## СОДЕРЖАНИЕ

# 1 ЛАБОРАТОРНАЯ РАБОТА №1 «РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ»

**Цели работы:**

закрепление имеющихся знаний о методах анализа и спецификации требований к информационным системам.

приобретение навыков анализа и формализации требований, предъявляемых к ИС.

приобретение навыков разработки технического задания на создание новой информационной системы.

**Задачи работы:**

1. Выявить требования к ИС.
2. Сгруппировать требования по группам: требования к системе в целом; требования к функциям (задачам), выполняемым системой; требования к видам обеспечения.
3. Определить приоритет требований (необходимые, желательные, дополнительные)
4. Выделить пользователей системы.
5. Выделить основные варианты использования системы.
6. Расписать сценарии по каждому выделенному варианту использования системы (с учетом выделенных требований) .
7. Представить их графически с помощью диаграмм прецедентов и диаграмм последовательности.
8. Выполнить анализ постановки задачи на создание ИС.
9. Выявить и сформулировать, функциональные и технические требования к информационной системе.
10. Разработать документ «Техническое задание на создание ИС», описывающий требования к ИС и содержащий другие, необходимые для разработки, сведения.

**Теоретические сведения**

Требования к программному обеспечению – это совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации. Требования могут выражаться в виде текстовых утверждений и графических моделей. Требования функционального характера определяют требуемое поведение программной системы.

Выделяют следующие виды требований функционального характера:

1. Бизнес-требования – определяют назначение ПО, описываются в документе о видении и границах программного проекта.

2. Пользовательские требования – определяют набор пользовательских задач, которые должен решать программный продукт, а также способы их решения. Пользовательские требования могут выражаться в виде фраз утверждений, сценариев использования, сценариев взаимодействия, пользовательских историй.

3. Функциональные требования – охватывают предполагаемое поведение системы, определяя действия, которые система способна выполнять. Процесс разработки требований включает в себя выполнение следующих этапов:

1) выявление требований (сбор, понимание, рассмотрение и выяснение потребностей заинтересованных лиц);

2) анализ (проверка целостности и законченности);

3) спецификация (документирование требований);

4) проверка правильности.

Разработка ТЗ включает в себя подготовку специального документа с аналогичным названием. В Техническом задании обязательно должны быть описаны:

- ограничения, риски, критические факторы, влияющие на успешность проекта, например время реакции системы на запрос является заданным ограничением, а не желательным фактором;
- совокупность условий, при которых предполагается эксплуатировать будущую систему: архитектура системы, аппаратные и программные ресурсы, предоставляемые системе, внешние условия её функционирования, состав людей и работ, которые обеспечивают бесперебойное функционирование системы;
- сроки завершения отдельных этапов, форма сдачи работ, ресурсы, привлекаемые в процессе разработки проекта, меры по защите информации;
- описание выполняемых системой функций;
- будущие требования к системе в случае её развития, например возможность работы пользователя с системой с помощью Интернета и т.п.;
- сущности, необходимые для выполнения функций системы;
- интерфейсы и распределение функций между человеком и системой;

– требования к программным и информационным компонентам ПО, требования к СУБД. Если проект предполагается реализовывать для нескольких СУБД, то требования к каждой из них, или общие требования к абстрактной (например, распределённой) СУБД и список рекомендуемых для данного проекта СУБД, которые удовлетворяют заданным условиям;
– что не будет реализовано в рамках проекта.

Разработка ТЗ ведётся в соответствии со стандартами:

– ГОСТ 34.601-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания.
– ГОСТ 34.602-89. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.

## I. Общие положения

ТЗ должно соответствовать современному уровню развития науки и техники, максимально точно отражать цели, замысел и требования к создаваемой системе и при этом не ограничивать разработчика в поиске и реализации наиболее эффективных технических, технико-экономических и других решений. В соответствии с ГОСТ 34.601-90, после согласования с Заказчиком, выполняется разработка, оформление, согласование и утверждение Технического задания на АИС (при необходимости – на части АИС). Данный стандарт также определяет состав участников проектирования и реализации проектных решений, которые участвуют в составлении и (или) согласовании ТЗ. В самом общем случае к ним относятся:

1. Организация-заказчик (пользователь), для которой создаётся АИС и которая обеспечивает финансирование, приёмку работ и эксплуатацию как по всей АИС, так и по отдельным её компонентам;

2. Организация-разработчик (генпроектировщик), осуществляющая работы по созданию АИС, представляя Заказчику совокупность научно-технических услуг на разных стадиях и этапах создания, а также разрабатывая и поставляя различные программные и технические средства АС. Данная (головная) организация может пользоваться услугами других организаций, работающих у неё на субподряде;

3. Организация-поставщик, изготавливающая и (или) поставляющая программные и технические средства по заказу Разработчика или Заказчика;

4. Организации, выполняющие строительные, электротехнические, санитарно-технические, монтажные, наладочные и другие подготовительные работы, связанные с созданием АИС.

ГОСТ 34.602-89 устанавливает порядок разработки, согласования и утверждения ТЗ на создание (развитие или модернизацию) автоматизированных систем различного назначения, а также состав и содержание указанного документа независимо от того, будет ли она работать самостоятельно или в составе другой системы. В зависимости от условий создания системы возможны различные совмещения функций заказчика, разработчика, поставщика и других организаций, участвующих в работах по созданию АИПС.

ТЗ на АИС разрабатываются на основании исходных данных.

Любые изменения к ТЗ оформляются дополнительными протоколами, подписанными заказчиком и разработчиком. Оформленные таким образом дополнения являются неотъемлемой частью ТЗ на АИС. На титульном листе ТЗ должна быть запись “Действует с …”.

## СОСТАВ И СОДЕРЖАНИЕ ТЕХНИЧЕСКОГО ЗАДАНИЯ

Рассмотрим состав ТЗ с учётом требований ГОСТ 34.602-89.

ТЗ на АИС содержит следующие разделы:

1. Общие сведения.
2. Назначение и цели создания (развития) системы.
3. Характеристика объектов автоматизации.
4. Требования к системе.
5. Состав и содержание работ по созданию системы.
6. Порядок контроля и приемки системы.
7. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу АИС в действие.
8. Требования к документированию.
9. Источники разработки.
10. Приложения.

В зависимости от вида, назначения, специфических особенностей объекта автоматизации и условий функционирования системы допускается оформлять разделы ТЗ в виде приложений, вводить дополнительные, исключать или объединять подразделы ТЗ.

Рассмотрим содержание основных разделов ТЗ с учётом требований ГОСТ 34.602-89.

### Раздел “Общие сведения”:

1. Полное наименование системы и её условное обозначение.

2. Наименование и реквизиты предприятий (объединений) разработчика и заказчика системы.

3. Перечень документов, явившихся основанием создания системы, кем и когда они утверждены.

4. Возможные сроки начала и окончания работ по созданию системы.

5. Сведения об источниках и порядке финансирования работ.

6. Порядок оформления и предъявления заказчику результатов работ по созданию системы или её частей, по изготовлению и наладке отдельных средств (технических, программных, информационных) и программно-технических комплексов системы.

**Раздел “Назначение и цели создания (развития) системы”:**

1. Под “Назначением системы” понимается вид автоматизируемых процессов (деятельности) и перечень предполагаемых к использованию объектов.

2. В пункте “Цели создания системы” приводятся наименования и требуемые значения технических, технологических, производственно-экономических и других показателей объекта автоматизации, достигаемые в результате создания АИС, указываются критерии оценки достижения целей создания системы.

**Раздел “Характеристики объекта автоматизации”:**

1. Краткие сведения об объекте автоматизации или ссылки на документы, содержащие эти данные.

2. Сведения об условиях эксплуатации объекта автоматизации.

3. Характеристики внешней среды, в которой функционирует объект автоматизации.

**Раздел “Требования к системе”** содержит подразделы с требованиями к системе в целом, функциям (задачам), выполняемым системой, видам обеспечения.

Требования к численности и квалификации персонала АИС содержат требования к численности персонала и пользователей АИС; квалификации персонала, порядку его подготовки, контроля знаний и навыков; режиму работы персонала АИС.

Требования по безопасности включают требования по обеспечению безопасности при монтаже, наладке, эксплуатации, обслуживании и ремонте технических средств системы (защита от воздействия электрического тока, электромагнитных полей, акустических шумов и т.п.), допустимым уровням освещённости, вибрационных и шумовых нагрузок.

Требования по сохранности информации содержат перечень событий: аварий, отказов технических средств (в т.ч. потерей питания) и т.п., при которых должна быть обеспечена сохранность информации в системе, а также требования к подсистеме резервного копирования и архивного хранения документов и данных.

В требования к защите информации от несанкционированного доступа включают требования, действующей в отрасли (ведомстве) заказчика.

В требования по эргономике и технической эстетике включают показатели АИС, задающие необходимое качество взаимодействия человека с машиной и комфортность условий работы персонала.

Требования к стандартизации и унификации включают показатели, устанавливающие соответствие с государственными стандартами, ведомственными и другими нормами.

В дополнительные требования могут быть включены:

- требования к оснащению системы устройствами для обучения персонала (тренажерами, другими устройствами аналогичного назначения) и документацией на них;
- требования к сервисным средствам, стендам для проверки элементов системы;
- требования к системе, связанные с особыми условиями эксплуатации;
- специальные требования по усмотрению разработчика или заказчика системы.

**Подраздел “Требования к видам обеспечения”** в зависимости от вида системы может содержать требования к математическому, информационному, лингвистическому, программному, техническому, организационному, методическому и другим видам обеспечения системы.

В части требований к математическому обеспечению системы приводятся требования к составу, области применения (ограничения) и способам использования в системе математических методов и моделей, типовых алгоритмов и алгоритмов, подлежащих разработке.

В части требований к информационному обеспечению системы приводят требования:

- к составу, структуре и способам организации фондов и машиночитаемых данных в системе;
- к информационному обмену между компонентами системы;

- к информационной совместимости со смежными системами;
- по использованию коммуникативных форматов, унифицированных документов, действующих в данной организации и (или) взаимодействующей группе организаций;
- к внутрисистемным форматам данных;
- по применению систем управления базами данных;
- к структуре процесса сбора, обработки, передачи данных в системе и представлению данных;
- к защите данных от разрушений при авариях и сбоях в электропитании системы;
- к контролю, хранению, обновлению и восстановлению данных;
- В части требований к лингвистическому обеспечению системы приводятся требования к применению в системе:
- классификаторов и тезаурусов,
- языков взаимодействия пользователей и технических средств системы,
- средств кодирования и декодирования данных,
- конверторов,
- языков ввода-вывода данных,
- языков манипулирования данными,
- способов организации диалога.

В части требований к программному обеспечению АИС приводятся общие функциональные и общесистемные требования к приобретаемым и вновь разрабатываемым программным продуктам. При этом следует предусмотреть:

- решение средствами ПО системы полного комплекса служебных и пользовательских задач;
- поддержку возможностей обработки, хранения и актуализации заданных видов документов и данных с учётом необходимых их количественных показателей;
- поддержку возможности настройки на заданные входные и выходные формы документов;
- поддержку необходимых форматов данных и средств лингвистического обеспечения;
- поддержку требований протоколов телекоммуникационного обмена данными, действующими в области функционирования АИС,

- обеспечение необходимой для создаваемой АИС скорости обработки и поиска данных,
- обеспечение требований стандартизации, унификации, эргономики, защиты информации и соответствия другим, не перечисленным в данном пункте, требованиям, включённым в другие пункты ТЗ.

В части требований к средствам технического обеспечения системы приводят требования к видам технических средств, в т.ч. к видам комплексов технических средств, программно-технических комплексов и других комплектующих изделий, допустимых к использованию в системе, а также к функциональным, конструктивным и эксплуатационным характеристикам средств технического обеспечения системы.

В части требований к организационному обеспечению приводят требования к структуре и функциям подразделений, участвующих в функционировании системы или обеспечивающих эксплуатацию; организации функционирования системы и порядку взаимодействия персонала АИС с персоналом объекта автоматизации; защите от ошибочных действий персонала системы.

В требования по обеспечению управления и контроля включают:

- перечень контролируемых параметров технологической цепи обработки входных документов и обслуживания пользователей,
- требования к регламенту обработки входных документов и обслуживания пользователей,
- требования к видам статистической обработки контролируемых данных, а также их выходным формам,
- требования к средствам формально-логического контроля.

**Раздел “Состав и содержание работ по созданию (развитию) системы”** должен содержать перечень стадий и этапов работ по созданию системы в соответствии с ГОСТ 34.601-90, сроки их выполнения, перечень организаций-исполнителей работ, ссылки на документы, подтверждающие их согласие на участие в создании системы и т.п.

**В разделе “Порядок контроля и приемки системы” указывают:**

1. Виды, состав, объём и методы испытаний системы и её составных частей (виды испытаний в соответствии с действующими нормами, распространяющимися на разрабатываемую систему);

2. Общие требования к приемке работ по стадиям (перечень участвующих организаций, и/или юридических и физических лиц, место и

сроки проведения), порядок согласования и утверждения приёмочной документации;

3. Статус приёмочной комиссии (государственная, межведомственная, ведомственная и т.п.).

**В разделе “Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие”** необходимо привести перечень основных мероприятий, которые следует выполнить при подготовке объекта автоматизации к вводу АИС в действие, и их исполнителей.

**В разделе “Требования к документированию” приводят:**

1. Согласованный разработчиком и заказчиком системы перечень подлежащих разработке комплектов и видов документов, в т.ч. выпускаемых на машинных носителях;

2. Требования по документированию комплектующих элементов межотраслевого применения в соответствии с требованиями ЕСКД и ЕСПД;

3. При отсутствии государственных стандартов, определяющих требования к документированию элементов системы, дополнительно включают требования к составу и содержанию таких документов.

Обеспечение качества проектной документации относится к возможностям средств проектирования анализировать и проверять описания и документацию на полноту и непротиворечивость, а также на соответствие принятым стандартам и правилам (включая ГОСТ, ЕСПД).

В разделе “Источники разработки” должны быть перечислены документы и информационные материалы (технико-экономическое обоснование, отчеты о законченных научно-исследовательских работах, информационные материалы на отечественные, зарубежные системы-аналоги и др.), на основании которых разрабатывалось ТЗ и которые должны быть использованы при создании системы.

В состав ТЗ на АИС включают приложения, содержащие расчёт ожидаемой эффективности системы; оценку научно-технического уровня системы; использованные при разработке ТЗ методические и наиболее важные информационные материалы из состава документов указанных в разделе “Источники разработки”.

Дополнительные рекомендации по составу и содержанию ТЗ на автоматизированные системы различного назначения и приложений к ним содержатся также в РД 50-640-87 и ГОСТ 24.602-86.

## ПРАВИЛА ОФОРМЛЕНИЯ ТЗ НА АИС

ТЗ оформляют на листах формата А4 без рамки, основной надписи и дополнительных граф к ней. Номера листов (страниц) проставляют, начиная с первого листа, следующего за титульным листом, в верхней части листа (над текстом, посередине).

