Министерство образования Республики Беларусь

Учреждение образования «[Институт информационных технологий](https://www.google.by/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&sqi=2&ved=0CDcQFjAC&url=http%3A%2F%2Fwww.iit-bsuir.by%2F&ei=unOvUpDmGemB4gSZ3IGQDQ&usg=AFQjCNHEr4the3QhkSSjmxbzcNJBZi5-Tg&sig2=GxV3hq7_8t34Csduk0HElg&bvm=bv.57967247,d.bGE&cad=rja) Белорусского государственного университета информатики и радиоэлектроники»

Факультет компьютерных технологий

Дисциплина: «Технологии разработки программного обеспечения»

Контрольная работа

на тему

«Автоматизация работы адвокатской конторы»

Студент: гр. 581072 Вежновец Виталина

Руководитель: Бакунова О.М.

Минск 2017

# ВВЕДЕНИЕ

Современная жизнь немыслима без использования информационных технологий в самых различных областях жизнедеятельности человека: на производстве, в организациях различного профиля, в научных исследованиях, в процессе обучения. На производственных предприятиях и в других организациях, заинтересованных в увеличении показателя эффективности своей деятельности, программные средства используются для управления и автоматизации различных задач прикладного характера. Высокое качество программных продуктов дает разработчикам не только конкурентные преимущества и доверие клиентов, но и облегчает сопровождение и развитие программного обеспечения. Среди широкого разнообразия программных средств, используемых организациями для автоматизации решения основных задач их деятельности можно выделить системы обработки информации, построенные по принципу систем управления базами данных. Системы управления базами данных прошли большой путь развития вместе с вычислительной техникой, и в настоящее время пользуются популярностью. Это связано в первую очередь с их востребованностью в учреждениях и на предприятиях, а также с доступностью персональных ЭВМ для любой организации. Возможности таких систем заключаются в создании на ЭВМ общей базы данных для множества приложений, структурировании этих данных и поддержании их в актуальном состоянии для обеспечения эффективного доступа к ним пользователей в рамках предоставленных полномочий. Потоки информации, которые необходимо обрабатывать таким системам, растут с каждым годом в связи с расширением круга решаемых задач.

В настоящее время автоматизированные системы на основе систем управления базами данных широко распространены в банковской сфере. Эти системы позволяют облегчить и улучшить работу сотрудников банка, что является не менее актуальным, чем разработка и использование автоматизированных систем для любой другой сферы деятельности.

Современный компьютерный мир представляет собой разнообразную и весьма сложную совокупность вычислительных устройств, систем обработки информации, телекоммуникационных технологий, программного обеспечения и высокоэффективных средств его проектирования. Вся эта многогранная и взаимосвязанная метасистема решает огромный круг проблем в различных областях человеческой деятельности, от простого решения школьных задач на домашнем персональном компьютере до управления сложными технологическими процессами.

Чем сложнее задача автоматизации и чем ответственнее область, в которой используются компьютерные информационные технологии, тем все более и более критичными становятся такие свойства как надежность и безопасность информационных ресурсов, задействованных в процессе сбора, накопления, обработки, передачи и хранения компьютерных данных. Вредоносные воздействия на информацию в процессе функционирования компьютерных систем (КС) различного назначения осуществляется с целью нарушения ее конфиденциальности, целостности и доступности. Решение задач, связанных с предотвращением воздействия непосредственно на информацию, осуществляется в рамках комплексной проблемы обеспечения безопасности информации и имеет достаточно развитую научно-методическую базу. При этом, рассматривая информацию как активный эксплуатируемый ресурс, можно говорить о том, что процесс обеспечения безопасности информации включает в себя и обеспечение безопасности программного обеспечения КС. Данный аспект обеспечения безопасности информации и средств ее обработки именуется эксплуатационной безопасностью, так как соответствует этапу применения КС. В то же время, в последнее время появились новые проблемы обеспечения безопасности, связанные с информационными технологиями, которые, по мнению ряда зарубежных и отечественных экспертов в области их создания и применения, в значительной степени определяют эффективность создаваемых компьютерных систем.

Мировые исследования последних лет показали, что функциональные и надежностные характеристики КС определяются качеством и надежностью программного обеспечения, входящего в их состав. Кроме проблем качества и надежности программного обеспечения при создании КС фундаментальная проблема его безопасности приобретает все большую актуальность. При этом в рамках данной проблемы на первый план выдвигается безопасность технологий создания программного обеспечения компьютерных систем. Данный аспект проблемы безопасности программных комплексов является сравнительно новым и связан с возможностью внедрения в тело программных средств на этапе их разработки (или модификации в ходе авторского сопровождения) так называемых "программных закладок". В связи с этим все более актуальным становится проблема обеспечения технологической безопасности программного обеспечения КС различного уровня и назначения.

### 1 Среда разработки JAVA

Java — объектно-ориентированный язык программирования, разрабатываемый компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 года. Изначально новый язык программирования назывался Oak (James Gosling) и разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения.

Программы на Java могут быть транслированы в байт-код, выполняемый на виртуальной java-машине (JVM) — программе, обрабатывающей байт-код и передающей инструкции оборудованию, как интерпретатор, но с тем отличием, что байт-код, в отличие от текста, обрабатывается значительно быстрее.

Язык Java зародился как часть проекта создания передового программного обеспечения для различных бытовых приборов. Реализация проекта была начата на языке C++, но вскоре возник ряд проблем, наилучшим средством борьбы с которыми было изменение самого инструмента — языка программирования. Стало очевидным, что необходим платформо-независимый язык программирования, позволяющий создавать программы, которые не приходилось бы компилировать отдельно для каждой архитектуры и можно было бы использовать на различных процессорах под различными операционными системами.

