотъемлемой частью требований (и соответствующих спецификаций) вместо того, чтобы рассматриваться независимой деятельностью по проверке удовлетворения требований программной системой.

– Приёмочное тестирование (Acceptance/qualification testing). Проверяет поведение системы на предмет удовлетворения требованиям заказчика. Это возможно в том случае, если заказчик берет на себя ответственность, связанную с проведением таких работ, как сторона «принимающая» программную систему, или специфицированы типовые задачи, успешная проверка (тестирование) которых позволяет говорить об удовлетворении требований заказчика. Такие тесты могут проводиться как с привлечением разработчиков системы, так и без них.

Стандарт *ISO 9126* выделяет также 6 аспектов качества, у каждого из которых еще выделяется некоторое количество подсвойств:

– Функциональность. Это один из основных аспектов качества. И наиболее важным его подсвойством является пригодность к использованию (suitability). То есть программа, во-первых, должна делать то, для чего она предназначена. Во-вторых, она должна делать это правильно (accuracy), то есть она должна вычислять с определенной точностью, она должна правильно конвертировать данные и т.д. В-третьих, программа должна обладать способностью к взаимодействию с другими программами (interoperability), с ОС, с другими версиями той же самой программы, в частности, она должна поддерживать выбранные

разработчиками стандарты (compliance), удовлетворять определенным правилам.

Имеется ещё одна особенность данного стандарта - защищенность или безопасность (security), то есть мало того, что программа должна делать то, для чего она предназначена, она при этом должна не делать ничего другого. Она не должна разрушать пользовательские данные, не должна мешать работать другим программам, не должна предоставлять доступ к данным тем, кому этот доступ не разрешен.

– Надежность. Это второй аспект качества, он достаточно часто неявно включается в какие-то другие аспекты, либо в функциональность, либо в производительность. Но в этом стандарте он выделяется отдельно.

К нему относятся такие подсвойства, как зрелость (maturity), которая является обратной величиной к частоте отказов, и устойчивость к отказам (fault tolerance), то есть способность системы не реагировать на внутренние проблемы. В том числе сюда относится транзакционная целостность и способность к восстановлению работоспособности при отказах (recoverability). То есть если сервер «упал», то он должен самостоятельно восстановиться, «вернуть» все нужные данные, ничего не потерять, и продолжить работу.

– Практичность. Это понятность программы (understandability), то есть пользователь должен понять, как воспользоваться ею для достижения своих целей. Это удобство обучения (learnability) или изучения. В частности, у программы должна быть понятная и удобная документация. Это работоспособность (operability) или управляемость, то есть пользователь должен иметь возможность управлять поведением программы, она должна правильно реагировать на его действия. И привлекательность (attractiveness), эстетическая привлекательность, что также немаловажно.

– Эффективность, Она же – производительность, четвертый аспект качества. Сюда относятся временные

характеристики (time behaviour): время отклика, скорость работы программы, скорость обработки определенных данных и др. И использование ресурсов (resource utilisation): использование дисковых ресурсов, использование ресурсов процессора, использование оперативной памяти, использование сетевых ресурсов.

– Сопровождаемость. Этот аспект качества в большей степени не внешний, а внутренний. Он важен не столько конечным пользователям, не столько потребителям программы, сколько самим разработчикам и ее тестировщикам. Является единственным аспектом качества, который плохо совместим с тестированием. Все его подсвойства тестированию практически не поддаются, и для них применяются, как правило, аналитические способы контроля качества. Статический анализ кода, анализ документации и так далее – все это не решается средствами тестирования. Подразумевает анализируемость кода (analyzability), изменяемость (changeability), то есть удобство внесения изменений в программный код, риск возникновения неожиданных эффектов после того, как изменения внесены (stability), а также контролируемость (testability) (удобство тестирования программы).

– Переносимость. Программа должна уметь работать в различных окружениях (adaptibility), она должна быть достаточно проста в установке (installability), она должна работать одновременно с другими программами и не мешать им (coexistence), и другие программы не должны влиять на работу тестируемой программы. И, наконец, хорошая программа должна предоставлять возможность пользователю перейти с какого-то другого аналогичного программного обеспечения на использование этой программы – замена другого ПО данным (replaceability). То есть предоставить какие-то возможности по миграции, чаще всего по миграции данных.

Соответственно, если исходить из описанной классификации, то для каждого аспекта качества можно выделить соответствующий вид тестирования.

В этой связи получаем разные виды тестирования:

- тестирование функциональности,
- тестирование надежности,
- тестирование эффективности,
- тестирование практичности,
- тестирование сопровождаемости,
- тестирование переносимости.