На титульном листе помещают подписи заказчика, разработчика и согласующих организаций, которые скрепляют гербовой печатью. При необходимости титульный лист оформляют на нескольких страницах. Подписи разработчиков ТЗ на АИС и должностных лиц, участвующих в согласовании и рассмотрении проекта ТЗ на АИС, помещают на последнем листе.

При необходимости на титульном листе ТЗ допускается помещать установленные в отрасли коды, например: код работы, регистрационный номер ТЗ и др.

Разделы и подразделы ТЗ должны быть размещены в порядке, установленном ГОСТ 34.602-89.

Если конкретные значения показателей, норм и требований не могут быть установлены в процессе разработки ТЗ на АИС, в нём делают запись о порядке установления и согласования этих показателей, норм и требований “Окончательное требование (значение) уточняется в процессе ... и согласовывается протоколом с ... на стадии ...”. При этом в текст ТЗ на АИС изменений не вносят.

Титульный лист дополнения к ТЗ на АИС оформляют аналогично титульному листу Технического задания. Вместо наименования “Техническое задание” пишут “Дополнение 1... к ТЗ на АИС...”.

На следующих листах дополнения к ТЗ на АИС помешают основание для изменения, содержание изменения и ссылки на документы, в соответствии с которыми вносятся эти изменения.

При изложении текста дополнения к ТЗ следует указывать номера соответствующих пунктов, подпунктов, таблиц основного ТЗ и применять слова: “заменить”, “дополнить”, “исключить”, “изложить в новой редакции”.

Реально сложившаяся практика проектирования АИС предусматривает следующие этапы (стадии) проектирования:

1. Предпроектное обследование, включающее:

- краткую характеристику исходного состояния объекта автоматизации и среды, в которой он функционирует;
- указание основных целей и перечень задач автоматизации;

- описание укрупнённой организационно-функциональной структуры выбранного варианта (вариантов) построения создаваемой системы;
- технико-экономическое обоснование системы;
- укрупнённое описание и основные требования к средствам информационного и лингвистического обеспечения;
- перечень и общие требования к средствам программно-аппаратного обеспечения;
- перечень и укрупнённую характеристику этапов создания системы, сроки их выполнения;
- исходную оценку стоимостных показателей выполнения работ;

2. Техническое задание на систему в целом и (или) её основные составные части (подсистемы, программно-технические комплексы и средства, отдельные задачи и т.д.), выполненное в соответствии с ГОСТ 34.601-90.

3. Эскизное проектирование. При проектировании программного обеспечения системы Эскизный проект должен содержать полную спецификацию разрабатываемых программ.

4. Опытная и промышленная эксплуатация разработанной АИС.

По результатам опытной, а порой и промышленной эксплуатации системы оптимизируют работу её составляющих и взаимодействие между ними, в том числе с учётом выполнения работниками действий, определённых для них Техническим заданием и Проектом. Это не означает, что с течением времени цели, задачи, способы их достижения, используемые технические и программные средства остаются неизменными.

В процессе реального проектирования затруднительно осуществить все рекомендации, связанные с реализацией наиболее эффективных решений. В связи с этим оценка полученных результатов осуществляется методом сравнения основных показателей с аналогичными, реализованными в существующих проектах. Формирование нескольких вариантов проектных решений и выбор наилучших из них позволяет достигать оптимальных решений. Такие проекты принято называть квазиоптимальными (т.е. лучшими из числа ранее созданных аналогичных проектов).

**Порядок выполнения лабораторной работы**

1. Ознакомиться с предложенным вариантом описания предметной области. Проанализировать предметную область, уточнив и дополнив ее,

руководствуясь собственным опытом, консультациями и другими источниками.

2. Выполнить структурное разбиение предметной области на отдельные подразделения (отделы, службы, подсистемы, группы и пр.) согласно выполняемым ими функциям.

3. Определить задачи и функции системы в целом и функции каждого подразделения (подсистемы).

4. Выполнить словесное описание работы каждого подразделения (подсистемы), алгоритмов и сценариев выполнения ими отдельных работ.

5. Изучить требования к структуре и содержанию документа «Техническое задание на создание ИС». Составить план документа.

6. Сформулировать цели и задачи создания ИС. Охарактеризовать вид ИС, её назначение, используемые в работе системы данные. Сформулировать концептуальные требования к ИС.

7. Дать характеристику типового объекта автоматизации (организации, предприятия) для которого создаётся и на котором будет внедрена ИС. Описать автоматизируемые бизнес-процессы.

8. Сформулировать требования к системе в целом. Описать структуру ИС. Перечислить функциональные подсистемы.

9. Сформулировать функциональные требования. Описать требования к функциям и задачам, выполняемым системой. Описать назначение и состав функций каждой из подсистем.

10. Описать предметную область. Разработать концептуальную модель данных предметной области. Сформулировать требования к информационному обеспечению системы.

11. Сформулировать требования к программному обеспечению системы. Описать требования к пользовательскому интерфейсу. Сформулировать технические требования к реализации и режимам работы ИС.

12. Используя полученные результаты, подготовить документ «Техническое задание на создание ИС», включающий в себя полное описание концептуальных, функциональных и технических требований к создаваемой системе.

**Контрольные вопросы**

1. Требования к информационной системе.
2. Методы анализа и спецификации требований.
3. Анализ предметной области.

4. Разработка технического задания на создание информационной системы.
5. Концептуальные требования.
6. Функциональные требования.
7. Технические требования.
8. Технологии и методологии управления требованиями.

# 2 ЛАБОРАТОРНАЯ РАБОТА №2 «МОДЕЛИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ»

**Цели работы:**

закрепление имеющихся знаний о технологиях и методологиях моделирования информационных систем;

приобретение навыков объектно-ориентированного анализа, моделирования и проектирования ИС;

приобретение навыков разработки моделей с применением унифицированного языка моделирования UML.

**Задачи работы:**

1. Разработать модель прецедентов, описывающую бизнес-процессы организации с точки зрения внешнего пользователя (клиента) и отражающую взгляд на деятельность организации извне. Результатом моделирования являются диаграммы деятельности и диаграммы прецедентов.

2. Разработать модель бизнес-объектов, описывающую выполнение бизнес-процессов организации ее внутренними исполнителями. Основными компонентами модели являются внешние и внутренние исполнители. Результатом моделирования являются диаграммы последовательности.

3. Разработать концептуальную модель данных, описывающую объекты предметной области и связи между ними. Результатом моделирования являются диаграммы классов и диаграммы объектов.

4. Разработать описание требований к системе. Результатом является исчерпывающий перечень функций, которые должны быть реализованы в системе, и подробное описание необходимой реализации этих функций.

5. Разработка моделей базы данных и приложений, представляющих собой детальное описание проекта базы данных и клиентских приложений информационной системы. Результатом моделирования являются диаграммы компонентов и диаграммы базы данных.

**Теоретические сведения**

Разработка информационной системы невозможна без ее тщательного проектирования: слишком велико влияние этого шага на

последующие этапы жизненного цикла информационной системы, в основе которой лежит создаваемая база данных.

Для целей проектирования информационной системы могут быть использованы следующие виды моделей:

- методология функционального моделирования работ SADT (Structured Analysis and Design Technique);
- диаграммы потоков данных DFD (Data Flow Diagrams);
- методология объектного проектирования на языке UML (UML-диаграммы).

Методология SADT (Structured Analisys and Design Technique - технология структурного анализа и проектирования) разработана Дугласом Т. Россом и является одной из самых известных и широко используемых методик проектирования. Новое название методики, принятое в качестве стандарта, -IDEF0 (Icam DEFinition) является частью программы ICAM (Integrated Computer -Aided Manufacturing - интегрированная компьютеризация производства).

Процесс моделирования в SADT включает сбор информации об исследуемой области, документирование полученной информации, представление ее в виде модели и уточнение модели. Кроме того, этот процесс подсказывает вполне определенный путь выполнения согласованной и достоверной структурной декомпозиции, что является ключевым моментом в квалифицированном анализе системы.

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

Основными понятиями методологии функционального моделирования работ являются:

*Работы (activity)* - поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты. На диаграмме работы изображаются прямоугольниками.

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

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

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

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

*Вызов (Call)* представляет собой взаимодействие одной модели работ с другой (стрелка, исходящая из нижней грани).

Различают в IDEF0 пять типов связей работ.

*Связь по входу (input-output)* имеет место, когда выход вышестоящей работы направляется на вход следующей работы.

*Связь по управлению (output-control)* обозначает ситуацию, когда выход вышестоящей работы направляется на управление следующей работы. Связь показывает доминирование вышестоящей работы.

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

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

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

Из перечисленных блоков строятся диаграммы работ, описывающие принципы функционирования системы.

Диаграммы потоков данных (Data Flow Diagrams - DFD) используются для описания движения документов и обработки информации как дополнение к IDEF0. В отличие от IDEF0, где система рассматривается как взаимосвязанные работы, стрелки в DFD показывают лишь то, как объекты (включая данные) движутся от одной работы к другой. DFD отражает функциональные зависимости значений, вычисляемых в системе, включая входные значения, выходные значения и внутренние хранилища данных. DFD - это граф, на котором показано движение значений данных от их источников через преобразующие их процессы к их потребителям в других объектах.

DFD содержит процессы, которые преобразуют данные, потоки данных, которые переносят данные, активные объекты, которые

производят и потребляют данные, и хранилища данных, которые пассивно хранят данные.

Диаграмма потоков данных содержит:

- процессы, которые преобразуют данные;
- потоки данных, переносящие данные;
- активные объекты, которые производят и потребляют данные;
- хранилища данных, которые пассивно хранят данные.

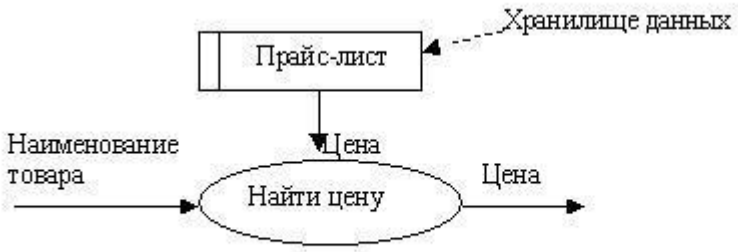
Процесс DFD преобразует значения данных и изображается в виде эллипса, внутри которого помещается имя процесса.


Поток данных соединяет выход объекта (или процесса) с входом другого объекта (или процесса) и представляет собой промежуточные данные вычислений. Поток данных изображается в виде стрелки между производителем и потребителем данных, помеченной именами соответствующих данных. Дуги могут разветвляться или сливаться, что означает соответственно разделение потока данных на части либо слияние объектов.

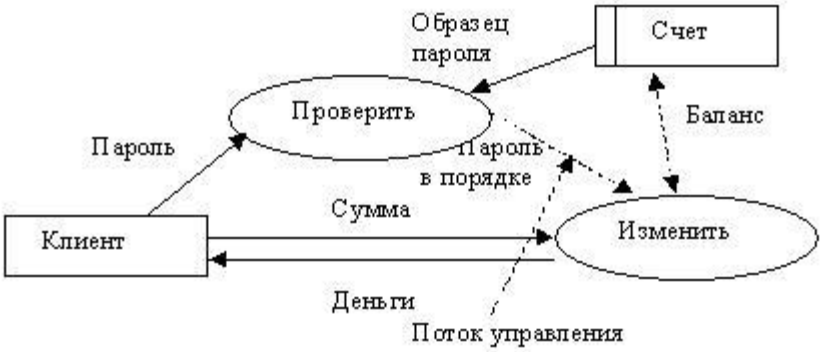
Активным объектом является объект, который обеспечивает движение данных, поставляя или потребляя их. Хранилище данных - это пассивный объект в составе DFD, в котором данные сохраняются для последующего доступа.


**Хранилища данных.** Хранилище данных - это пассивный объект в составе DFD, в котором данные сохраняются для последующего доступа. Хранилище данных допускает доступ к хранимым в нем данным в

порядке, отличном от того, в котором они были туда помещены. Агрегатные хранилища данных, как, например, списки и таблицы, обеспечивают доступ к данным в порядке их поступления, либо по ключам.




**Потоки управления.** DFD показывает все пути вычисления значений, но не показывает в каком порядке значения вычисляются. Решения о порядке вычислений связаны с управлением программой, которое отражается в динамической модели. Эти решения, вырабатываемые специальными функциями, или предикатами, определяют, будет ли выполнен тот или иной процесс, но при этом не передают процессу никаких данных, так что их включение в функциональную модель необязательно. Тем не менее, иногда бывает полезно включать указанные предикаты в функциональную модель, чтобы в ней были отражены условия выполнения соответствующего процесса.Функция, принимающая решение о запуске процесса, будучи включенной в DFD, порождает в диаграмме поток управления и изображается пунктирной стрелкой.