Система программирования Java может служить основой для совместной разработки больших программных систем коллективом разработчиков, связанных между собой только через WWW (они и знакомы между собой могут быть лишь заочно, через e-mail, а когда они наконец повстречаются где-нибудь на международном симпозиуме, в их активе уже может быть совместно разработанная программная система). Java и WWW являются первыми системами, обеспечивающими такую возможность, поэтому их внедрение и распространение многие программисты справедливо называют революцией в разработке программного обеспечения. Ясно, что ведущую роль в обеспечении указанной возможности играет именно Java, так как именно Java позволяет распространять не просто тексты, а работающие программы и их фрагменты (апплеты) по WWW.

Основным свойством апплетов является возможность выполнять их на различных платформах и в различных окружениях, не оказывая вредного влияния на аппаратуру, программы и данные их пользователей. Язык, ориентированный на программирование апплетов должен прежде всего обеспечивать надежность и безопасность. Этого легче всего достичь в интерпретируемом языке, хотя интерпретация, как правило, ведет к существенной потере эффективности программы, причем эти потери растут с ростом объема программы и объема обрабатываемых ею данных.

За более чем десятилетнюю историю языка Java, сменилось ни одно поколение интегрированных сред разработки (Integrated Development Environment - IDE). Эволюция IDE средств обусловлена множеством факторов, совокупность которых называется - информационные технологии, включающих в себя программную и аппаратную составляющие, а так же развитие самого языка, которое происходит не только в глубь, оптимизация каких то возможностей, что ярко демонстрирует приход новых библиотек Swing вместо устаревших AWT, но и "вширь", это появление технологий JSP, большая интеграция с СУБД и серверами приложений, поддержка технологий Spring, Hibernate и пр.

Язык Java потребовался для создания интерактивных продуктов для сети Internet. Фактически, большинство архитектурных решений, принятых при создании Java, было продиктовано желанием предоставить синтаксис, сходный с C и C++. В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. В Java добавлены все хорошие черты C++. Три ключевых элемента объединились в технологии языка Java.

* Java предоставляет для широкого использования свои апплеты (applets) — небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML
* Java высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты
* Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода.

2 Разработка функциональной модели

AllFusion ERwin Data Modeler (ранее ERwin) — CASE-средство для проектирования и документирования баз данных, которое позволяет создавать, документировать и сопровождать базы данных, хранилища и витрины данных. Модели данных помогают визуализировать структуру данных, обеспечивая эффективный процесс организации, управления и администрирования таких аспектов деятельности предприятия, как уровень сложности данных, технологий баз данных и среды развертывания.

AllFusion ERwin Data Modeler (ERwin) предназначен для всех компаний, разрабатывающих и использующих базы данных, для администраторов баз данных, системных аналитиков, проектировщиков баз данных, разработчиков, руководителей проектов, — и позволяет управлять данными в процессе корпоративных изменений, а также в условиях стремительно изменяющихся технологий.

AllFusion ERwin Data Modeler (ERwin) позволяет наглядно отображать сложные структуры данных. Удобная в использовании графическая среда системы упрощает разработку базы данных и автоматизирует множество трудоёмких задач, уменьшая сроки создания высококачественных и высокопроизводительных транзакционных баз данных и хранилищ данных. Продукт улучшает коммуникацию организации, обеспечивая совместную работу администраторов и разработчиков баз данных, многократное использование модели, а также наглядное представление комплексных активов данных в удобном для понимания и обслуживания формате.

* 1. **Концепция IDEF0**

Общая методология IDEF состоит из трех частных методологий моделирования, основанных на графическом представлении систем:

• IDEF0 используется для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающие эти функции.

• IDEF1 применяется для построения информационной модели, отображающей структуру и содержание информационных потоков, необходимых для поддержки функций системы;

• IDEF2 позволяет построить динамическую модель меняющихся во времени поведения функций, информации и ресурсов системы.

Методология IDEF0 основана на следующих концептуальных положениях. Модель – искусственный объект, представляющий собой отображение (образ) системы и ее компонентов. Модель разрабатывают для понимания, анализа и принятия решений о реконструкции (реинжиниринге) или замене существующей, либо проектировании новой системы. Система представляет собой совокупность взаимосвязанных и взаимодействующих частей, выполняющих некоторую полезную работу. Частями (элементами) системы могут быть любые комбинации разнообразных сущностей, включающие людей, информацию, программное обеспечение, оборудование, изделия, сырье или энергию (энергоносители). Модель описывает, что происходит в системе, как ею управляют, какие сущности она преобразует, какие средства использует для выполнения своих функций и что производит.

Блочное моделирование и его графическое представление. Основной концептуальный принцип методологии IDEF – представление любой изучаемой системы в виде набора взаимодействующих и взаимосвязанных блоков, отображающих процессы, операции, действия (определения – см. ниже), происходящие в изучаемой системе.

В IDEF0 все, что происходит в системе и ее элементах, принято называть функциями. Каждой функции ставится в соответствие блок. На IDEF0 –диаграмме, основном документе при анализе и проектировании систем, блок представляет собой прямоугольник. Интерфейсы, посредством которых блок взаимодействует с другими блоками или с внешней по отношению к моделируемой системе средой, представляются стрелками), входящими в блок или выходящими из него. Входящие стрелки показывают, какие условия должны быть одновременно выполнены, чтобы функция, описываемая блоком, осуществилась.

* 1. **Лаконичность и точность**

Документация, описывающая систему, должна быть точной и лаконичной. Многословные характеристики, изложенные в форме традиционных текстов, неудовлетворительны. Графический язык позволяет лаконично, однозначно и точно показать все элементы (блоки) системы и все отношения и связи между ними, выявить ошибочные, лишние или дублирующие связи и т.д.