7.3 Автоматизация тестирования

Автоматизация тестирования – это процесс написания компьютерной программы в виде скриптов для тестирования, который обычно делается вручную. Некоторыми популярными средствами автоматизации являются Winrunner, Quick Test Professional (QTP), LoadRunner, SilkTest, Rational Robot, и др. Средства автоматизации также включает в себя сервисные инструменты, такие как TestDirector и многие другие.

Авто-тесты целесообразно писать только при стабильной функциональности. Если ПО изменяется, авто-тесты писать более затратно, чем тестировать в ручном режиме.

7.4 Примеры последствий ошибок тестирования

Тестирование ПО является важным и завершающим этапом основной части процесса разработки ПО (аналитика → разработка → тестирование), поэтому если в разрабатываемом ПО есть какие-либо несоответствия требованиям или ошибки, то они должны выявиться именно на этом этапе. Если этого не произойдет и ошибки проявят себя в процессе промышленной эксплуатации ПО, то это может привести к серьезным, а иногда и катастрофическим последствиям.

Рассмотрим несколько примеров подобного рода последствий, которые имели место в реальной жизни.

2005 год, вредоносная защита от копирования на дисках Sony. В 2005 году музыкальная компания Sony BGM ввела на некоторых своих аудиодисках новую защиту от копирования. Защита срабатывала, когда диск проигрывали в операционной системе Windows – в систему внедрялся руткит (rootkit), который позволял взломать операционную систему и на системном уровне запрещать копирование информации с аудиодиска, что помогло бы снизить уровень «пиратства» для их продукции. Обычно руткиты используются компьютерными вирусами или троянами для встраивания своего функционала или изменения настроек операционной системы и являются трудноуловимыми. При этом данный руткит был намеренно реализован так, чтобы скрываться от пользователя и операционной системы и, в конечном счете, был классифицирован компаниями, занимающимися компьютерной безопасностью, как вредоносное ПО. Ситуация дошла до судебного разбирательства, в результате которого данные аудиодиски были отозваны компанией Sony. Несмотря на это, репутация компании серьезно пострадала не только в глазах компаний по кибербезопасности, но и в глазах покупателей продукции компании, которые были ущемлены в своих правах попытками компании управлять их частными компьютерами без их ведома и согласия.

2008 год, пятый терминал аэропорта Хитроу. В новом на тот момент терминале аэропорта Хитроу установили сверхсовременную систему контроля и управления багажом, которая должна была контролировать и регулировать перемещение огромного количества багажа и грузов внутри аэропорта. Перед внедрением система была протестирована с более чем 12 тысяч сумок и чемоданов. Однако в первый же день работы системы в аэропорте потерялось 42 тысячи чемоданов, из-за чего было отменено более 500 авиарейсов и временно приостановлена посадка на остальные рейсы. Причиной стало то, что при тестировании не были проверены некоторые реальные сценарии использования данной системы. Например, когда пассажир внезапно снимал багаж с

транспортировочной багажной ленты, вспомнив, что оставил в нем какую-то необходимую ему вещь, система, не имея инструкций на данный случай, не знала как его обработать и выключалась.

1993 год, Pentium и ошибка точности округления чисел с плавающей точкой. Новый Pentium-чип компании Intel периодически допускал ошибки при делении чисел с плавающей точкой из определенного диапазона. К примеру, в результате деления числа *4195835.0* на число *3145727.0* получалось *1.33374* вместо *1.33382*, то есть ошибка составляла *0.006%*. Для обычного домашнего использования данная ошибка могла и не привести к серьезным последствиям, но для коммерческих и, тем более, высоконагруженных систем это серьезный риск появления критических сбоев. В результате, компания была вынуждена заменить 5 миллионов дефектных чипов, понеся финансовые потери в размере 475 миллионов долларов США. Также данный инцидент нанес серьезный урон репутации компании.