Первым шагом при построении иерархии DFD является построение контекстных диаграмм. Обычно при проектировании относительно простых информационных систем строится единственная контекстная диаграмма со звездообразной топологией, в центре которой находится так называемый главный процесс, соединенный с приемниками и источниками

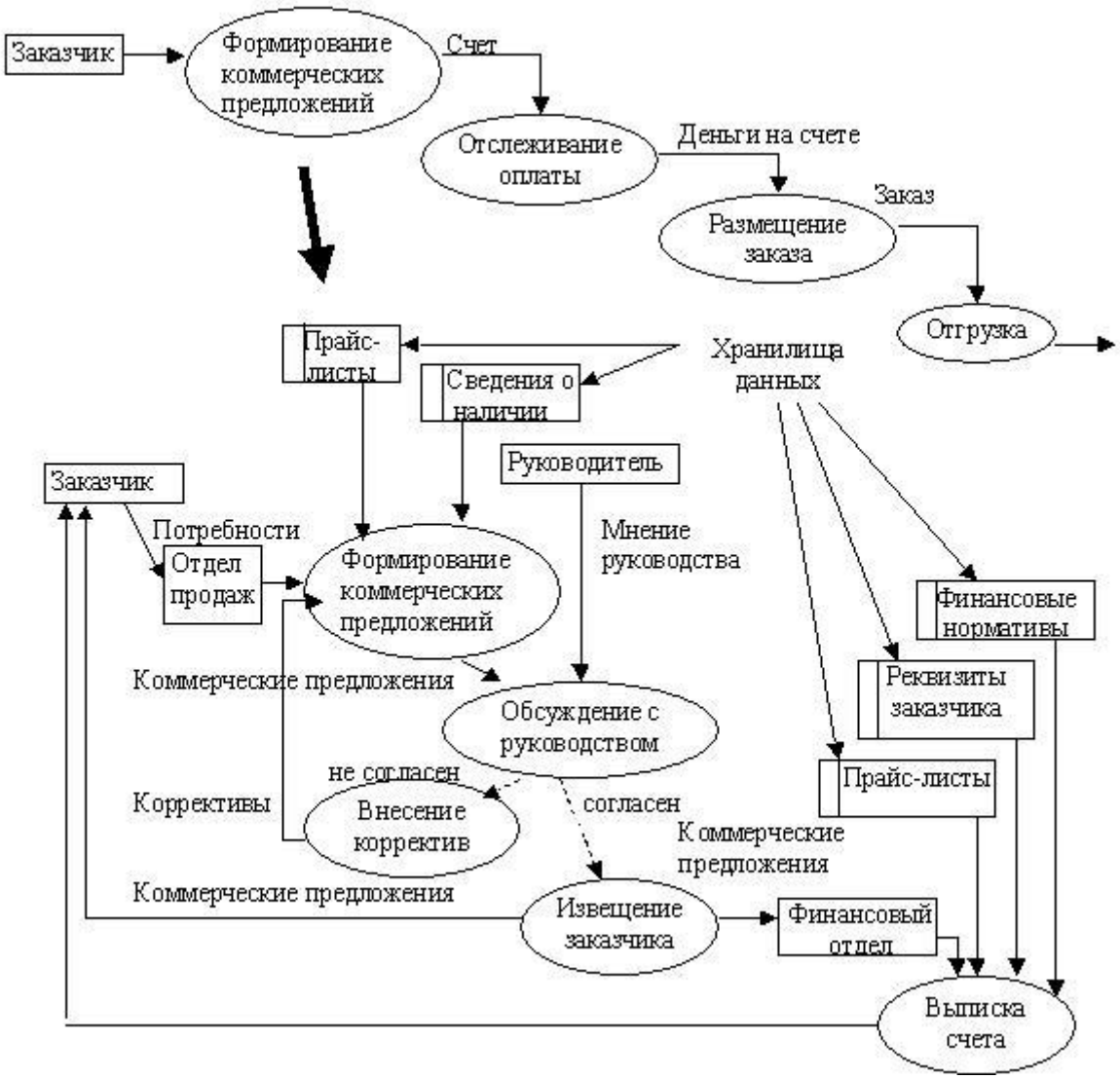
информации, посредством которых с системой взаимодействуют пользователи и другие внешние системы.

Если же для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и, кроме того, главный единственный процесс не раскрывает структуры распределенной системы.

Для сложных информационных систем строится иерархия контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит не главный единственный процесс, а набор подсистем, соединенных потоками данных. Контекстные диаграммы следующего уровня детализируют контекст и структуру подсистем.

При построении иерархии DFD переходить к детализации процессов следует только после определения содержания всех потоков и накопителей данных, которое описывается при помощи структур данных. Структуры данных конструируются из элементов данных и могут содержать альтернативы, условные вхождения и итерации. Условное вхождение означает, что данный компонент может отсутствовать в структуре. Альтернатива означает, что в структуру может входить один из перечисленных элементов. Итерация означает вхождение любого числа элементов в указанном диапазоне. Для каждого элемента данных может указываться его тип (непрерывные или дискретные данные). Для непрерывных данных может указываться единица измерения (кг, см и т.п.), диапазон значений, точность представления и форма физического кодирования. Для дискретных данных может указываться таблица допустимых значений.

Ниже приведена диаграмма потоков данных верхнего уровня с ее последующим уточнением:




Унифицированный язык моделирования UML (Universal Modeling Language) – это графический язык моделирования общего назначения, предназначенный для спецификации, визуализации, проектирования и документирования всех компонентов, создаваемых при разработке программных систем. Язык UML является объектно-ориентированным языком.

Его использование основывается на понимании общих принципов объектно-ориентированного анализа и проектирования:

1. Принцип абстрагирования предписывает включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций.

2. Принцип многомодельности означает, что никакое единственное представление системы не является достаточным для адекватного выражения всех ее особенностей.

3. Принцип иерархического построения моделей сложных систем предписывает рассматривать процесс построения моделей на разных

уровнях абстрагирования или детализации в рамках фиксированных представлений. Диаграмма UML – это графическое представление набора элементов, изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями), используемое для визуализации системы с разных точек зрения. Диаграммы UML используются для описания различных аспектов функционирования и структуры ИС на разных стадиях создания системы и, соответственно, на разных этапах моделирования: концептуального, логического и физического.

UML был разработан компанией Rational Software с целью создания наиболее оптимального и универсального языка для описания как предметной области, так и конкретной задачи в программировании. Визуальное моделирование в UML можно представить как некоторый процесс поуровневого спуска от наиболее обшей и абстрактной концептуальной модели системы к логической, а затем и к физической модели соответствующей системы. Любая задача, таким образом, моделируется при помощи некоторого набора иерархических диаграмм, каждая из которых представляет собой некоторую проекцию системы.

Диаграмма (Diagram) - это графическое представление множества элементов. Чаще всего она изображается в виде связного графа с вершинами (сущностями) и ребрами (отношениями).

В UML определено восемь видов диаграмм:

- диаграмма прецедентов (Use case diagram) - диаграмма поведения, на которой показано множество прецедентов и актеров, а также отношения между ними;
- диаграмма деятельности (Activity diagram) - диаграмма поведения, на которой показан автомат и подчеркнуты переходы потока управления от одной деятельности к другой;
- диаграмма классов (Class diagram) - структурная диаграмма, на которой показано множество классов, интерфейсов, кооперацией и отношения между ними;
- диаграмма состояний (Statechart diagram) - диаграмма поведения, на которой показан автомат и подчеркнуто поведение объектов с точки зрения порядка получения событий;
- диаграмма последовательностей (Sequence diagram) - диаграмма поведения, на которой показано взаимодействие и подчеркнута временная последовательность событий;
- диаграмма кооперации (Collaboration diagram) - диаграмма поведения, на которой показано взаимодействие и подчеркнута

структурная организация объектов, посылающих и принимающих сообщения;

- диаграмма компонентов (Component diagram) - диаграмма поведения, на которой показан автомат и подчеркнуто поведение объектов с точки зрения порядка получения событий
- диаграмма развертывания (Deployment diagram) - структурная диаграмма, на которой показаны узлы и отношения между ними.

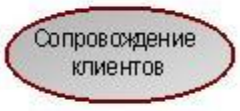
**Диаграмма прецедентов**

Диаграммы прецедентов применяются для моделирования вида системы с точки зрения внешнего наблюдателя. На диаграмме прецедентов графически показана совокупность прецедентов и Субъектов, а также отношения между ними.

Рассмотрим основные элементы диаграммы прецедентов.

Субъект (actor) - **любая сущность, взаимодействующая с системой извне [2] или множество логически связанных ролей, исполняемых при взаимодействии с прецедентами [1].** Стандартным графическим обозначением субъекта на диаграммах является фигурка "человечка", под которой записывается конкретное имя субъекта, однако субъектом может быть не только человек, но и техническое устройств о, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик.


Прецеденты (use case) - **это описание множества последовательностей действий (включая их варианты), которые выполняются системой для того, чтобы актер получил результат, имеющий для него определенное значение.** При этом ничего не говорится о том, каким образом будет реализовано взаимодействие субъектов с системой, это одна из важнейших особенностей разработки прецедентов. Стандартным графическим обозначением прецедента на диаграммах является эллипс, внутри которого содержится краткое название прецедента или имя в форме глагола с пояснительными словами.



Сущность концепции прецедентов подразумевает несколько важных пунктов.

- Прецедент представляет собой **завершенный** фрагмент функциональных возможностей (включая основной поток логики управления, его любые вариации (подпотоки) и исключительные условия (альтернативные потоки)).
- Фрагмент **внешне наблюдаемых функций** (отличных от внутренних функций).
- **Ортогональный** фрагмент функциональных возможностей (прецеденты могут при выполнении совместно использовать объекты, но выполнение каждого прецедента независимо от других прецедентов).
- Фрагмент функциональных возможностей, **инициируемый субъектом.** Будучи инициирован, прецедент может взаимодействовать с другими субъектоми. При этом возможно, что субъект окажется только на принимающем конце прецедента, опосредованно инициированного другим субъектом.
- Фрагмент функциональных возможностей, который предоставляет субъекту **ощутимый полезный результа**т (и этот результат достигается в пределах одного прецедента).

Между субъектами и прецедентами - основными компонентами диаграммы прецедентов - могут существовать различные отношения, которые описывают взаимодействие экземпляров одних субъектов и прецедентов с экземплярами других субъектов и прецедентов. В языке UML имеется несколько стандартных видов отношений между субъектами и прецедентами:

*Отношение ассоциации (association)* - **определяет наличие канала связи между экземплярами субъекта и прецедента (или между экземплярами двумх субъектов).** Обозначается сплошной линией, возможно наличие стрелки и указание мощности связи.

*Отношение расширения (extend)* - **определяет взаимосвязь экземпляров отдельного прецедента с более общим прецедентом, свойства которого определяются на основе способа совместного объединения данных экземпляров.** Обозначается пунктирной линией со стрелкой, направленной от того прецедента, который является расширением для исходного прецедента, и помечается ключевым словом "extend" ("расширяет").

*Отношение включения (include)* - **указывает, что некоторое заданное поведение для одного прецедента включает в качестве составного компонента поведение другого прецедента.** Данное отношение является направленным бинарным отношением в том смысле, что пара экземпляров прецедентов всегда упорядочена в отношении включения. Обозначается пунктирной линией со стрелкой, направленной от базового прецедента к включаемому, и помечается ключевым словом "include" ("включает").

*Отношение обобщения (generalization)* - **служит для указания того факта, что некоторый прецедент А может быть обобщен до прецедента В. В этом случае прецедент А будет являться специализацией прецедента В.** При этом В называется предком или родителем по отношению к А, а прецедент А - потомком по отношению к прецеденту В. Следует подчеркнуть, что потомок наследует все свойства и поведение своего родителя, а также может быть дополнен новыми свойствами и особенностями поведения. Графически данное отношение обозначается сплошной линией со стрелкой в форме незакрашенного треугольника, которая указывает на родительский прецедент.

**Пример. Магазин видеопродукции.**

Магазин продает видеокассеты, DVD-диски, аудио-кассеты, CD-диски и т.д. , а также предлагает широкой публике прокат видеокассет и DVD-дисков.

Товары поставляются несколькими поставщиками. Каждая партия товара предварительно заказывается магазином у некоторого поставщика и доставляется после оплаты счета. Вновь поступивший товар маркируется, заносится в базу данных и затем распределяется в торговый зал или прокат.

Видеоносители выдаются в прокат на срок от 1 до 7 дней. При прокате с клиента взимается залоговая стоимость видеоносителя. При возврате видеоносителя возвращается залоговая стоимость минус сумма за прокат. Если возврат задержан менее чем на 2 дня, взимается штраф в размере суммы за прокат за 1 день* кол-во дней задержки. При задержке возврата более чем на 2 дня - залоговая сумма не возвращается. Клиент может взять одновременно до 4 видеоносителей (прокат-заказ ). На каждый видеоноситель оформляется квитанция.

Клиенты могут стать членами видео-клуба и получить пластиковые карточки. С членов клуба не берется залог (за исключением случая описанного ниже), устанавливается скидка на ставку проката и покупку

товаров. Члены клуба могут делать предварительные заказы на подбор видеоматериалов для проката или покупки.

Каждый член клуба имеет некоторый стутус. Первоначально - "новичок". При возврате всрок 5 прокат-заказов , статус меняется на "надежный". При задержке хотя бы одного видеоносителя более чем на 2 дня , статус "новичок" или "надежный" меняется на "ненадежный" и клиенту высылается предупреждение. При повторном нарушении правил статус меняется на "нарушитель". Члены клуба со статусом "надежный" могут брать до 8 видеоносителей единовременно, все остальные - 4. С членов клуба со статусом "нарушитель" берется залоговая сумма.

Клиенты при покупке товара или получении видеоносителя в прокат могут расплачиваться наличными или кредитной картой.

Прокатные видеоносители через определенное количество дней проката списываются и утилизируются по акту. Списываются также товары и прокатные видеоносители, у которых обнаружился брак.

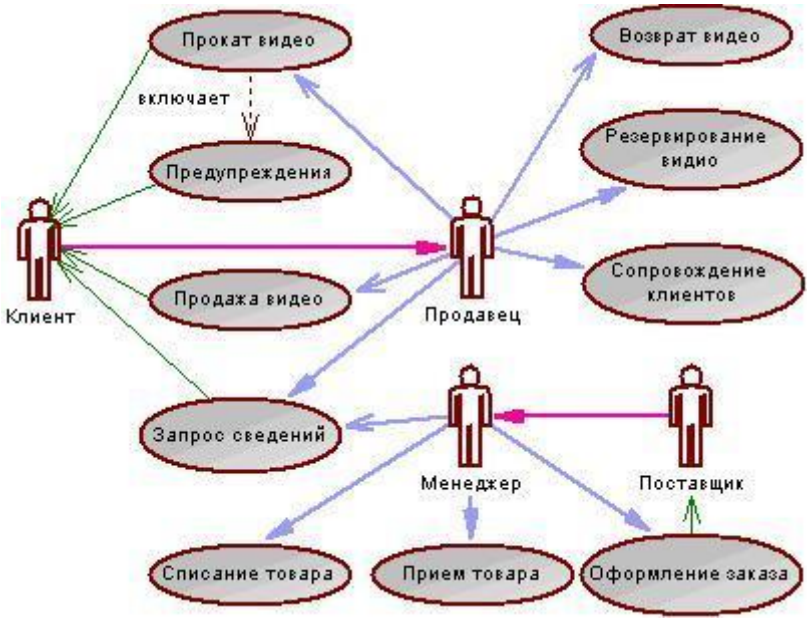
На рисунке ниже приведена диаграмма прецедентов для рассматриваемого примера. В этом примере можно выделить следующие субъекты и соответствующие им прецеденты:

- **Менеджер** - изучает рынок видеопродукции, анализирует продажи (прецедент "Запрос сведений"), работает с поставщиками: составляет заявки на поставки товара (прецедент "Оформление заказа" ), оплачивает и принимает товар (прецедент "Прием товара"), списывает товар (прецедент "Списание товара").
- **Продавец** - работает с клиентами: продает товар (прецедент "Продажа видео"), оформляет членство в клубе (прецедент "Сопровождение клиентов"), резервирует (прецедент "Резервирование видио"), выдает в прокат (прецедент "Прокат видео") и принимает назад видеоносители (прецедент "Возврат видео"), отвечает на вопросы клиента (прецедент "Запрос сведений").
- **Поставщик** - оформляет документы для оплаты товара (прецедент "Оформление заказа"), поставляет товар (прецедент "Прием товара"))
- **Клиент** - покупает(прецедент "Продажа видео"), берет на прокат и возвращает видеоносители (прецеденты "Прокат видео" и "Возврат видео"), вступает в клуб (прецедент "Сопровождение клиентов"), задает вопросы (прецедент "Запрос сведений").