* 1. **Передача информации**

Средства IDEF0 облегчают передачу информации от одного участника разработки модели (отдельного разработчика или рабочей группы) к другому. К числу таких средств относятся:

* диаграммы, основанные на простой графике блоков и стрелок, легко читаемые и понимаемые;
* метки на естественном языке для описания блоков и стрелок, а также глоссарий и сопроводительный текст для уточнения смысла элементов диаграммы;
* последовательная декомпозиция диаграмм, строящаяся по иерархическому принципу, при котором на верхнем уровне отображаются основные функции, а затем происходит их детализация и уточнение;
* древовидные схемы иерархии диаграмм и блоков, обеспечивающие обозримость модели в целом и входящих в нее деталей.
  1. **Строгость и формализм**

Разработка моделей IDEF0 требует соблюдения ряда строгих формальных правил, обеспечивающих преимущества методологии в отношении однозначности, точности и целостности сложных многоуровневых моделей. Эти правила описываются ниже. Здесь отмечается только основное из них: все стадии и этапы разработки и корректировки модели должны строго, формально документироваться с тем, чтобы при ее эксплуатации не возникало вопросов, связанных с неполнотой или некорректностью документации.

* 1. **Итеративное моделирование**

Разработка модели в IDEF0 представляет собой пошаговую, итеративную процедуру. На каждом шаге итерации разработчик предлагает вариант модели, который подвергают обсуждению, рецензированию и последующему редактированию, после чего цикл повторяется. Такая организация работы способствует оптимальному использованию знаний системного аналитика, владеющего методологией и техникой IDEF0, и знаний специалистов – экспертов в предметной области, к которой относится объект моделирования.

* 1. **Отделение «организации» от «функций».**

При разработке моделей следует избегать изначальной «привязки» функций исследуемой системы к существующей организационной структуре моделируемого объекта (предприятия, фирмы). Это помогает избежать субъективной точки зрения, навязанной организацией и ее руководством. Организационная структура должна явиться результатом использования (применения) модели. Сравнение результата с существующей структурой позволяет, во-первых, оценить адекватность модели, а во-вторых – предложить решения, направленные на совершенствование этой структуры.

Контекстная диаграмма, представленная на рисунках 2.1и 2.2, дает общее описание системы и факторы влияющее на нее.