2003 год, отсутствие электричества в Северной Америке. В 2003 году на северо-востоке США и в Онтарио, Канада произошло аварийное отключение электроэнергии, которое повлияло на 55 миллионов человек и стало одной из самых крупных за всю историю аварий в подобного рода энергосистемах. Авария началась в тот момент, когда электростанция на Южном берегу реки Эри в штате Огайо прекратила работу из-за слишком высокого энергопотребления, что повлекло за собой увеличение нагрузки на оставшуюся сеть. В результате повышенной нагрузки линии электропередач были перегружены, и из-за нагревания произошло тепловое расширение проводов. Несколько таких линий провисли настолько сильно, что задели деревья, и произошло короткое замыкание, в результате которого нагрузка на всю сеть возросла еще больше. Все это вызвало каскадный эффект, который привел к снижению мощности всей энергосистемы на 80%. Сама причина аварии в данном случае не связана напрямую с

ошибкой в программном обеспечении, но её можно было предотвратить, если бы не ошибка в программе, которая отвечает за оповещение о перегрузке в центре управления энергосистемами. Ошибка в ПО произошла по причине того, что две части данной системы оповещения попали в состояние «конкуренции» за один и тот же программный ресурс и никак не могли разрешить этот логический конфликт (ошибка проектирования многопоточной программной системы, известная в программировании как «состояние гонки» – англ. race condition), в результате чего система оповещения зависла и перестала мониторить аварийные ситуации. По той же причине система оповещения не смогла оповестить работников энергоцентра и о своей поломке. В результате данной аварии большие территории, десятки миллионов людей остались на несколько дней без электричества. Это серьезно повлияло на промышленность, связь, коммунальные услуги в регионе.

1996 год, ракета Ariane-5. Беспилотная ракета Ariane-5, предназначенная для вывода на околоземную орбиту четырех научных спутников для изучения взаимодействия магнитного поля Земли с солнечными ветрами, взорвалась вскоре после запуска из-за ошибки в работе бортового компьютера. Программный модуль, унаследованный от предыдущей версии ракеты, не был достаточно хорошо протестирован в новом аппаратно-программном окружении новой версии ракеты и во время взлета ракеты попытался произвести вычисления для выравнивания сопл твердотопливного ускорителя ракеты точно так, как это производилось для Ariane-4, но вычисляемые значения для Ariane-5 оказались значительно больше, в связи с чем возникла ошибка: программа попыталась сконвертировать 64-битное вещественное число в 16-битное знаковое целое, что вызвало арифметическое переполнение последнего. Механизм обработки ошибок также не был адаптирован под новую ракету, и ошибка не была правильно обработана. Это привело к разрушению ракеты и всего полезного груза на высоте около 4 тысяч метров. Потери составили около 500

миллионов долларов США (стоимость полезной нагрузки – спутников) и 8 миллиардов долларов США затрат на разработку самой ракеты.

1998 год, миссия Mars Climate Orbiter. На 286-е сутки полета спутник-транслятор Mars Climate Orbiter начал переход на высокоэллиптическую орбиту Марса. Аппарат включил двигатели на торможение, но пролетел мимо целевой орбиты и был утерян. Это произошло из-за того что программное обеспечение, контролирующее тягу двигателей, использовало фунт-силу вместо ньютон в качестве единицы измерения, тогда как команды для тяги двигателей предполагали именно ньютоны. Ущерб составил 125 миллионов долларов США.

1969 год, высадка человека на Луну (миссия «Аполлон-11»). Данный факт малоизвестен, но при определенных обстоятельствах высадка человека на Луну могла закончиться катастрофой. Космическая миссия «Аполлон-11», целью которой был а высадка человека на Луну, могла не завершиться из-за особенностей вычислительных алгоритмов компьютерной программы посадочного лунного модуля. Данный софт на протяжении семи лет писало около 300 человек. Суть ошибки была в том, что программа нацеливала лунный модуль в точку над местом посадки. Однако, поскольку для расчета траектории использовались полиномы высоких степеней, траектория, выходя в точку прицеливания, могла оказаться под поверхностью Луны, то есть посчитать, что поверхность Луны дальше, чем она есть на самом деле. Потому что полином высокой степени мог «прыгнуть» в сторону. Ирония с этим багом состоит в том, что его никак не исправили. Программа не отслеживала потенциально опасные кривые. Баг не проявился в реальных посадках, но его мог вызвать не такой уж невозможный случай. Если бы лунный модуль немного сбился с курса и оказался над неучтенным достаточно глубоким кратером, компьютер, получая данные от посадочного радара, мог бы подумать, что оказался выше траектории, пересчитать

кривую на более крутую и направить лунный модуль к прицельной точке, нырнув сначала в Луну.