Последние два субъекта **Поставщик и Клиент** не будут иметь непосредственного доступа к разрабатываемой системе (второстепенные субъекты), однако именно они являются основным источником событий,

инициализирующих прецеденты, и получателями результата работы прецедентов

От прецедента "Прокат видео" к прецеденту "Предупреждения" установлено отношение включения на том основании, что каждый выданный видеоноситель должен быть проверен на своевременный возврат и, в случае необходимости, выдано предупреждение клиенту.




Дальнейшее развитие модели поведения системы предполагает спецификацию прецедентов. Для этого традиционно используют два способа. Первый - описание с помощью текстового документа. Такой документ описывает, что должна делать система, когда субъект инициировал прецедент. Типичное описание содержит следующие разделы:

- Краткое описание
- Участвующие субъекты
- Предусловия, необходимые для инициирования прецедента
- Поток событий (основной и, возможно, подпотоки, альтернативный
- Постусловия, определяющие состояние системы, по достижении которого прецедент завершается.

Диаграммы деятельности (Activity diagram), называемые также диаграммами активности или диаграммами видов деятельности, были введены в язык UML сравнительно недавно. Диаграмма деятельности - это, по существу, блок-схема, которая показывает, как поток управления переходит от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Результат может привести к изменению состояния системы или возвращению некоторого значения.

Диаграмма деятельности отличается от традиционной блок-схемы

- более высоким уровнем абстракции;
- возможностью представления с помощью диаграмм деятельности управления параллельными потоками наряду с последовательным управлением.

Основными направлениями использования диаграмм деятельности являются

- визуализация особенностей реализации операций классов;
- отображение внутрисистемной точки зрения на прецедент.

В последнем случае диаграммы деятельности применяют для описания шагов, которые должна предпринять система после того, как инициирован прецедент

Разработка диаграммы деятельности преследует цели:

- детализировать особенности алгоритмической и логической реализации выполняемых системой операций и прецедентов;
- выделить последовательные и параллельные потоки управления;
- подготовить детальную документацию для взаимодействия разработчиков системы с ее заказчиками и проектировщиками.

Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия или состояния деятельности, а дугами - переходы от одного состояния действия/деятельности к другому. Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояния (на практике иногда можно видеть несколько конечных состояний на одной диаграмме, но это одно и тоже состояние, изображенное несколько раз для лучшей читабельности диаграммы). Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз. В этом случае начальное состояние будет изображаться в верхней части диаграммы, а конечное - в ее нижней части.

Рассмотрим основные элементы диаграммы деятельности.

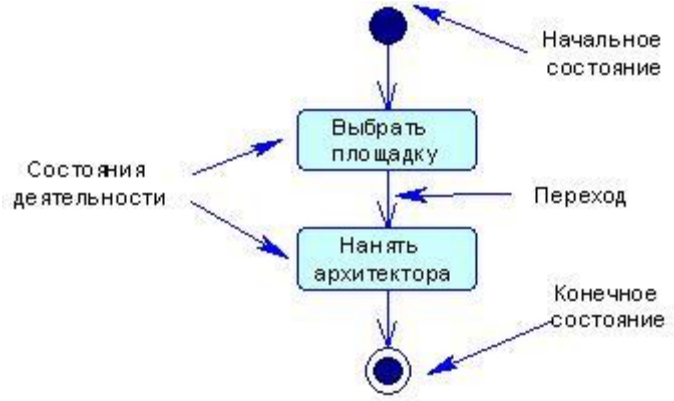
*Состояние деятельности (Activity, Process)* - **это продолжающийся во времени неатомарный шаг вычислений в автомате.** Состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время.

*Состояния действия (action state*) - **состояние, которое представляет вычисление атомарного действия,** как правило - вызов операции. Состояния действия не могут быть подвергнуты декомпозиции. Они атомарны, то есть внутри них могут происходить различные события, но выполняемая в состоянии действия работа не может быть прервана. И наконец, обычно предполагается, что длительность одного состояния действия занимает неощутимо малое время. Действие может заключаться в вызове другой операции, посылке сигнала, создании или уничтожении объекта либо в простом вычислении - скажем, значения выражения.

Состояния деятельности и состояния действия имеют одинаковое стандартное графическое обозначение - прямоугольник с закругленными краями. Внутри такого символа записывают произвольное выражение (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.

*Начальное и конечное состояния* на диаграммах деятельности изображаются как закрашенный кружок и закрашенный кружок внутри окружности, соответственно.

Ниже приведен пример диаграммы деятельности:



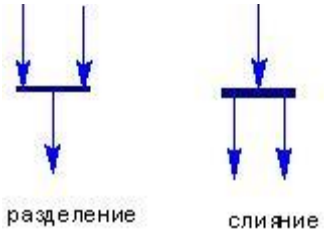

*Переход (Transitions)* - отношение между двумя состояниями, показывающее, что объект, находящийся в первом состоянии, должен выполнить некоторые действия и перейти во второе состояние. Когда действие или деятельность в некотором состоянии завершается, поток управления сразу переходит в следующее состояние действия или деятельности. Для описания этого потока и используются переходы, показывающие путь из одного состояния действия или деятельности в другое. В UML переход представляется простой линией со стрелкой.

*Ветвления.* Простые последовательные переходы встречаются наиболее часто, но их одних недостаточно для моделирования любого потока управления. Как и в блок-схему, в диаграмму деятельности может

быть включено *ветвление или множественный переход со сторожевыми условиями.* Ветвление описывает различные пути выполнения в зависимости от значения некоторого булевского выражения. Графически точка ветвления представляется ромбом. В точку ветвления может входить ровно один переход, а выходить - два или более. Для каждого исходящего перехода задается булевское выражение, которое вычисляется только один раз при входе в точку ветвления. Ни для каких двух исходящих переходов сторожевые условия не должны одновременно принимать значение "истина", иначе поток управления окажется неоднозначным. Но эти условия должны покрывать все возможные варианты, иначе поток остановится.

*Разделения и слияния.* Простые и ветвящиеся последовательные переходы в диаграммах деятельности используются чаще всего. Однако часто возникает потребность изображения параллельных потоков, и это особенно характерно для моделирования бизнес-процессов. В UML для обозначения разделения и слияния таких параллельных потоков выполнения используется синхронизационная черта, которая рисуется в виде жирной вертикальной или горизонтальной линии. При этом *разделение (concurrent fork)* имеет один входящий переход и несколько выходящих, *слияние (concurrent join)*, наоборот, имеет несколько входящих переходов и один выходящий.

Пример разделения и слияния потоков приведен ниже:

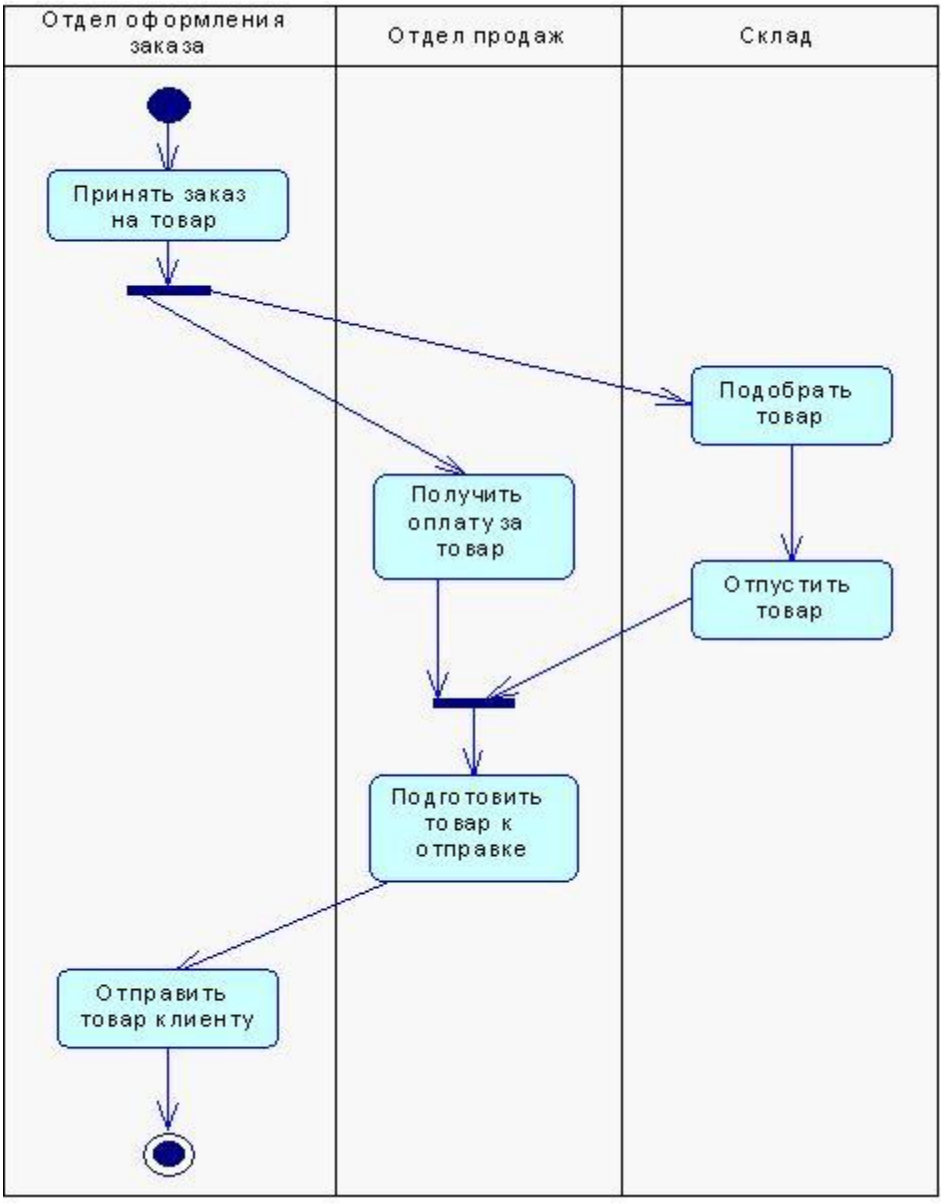



*Дорожки.* При моделировании течения бизнес-процессов иногда бывает полезно разбить состояния деятельности на диаграммах деятельности на группы, каждая из которых представляет отдел компании, отвечающий за ту или иную работу. В UML такие группы называются дорожками (Swimlanes), поскольку визуально каждая группа отделяется от соседних вертикальной чертой, как плавательные дорожки в бассейне (см. рис. 3). Дорожки - это разновидность пакетов, описывающие связанную совокупность работ.

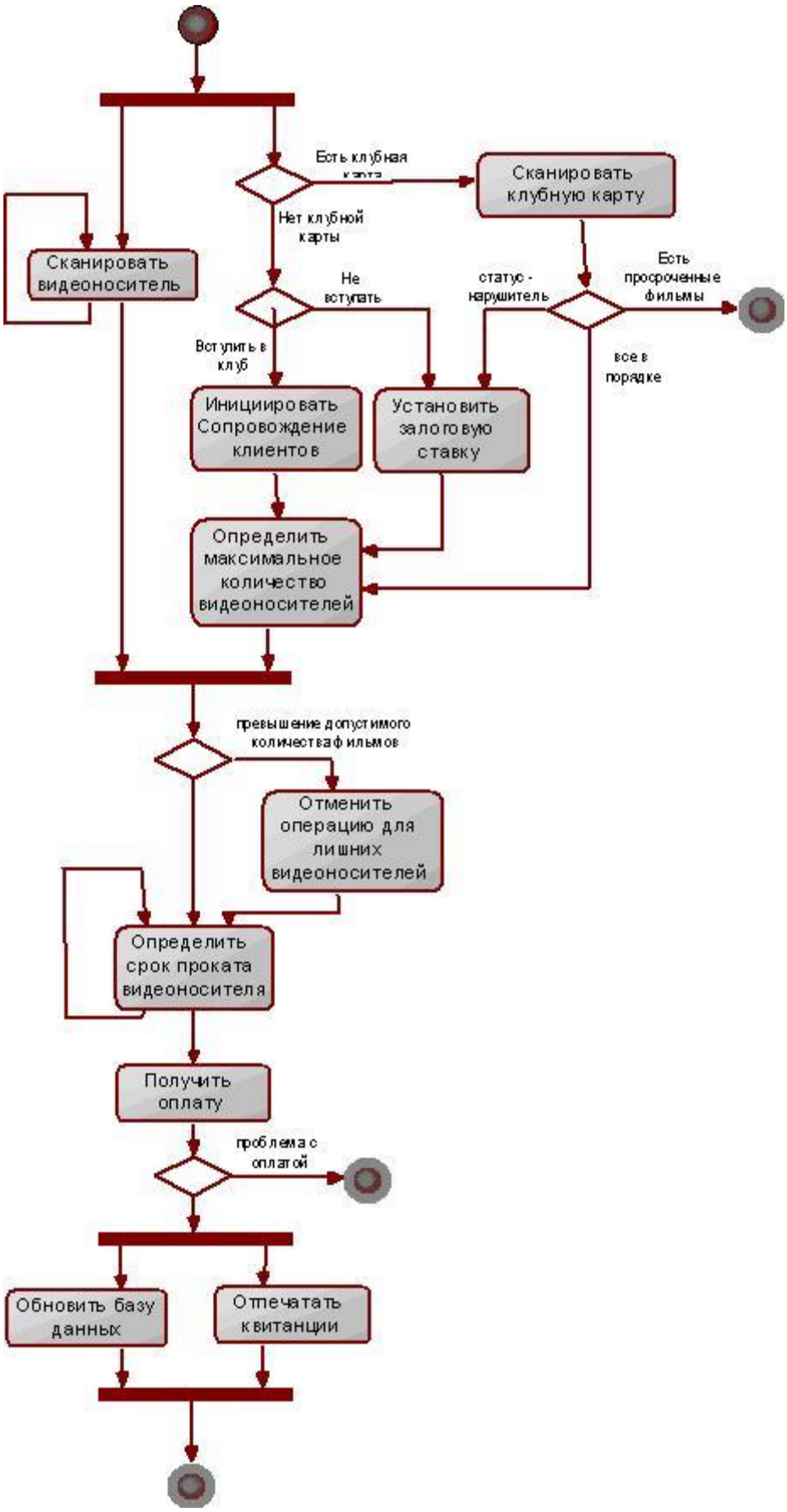
Каждой присутствующей на диаграмме дорожке присваивается уникальное имя. Никакой глубокой семантики дорожка не несет, разве что может отражать некоторую сущность реального мира. Каждая дорожка

представляет сферу ответственности за часть всей работы, изображенной на диаграмме. На диаграмме деятельности, разбитой на дорожки, каждая деятельность принадлежит ровно одной дорожке, но переходы могут пересекать границы дорожек.

Имеется некоторая связь между дорожками и параллельными потоками выполнения. Концептуально деятельность внутри каждой дорожки обычно - но не всегда - рассматривается отдельно от деятельности в соседних дорожках. Это разумно, поскольку в реальном мире подразделения организации, представленные дорожками, как правило, независимы и функционируют параллельно.