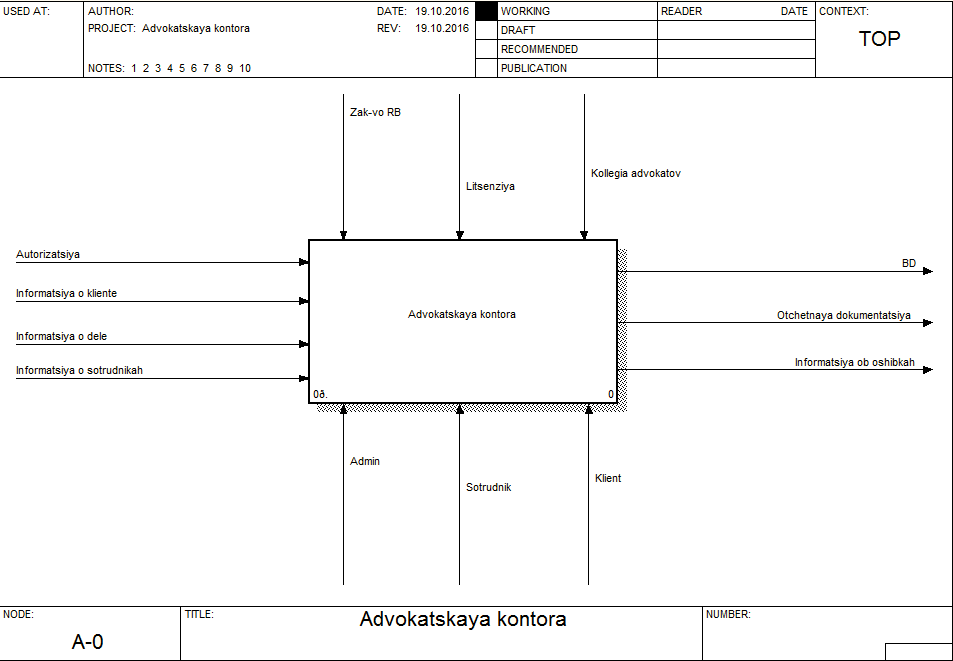


Рисунок 2.1 **–** Контекстнаядиаграмма

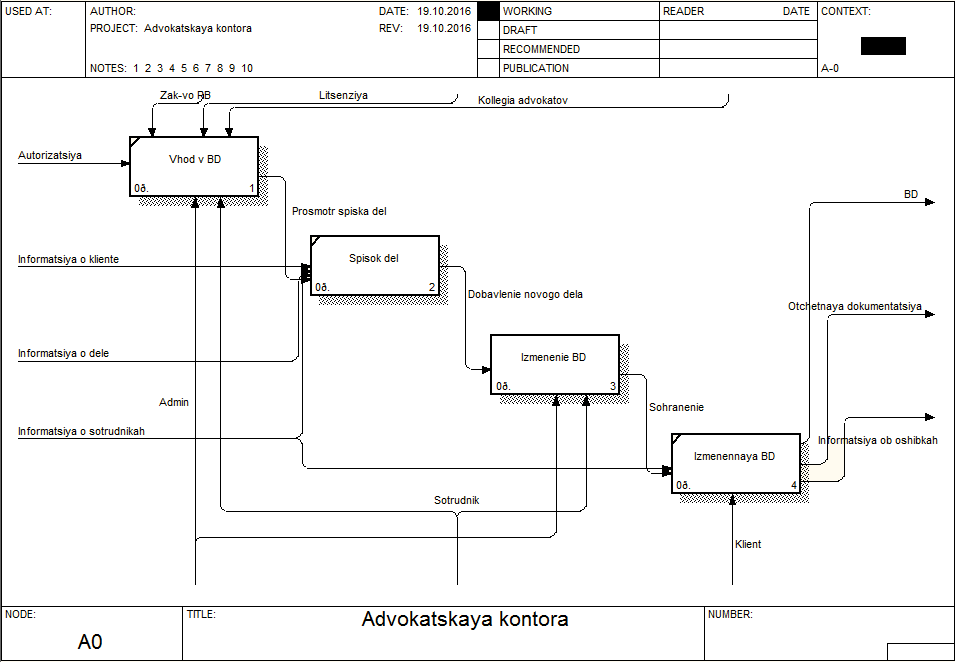


Рисунок 2.2 – Декомпозиция

* 1. **Разработка структурной схемы**

Работа пользователя с базами данных в ПС реализована посредством графического интерфейса. Именно этот блок объединят все модули ПС и предоставляет пользователю возможность управления процессом.

Блок экспорта отвечает за процесс экспорта, управление процессом экспорта, визуализацию процесса экспорта, индикацию процесса экспорта.

Процесс разделен и протекает в несколько этапов. Соответственно блок экспорта имеет в своем составе блок экспорта метаданных и блок экспорта данных, а каждый из этих блоков имеет в своем составе блоки извлечения, структурирования, форматирования и выгрузки. Каждый блок отвечает за работу одного этапа экспорта БД.

Блок конфигурирования баз данных отвечает за редактирование БД. Так как настроек довольно много и их структура довольно сложна, блок имеет в своем составе множество блоков, каждый из которых выполняет часть задачи. Каждый из этих блоков отвечает за предоставление пользователю удобного интерфейса и функционала по создание и редактированию настроек влияющих на работу определенного этапа работу процесса экспорта базы данных. Работа с настройками соединения с БД и настройками выгрузки выделена в отдельные блоки, так как в дополнение к задачам, которые выполняют остальные блоки, эти блоки взаимодействуют еще и с внешними системами.

Блок редактирования общих настроек служит для редактирования общих настроек. Блок обработки ошибок и исключений отвечает за обработку штатных исключений предусмотренных разработкой ПС и внештатных ошибок. Блок настройки диалогов предоставляет доступ к списку диалогов, которые выдает ПС для пользователя и позволяет исключить либо же наоборот вернуть определенные диалоги в работу ПС. Объектно-ориентированная структура программного средства представлена в виде диаграммы классов на рисунке 3.1.

## 3.1 Модель вариантов использования

Модель вариантов использования предназначается для определения требований к системе. Она включает в себя актеров, варианты использования и связи между ними. Для отображения этой модели язык UML предлагает использовать диаграммы Use Case (вариант использования) совместно с моделями State Diagram (диаграммы состояний) и Activity Diagram (диаграммы деятельности/активности). Последние используются для конкретизации вариантов использования системы.

Разработка диаграммы вариантов использования преследует цели:

* определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
* сформулировать общие требования к функциональному поведению проектируемой системы;
* разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
* подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

### 3.1.2 Варианты использования

Работа с вариантами использования является одной из самых важных на стадии уточнения. Каждый вариант использования — это требование к системе. Диаграммы отображают взаимодействие между вариантами использования системы и действующими лицами, передающими информацию в данную систему. Такой пояснительный текст получил название примечания или сценария.

Для разрабатываемой системы, исходя из потребностей действующих лиц, можно выделить следующие варианты использования:

* просмотр записи об открытии дела;
* создание записи о звонке клиента;
* создание записи о судебном заседании;
* создание записи о ведении дела;
* редактирование записи о приеме клиента;
* ведение БД о выполненных заказах;
* формирование документации.

### 3.1.3 Диаграмма вариантов использования

На диаграмме вариантов использования показывают взаимодействия между всеми действующими лицами и вариантами использования. Диаграмма должна показывать, какие действующие лица инициируют варианты использования, а также должна отображать, когда действующие лица получают информацию от вариантов использования.

Основные взаимодействия между действующими лицами и вариантами использования задаются с помощью связи коммуникации. Задается в виде простой стрелки. Направление стрелки показывает, кто инициирует связь (всегда действующее лицо) и какой вариант использования отправляет информацию внешнему действующему лицу.

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Каждая такая схема акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата.

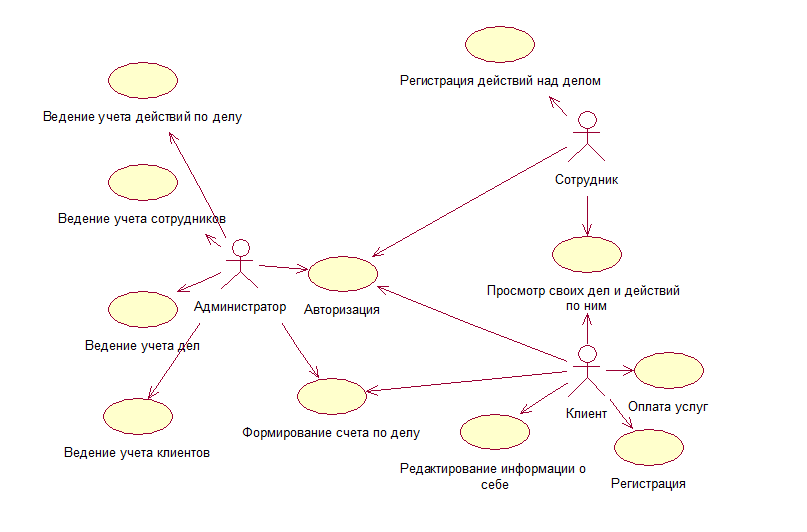


Рисунок 3.1 – Диаграмма вариантов использования (Use Case)

Основное назначение логического представления состоит в анализе структурных и функциональных отношений между элементами модели системы. Различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения.