Проблемы 2000 и 2038 года. Проблема 2000 года – это самый известный компьютерный баг. Во многих компьютерных системах для обозначения года в датах использовалось две последние цифры, к примеру, 98 вместо 1998 – это казалось достаточно логичным решением на момент создания первых операционных систем в середине прошлого столетия. Впрочем, многие не предвидели, что может случиться проблема, когда дата превысит 2000 год. Используя те системы, 2000 год мог представляться только как «00», из-за чего программное обеспечение могло воспринять его как 1900 год. Такое представление повлекло бы за собой неправильные результаты в вычислениях на промежутках лет, находящихся по две стороны от 2000 года. К примеру, кто-то родившийся в 1920 году и умерший в 2001 мог получить значение возраст -19 лет. В ответ на эту проблему, производители ПО быстро обновили свои продукты, задействованные в управлении банковскими структурами, больничным оборудованием и другими важными сферами. Для подтверждения того, что баг может повлиять на компьютеры всего мира, в феврале 1999 года был создан «Интернациональный центр по разрешению проблемы 2000 года». Задачей центра была координация работы, необходимой для подготовки к новому тысячелетию. В конце концов, новый год наступил без каких-либо проблем. Однако, не все компьютеры одинаково работают с датами. Большинство операционных систем семейства UNIX представляют даты в виде прошедшего количества секунд с 1 января 1970 года (стандартная функция timestamp). К примеру: 1 января 1980 года представляется, как 315532800 секунд после 1 января 1970 года. Это число хранится в компьютерах, как беззнаковое 32-битное целое число (integer), которое может держать максимальное значение в 2147483647. Это означает, что компьютер может хранить 2147483647 в качестве даты, чего хватит только до 19 января 2038 года, после чего опять могут появиться проблемы. Баг

может принести намного больше вреда, если учитывать что UNIX используется во встраиваемых системах, в которых связь между «железной» и софтверной частью намного сильнее – к примеру, в роботах, используемых на конвейерах, в часах, в роутерах и т.д.

2012 год, крах Knight Capital Group. Один из крупнейших игроков трейдерского рынка в США потерял 75% своих акций (440 миллионов долларов США) за всего 30 минут из-за действий компьютерных программ для трейдинга – торговых роботов. Причиной послужила ошибка в алгоритме программы, которая позволяла проводить ошибочные сделки, покупая акции задорого и продавая дешево.

1985 год, Терак-25 (Therac-25). Медицинское устройство для радиационной терапии Therac-25 работало некорректно и облучало пациентов смертельной дозой радиации. Четыре человека погибли, здоровью еще двоих был нанесен тяжкий ущерб. Причина заключалась в том, что у устройства было два режима работы: первый, когда луч электронов направлялся напрямую на пациента в небольших дозах, и второй – когда луч сначала нацеливался на металлическую "цель", которая превращала его в радиационные лучи и передавала пациенту. В предыдущих моделях устройства второй режим был оснащен физическими детекторами наличия "цели" перед излучателем, чтобы лучи не направлялись напрямую на пациента в колоссальных дозах. В Therac-25 физические детекторы были заменены на программные. К сожалению, ПО было подвержено "арифметической перегрузке" – система начинала использовать во внутренних расчетах число, которые было чересчур большим для операций с ним. Если именно в этот момент оператор настраивал машину, проверки безопасности отказывали, и "цель" не перемещалась на место. В результате пациент получал в 100 раз б***о***льшую дозу радиации.

1983 год, потенциальное начало ядерной войны во время Холодной Войны.

Вечером 26 сентября 1983 года полковник Станислав Петров заступал на боевое дежурство в качестве руководителя центрального командного пункта советской системы предупреждения о ракетном нападении военного городка “Серпухов-15”. Через нескольких минут прозвучал сигнал тревоги “Ракетное нападение”, а настенная машинно-генерируемая карта показала, что в направлении Советского Союза летит американская ядерная ракета. В последующие несколько минут на экране компьютера появились отметки еще пяти ракет. В это время "холодная война" находилась на своём пике, – за три с половиной недели до этого был сбит южнокорейский "Боинг-747". По инструкции, в случае ракетного нападения дежурный был обязан немедленно поставить в известность руководство страны, которое и принимало решение об ответно-встречном ударе. Руководствуясь здравым смыслом (мол, 5 ракет — это слишком мало для первого удара в войне), Петров решил, что компьютер дал сбой. В результате он оказался прав: в системе оповещения действительно произошел сбой. После длившегося год секретного расследования инцидента 26 сентября 1983 года, был сделан вывод, что показания системы были вызваны редким, но предсказуемым эффектом отражения сигнала от поверхности Земли. Причиной послужила засветка датчиков спутника солнечным светом, отраженным от высотных облаков. Позднее в космическую систему были внесены изменения, позволяющие исключить такие ситуации. Правда, система дала сбой еще раз в 1995 году, когда россияне в течение короткого времени принимали научную ракету, стартовавшую с территории Норвегии, за летящую американскую ядерную ракету. Бывали случаи, когда за ракетное нападение принимались пуски метеорологических спутников, восход полной Луны, стаи гусей. Решить проблему сбоев в системе предупреждения намеревались путем развертывания в

Москве централизованной системы предупреждения о ракетном нападении, но построить его так и не успели.