На следующем рисунке приведена диаграмма деятельности прецедента "Прокат видео" для рассмотренного в лабораторной работе 4 примера "Магазин видеопродукции"




**Порядок выполнения лабораторной работы**

1. Ознакомиться с методологией структурного моделирования работ.

2. Построить серию диаграмм работ для всей информационной системы в целом и для отдельных сценариев работ, отражающих логику и взаимоотношение подразделений (подсистем).

3. Ознакомиться с методологией диаграмм потоков данных.

4. Построить серию диаграмм потоков данных для отдельных сценариев работ, отражающих логику и взаимоотношение подразделений (подсистем).

5. Ознакомиться с методологией моделирования прецедентов на основе языка UML.

6. Построить диаграмму прецедентов для своей предметной области.

7. Описать несколько (2-3) прецедентов.

8. Ознакомиться с методологией моделирования деятельности на основе языка UML.

9 Построить диаграммы деятельности для каждого прецедента присутствующего на диаграмме прецедентов.

**Контрольные вопросы**

1. Моделирование информационных систем.
2. Виды моделей.
3. Структурные модели ИС.
4. Объектно-ориентированный анализ и проектирование.
5. Технологии, языки и средства моделирования.
6. Язык унифицированного моделирования UML.
7. Диаграммы языка UML: структурные диаграммы, диаграммы поведения, диаграммы взаимодействия.
8. Инструментальные средства моделирования ИС.
9. Применение UML при проектировании ИС.

# 3 ЛАБОРАТОРНАЯ РАБОТА №3 «ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ»

**Цель работы:**

1. Закрепить имеющиеся знания о базах данных.

2. Изучить методологию проектирования базы данных как основы информационной системы.

**Задачи работы**

1. Приобрести навыки анализа и формализованного описания заданной предметной области.

2. Приобрести навыки разработки проекта базы данных с учётом её использования в составе некоторой информационной системы.

3. Научиться строить инфологическую, даталогическую, физическая модели, создаваемые в процессе проектирования баз данных:

**Теоретические сведения**

**База данных** (БД) – это совокупность данных, отображающая состояние объектов и их отношения в рассматриваемой предметной области. База данных является основой любой информационной системы. Модель данных – это некоторая абстракция, которая в приложении к конкретным данным позволяет пользователям и разработчикам трактовать их как информацию, т. е. рассматривать их как сведения, содержащие не только данные, но и взаимосвязи между ними.

Реляционная модель данных основана на понятии отношения, физическим представлением которого является двухмерная таблица, состоящая из строк одинаковой структуры. Логическая структура данных представляется набором связанных таблиц. Система управления базами данных (СУБД) – это совокупность лингвистических и программных средств, необходимых для создания и использования БД. СУБД предоставляют прикладным программам, разработчикам и пользователям множество различных представлений данных, хранящихся в БД.

**Инфологическая модель** применяется на втором этапе проектирования БД, то есть после словесного описания предметной области. Инфологическая модель представляет собой описание будущей базы данных, представленное с помощью естественного языка, формул,

графиков, диаграмм, таблиц и других средств, понятных как разработчикам базы данных (БД), так и обычным пользователям. Назначение такой модели состоит в адекватном описании процессов, информационных потоков, функций системы с помощью общедоступного и понятного языка, что делает возможным привлечение экспертов предметной области, консультантов, пользователей для обсуждения модели и внесения исправлений. В данном случае под созданием инфологической модели будем понимать именно ее создание для БД. В общем случае, инфологическая модель может создаваться для любой проектируемой системы и представляет ее описание (в общем случае в произвольной форме).

Создание инфологической модели является естественным продолжением исследований предметной области, но в отличие от него является представлением БД с точки зрения проектировщика (разработчика). Наглядность представления такой модели позволяет экспертам предметной области оценить ее точность и внести исправления. От правильности модели зависит успех дальнейшей разработки.

Инфологическую модель можно представить в виде словесного описания, однако наиболее наглядным является использование специальных графических нотаций, разработанных для проведения подобного рода моделирования.

Важно отметить, что создаваемая на этом этапе модель полностью не зависит от физической реализации будущей системы. В случае с БД это означает, что совершенно не важно где физически будут храниться данные: на бумаге, в памяти компьютера и кто или что эти данные будет обрабатывать. В этом случае, когда структуры данных не зависят от их физической реализации, а моделируются исходя из их смыслового назначения, моделирование называется семантическим.

Существует несколько способов описания инфологической модели, однако, в настоящее время одним из наиболее широко распространенных подходов, применяемых при инфологическом моделировании, является подход, основанный на применении диаграмм «сущность-связь» (ER – Entity Relationship). При рассмотрении последующих примеров будем использовать одну из самых распространенных в рамках ER моделей нотацию IDEF1X. Данный стандарт был разработан в 1993 г. Национальным институтом стандартизации и технологий и является федеральным стандартом обработки информации (США), описывающим

семантику и синтаксис языка, правила и технологии для разработки логической модели данных.

Для построения инфологической модели важно знать элементы этой модели. Базовыми элементами модели сущность-связь являются сущности.

Сущность по форме представляет собой только некоторое реального описание объекта, точнее набор описаний его значимых признаков-атрибутов. Конкретный набор значений атрибутов объекта будет называться экземпляром сущности.

**Даталогическая модель** (ДЛМ) строится на основе ИЛМ. ДЛМ БД является концептуальной моделью БД и отражает логические связи между информационными элементами ДЛМ. В ДЛМ фиксируются данные и связи данных между ними.

ДЛМ строится в терминах информационных единиц, допустимых в той конкретной СУБД, в среде которой проектируется БД. ДЛМ зависит от выбора СУБД для разработки БнД или информационной модели.

Схема БД - описание ДЛМ на языке выбранной СУБД.

**Физическая модель** используется для привязки ДЛМ к среде хранения данных физического уровня. Эта модель БД определяется используемыми ЗУ, способами физической организации данных в среде хранения.

Физическая модель, также как и ДЛМ, строится с учетом особенностей выбранной СУБД.

Физическое проектирование — описание физической структуры БД.

Рассмотрим проектирование базы данных на примере модели данных «Библиотека».

К стержневым сущностям можно отнести:

*1. Создатели (Код создателя, Создатель)*.

Эта сущность отводится для хранения сведений об основных людях, принимавших участие в подготовке рукописи издания (авторах, составителях, титульных редакторах, переводчиках и художниках). Такое объединение допустимо, так как данные о разных создателях выбираются из одного домена (фамилия и имена) и исключает дублирование данных (один и тот же человек может играть разные роли в подготовке разных изданий). Например, С.Я.Маршак писал стихи (Сказка о глупом мышонке) и пьесы (Двенадцать месяцев), переводил Дж.Байрона, Р.Бернса, Г.Гейне и составлял сборники стихов.

Так как фамилия и имена (инициалы) создателя могут быть достаточно громоздкими (М.Е. Салтыков-Щедрин, Франсуа Рене де Шатобриан, Остен Жюль Жан-Батист Ипполит и т.п.) и будут многократно встречаться в разных изданиях, то их целесообразно нумеровать и ссылаться на эти номера. Для этого вводится целочисленный атрибут "Код_создателя", который будет автоматически наращиваться на единицу при вводе в базу данных нового автора, переводчика или другого создателя.

Аналогично создаются: Код_издательства, Код_заглавия, Вид_издания, Код_характера, Код_языка, Номер_билета, Номер_переплета, Код_места и Код_издания, замещающие от одного до девяти атрибутов.

*2. Издательства (Код_издательства, Название, Город).*

*3. Заглавия (Код_заглавия, Заглавие).*

Выделение этой сущности позволит сократить объем данных и снизить вероятность возникновения противоречивости (исключается необходимость ввода длинных текстовых названий для различных томов собраний сочинений, повторных изданий, учебников и т.п.).

*4. Вид_издания (Вид_издания, Название_вида).*

*5. Характеры (Код_характера, Характер_переиздания).*

*6. Языки (Код_языка, Язык, Сокращение).*

Кроме названия языка хранится его общепринятое сокращение (англ., исп., нем., фр.), если оно существует.

*7. Места (Код_места, Номер_комнаты, Номер_стеллажа, Номер_полки).*

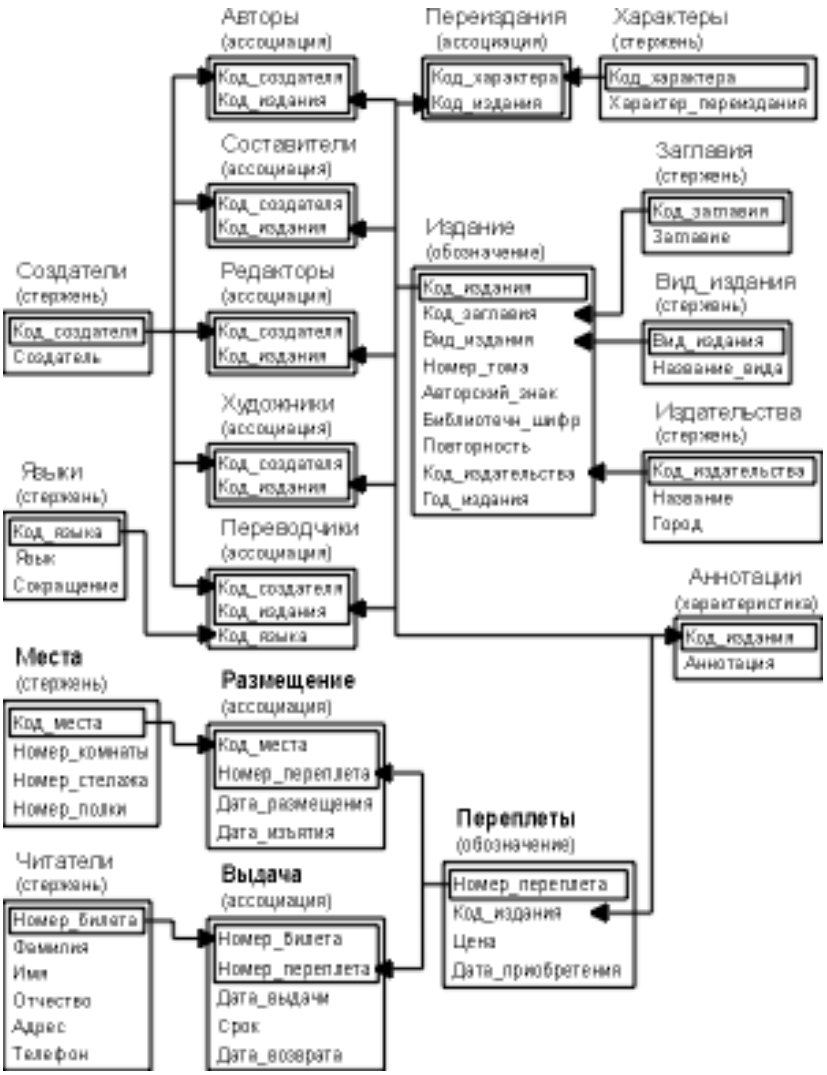
Один из кодов этой сущности (например, "-1") отведен для описания обобщенного места, находящегося за стенами хранилища книг (издание выдано читателю, временно передано другой библиотеке или организации).

*8. Читатели (Номер_билета, Фамилия, Имя, Отчество, Адрес, Телефон).*

Две ключевые сущности, описывающие издание и его конкретные экземпляры, оказываются зависимыми от других сущностей и попадают в класс обозначений:

1. *Издание (Код_издания, Код_заглавия, Вид_издания, Номер_тома, Авторский_знак, Библиотечн_шифр, Повторность, Код_издательства, Год_издания, Аннотация) [Заглавия, Вид_издания, Издательства];*

2. *Переплеты (Номер_переплета, Код_издания, Цена, Дата_приобретения)[Издания];*




Инфологическая модель предметной области "Библиотека"

Стержневые сущности и обозначения связаны между собой ассоциациями:

1. *Авторы [Создатели M, Издание N] (Код_создателя, Код_издания).*
2. *Составители [Создатели M, Издания N] (Код_создателя, Код_издания).*
3. *Редакторы [Создатели M, Издания N] (Код_создателя, Код_издания).*
4. *Художники [Создатели M, Издания N] (Код_создателя, Код_издания).*

5. *Переводчики [Создатели M, Издания N] (Код_создателя, Код_издания, Язык).*
6. *Переиздания [Характеры M, Издания N] (Код_характера, Код_издания).*
7. *Размещение [Места M, Переплеты N] (Код_места, Номер_переплета, Дата_размещения, Дата_изъятия).*
8. *Выдача [Читатели M, Переплеты N] (Номер_билета, Номер_переплета, Дата_выдачи, Срок, Дата_возврата).*

Теперь следует проверить, не нарушены ли в данном проекте какие-либо принципы нормализации, т.е. что любое неключевое поле каждой таблицы:

- функционально зависит от полного первичного ключа, а не от его части (если ключ составной);
- не имеет функциональной зависимости от другого неключевого поля.

Сущности Авторы, Составители, Редакторы, Художники и Переиздания, не имеющие неключевых полей, безусловно нормализованы. Нормализованы и сущности Создатели, Характеры, Заглавия, Вид_издания, состоящие из несоставного ключа и единственного неключевого поля.

Анализ сущностей Переводчики, Размещение и Выдача, состоящих из составного ключа и неключевых полей, показал, что в них нет функциональных связей между неключевыми полями. Последние же не зависят функционально от какой-либо части составного ключа.

Наконец, анализ сущностей Издания, Переплеты, Места, Читатели и Языки, показал, что единственной "подозрительной" сущностью является стержень Языки, имеющий два функционально связанных неключевых поля: Язык и Сокращение.

Поле Язык стало неключевым из-за ввода цифрового первичного ключа Код_языка, заменяющего текстовый возможный ключ Язык. Это позволило уменьшить объем хранимых данных в таблице Переводчики, затраты труда на ввод множества текстовых значений и возможной противоречивости, которая часто возникает из-за ввода в разные поля ошибочных дубликатов (например, "Английский", "Англиский", "Анлийский", "Англйский" и т.п.).

Процесс проектирования БД на основе принципов нормализации представляет собой последовательность переходов от неформального словесного описания информационной структуры предметной области к

формализованному описанию объектов предметной области в терминах некоторой модели.

Процесс проектирования длительный и требует обсуждений с заказчиком и со специалистами в предметной области. Наконец, при разработке серьезных корпоративных информационных систем проект базы данных является тем фундаментом, на котором строится вся система в целом, и вопрос о возможном кредитовании часто решается экспертами банка на основании именно грамотно сделанного инфологического проекта БД. Следовательно, инфологическая модель должна включать такое формализованное описание предметной области, которое легко будет «читаться» не только специалистами по базам данных. И это описание должно быть настолько емким, чтобы можно было оценить глубину и корректность проработки проекта БД, и конечно, оно не должно быть привязано к конкретной СУБД. Выбор СУБД – это отдельная задача, для корректного ее решения необходимо иметь проект, который не привязан ни к какой конкретной СУБД.