Алгоритмические и логические операции, требующие выполнения в определенной последовательности, окружают нас постоянно. Конечно, мы не всегда задумываемся о том, что подобные операции относятся к столь научным категориям. Например, чтобы позвонить по телефону, нам предварительно нужно снять трубку или включить его. Для приготовления кофе или заваривания чая необходимо вначале вскипятить воду. Чтобы выполнить ремонт двигателя автомобиля, требуется осуществить целый ряд нетривиальных операций, таких как разборка силового агрегата, снятие генератора и некоторых других.

Важно подчеркнуть то обстоятельство, что с увеличением сложности системы строгое соблюдение последовательности выполняемых операций приобретает все более важное значение. Если попытаться заварить кофе холодной водой, то мы можем только испортить одну порцию напитка. Нарушение последовательности операций при ремонте двигателя может привести к его поломке или выходу из строя. Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой, операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому.

Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний. Именно они позволяют реализовать в языке UML особенности процедурного и синхронного управления, обусловленного завершением внутренних деятельностей и действий. Метамодель UML предоставляет для этого необходимые термины и семантику. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции некоторого класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы.

На диаграмме последовательности изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами. Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени. При этом диаграмма последовательности имеет как бы два измерения. Одно - слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Графически каждый объект изображается прямоугольником и располагается в верхней части своей линии жизни. Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то и его линия жизни должна продолжаться по всей плоскости диаграммы последовательности от самой верхней ее части до самой нижней. Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены (разрушены), чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы "X". На рисунке 3.2 представлен пример диаграммы последовательности действий:

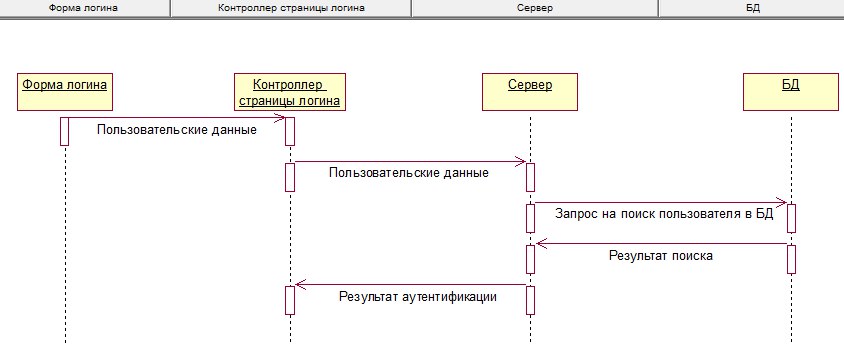


Рисунок 3.2 – Диаграмма Sequence

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Традиционно для этой цели использовались блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата.

Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой, операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами - переходы от одного состояния действия к другому.

Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний. Именно они позволяют реализовать в языке UML особенности процедурного и синхронного управления, обусловленного завершением внутренних деятельностей и действий. Метамодель UML предоставляет для этого необходимые термины и семантику. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции некоторого класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы. Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и по крайней мере одним выходящим из состояния переходом. Этот переход неявно предполагает, что входное действие уже завершилось. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Обычное использование состояния действия заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления.

Графически состояние действия изображается фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами . Внутри этой фигуры записывается выражение действия (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.

Пример диаграммы activity представлен на рисунке 3.3:

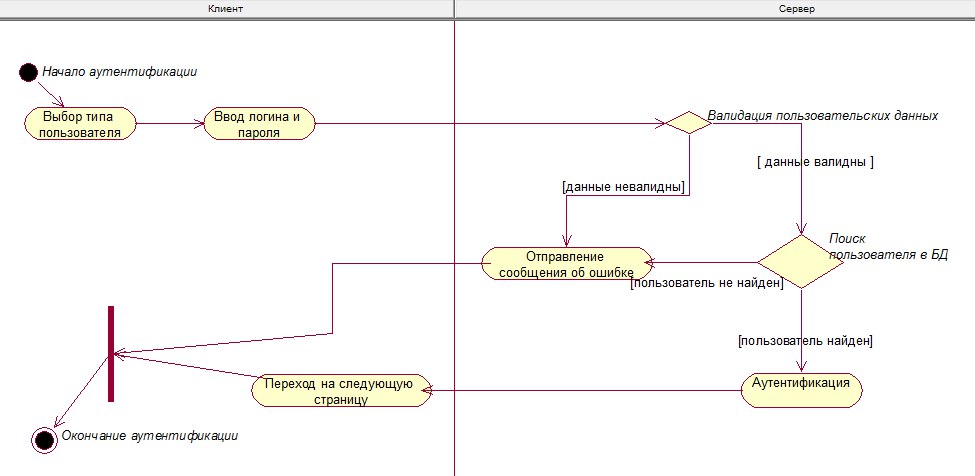


Рисунок 3.3 – Диаграмма activity

Диаграмма кооперации является другим способом визуализации взаимодействия в модели и, как и диаграмма последовательности, оперирует объектами и сообщениями. Особенность работы в среде Rational Rose заключается в том, что этот вид канонической диаграммы создается автоматически после построения диаграммы последовательности и нажатия клавиши . С помощью этой же клавиши осуществляется переключение между диаграммами последовательности и кооперации.