Выводы

Можно выделить основные тенденции в сфере тестирования. ЯП Python становится все более популярным для автоматизации тестирования. Тестирование мобильных приложений вытесняет тестирование веб-приложений.

Тестирование мобильных приложений выросло очень большим темпом за последние годы в силу развития популярности смартфонов. Также продолжаются развиваться облачные платформы для тестирования работы приложений, использующих iOS и Android на различных мобильных устройствах в облаке. Самые популярные от гигантов индустрии – это «Amazon AWS Device Farm» и «Google Firebase Test Lab for Android».

Тестирование ПО является важным этапом жизненного цикла и процесса разработки ПО, так как именно оно направлено на проверку корректности работы разрабатываемого функционала и является последним этапом перед внедрением ПО в промышленную эксплуатацию. Плохо протестированный функционал, наличие в нем ошибок, даже незначительных неточностей в работе, способно привести к серьезным, а иногда катастрофическим последствиям.

Раздел 8 Составление заявки на изобретение

8.1 Понятие изобретения и объекта изобретения

Изобретение – новое и обладающее существенными отличиями техническое решение задачи, дающее положительный эффект.

Изобретение должно содержать указания на конкретные технические средства и способы решения задачи, то есть она должно быть осуществимо промышленным путем.

Новизна как критерий изобретения связана с приоритетом поданной на него заявки. Если установлено, что заявленное техническое решение по сравнению с прототипом имеет существенные отличия, то это означает, что оно отвечает критерию «новизна».

Критерий «*существенные отличия*» означает, что заявляемое в качестве изобретения техническое решение задачи содержит такую совокупность признаков, каждый из которых необходим, а все вместе взятые достаточны, чтобы отличить объект изобретения от других объектов и охарактеризовать его в том качестве, которое проявляется в положительном эффекте.

Объектами изобретений могут являться: новое устройство, новый способ, новое вещество; применение ранее известных устройств, способов, веществ по новому назначению (без их изменения по существу, когда положительный эффект получается именно благодаря такому применению). [1]

Продуктом как объектом изобретения является, в частности, устройство, вещество, штамм микроорганизма, генетическая конструкция и др. К *устройствам* относятся конструкции и изделия.

Способом как объектом изобретения является процесс осуществления действий над материальным объектом с помощью материальных средств.

В России можно патентовать только технические решения (способ, устройство, вещество), поэтому в тексте заявки

следует избегать элементов изобретения, представленных людьми (исполнителей, руководителей и т.д.).

Экспертизу заявки проводит эксперт в области патентного права, но не эксперт в области заявляемого изобретения, поэтому заявку следует составлять так, чтобы было понятно любому специалисту, независимо от области его деятельности.

8.2 Требования и структура описания изобретения

Описание изобретения с формулой изобретения и графические материалы, если они необходимы, являются основными документами заявки, отображающими созданное изобретение.

Описание изобретения должно иметь следующую структуру:

1. Название изобретения и класс международной классификации изобретений, к которому, по мнению заявителя, оно относится.

Пример 1. Предлагаемое изобретение относится к области систем управления сложными объектами с целью повышения эффективности их функционирования.

Пример 2. Система относится к классу управляющих систем общего назначения и может быть использована для управления организационно-техническими или социально-экономическими системами.

2. Характеристика аналогов изобретения. Аналоги выбираются из наиболее близких к заявленному объекту и наиболее прогрессивных в этой области технических решений. Аналоги описываются кратко, выделяются их существенные недостатки.

Пример. Известны способы «реанимирования» нефтяных скважин, основанные на изменении режима их эксплуатации или замене оборудования и неисправных частей в целях восстановления работоспособности объекта [Источник 1]. …Способ, основанный на изменении режима эксплуатации скважины, подразумевает варьирование значениями параметров объекта, но, поскольку на

практике значения этих параметров выбираются «вслепую», без какого-либо предварительного анализа и обоснования (фактически произвольно), нельзя предугадать, к какому результату приведет то или иное изменение каждого из параметров. Поэтому данный способ также часто оказывается неэффективным.

Известна система [Источник 2], основанная на использовании в процессе управления ситуационной модели, которая актуализируется в соответствии с текущими данными о состоянии объекта. Система предназначена для автоматизированного управления объектом и отображения информации о его состоянии, причем прогнозирование осуществляется на основе текущих показателей объекта и не затрагивает ретроспективные статистические данные о нем.