**Порядок выполнения лабораторной работы**

1. Составить план разработки проекта базы данных для заданной предметной области. Базу данных следует рассматривать как часть будущей информационной системы, автоматизирующей бизнес-процессы некоторой организации.

2. Выполнить анализ заданной предметной области. Сформулировать словесное описание информационных объектов. Описать типовые запросы для поиска и анализа информации об объектах предметной области.

3. Построить инфологическую модель данных, описывающую предметную область в рамках ER-модели «сущность – связь». Получить визуальное представление инфологической модели путём построения ER-диаграмм.

4. Построить даталогическую модель базы данных. Преобразовать полученные ранее ER-модели в конкретную схему реляционной базы данных.

5. Проверить полноту и корректность даталогической модели базы данных путём составления типовых запросов для поиска и анализа информации.

6. Создать таблицы базы данных на основе даталогической модели.

7. Определить ограничения целостности базы данных.

8. Построить схему данных.

9.Разработать типовые запросы и проверить правильность создания базы данных.

**Контрольные вопросы**

1. Базы данных как основа информационной системы.
2. Виды и назначение баз данных.
3. Модели данных.
4. Системы управления базами данных.
5. Реляционные БД и СУБД.
6. Технологии проектирования баз данных.
7. Как можно представить инфологическую модель?
8. Что представляет собой сущности?
9. Каким образом представляются базовые таблица?
10. Каким образом строятся даталогическая и физическая модели?
11. Опишите принципы нормализации.

# 4 ЛАБОРАТОРНАЯ РАБОТА №4 «ПРОЕКТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕФЕЙСА»

**Цель работы:** закрепить имеющиеся знания о проектировании пользовательского интерфейса.

**Задачи работы:**

1. Получить навыки практического применение методов сбора требований к пользовательскому интерфейсу ИС.
2. Сформировать представление о содержании анализа требований к интерфейсу пользователя.
3. Получить навыки проектирования интерфейса пользователя и выборы дизайнерских решений.

**Теоретические сведения**

**Интерфейс пользователя —** это совокупность логических и физических принципов взаимодействия пользователя и компонентов информационной системы

В дизайне пользовательского интерфейса можно условно выделить две составляющие:

*декоративную;*

*активную.*

К декоративной составляющей относятся элементы, отвечающие за *эстетическую* привлекательность программного изделия.

Активные элементы пользовательского интерфейса подразделяются на:

*операционные* – для отображения результатов выполнения операций над данными;

*информационные* – для отображения информации, сопровождающей выполнение операций: пояснения, комментарии, подсказки, уведомления и др.;

*управляющие* – для управления программой.

Важным принципом построения дизайна интерфейса является баланс между активными и декоративными составляющими. Так как при создании игр главным является баланс между сложностью игры и ее увлекательностью, то и интерфейсе должен обеспечиваться баланс между

функциональными возможностями программы, возможностями манипуляции ею и ее изобразительным рядом.

**Символьный интерфейс**

Символьный интерфейс основан на использовании различных символов (букв, цифр, знаков) для вывода информации на экран монитора и ввода пользователем данных и команд для выполнения.

Различают два типа символьных интерфейсов:

командная строка;

символьные формы.

*Командная строка* осуществляет взаимодействие пользователя с компьютером путем подачи компьютеру команд, которые он выполняет и выдает результат их выполнения пользователю.

При этой технологии в качестве способа ввода информации оператором в ЭВМ служит клавиатура, а компьютер выводит информацию человеку с помощью алфавитно-цифрового дисплея (монитора). Комбинацию монитор-клавиатура стали называть терминалом или консолью. Команды набираются в командной строке, представляющей собой символ приглашения и мигающий курсор, при этом набранные символы можно стирать и редактировать. По нажатию клавиши «Enter» («Ввод») ЭВМ принимает команду и начинает ее выполнять. После перехода в начало следующей строки компьютер выдает на монитор результаты своей работы. Наиболее распространенным командный интерфейс был в операционной системе MS DOS.

*Символьные формы* формируют на экране монитора области для ввода информации (форма) с использованием только символов. Ввод данных осуществляется также исключительно с помощью символов. Для завершения ввода формы и начала ее обработки ЭВМ команды подаются с помощьтю упарвляющих клавиш клавиатуры.

**Графический интерфейс**

Графический интерфейс использует растровую и векторную технологии построения изображений на экране монитора.

Растровая технология формирует изображения посредством окрашивания в различные цвета пикселей экрана.

Векторная технология строит изображения на основе простейших геометрических фигур (отрезков, кривых, окружностей и др.)

Различают несколько типов графических интерфейсов:

WIMP-интерфейс;

трехмерные интерфейсы.

В настоящее время широкое распространение получил графический WIMP-интерфейс.

***WIMP («Window, Icon, Menu, Pointer»)*** – интерфейс человеко-машинного взаимодействия на базе элементов: «окно, значок, меню, указатель). Данная технология предложена Мерзугой Уильбертсом (англ.) в 1980 году. Хотя его популярность постепенно падает, это слово часто используется в качестве приближённого синонима «графического интерфейса пользователя».

WIMP был разработан в корпорации **Xerox PARC** и популяризирован компьютером Macintosh в 1984 году (в нем были добавлены понятие «строка меню» и концепция расширенного управления окном).

Характерной чертой интерфейса WIMP является то, что диалог пользователя с компьютером ведется не с помощью командной строки, а с помощью окон, графических образов меню, курсора и других элементов. Хотя в этом интерфейсе подаются команды машине, но это делается через графические образы.

Идея графического интерфейса зародилась в средине 70-х годов в исследовательском центре фирмы Xerox Palo Alto Research Center (PARC). Предпосылкой графического интерфейса явилось уменьшение времени реакции компьютера на команду, увеличение объема оперативной памяти, а также развитие элементной базы, технических характеристик ЭВМ и в частности мониторов. После появления графических дисплеев с возможностью вывода любых графических изображений различного цвета графический интерфейс стал неотъемлемой частью всех компьютеров. Постепенно проходил процесс унификации в использовании клавиатуры и мыши прикладными программами. Слияние этих двух тенденций привело к созданию такого пользовательского интерфейса, с помощью которого при минимальных затратах времени и средств на переучивание персонала можно работать с любыми программными приложениями

Этот вид интерфейса реализован в виде двух уровней:

простой графический интерфейс;

полный WINP — интерфейс.

*Простой графический интерфейс*, который на первом этапе очень походил на технологию командной строки со следующими отличиями:

при отображении символов с целью повышения выразительности изображения допускалось выделение части символов цветом, инверсным изображением, подчеркиванием и мерцанием;

курсор мог быть представлен некоторой областью, выделенной цветом и охватывающей несколько символов и даже часть экрана;

реакция на нажатие любой клавиши во многом стало зависеть от того, в какой части находится курсор.

кроме часто используемых клавиш управлением курсором стали использоваться манипуляторы типа мыши, трекбола и т.п., которые позволяли быстро выделять нужную область экрана и перемещать курсор;

широкое использование цветных мониторов.

Появление простого графического интерфейса совпадает с широким распространением операционной системы MS DOS. Типичным примером его использования является файловая оболочка Norton Commander и текстовые редакторы MaltiEdit, ChiWriter, Microsoft Word для DOS, Лексикон и др.

*Полный WIMP-интерфейс*, явился вторым этапом развития графического интерфейса, который характеризуется следующими особенностями:

вся работа с программами, файлами и документами происходит в окнах;

программы, файлы, документы, устройства и другие объекты представляются в виде значков (иконок), которые при открытии превращаются в окна;

все действия с объектами осуществляются с помощью меню, которое становится основным элементом управления;

манипулятор выступает в качестве главного средства управления.

Следует отметить, что WIMP-интерфейс требует для своей реализации повышенного требования к производительности компьютера, объему его памяти качественного растрового цветного дисплея программного обеспечения, ориентированного на этот вид интерфейса. В настоящее время WIMP-интерфейс стал стандартом де-факто, а операционная система Microsoft Windows стала ярким его представителем.

Кроме того в фирме ***Xerox PARC*** разработаны два ***трехмерных интерфейса***:

конические деревья;

стена в перспективе.

Интерфейс *«конические деревья»* является визуализацией файловой системы компьютера и похож на систему детских пирамидок, каждый уровень которой соответствует уровню файлового каталога. Сами файлы из каталога отображаются в виде 3-мерной карусели под своим каталогом. Соль модели в том, что нужный файл можно "приблизить" поворотом карусели (может быть, не одной), идущим в режиме анимации.

Интерфейс *«стена в перспективе»* также отображает файловую систему, но вне ее иерархии, согласно двум каким-либо параметрам, например частоте обращения к файлу и его размеру. Это нормальная стена, только очень длинная, разбитая на три отрезка. Средний из них отображается на экране плоско, а два крайних уходят в перспективу. Пользователь может сделать средним любой отрезок стены, причем это тоже происходит в режиме анимации.

**Естественный интерфейс**

Естественный интерфейс основан на использовании природных (естественных) качеств, присущих пользователям. Различают следующие типы естественных интерфейсов:

SILK (речевой);

биометрический (мимический);

семантический (общественный).

***SILK (Speech, Image, Language, Knowledge)*** – интерфейс, основанный на четырех понятиях: «речь», «образ», «язык», «знания». Этот интерфейс наиболее приближен к обычной человеческой форме общения. В рамках этого интерфейса идет обычный разговор человека и компьютера. При этом компьютер находит для себя команды, анализируя человеческую речь и находя в ней ключевые фразы. Результаты выполнения команд он также преобразует в понятную человеку форму. Этот вид интерфейса требует больших аппаратурных затрат, поэтому находится в стадии разработки и совершенствования и используется пока только в военных целях.

Первая *речевая технология* появилась в середине 90-х годов после появления недорогих звуковых карт и широкого распространения технологий распознавания речи. Команды подавались голосом путем произнесения специальных стандартных слов (команд), которые должны выговариваться четко, в одном темпе с обязательными паузами между словами. Учитывая, что алгоритмы распознавания речи были недостаточно

развиты, требовалась индивидуальная предварительная настройка компьютерной системы на конкретного пользователя.

В настоящее время данная речевая технология получила развитие и реализована во многих программных продуктах (речевые команды в MAC OS, речевой поиск в Google, голосовой блокнот др.).

*Биометрическая технология («Мимический интерфейс»)* возникла в конце 90-х годов и в настоящее время находится в стадии разработки. Для управления компьютером используется выражение лица, направление взгляда, размер зрачка и другие признаки человека. Для идентификации пользователя используется рисунок радужной оболочки его глаз, отпечатки пальцев и другая уникальная информация, которая считывается с цифровой камеры, а затем с помощью программы распознавания образов из этого изображения выделяются команды.

*Семантический (общественный) интерфейс* возник еще в конце 70-х годов XX века, с развитием искусственного интеллекта. Его трудно назвать самостоятельным видом интерфейса, так как он включает в себя и интерфейс командной строки, и графический, и речевой, и мимический интерфейсы. Основной его особенностью является отсутствие команд при общении с компьютером. Запрос формируется на естественном языке, в виде связанного текста и образов. По сути - это моделирование общения человека с компьютером. В настоящее время используется для военных целей. Такой интерфейс крайне необходим в обстановке ведения воздушного боя.

**Свойства эффективного интерфейса**

Для того, чтобы ПИ был качественным (эффективным), он должен обладать следующими свойствами:

1. *Естественность интерфейса*, т.е. его способность выдавать сообщения и результаты, которые не требуют дополнительных пояснений.

2. *Согласованность интерфейса*, т.е. его способность предоставлять пользователям возможность переносить имеющиеся знания на новые задания, осваивать новые аспекты быстрее, и благодаря этому фокусировать внимание на решаемой задаче, а не тратить время на уяснение различий в использовании тех или иных элементов управления, команд и т.д.

Согласованность рассматривают в трех аспектах:

согласованность в пределах приложения (одна и та же команда должна выполнять одни и те же функции, где бы она не встретилась, причем одним и тем же образом);

согласованность в пределах рабочей среды (приложение должно "опираться" на знания и навыки пользователей, которые он получил ранее при работе в среде ОС);

согласованность в использовании названий (поведение каждого объекта интерфейса должно соответствовать тому названию, которое ему присвоено).

3. *Дружественность интерфейса*, т.е. его способность предотвращать ситуации, которые, вероятно, закончатся ошибками, вследствие неправильного ввода команды или данных пользователем.

4. *Обратимость интерфейса*, т.е. его способность каждое действие пользователей сопровождать визуальным или звуковым подтверждением того, что приложение восприняло команду.

5. *Простота интерфейса*, т.е. легкость в его использовании, изучении и в предоставлении доступа ко всему перечню функциональных возможностей, предусмотренных данным приложением.

6. *Гибкость (адаптивность) интерфейса*, т.е. его способность учитывать уровень подготовки и производительность труда пользователя.

7. *Эстетическая привлекательность*, т.е. его способность приложения обеспечить формирование на экране такой среды, которая не только содействовала бы пониманию пользователем представленной информации, но и позволяла бы сосредоточиться на наиболее важных ее аспектах.

**Технологические стандарты интерфейсов**

Технологические стандарты устанавливают вид элементов графического пользовательского интерфейса, а также определяют словарь графических управляющих элементов. Каждый такой элемент обладает стандартизированными свойствами описанного вида, составляющими его регламент. Нарушение данного регламента следует рассматривать как ***ошибку разработки пользовательского интерфейса***.

Примерами стандартов, которые относятся к группе технологических стандартов, являются стандарты ISO 9241, ISO/IEC 10741, 11581.

**Эргономические стандарты интерфейсов**

Эргономические стандарты устанавливают характеристики безопасности и производительности пользовательского интерфейса.

*Главное внимание* в эргономических стандартах уделяется качеству функциональных характеристик пользовательского интерфейса, к которым относятся:

соответствие задаче;

самоописательность;

контролируемсть;

соответствие ожиданиям пользователя;

толерантность (терпимость, невосприимчивость) к ошибкам;

настраиваемость;

изучаемость;

практичность (понятность, обозримость, обучаемость, удобство, простота).

Если учесть, что пользовательский интерфейс обладает свойствами обычного текстового объекта, взаимодействующего с пользователем, то важным представляется учет таких характеристик как:

размер шрифта;

цветовое оформление;

навигация по элементам интерфейса.

Для компьютерного интерфейса также важен учет особенностей, связанных с комфортностью экранного представления, достаточной оперативностью реакции программы на действия пользователя, удобством манипулирования мышью и клавиатурой (и их скоростными показателями).

К группе эргономических стандартов относятся ISO 9241, ISO/IEC 13407, 12119, 9126

## Принципы разработки пользовательских интерфейсов

### *1. Интерфейс создан для интерактивности*

Интерфейсы нужны для взаимодействия человека с миром. Они помогают сделать сложное простым, показывают соотношения элементов, собирают нас вместе или напротив разъединяют, удовлетворяют наши ожидания и предоставляют доступ к услугам. Проектирование интерфейсов имеет мало общего с искусством, ведь эффективность интерфейса может быть измерена. С другой стороны, лучшие интерфейсы — это больше чем решенная задача. Они могут вдохновлять, захватывать и погружать в другой мир.