После того как диаграмма кооперации активизирована, специальная панель инструментов приобретает следующий вид:

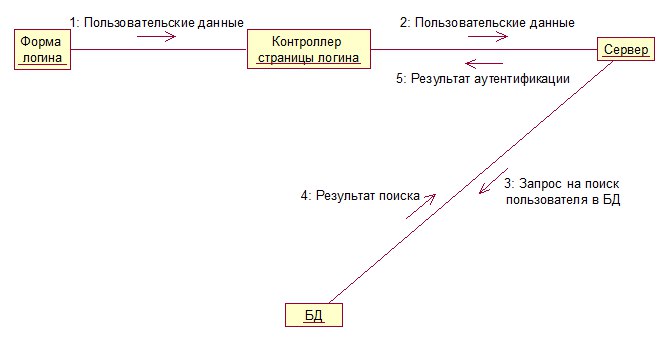


Рисунок 3.4 – Диаграмма кооперации

# 

# 

# Рисунок 3.5 – Диаграмма классов

Центральное место в объектно-ориентированном программировании занимает разработка логической модели системы в виде диаграммы классов. Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывать их внутреннюю структуру и типы отношений.

Диаграмма классов представляет собой граф, вершинами которого являются элементы типа «классификатор», связанные различными типами структурных отношений. Диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи.

**Классы**

***Класс*** (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции. В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

***Имя класса*** должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов или одной диаграммой. Оно указывается в первой верхней секции прямоугольника. В дополнение к общему правилу наименования элементов языка UML, имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется в качестве имен классов использовать существительные, записанные без пробелов. Имена классов образуют словарь предметной области.

В первой секции обозначения класса могут находиться ссылки на стандартные шаблоны или абстрактные классы, от которых образован данный класс и от которых он наследует свойства и методы. Дополнительно в этой секции может приводиться информация о разработчике данного класса и статусе состояния разработки, а также  записываться и другие общие свойства этого класса, имеющие отношение к другим классам диаграммы или стандартным элементам языка UML.

Именами классов могут быть такие существительные, как «Сотрудник», «Компания, «Руководитель», «Клиент», «Продавец», «Менеджер», «Офис» и другие, имеющие непосредственное отношение к моделируемой предметной области и функциональному назначению проектируемой системы.

Атрибуты класса или свойства записываются во второй сверху секции прямоугольника класса. В языке UML каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения:

<квантор видимости><имя атрибута>[кратность]:

<тип атрибута> = <исходное значение>{строка-свойство}

Квантор видимости может принимать одно из трех возможных значений и  отображается при помощи соответствующих специальных символов:

* «+» обозначает атрибут с областью видимости типа общедоступный (public). Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма;
* «#» обозначает атрибут с областью видимости типа защищенный (protected). Атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса;
* «-»  обозначает атрибут с областью видимости типа закрытый (private). Атрибут с этой областью видимости недоступен или невиден для всех классов без исключения.

Квантор видимости может быть опущен. В этом случае его отсутствие просто означает, что видимость атрибута не указывается. Эта ситуация отличается от принятых по умолчанию соглашений в традиционных языках программирования, когда отсутствие квантора видимости трактуется как public или private. Однако вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private.

Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и поэтому должна быть уникальной в пределах данного класса. Имя атрибута является единственным обязательным элементом синтаксического обозначения атрибута. Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса. В общем случае кратность записывается в форме строки текста в квадратных скобках после имени соответствующего атрибута:

[нижняя\_граница1 .. верхняя\_граница1, нижняя\_граница2.. верхняя\_грашца2, ..., нuжняя\_гpaнuцak .. верхняя\_границаk],

где нижняя граница и верхняя граница являются положительными целыми числами, каждая пара которых служит для обозначения отдельного замкнутого интервала целых чисел. В качестве верхней границы может использоваться специальный символ «\*», который означает произвольное положительное целое число.  Значения кратности следуют в возрастающем порядке без пропуска чисел, лежащих между нижней и верхней границами интервала.

### Способы идентификации классов анализа

Для идентификации классов и объектов используются:

* глоссарий предметной области;
* развертывание предметной области;
* анализ поведения;
* анализ вариантов.

**3.2 Проектирование логической модели данных**

Перед проектированием физической модели данных необходимо смоделировать логическую модель. Логическая модель описывает всю базу данных как единое целое. Кроме того, необходимое пользователю логическое представление данных может существенно отличаться от общей модели данных.

## 3.3 Разработка сценариев и макетов экранных форм

Для организации эффективной работы с объектами баз данных необходимо обеспечить удобный пользовательский интерфейс.

В настоящее время для приложений, разрабатываемых для среды Windows, используется три типа интерфейса: однодокументный SDI (Single-Document Interface), многодокументный MDI (Multiple-Document Interface) и интерфейс типа проводник (Explorer). Однодокументный интерфейс - это тип интерфейса, в котором предоставляется возможность работы только с одним документом в одном окне. Для работы с несколькими документами в таком интерфейсе необходимо многократно запускать приложение. Для каждого типа данных и документов требуется своя форма и, соответственно, свое приложение с интерфейсом типа SDI. Каждый раз при запуске SDI-приложения в память загружаются одни и те же данные (меню, панель и элементы управления), выполняющие одинаковые действия, что приводит к неэффективной и медленной работе запускаемых приложений.

Есть и положительные стороны приложений такого типа интерфейса - они занимают меньше места на диске и в оперативной памяти, да и на их разработку уходит гораздо меньше времени, что также немаловажно.

Приложение дипломного проекта построено на основе многодокументного интерфейса (MDI). Это позволяет разработчику проектировать одновременно несколько ЛВС или разрабатывать несколько вариантов одной и той же сети.

Главная особенность MDI заключается в том, что для этого типа интерфейса можно многократно открывать форму одного вида документа для нескольких разных по содержанию документов (например, программа Microsoft Word).

Для интерфейса такого типа характерно наличие одного главного окна (MDI-окно), которое обычно именуется родительским окном, и необходимого для работы количества подчиненных (вложенных) окон, называемых дочерними. Количество открытых дочерних окон ограничено лишь возможностями компьютера.

Родительское окно для MDI-интерфейса может быть только одно, при этом оно является контейнером для всех дочерних окон. Это означает, что при минимизации родительского окна вместе с ним минимизируются и все дочерние окна.