3. Характеристика выбранного заявителем прототипа.

3.1. Описание в статике (описание составных элементов прототипа изобретения)

Пример. Наиболее близким является способ [Источник 3], схема которого представлена на Фиг. 1, являющийся прототипом предлагаемого изобретения. Способ-прототип представляет собой последовательность технологических операций, сущность которых состоит в восстановлении работоспособности нефтяной скважины – объекта 1 посредством выбора подсистемой выбора 3 на основе Д2 – текущей информации о состоянии объекта из базы данных (БД) информационной системы (ИС) 2 нефтедобывающего предприятия и разработанной на предприятии методики восстановления работоспособности скважин [Источник 3] варианта реанимирования Д3 для конкретной скважины, принятия решения подсистемой управления 4 о применении на объекте 1 одного или последовательно нескольких из заданного множества вариантов реанимирования Д6.

Описание в динамике (описание работы прототипа изобретения)

Пример. Результаты реанимирования Д1 объекта в случае успеха реанимационных работ сохраняются в БД ИС 2 нефтедобывающего предприятия. В противном случае, на основе информации Д4 о неуспехе применения выбранного варианта реанимирования, происходит принятие решения подсистемой

управления 4 об остановке скважины и постановке ее в очередь на подземный ремонт.

3.3. Выделение главных недостатков прототипа, обосновывающих необходимость его замены предлагаемым изобретением

Пример. Основной недостаток прототипа состоит в неадаптивной процедуре реанимирования, которая включает последовательный перебор всех возможных его вариантов до тех пор, пока не будет подобран наиболее эффективный в текущих условиях и для рассматриваемой конкретной скважины. Такой перебор «вслепую» экономически неэффективен, поскольку каждый подход к скважине, подлежащей реанимированию без подземного ремонта (таких объектов на месторождении может быть достаточно много), связан с транспортировкой техники и специалистов, материальными (от 5 до 15 тыс. рублей на каждый подход реанимационного звена плюс потери возможной прибыли из-за простоя скважины), временными (до 2 суток на одну скважину) и эксплуатационными (амортизация оборудования и т.д.) затратами.

В этом разделе должна быть приведена библиографическая ссылка на источник, в котором описан выбранный заявителем прототип.

4. Цель изобретения. Объективность цели определяется необходимостью удовлетворения какой-либо общественной потребности.

Пример. Целью изобретения является устранение недостатков способа-прототипа и повышение эффективности процесса реанимирования нефтяных скважин.

5. Сущность изобретения и его отличительные от прототипа признаки. Здесь также приводятся доказательства возможности достижения положительного эффекта при осуществлении изобретения, основанные на проведенном анализе, и формула изобретения.

Пример. Технической сущностью предлагаемого изобретения является внедрение в способ реанимирования интеллектуальной обработки данных с помощью ИС для повышения эффективности восстановления объекта.

Эта сущность достигается тем, что в способ реанимирования нефтяных скважин, включающий получение информации от информационного выхода объекта 1 и занесение ее в БД ИС 2 через первый вход; выбор варианта реанимирования подсистемой выбора 3; оценку результатов реанимирования, принятие решения о целесообразности реанимирования, управление объектом 1 подсистемой управления 4 путем подачи воздействия от первого выхода подсистемы 4 на управляющий вход объекта 1 и занесение результатов реанимирования в ИС 2 от второго выхода подсистемы 4 на второй вход ИС 2, вводятся операции формирования исходных данных об объекте 1 подсистемой формирования исходных данных 5 на основе информации, поступающей с выхода ИС 2 на первый вход подсистемы 5 …

6. Перечень фигур графических изображений с кратким указанием, что изображено на каждой из них.

Пример. На Фиг. 1 представлена структурная схема существующего способа реанимирования нефтяных скважин (прототип предполагаемого изобретения).

На Фиг. 2 представлена структурная схема предлагаемого способа реанимирования нефтяных скважин с применением информационных технологий.

7. Примеры конкретного выполнения. Описываются примеры, подтверждающие возможность осуществления изобретения с получением положительного эффекта при использовании всей совокупности существенных признаков изобретения, указанной в его формуле.