### *2. Ясность — задача номер 1*

Ясность лежит в основе любого интерфейса. Люди должны быстро понять, с чем они работают, зачем это нужно и что с помощью этого можно сделать, предположить, что будет получаться в результате их действий и затем начать использовать это. Ясность подпитывает уверенность пользователей в своих действиях и делает использование интерфейса приятным. Сотня кристально ясных страниц лучше, чем одна страница, на которой царит хаос из-за переизбытка информации.

***3. Удерживайте внимание любой ценой***

Мы живем в мире, где нас часто отвлекают и прерывают. Уже сложно спокойно почитать, чтобы ничего не отвлекало нас. Внимание — золото. Поэтому всегда держите в голове цель того или иного экрана. Если цель — чтение статьи, дайте пользователю дочитать ее перед тем, как показывать рекламу. Не заполняйте боковую часть страницы всякой отвлекающей ерундой. Помня о внимании, вы не только сделаете читателей счастливее, но и повысите эффективность взаимодействия.

***4. Предоставьте пользователю контроль над ситуацией***

Люди чувствуют себя уверенно, когда владеют ситуацией. Плохо спроектированные программы заставляют людей делать незапланированные действия, мешают на пути к цели и даже неожиданно закрываются. Держите пользователя в курсе дел, периодически показывая системный статус, объясняя что произойдет, если вы сделаете то или иное действие и давая представление о том, что их ждет на каждом шагу. Не бойтесь быть слишком очевидным. Очевидности много не бывает

***5. Прямое взаимодействие лучше***

Лучший интерфейс — его отсутствие, то есть возможность напрямую взаимодействовать с физическими объектами. Учитывая, что это не всегда возможно и то, что объекты становятся все больше информационными, мы создаем интерфейсы, чтобы помочь взаимодействовать с ними. Проще всего набабахать множество слоев из глянцевых кнопок, хрома, графики, опций, свойств, окон, приложений. Однако пользователь будет вынужден обращаться с ними, а не с реальным объектами, над которыми выполняются действия. Вместо этого создавайте интерфейсы, с которыми можно обращаться жестами, как и в жизни. Идеальный интерфейс оставляет ощущение прямого контакта с объектами на экране.

***6. Одно главное действие на экран***

Каждый экран должен быть спроектирован ради единственного действия, которое действительно важно в данный момент. Так легче

учиться, легче использовать, легче надстраивать по необходимости. Экраны с двумя и более основными действиями быстро становятся запутывающими. Так же как написанная статья должна иметь одну объединяющую идею, экран должен иметь одно главное действие, ради которого он создан.

***7. Оставьте второстепенные действия второстепенными***

На экранах с одним главным действием может быть множество второстепенных действий, но они должны оставаться таковыми. Вы пишете статью не для того, чтобы люди делились ею в Твиттере, а чтобы ее прочитали и поняли. Сделайте второстепенное визуально легче или вообще показывайте его после того, как главное действие выполнено.

***8. Обеспечьте естественный следующий шаг***

Редкие действия выполняются за один шаг. Поэтому подумайте над шагами для каждого действия, которое есть в вашем интерфейсе. Предскажите следующее действие и дайте ему интерфейс. Так же, как и в человеческих отношениях, дайте понять, каким будет следующий шаг. Не оставляйте человека в ожидании только потому, что он сделал, что вы хотели. Нарисуйте ему дорогу к цели и ненавязчиво помогайте на каждом шагу.

***9. Форма следует за функцией***

Люди уверенно себя чувствуют, когда обращаются с предсказуемыми вещами. Когда другие люди, животные, предметы, программы ведут себя, так, как мы ожидаем, нам это нравится. Поэтому важно проектировать элементы интерфейса так, чтобы по ним можно было предсказать, какие действия они выполняют. Если что-то выглядит как кнопка, она должна работать как кнопка. Не выпендривайтесь на основах взаимодействия, оставьте свою креативность для более важных вещей.

***10. Последовательность имеет значение***

Элементы, которые выполняют похожие функции, должны выглядеть похоже. Это кажется логичным и очевидным. Но в этом принципе часто забывается обратное следствие: непохожие действия должны обозначаться непохожими элементами. В попытках создать целостный интерфейс, начинающие проектировщики игнорируют важные различия, используя ограниченный набор визуальных элементов.

***11. Строгие визуальные иерархии работают лучшим образом***

Строгая визуальная иерархия прослеживается там, где есть четкая последовательность элементов на экране. То есть порядок расположения схожих элементов одинаков на всех однотипных экранах. Если иерархия

отсутствует, то пользователю сложно понять, куда надо смотреть, чтобы увидеть то, что нужно. Мало кто замечает эту иерархию, но она является одним из простых способов сделать интерфейс более четким.

***12. Порядок разгружает мозг***

Как сказал Джон Маэда (John Maeda) в его книге “Простота”, порядок в расположении элементов может дать ощущение, что многое выглядит как малое. Это помогает людям понять ваш интерфейс лучше и быстрее, так как вы показываете невидимые связи между элементами. Группируйте схожие элементы, акцентируйте связи расположением и визуальной иерархией. Этим вы помогаете пользователю, который не обязан разбираться в вашем интерфейсе, просто потому, что вы его сделали. Не заставляйте пользователя думать и он будет вам благодарен.

***13. Используйте цвет для выделения, а не создания смысла***

Цвет физических предметов меняется в зависимости от освещения. При свете дня мы видим дерево со всеми его деталями и оттенками, однако если смотреть на него против закатного солнца — мы увидим только черный контур. Так же, как и в физическом мире, где цвет предмета может сильно различаться, цвет не должен быть главным в интерфейсе. Он может помочь, когда нужно направить внимание на определенный элемент, но не стоит использовать его как единственный способ различения элементов. Используйте легкие или приглушенные цвета фона, оставив яркие для акцентирования. Конечно, вы можете использовать яркие цвета, но только когда вы уверены, что это будет позитивно воспринято вашей аудиторией.

***14. Постепенное появление***

Показывайте только то, что нужно на текущем экране. Если люди должны сделать выбор, дайте им достаточно информации для него, а в детали погружайтесь на следующем экране. Избегайте привычки рассказать и показать все сразу. Когда возможно, разделите принятие нескольких решений на разные экраны. Это поможет сохранить ясность во взаимодействии.

***15. Помогайте пользователям по ходу***

В идеальных интерфейсах помощь не нужна, т.к. они понятны и легко используемы. В реальной жизни интерфейсы иногда требуют помощи, однако ее нужно оказывать порционно и там, где она действительно нужна, скрывая в остальное время. Можно попросить людей пройти в раздел справки и найти там ответ на свой вопрос, но лучше встроить подсказки там, где они нужны. Только убедитесь, что пользователи знакомы с подобными интерфейсами.

***16. Решающий момент: нулевой шаг***

Первый опыт взаимодействия с интерфейсом имеет решающее значение, которое часто недооценивают дизайнеры. Сложности с интерфейсом чаще всего проявляются на этой стадии. Чтобы помочь пользователям побыстрее разобраться в интерфейсе, спроектируйте справочную страницу перед загрузкой основного содержимого. Расскажите и покажите, что и как нужно будет делать. Когда пользователи поймут правила, им будет приятнее пользоваться интерфейсом.

***17. Существующие проблемы — самые главные***

Люди ищут решения своих текущих проблем, а не потенциальных проблем, которые ждут их в будущем. Поэтому не стоит проектировать интерфейс, заточенный под гипотетические проблемы. Исследуйте текущие проблемы и создайте интерфейс под них. Это не так увлекательно, но однозначно даст свои плоды, когда реальные пользователи начнут пользоваться вашим интерфейсом.

***18. Лучший дизайн незаметен***

Парадоксально, но факт: отличный дизайн незаметен. Причина в том, что пользователь может сконцентрироваться на решении своих задач, вместо того, чтобы разбираться с интерфейсом. Когда пользователь успешно решил свою задачу, он счастлив и не видит причин благодарить за это интерфейс. Возможно это звучит разочаровывающе для дизайнеров, ведь никто не скажет, как хорош ваш дизайн. Однако, великие проектировщики знают, что счастливые пользователи молчат.

***19. Черпайте вдохновение в других областях***

Дизайн, типографика, копирайт, информационная архитектура и визуализация. Все эти области являются частью интерфейса. Они могут быть второстепенными или главными в вашем интерфейсе. Не замыкайтесь на проектировании, расширьте свой кругозор и поищите вдохновения в принципах других дисциплин. Подумайте, чему вы можете научиться из написания программного кода, правил оформления книг, скейтбординга или карате.

***20. Интерфейсы создаются, чтобы ими пользовались***

Как и с другими видами дизайна, проектирование успешно тогда, когда люди пользуются этим. Как красивый стул, на котором неудобно сидеть, интерфейс считается провальным, когда пользователи не хотят им пользоваться. Поэтому важно помнить, что проектирование интерфейса совмещает в себе как создание среды для решения задач, так и продукт, которым хочется пользоваться. Интерфейс, который удовлетворяет только

эго дизайнера, не может считаться хорошим. Интерфейс хорош тогда, когда им активно пользуются.

**Порядок выполнения лабораторной работы**

1. Выявить требования к пользовательскому интерфейсу ИС.
2. Сформировать профили потенциальных пользователей программного обеспечения информационной системы в табличном виде.
3. Определить функциональность приложения, исходя из целей и задач пользователей
4. Сформировать множество пользовательских сценариев для выделенных профилей пользователей
5. Выделить требования к пользовательскому интерфейсу, основываясь на сценариях и группировке операций:
   основное и вложенные меню (текстовое и графическое описание),
   количество страниц (для веб-систем) и их назначение,
   структуры главной и вложенных страниц,
   количество и структуру диалоговых окон,
   количество и структуру системных сообщений (об ошибках, о завершении процесса и т.п.)
   наличие обратной связи и др.
6. Определить приоритет требований (необходимые, желательные, дополнительные)
7. Выделить основные требования к дизайну.
8. Построить схему пользовательского интерфейса.
9. Разработать дизайн-макеты главной страницы системы, дополнительных страниц (вкладок), диалоговых окон и сообщений
10. Обосновать выбор дизайна.

**Контрольные вопросы**

1. Понятие пользовательского интерфейса.
2. Виды пользовательских интерфейсов.
3. Свойства пользовательского интерфейса.
4. Стандарты пользовательских интерфейсов.
5. Принципы разработки пользовательских интерфейсов.
6. Этапы разработки пользовательского интерфейса.

# 5 ЛАБОРАТОРНАЯ РАБОТА №5 «РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ»

**Цели работы:**

1. Закрепление имеющихся знаний о средствах разработки программного обеспечения информационных систем.
2. Приобретение навыков работы в современных интегрированных средах разработки программного обеспечения.
3. Приобретение навыков разработки клиентского программного обеспечения ИС.

**Задачи работы:**

1. Разработать прототип приложения.
2. Разработать программный код приложения.
3. Провести отладку разработанного приложения.
4. Разработать документ «Руководство пользователя» с описанием назначения и функциональных возможностей приложения

**Теоретические сведения**

CASE-технология представляет собой совокупность методологий анализа, проектирования, разработки и сопровождения сложных программных систем, которая поддерживается комплексом взаимосвязанных программных средств автоматизации. Основой CASE-технологии является использование единой базы данных (репозитория) для хранения всей информации, которая может использоваться в процессе создания системы. Репозиторий может хранить объекты различных типов: структурные диаграммы, эскизы экранных форм, модели данных, описание алгоритмов обработки данных и т.д. CASE-средства – это программные средства, поддерживающие процессы создания и сопровождения ИС, включая анализ и формулирование требований, проектирование прикладного ПО и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление, управление проектом и т.д.

К CASE-средствам относят любой программный продукт, обладающий следующими основными характерными особенностями:

1) наличие мощных графических средств для описания и документирования ИС;

2) интеграция отдельных компонентов CASE-средств, обеспечивающая управляемость процесса разработки ИС;

3) использование специальным образом организованного хранилища проектных метаданных (репозитория).

Быстрая разработка приложений RAD (Rapid Application Development) является одной из современных методологий разработки программного обеспечения. Как и другие методологии (MSF, RUP и др.) RAD описывает итеративный подход к организации процесса разработки ПО и соответствующую модель жизненного цикла. Методологию RAD также часто связывают с технологией визуального программирования и применением современных интегрированных сред разработки программного обеспечения. Методология RAD основывается на визуализации процесса создания программного кода приложений и поддерживается инструментальным ПО, которое предоставляет разработчикам средства визуального программирования. Применение средств визуального программирования позволяет значительного ускорить процесс разработки приложений, а также уменьшить трудоёмкость работы по модификации уже готовой программы, внесению в неё необходимых дополнений или изменений. Средства быстрой разработки приложений, как правило, основываются на объектно-ориентированной компонентной архитектуре.

Процедура разработки интерфейса средствами RAD сводится к набору последовательных операций, включающих:

1) размещение компонентов интерфейса в нужном месте;

2) задание моментов времени их появления на экране;

3) настройку связанных с ними атрибутов и событий.

Интегрированная среда разработки (ИСР) является средством, с помощью которого выполняются проектирование, программирование, тестирование и отладка прикладных программ. Примерами современных ИСР, поддерживающих методологию RAD и технологию визуального программирования, являются Microsoft Visual Studio, Embarcadero RAD Studio, IntelliJ IDEA, MonoDevelop и др.

**Порядок выполнения лабораторной работы**

1.Разработать алгоритмы программы.

2. Построить дерево диалога.

3.На основе уточненных и доработанных алгоритмов разработать структурную схему приложения.

4. Разработать функциональную схему приложения.

**Контрольные вопросы**

1.Назовите этапы разработки программного обеспечения.

2.В чем заключается проектирование программного обеспечения?

3.Перечислите составляющие технического проекта.

4.Охарактеризуйте структурный подход к программированию.

# 6 ЛАБОРАТОРНАЯ РАБОТА №6 «ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»

## 6. Тестирование программного обеспечения

**Цель работы:** изучение основных методов тестирования программных средств, получение навыков тестирования программного обеспечения.

**Задачи работы:**

1. Изучить основные методы тестирования программных систем.
2. Сформировать единое представление о реализуемом ПО.
3. Определить несоответствия функционала реализуемого ПО с ТЗ.

### Теоретические сведения

**Тестирование** (software testing) – деятельность, выполняемая для оценки и улучшения качества программного обеспечения. 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. [SWEBOK].

**Тестирование** - запуск исполняемого кода с тестовыми данными и исследование выходных данных и рабочих характеристик программного продукта для проверки правильности работы системы.

**Тестирование** - процесс выполнения его программ на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется тестовым или просто тестом.