В свою очередь, дочерние окна могут находиться только внутри родительского, то есть при раскрытии на весь экран дочерние окна раскрываются полностью только в границах родительского окна и не могут быть вынесены или перемещены за эти границы.

В состав интерфейса MDI разработанного приложения входят следующие элементы:

* главное меню;
* панель инструментов с элементами управления;
* главное окно приложения (MDI-окно);
* дочерние окна.

**3.4 Подготовка модели к генерации кода**

Одним из наиболее важных свойств программы IBM Rational Rose является возможность генерации программного кода на нескольких языках программирования, которая может быть использована разработчиком после построения модели. Для этой цели в среде IBM Rational Rose присутствует достаточно большой выбор языков программирования и схем баз данных. Однако возможность генерации текста программы на том или ином языке программирования зависит от установленной версии IBM Rational Rose.

Общая последовательность действий, которые необходимо выполнить для генерации программного кода в среде IBM Rational Rose, состоит из следующих этапов:

* Проверка модели на отсутствие ошибок.
* Создание компонентов для реализации классов.
* Отображение классов на компоненты.
* Выбор языка программирования для генерации текста программного кода.
* Установка свойств генерации программного кода.
* Выбор класса, компонента или пакета.
* Генерация программного кода.

Главное окно:

@SuppressWarnings("ALL")  
public class Gen\_Form extends javax.swing.JFrame {  
 public Client\_Form client = new Client\_Form();  
 public Gen\_Form() {  
 initComponents();  
 }  
 public Manager\_Form manager = new Manager\_Form();  
  
  
 @SuppressWarnings("unchecked")  
 // <editor-fold defaultstate="collapsed" desc="Generated Code">  
 private void initComponents() {  
  
 button1 = new java.awt.Button();  
 button2 = new java.awt.Button();  
  
 setDefaultCloseOperation(javax.swing.WindowConstants.*EXIT\_ON\_CLOSE*);  
  
 button1.setLabel("Клиент");  
 button1.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 button1ActionPerformed(evt);  
 }  
 });  
  
 button2.setLabel("Менеджер");  
 button2.addActionListener(new java.awt.event.ActionListener() {  
 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 button2ActionPerformed(evt);  
 }  
 });  
  
 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());  
 getContentPane().setLayout(layout);  
 layout.setHorizontalGroup(  
 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*)  
 .addGroup(layout.createSequentialGroup()  
 .addContainerGap()  
 .addComponent(button1, javax.swing.GroupLayout.*PREFERRED\_SIZE*, 113, javax.swing.GroupLayout.*PREFERRED\_SIZE*)  
 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.*RELATED*, 76, Short.*MAX\_VALUE*)  
 .addComponent(button2, javax.swing.GroupLayout.*PREFERRED\_SIZE*, 110, javax.swing.GroupLayout.*PREFERRED\_SIZE*)  
 .addContainerGap())  
 );  
 layout.setVerticalGroup(  
 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*)  
 .addGroup(layout.createSequentialGroup()  
 .addContainerGap()  
 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.*LEADING*, false)  
 .addComponent(button1, javax.swing.GroupLayout.*DEFAULT\_SIZE*, 59, Short.*MAX\_VALUE*)  
 .addComponent(button2, javax.swing.GroupLayout.*DEFAULT\_SIZE*, javax.swing.GroupLayout.*DEFAULT\_SIZE*, Short.*MAX\_VALUE*))  
 .addContainerGap(javax.swing.GroupLayout.*DEFAULT\_SIZE*, Short.*MAX\_VALUE*))  
 );  
  
 pack();  
 }// </editor-fold>  
  
 private void button1ActionPerformed(java.awt.event.ActionEvent evt) {  
 client.setVisible(true);  
 }  
  
 private void button2ActionPerformed(java.awt.event.ActionEvent evt) {  
 manager.setVisible(true);  
 }  
  
 */\*\*  
 \** ***@param*** *args the command line arguments  
 \*/* public static void main(String args[]) {  
 /\* Set the Nimbus look and feel \*/  
 //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">  
 /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.  
 \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html  
 \*/  
 try {  
 for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.*getInstalledLookAndFeels*()) {  
 if ("Nimbus".equals(info.getName())) {  
 javax.swing.UIManager.*setLookAndFeel*(info.getClassName());  
 break;  
 }  
 }  
 } catch (ClassNotFoundException ex) {  
 java.util.logging.Logger.*getLogger*(Gen\_Form.class.getName()).log(java.util.logging.Level.*SEVERE*, null, ex);  
 } catch (InstantiationException ex) {  
 java.util.logging.Logger.*getLogger*(Gen\_Form.class.getName()).log(java.util.logging.Level.*SEVERE*, null, ex);  
 } catch (IllegalAccessException ex) {  
 java.util.logging.Logger.*getLogger*(Gen\_Form.class.getName()).log(java.util.logging.Level.*SEVERE*, null, ex);  
 } catch (javax.swing.UnsupportedLookAndFeelException ex) {  
 java.util.logging.Logger.*getLogger*(Gen\_Form.class.getName()).log(java.util.logging.Level.*SEVERE*, null, ex);  
 }  
 //</editor-fold>  
  
 /\* Create and display the form \*/  
 java.awt.EventQueue.*invokeLater*(new Runnable() {  
 public void run() {  
 new Gen\_Form().setVisible(true);  
 }  
 });  
 }  
  
 // Variables declaration - do not modify  
 private java.awt.Button button1;  
 private java.awt.Button button2;  
}

Классы:

class Client   
{  
 private String FIO;  
 private String Paspot;  
 private double Dohod;  
 private int ID;  
 public Bank theBank;  
 public Manager theManager;  
   