7.1. Описание в статике (описание составных элементов изобретения)

Пример. Предлагаемый способ реанимирования содержит операции получения информации об объекте 1; занесения данных БД ИС 2; выбора варианта реанимирования подсистемой выбора варианта реанимирования 3; оценки результатов реанимирования,

принятия решения о целесообразности реанимирования, управления объектом 1 подсистемой управления 4; формирования набора исходных данных об объекте 1 подсистемой формирования набора исходных данных 5; корреляционного анализа параметров процесса реанимирования подсистемой корреляционного анализа 6; статистического анализа факторов, дестабилизирующих процесс реанимирования, подсистемой статистического анализа 7. ...

7.2. Описание в динамике (описание работы изобретения)

Пример. Данные Д1, собираемые с объекта 1 (см. Фиг. 2), актуализируют информацию о скважине, подлежащей реанимированию, в БД ИС предприятия 2, откуда она (см. информационный поток Д2) запрашивается в процессе формировании выборки исходных данных для анализа (блок 3) с целью принятия управленческих решений по реанимированию данной скважины. …

Таким образом, реанимирование объекта 1 в рамках предлагаемого способа осуществляется с использованием наиболее полного и объективного информационного материала – актуальных и ретроспективных (статистических) данных по текущей скважине и по схожим с ней по характеристикам объектам, что обеспечивает необходимую точность выбора варианта реанимирования….

8. Технико-экономическая или иная эффективность.

Приводится качественная оценка технико-экономических преимуществ изобретения в сравнении с прототипом.

Если по изобретению не представляется возможным определить экономическую эффективность, то должно быть показано, какие социальные или другие задачи решает изобретение (например, улучшение условий труда, повышение техники безопасности и др.). Предоставляются материалы, подтверждающие достижение цели, полезность и эффективность (акты внедрений, испытаний и т.д.).

9. Вывод

Пример. Предлагаемый способ обеспечивает повышение эффективности процесса реанимирования нефтяных скважин за счет выбора (на основе анализа априорной и динамически

меняющейся в процессе функционирования объекта информации о нем) наилучшего из совокупности имеющихся и наиболее экономически выгодного варианта реанимирования, без проведения затратного эксперимента, включающего последовательный перебор «вслепую» всех возможных вариантов реанимирования на реальном объекте. Изобретение способствует повышению точности принимаемых решений, сокращению продолжительности процесса восстановления скважин (в ряде случаев – с 2 суток до 2 часов) и снижению нагрузки на специалистов, реализующих реанимирование, за счет уменьшения объема проводимых на скважине работ (для восстановления применяется один или несколько вариантов реанимирования, но не все последовательно до нахождения подходящего) и, как следствие, повышению эффективности нефтедобычи на месторождении в целом.

10. Список литературы (только та литература, на которую имеется ссылка в тексте).

Пример.

1. Рекомендации по определению видов ремонтных работ в скважинах, эксплуатируемых организациями нефтедобывающей, нефтеперерабатывающей, газовой и нефтехимической промышленности. Минэнерго России // Промышленный еженедельник. №28, 2008.

2. Система ситуационного управления // Андреев Г.И., Латышев Н.В., Остапенко С.Н. и др. Свидетельство на полезную модель 13103 U1 РФ, МПК 7 G05B17/00, № 99124885/20 ; заявл. 01.12.1999 ; опубл. 20.03.2000.

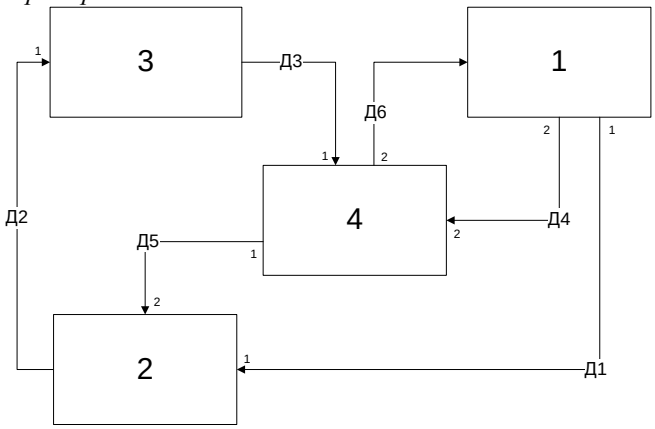
11. Формула изобретения (на отдельном листе). Дается описание состава (элементов) изобретения с указанием существенных отличительных признаков в максимально сжатой (краткой) форме. Существенным признаком можно признать лишь такой признак из общей массы признаков объекта изобретений, отсутствие которого в совокупности признаков не дает возможность получить тот положительный эффект, который является целью изобретения.