**Тестирование** – проверка работы программ с данными, подобными реальным, которые будут обрабатываться в процессе эксплуатации системы. Наличие в программе дефектов и несоответствий требованиям обнаруживается путем исследования выходных данных и выявления среди них аномальных.

**Объект тестирования** (Objectivies of Testing). 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.

**Цели тестирования** (The target of the test). Тестирование проводится в соответствии с определенными целями (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).

К *объектам тестирования* относятся:

1. **Модульное тестирование** (Unit testing). Этот уровень тестирования позволяет проверить функционирование отдельно взятого элемента системы. Что считать элементом – модулем системы определяется контекстом.
2. **Интеграционное тестирование** (Integration testing). Данный уровень тестирования является процессом проверки взаимодействия между программными компонентами/модулями. Интеграционное тестирование – постоянно проводимая деятельность, предполагающая работу на достаточно высоком уровне абстракции. Наиболее успешная практика интеграционного тестирования базируется на инкрементальном подходе.
3. **Системное тестирование** (System testing). 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 т.п. Также тестируются интерфейсы к внешним приложениям, аппаратному обеспечению, операционной среде и т.д.

*Целями тестирования* являются:

***Приёмочное тестирование*** (Acceptance/qualification testing). Проверяет поведение системы на предмет удовлетворения требований

заказчика. Это возможно в том случае, если заказчик берет на себя ответственность, связанную с проведением таких работ.

***Установочное тестирование*** (Installation testing). С целью проверки процедуры инсталляции системы в целевом окружении.

***Альфа- и бета-тестирование*** (Alpha and beta testing). Перед выпуском ПО, как минимум, должно проходить стадии альфа (внутреннее пробное использование) и бета (пробное использование с привлечением отобранных внешних пользователей) версий.

***Функциональные тесты/тесты соответствия*** (Conformance testing/Functional testing/Correctness testing). Проверка соответствия системы, предъявляемым к ней требованиям, описанным на уровне спецификации поведенческих характеристик.

***Достижение и оценка надежности*** (Reliability achievement and evaluation)

***Регрессионное тестирование*** (Regression testing). Тестирования программного обеспечения, направленное на обнаружение ошибок в уже протестированных участках исходного кода. Такие ошибки, связанные с тем, что после внесения изменений в программу перестает работать то, что должно было работать, называют *регрессионными ошибками* (англ. *regression bugs*).

***Определение успешности регрессионных тестов*** (IEEE 610-90 “Standard Glossary of Software Engineering Terminology”): “повторное выборочное тестирование системы или компонент для проверки сделанных модификаций не должно приводить к непредусмотренным эффектам”.

***Тестирование производительности*** (Performance testing). Делается попытка достижения количественных пределов, обусловленных характеристиками самой системы и ее операционного окружения.

***Нагрузочное тестирование*** (Stress testing). С целью достижения ее реальных/достижимых (т производительности) возможностей производительности и повышением нагрузки, вплоть до достижения запланированных характеристик и далее, с отслеживанием поведения на всем протяжении повышения загрузки системы.

***Сравнительное тестирование*** (Back-to-back testing)

***Восстановительные тесты*** (Recovery testing)

***Конфигурационное тестирование*** (Configuration testing)

***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; наконец, насколько хорошо система застрахована (с точки зрения потенциальных сбоев) от ошибок пользователя.

***Разработка, управляемая тестированием*** (Test-driven development)

*Обобщенно* можно цели тестирования определить следующим образом:

**1. Тестирование дефектов** проводится для обнаружения несоответствий между ПО и его спецификацией, которые обусловлены ошибками или дефектами. Такие тесты разрабатываются для выявления ошибок в системе, а не для имитации ее работы. Целью тестирования дефектов является выявление в программной системе скрытых дефектов до того, как она будет сдана заказчику. Тестирование дефектов противоположно аттестации, в ходе которой проверяется соответствие системы своей спецификации. Во время аттестации система должна корректно работать со всеми заданными тестовыми данными. При тестировании дефектов запускается такой тест, который вызывает некорректную работу программы и, следовательно, выявляет дефект. Обратите внимание на эту важную особенность: тестирование дефектов демонстрирует наличие, а не отсутствие дефектов в программе.

**2. Статистическое тестирование** оценивает производительность и надежность ПО, а также работу в различных режимах эксплуатации. Тесты разрабатываются так, чтобы имитировать реальную работу системы с реальными входными данными.

**Структурное тестирования**

Метод предполагает создание тестов на основе **структуры системы** и ее реализации.

Такой подход иногда называют тестированием методом *«белого ящика»* (*White box*), «стеклянного ящика» или «прозрачного ящика», который как бы позволяет заглянуть внутрь программного обеспечения.

Как правило, структурное тестирование применяется к относительно небольшим программным элементам, например к подпрограммам или методам, ассоциированным с объектами. Например, из анализа кода можно определить, сколько контрольных тестов нужно выполнить для того,

чтобы в процессе тестирования все операторы выполнились по крайней мере один раз.

**Функциональное тестирование**

Метод тестирования базируется на том, что все тесты основываются на проверке не самого ПО, а только выполняемых им функций.

ПО представляется как *«черный ящик»* (*Black-box*), поведение которого можно определить только посредством изучения ее входных и соответствующих выходных данных.

**Комбинированное тестирования**

Метод сочетает проверку как структуры ПО, так и его функционирования.

Такой подход называют тестирование методом «*серого ящика*» (*Gray-box Testing*).

**Деятельность по тестированию (Test Activities):**

1. **Планирование** (Planning)

2. **Генерация сценариев тестирования** (Test-case generation). Создание тестовых сценариев основывается

на уровне и конкретных техниках тестирования.

3. **Разработка тестового окружения** (Test environment development). Окружение должно обеспечивать разработку и контроль тестовых сценариев, ведение журнала тестирования, и возможности восстановления ожидаемых и отслеживаемых результатов тестирования, самих сценариев, а также других активов тестирования.

4. **Выполнение тестов** (Execution). Выполнение тестов должно содержать основные принципы ведения научного эксперимента:

должны фиксироваться все работы и результаты процесса тестирования;

форма журналирования таких работ и их результатов должна быть такой, чтобы соответствующее содержание было понятно, однозначно интерпретируемо и повторяемо другими лицами;

тестирование должно проводиться в соответствии с заданными и документированными процедурами;

тестирование должно производиться над однозначно идентифицируемой.

версией и конфигурацией программной системы

5. **Анализ** результатов тестирования (Test results evaluation).

6. **Отчёты** о проблемах/журнал тестирования (Problem reporting/Test log)

7. **Отслеживание дефектов** (Defect tracking). Дефекты могут (и, чаще всего, должны) анализироваться для определения момента и места первого появления данного дефекта в системе, какие типы ошибок стали причиной этих дефектов.

**Документы тестирования**

1. **План тестирования -** организационный документ, содержащий требования к тому, как должно выполняться тестирование в данном конкретном проекте.

Создается менеджером тестирования (test manager).

2. **Тест-требования** - документы, в которых подробно описано то, какие аспекты поведения системы должны быть протестированы. На основании описания архитектуры создаются низкоуровневые тест-требования, где описываются аспекты поведения конкретной программной реализации системы, которые необходимо протестировать.

Создаются разработчиками тестов (test procedure developers).

3. **Тест-планы** (**тестовые спецификации**) - документы, которые содержат подробное пошаговое описание того, как должны быть протестированы тест-требования. На основании тест-требований и проектной документации разработчиков также создается ***тестовое окружение***, необходимое для корректного выполнения тестов на тестовых стендах - драйверы, заглушки, настроечные файлы и т.п.

Создаются разработчиками тестов (test developers).

4. **Отчеты о выполнении тестирования** (они могут создаваться либо автоматически, либо вручную), которые содержат информацию о том, какие несоответствия требованиям были выявлены в результате тестирования.

Создаются тестировщиками (testers).

5. **Отчеты о проблемах** - документы, которые направляются на анализ в группу разработчиков с целью определения причины возникновения несоответствия.

**Тест-план**

Структура тест-плана может соответствовать структуре тест-требований или следовать логике внешнего поведения системы. Каждый

пункт тест-плана описывает, как производится проверка правильности функционирования программной реализации, и содержит:

**ссылку на требование**(я), которое проверяется этим пунктом;

конкретное **входное воздействие** на программу (значения входных данных);

**ожидаемую реакцию** программы (тексты сообщений, значения результатов);

описание **последовательности действий**, необходимых для выполнения пунктов тест-плана.

Возможные формы подготовки тест-планов:

1. В виде текстовых документов, в которых отдельные разделы представляют собой описания тестовых примеров, каждый пример включает в себя перечисление последовательности действий, которые необходимо выполнить - **сценария теста**, а также ожидаемые отклики системы на эти действия.
2. Для автоматизированного тестирования сценарий может записываться на **формальном языке**.
3. **Таблица** используется при четко и формально определенных входных потоках данных системы. Например, каждый столбец таблицы может представлять собой тестовый пример, каждая строка - описание входного потока данных, а в ячейке таблицы записывается передаваемое в данном тестовом примере в данный поток значение. Ожидаемые значения для данного теста записываются в аналогичной таблице, в которой в строках перечисляются выходные потоки данных.
4. В виде **конечного автомата** используется при тестировании протоколов связи или программных модулей, взаимодействие которых с внешним миром производится при помощи обмена сообщениями по заранее заданному интерфейсу.

**Порядок выполнения лабораторной работы**

1. Сформировать сценарий системного тестирования разработанного ПО.
2. Уточнить спецификации для тестирования отдельных, компонентов (функций).

3. Описать состояние окружения (входные данные) и ожидаемую последовательность событий в системе (ожидаемый результат в спецификации для каждого тестового случая.

4. Разработать тесты.

5. Выполнить тестирование общей работоспособности и отдельных функциональных возможностей разработанного приложения.

6. Исправить возможные ошибки в приложении.

7. Выполнить верификацию функциональных возможностей разработанного приложения, сравнивая их с имеющимся перечнем функциональных требований.

**Контрольные вопросы**

1. Понятие тестирования ПО.
2. Цели тестирования.
3. Виды тестирования.
4. Что представляет собой сценарий тестирования?
5. Какие виды тестирования обязательно применять в процессе разработки сложного ПО?
6. Модульное тестирование. Понятие модуля.
7. V-образная модель. Статическое и динамическое тестирование.
8. Валидация и верификация. Тестирование методом "чёрного" и "белого" ящика.
9. Тестовый случай и тестовое покрытие.

# 7 РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

***а) Основная литература:***


1. Технологии и методы программирования: учебное пособие для прикладного бакалавриата / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. — М.: Юрайт, 2019. – 235 с.: (Серия: Бакалавр. Прикладной курс). – https://biblio-online.ru/bcode/433611..
2. Программирование графики на C++. Теория и примеры: учеб. пособие / В.И. Корнеев, Л.Г. Гагарина, М.В. Корнеева. – М.: ИД «ФОРУМ»: ИНФРА-М, 2017. – 517 с. – URL: https://znanium.com/catalog/product/562914.
3. Проектирование современных баз данных: Учебно-методическое пособие / Э.Г. Дадян. – М.: НИЦ ИНФРА-М, 2017. – 120 с. – URL: https://znanium.com/catalog/product/959294.
4. Проектирование информационных систем: учеб. пособие / В.В. Коваленко. – М.: ФОРУМ: ИНФРА-М, 2018. – 320 с. – URL: https://znanium.com/catalog/product/980117.
5. Технология разработки программного обеспечения: учеб. пособие / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Сидорова-Виснадул; под ред. Л.Г. Гагариной. – М.: ИД «ФОРУМ»: ИНФРА-М, 2019. – 400 с. – URL: https://znanium.com/catalog/product/1011120.


***б) Дополнительная литература:***


1. Программные средства и механизмы разработки информационных систем: Учебное пособие / А.А. Лежебоков. – Таганрог: Южный федеральный университет, 2016. – 86 с. – URL: https://znanium.com/catalog/product/997088.
2. Введение в архитектуру программного обеспечения: Учебное пособие / Гагарина Л.Г., Федоров А.Р., Федоров П.А. – М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2017. – 320 с. – URL: https://znanium.com/catalog/product/615207.
3. Управление качеством программного обеспечения: учебник / Б.В. Черников. – М.: ИД «ФОРУМ»: ИНФРА-М, 2019. – 240 с. – URL: https://znanium.com/catalog/product/1018037.
4. Язык C++ и объектно-ориентированное программирование в C++. Лабораторный практикум: Учебное пособие для вузов / Ашарина И.В., Крупская Ж.Ф. – М.: Гор. линия-Телеком, 2016. – 232 с. – URL: https://znanium.com/catalog/product/973780.
5. Скрапинг веб-сайтов с помощью Python / Р. Митчелл; пер. с англ. А. В. Груздева. – Москва : ДМК Пресс, 2016. - 280 с. – Режим доступа: URL: http://znanium.com/catalog/product/1027754.

6. Басс Л., Клементс П., Кацман Р. Архитектура программного обеспечения на практике: пер. с англ. – 2-е изд. – СПб.: Питер, 2006. – 575 с.
7. Вендров А.М. Практикум по проектированию программного обеспечения экономических информационных систем: учеб. пособие. – М.: Финансы и статистика, 2004. – 192 с.
8. Брауде Э.Д. Технология разработки программного обеспечения: пер. с англ. – СПб.: Питер, 2004. – 655 с.
9. Бекаревич Ю.Б., Пушкина Н.В. Самоучитель Microsoft Access 2002. – СПб.: БХВ-СПб., 2003. – 720 с.
10. Голицина О.Л., Максимов Н.В., Попов И.И. Базы данных: Учебное пособие. – М.: ФОРУМ: ИНФРА-М, 2003. – 352 с.
11. Карпова Т.С. Базы данных: модели, разработка, реализация. – СПб.: Питер, 2002. – 304 с.

***в) Интернет-ресурс, базы данных:***

1. Система федеральных образовательных порталов. Информационно-коммуникационные технологии в образовании. http://www.ict.edu.ru/lib/.
2. Интернет университет информационных технологий. http://www.intuit.ru/.
3. Система федеральных образовательных порталов. Информационно-коммуникационные технологии в образовании. http://www.ict.edu.ru/lib/.
4. Российская национальная библиотека (РНБ). www.hbl-russia.ru.
5. Российская государственная библиотека (РГБ). http://www.rsl.ru.
6. ЭБС «Университетская библиотека онлайн» http:www.biblioclub.ru.
7. ЭБС «Znanium.com» http://znanium.com.
8. ЭБС «Юрайт» https://biblio-online.ru.
9. CIT-Forum. Обзор паттернов проектирования - http://citforum.ru/SE/project/pattern/
10. Source Making (eng) - http://sourcemaking.com/
11. Паттерны проектирования. Раздел на сайте для программистов Cpp-Reference - http://cpp-reference.ru/patterns/
12. Справочник. Паттерны проектирования. - http://design-pattern.ru