 */\*\*  
 \** ***@roseuid*** *582C9FDE01AD  
 \*/* public Client()   
 {   
 }  
 public String getFIO(){return FIO;}  
 public void setFIO(String s){FIO = s;}  
   
 public String getPaspot(){return Paspot;}  
 public void setPaspot(String s){Paspot = s ;}  
   
 public double getDohod(){return Dohod;}  
 public void setDohod(Double s){Dohod = s ;}  
   
 public int getID(){return ID;}  
 public void setID(int i){ID = i;}  
   
}

Director:

public class Director   
{  
 private int ID;  
 private String FIO;  
   
 */\*\*  
 \** ***@roseuid*** *582CE3EB0314  
 \*/* public Director()   
 {  
 }  
   
 public int getID(){return ID;}  
 public void setID(int s){ID = s;}  
   
 public void setFIO(String F,String I,String O)  
 {  
 FIO = F+" "+ I +" "+O;  
 }  
 public String setFIO()  
 {  
 return FIO;  
 }  
   
}

Dogovor:

import com.sun.javafx.scene.layout.region.Margins;  
import com.sun.org.apache.xerces.internal.impl.dv.xs.DateTimeDV;  
  
  
  
class Dogovor   
{  
 private int ID;  
 private Bank \_bank = new Bank();  
 private String Data;  
 private int Summ;  
 private Client \_clien = new Client();  
   
 */\*\*  
 \** ***@roseuid*** *582C9FD401EE  
 \*/* public Dogovor()   
 {  
 }  
 public void Create(int id, Bank bank, String date, int Summ, Client client)  
 {  
 ID = id;  
 \_bank = bank;  
 Data = date;  
 this.Summ = Summ;  
 \_clien = client;  
 }  
   
 public String PrintDogovor()  
 {  
 String rez = "Договор № "+ ID +"/n" ;   
 return rez;  
 }  
}

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Скопцов А. П. Локальная сеть в школе и малом офисе: создание и поддержка своими силами. – М.: [КУДИЦ-Пресс](http://www.knigalub.ru/publisher/141189.html), 2000. – 240 с.
2. Гук М. Аппаратные средства локальных сетей. – Спб.: Питер, 2000. – 544 c.
3. Капустин Н.М., Кузнецов П.М., Схиртладзе А.Г. Автоматизация производственных процессов в машиностроении. Учебник для втузов. – М.: Высшая школа, 2004. – 415 c.
4. [Минаев](http://www.knigalub.ru/author/127979.html) И. Я. 100% самоучитель. Локальная сеть своими руками. – М.: [Технолоджи-3000](http://www.knigalub.ru/publisher/136066.html), 2003. – 320 с.
5. Vito J. Amato Cisco Networking Essentials, Volume 2. – М.: Вильямс, 2002. – 464 с.
6. Единое окно образовательный ресурс [Электронный pecypc] Режим доступа:

http: // window.edu.ru., свободный .

1. Вишневский В.М. Теоретические основы проектирования компьютерных сетей. – М.: Техносфера, 2003. – 324 с.
2. Столингс В. Современные компьютерные сети. 2-е изд. – М.: Высшая школа, 2005. – 784 с.
3. Книга: Энциклопедия. – М.: Большая российская энциклопедия, 1999. – 800 с.
4. Ермишин П. Г. Основы экономической теории. – Спб.: Питер, 2004. – 340 с.
5. Каталог оборудованя cisco [Электронный pecypc] Режим доступа:

http: //[www.cisco.com](http://www.cisco.com), свободный.

1. Юрлов Ф.Ф. Технико-экономическая эффективность сложных радиоэлектронных систем. — М.: Сов. радио, 1980. — 280 с.
2. Юрлов Ф.Ф., Плеханова А.Ф., Маркитанов М.Ю. Выбор эффективных решений в экономике. — Н. Новгород: Изд-во НГТУ, 2004. — 115 с.
3. Сайт microsoft.com.
4. Гофман В., Хоменко А. Работа с базами данных в Delphi – Спб.: БХВ-Петербург, 2000. – 643 с.
5. Бобровский С.И. Delphi 7: Учебный курс. – Спб.: Питер, 2005. – 736 с
6. Jeff Cogswell LEARN DELPHITM 2 DATABASE PROGRAMMING TODAY!. – IDG Books Worldwide, Inc., 1996. – 689 с.
7. Фёдоров А. Г. Delphi 3.0 для всех. – М.: КомпьютерПресс, 1998. – 360 с.
8. Томас Коннолли, Каролин Бегг Базы данных. Проектирование, реализация и сопровождение. Теория и практика -- М.: Вильямс, 2003. – 1464 с Мамаев Е. SQL Server 2000. – Спб.: БХВ-Петербург, 2001. – 1480 с.
9. Кодд Е.Ф. Реляционная модель данных для больших совместно используемых банков данных. – Спб.: БХВ-Петербург, 1995. – 574 c.
10. Дейт К. Введение в системы баз данных //6-издание. - Киев: Диалектика, 1998. – 857 с.
11. Туманов В.Е., Гайфуллин Б.Н., Сгибнев В.Я. Введение в SQL для баз данных в архитектуре клиент/сервер. – Спб.: Интерфейс Пресс, 2000. – 188 с.
12. Уилсон Р. Введение в теорию графов. – М.: Мир, 1977. – 207 с.
13. Сайт cyberguru.ru.
14. [Бабаш А.В., Шанкин](http://www.bookid.info/author/3791.html) Г.П. Криптография. – М.: [СОЛОН - Р](http://www.bookid.info/publisher/779.html), 2002. – 105 с
15. Ковалев А.Н., Ковалев Ф.Н., Юрлов Ф.Ф. Выбор оптимального решения в многокритериальных задачах.- Издательство: Нижегородский государственный университет им. Н.И. Лобачевского, 2005. - 770 c.