Пример. Способ реанимирования нефтяных скважин, включающий получение информации от информационного выхода объекта 1 и занесение ее в БД ИС 2 через первый вход; выбор варианта реанимирования подсистемой выбора 3; оценку результатов реанимирования, принятие решения о целесообразности реанимирования, управление объектом 1 подсистемой управления 4 путем подачи воздействия от первого выхода подсистемы 4 на управляющий вход объекта 1 и занесение результатов реанимирования в информационную систему 2 от второго выхода подсистемы 4 на второй вход информационной системы 2, отличающийся тем, что вводятся операции формирования исходных данных об объекте 1 подсистемой формирования исходных данных 5 на основе информации, поступающей с выхода информационной системы 2 на первый вход подсистемы 5; корреляционного анализа параметров процесса реанимирования подсистемой корреляционного анализа 6 на основе информации, поступающей с первого выхода подсистемы 5 на вход подсистемы 6; статистического анализа факторов, дестабилизирующих процесс реанимирования, подсистемой статистического анализа 7 на основе информации, поступающей от второго выхода подсистемы 5 на вход подсистемы 7; статистического имитационного моделирования процесса реанимирования подсистемой статистического имитационного моделирования 8 на основе информации, поступающей с выхода подсистемы 7 на первый вход подсистемы 8, причем данные от информационного выхода объекта 1 подаются также на первый вход подсистемы 4; от выхода подсистемы 3 – на второй вход подсистемы 8; от выхода подсистемы 6 – на первый вход подсистемы 3; от выхода подсистемы 7 – на второй вход подсистемы 3; от первого выхода подсистемы 8 – на третий вход подсистемы 3; от второго выхода подсистемы 8 – на второй вход подсистемы 4; от выхода подсистемы 6 – на третий вход подсистемы 5.

В идеальном варианте формула изобретения так должна отражать в сжатом виде его суть, чтобы любой специалист мог воспроизвести по ней изобретение. Именно формула является объектом интеллектуальной собственности, поэтому в неё необходимо включать те отличительные черты изобретения, авторское право на которые планируется получить.

12. Фигуры (изображения).

Пример.



Фиг. 1

13. Аннотация: область, сущность, формула изобретения.

Пример. Изобретение относится к области систем управления сложными объектами с целью повышения эффективности их функционирования.

Сущностью предлагаемого изобретения является внедрение в способ реанимирования интеллектуальной обработки данных с помощью ИС для повышения эффективности восстановления объекта.

Эта сущность достигается тем, что в способ реанимирования нефтяных скважин, включающий получение информации от информационного выхода объекта 1 и занесение ее в БД ИС 2 через первый вход…

Список использованных источников информации

1. Мачулина, К.Т. Составление заявки на изобретение: методические указания [Текст] / К.Т.Мачулина, А.Л.Корнеева. - УЭЗ КуАИ: Куйбышев, 1985. – 25 стр.

2. Вигерс Карл. Разработка требований к программному обеспечению [Текст] /Пер, с англ. — М.: Издательсш-торговый дом «Русская Редакция», 2004. —576с.: ил.

3. Информационные системы и технологии в экономике и управлении. Проектирование информационных систем [Электронный ресурс]: учебное пособие/ Е.В. Акимова [и др.].— Электрон. текстовые данные.— Саратов: Вузовское образование, 2016.— 178 c.

4. Вичугова А.А. Инструментальные средства информационных систем [Электронный ресурс]: учебное пособие/ Вичугова А.А.— Электрон. текстовые данные.— Томск: Томский политехнический университет, 2015.— 136 c.

5. Пальмов, С. В. Методы и средства моделирования программного обеспечения [Электронный ресурс] : конспект лекций / С. В. Пальмов ; ПГУТИ, Каф. ИСТ. - Электрон. текстовые дан. (1 файл: 2,44 Мб). - Самара : ИНУЛ ПГУТИ, 2016.

6. Управление разработкой информационных систем [Текст] : учебник / А. Р.Диязитдинова, Н. В. Коныжева; ПГУТИ, Кафедра ЭИС. - Самара : ИУНЛ ПГУТИ, 2013. - 194 с.

7. Диязитдинова, А. Р. Интегрированное информационное пространство корпорации [Текст] : учебное пособие / А. Р. Диязитдинова, Н. В. Коныжева ; ПГУТИ. - Самара : ИУНЛ ПГУТИ, 2013. - 140 с.

ПРОЕКТНЫЙ ПРАКТИКУМ

Учебное пособие

Руслан Радикович Халимов, Евгения Ивановна Горожанина
Федеральное государственное бюджетное образовательное учреждение высшего образования
“Поволжский государственный университет телекоммуникаций и информатики”
443010, г. Самара, ул. Льва Толстого 23