МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО
ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
В Г. ТАГАНРОГЕ РОСТОВСКОЙ ОБЛАСТИ
ПИ (филиал) ДГТУ в г. Таганроге

ЦМК «Прикладная информатика»

Практикум

По выполнению практических работ

по МДК:

МДК 02.02 Инструментальные средства разработки ПО

для специальности 09.02.07 Информационные системы и программирование,

квалификации

«Специалист по информационным системам»

Таганрог
2023

Составители: А.А. Погорелов

Практикум по выполнению практических работ по МДК:

МДК 02.02 Инструментальные средства разработки ПО. ПИ (филиала) ДГТУ в г.Таганроге, 2023г.

В практикуме кратко изложены теоретические вопросы, необходимые для успешного выполнения практических работ, рабочее задание и контрольные вопросы для самопроверки.

Предназначено для обучающихся по специальности 09.02.07 «Информационные системы и программирование». Квалификации выпускника: «Специалист по информационным системам»

Ответственный за выпуск:

Председатель ЦМК: _____ О.В. Андриян

1 Область применения методических указаний (рекомендаций)

Методические рекомендации предназначены в качестве методического пособия при проведении практических работ по учебному предмету (модулю), практике и государственной итоговой аттестации для специальности СПО 09.02.07. Информационные системы и программирование.

Практические работы проводятся после изучения соответствующих разделов и тем по учебному предмету (модулю), практике и государственной итоговой аттестации. Выполнение обучающимися практических работ позволяет им понять, где и когда изучаемые теоретические положения, и практические умения могут быть использованы в будущей практической деятельности.

2 Основные задачи проведения практических работ

Цель:

− формирование практических умений, необходимых в последующей профессиональной и учебной деятельности.

Задачи:

− обобщить, систематизировать, углубить, закрепить полученные теоретические знания по конкретным темам дисциплин общепрофессионального цикла;

− формировать умения применять полученные знания на практике;

− выработать при решении поставленных задач таких профессионально значимых качеств, как самостоятельность, ответственность, точность, творческая инициатива.

На практических занятиях обучающиеся овладевают первоначальными профессиональными умениями и навыками, которые в дальнейшем закрепляются и совершенствуются в процессе учебной и производственной практики.

Освоение дисциплины является частью освоения основного вида профессиональной деятельности и соответствующих общих (ОК) компетенций:

ОК 1. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам

ОК 2. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие.

ОК 4 Планировать и реализовывать собственное профессиональное и личностное развитие.

ОК 5 Планировать и реализовывать собственное профессиональное и личностное развитие.

ОК 6 Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей

ОК 7 Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях.

ОК 8 Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности

ОК 9 Использовать информационные технологии в профессиональной деятельности.

ОК 10 Пользоваться профессиональной документацией на государственном и иностранном языке

ПК 2.1. Разрабатывать требования к программным модулям на основе анализа проектной и технической документации на предмет взаимодействия компонент

ПК 2.2. Выполнять интеграцию модулей в программное обеспечение

ПК 2.3 Выполнять отладку программного модуля с использованием специализированных программных средств

ПК 2.4 Осуществлять разработку тестовых наборов и тестовых сценариев для программного обеспечения.

ПК 2.5. Производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования

В результате выполнения практических работ, предусмотренных программой по учебному предмету (модулю), практике и государственной итоговой аттестации, обучающийся должен:

Уметь

использовать выбранную систему контроля

версий; использовать методы для получения

кода с заданной функциональностью и

степенью качества

Знать

модели процесса разработки программного обеспечения; основные принципы процесса разработки программного обеспечения; основные подходы к интегрированию программных модулей; основы верификации и аттестации программного обеспечения

О проведении практической работы обучающимся сообщается заблаговременно: когда предстоит практическая работа, какие вопросы нужно

повторить, чтобы ее выполнить. Просматриваются задания, оговаривается ее объем и время ее выполнения. Критерии оценки сообщаются перед выполнением каждой практической работы.

Перед выполнением практической работы повторяются правила техники безопасности. При выполнении практической (лабораторной) работы обучающийся придерживается следующего алгоритма:

1. Записать дату, тему и цель работы.
2. Ознакомиться с правилами и условиями выполнения практического (лабораторного) задания.
3. Повторить теоретические задания, необходимые для рациональной работы и других практических действий.
4. Выполнить работу по предложенному алгоритму действий.
5. Обобщить результаты работы, сформулировать выводы по работе.
6. Дать ответы на контрольные вопросы.
7. Объем может колебаться в пределах **5-10 печатных страниц, в зависимости от работы** (***оформление письменной работы согласно Правилам оформления письменных работ обучающихся для гуманитарных/технических направлений подготовки (приказ ректора Б.Ч. Месхи от 16.12.2020 года №242***); все приложения к работе не входят в ее объем.

Работа должна быть выполнена грамотно, с соблюдением культуры изложения.

Обязательно должны иметься ссылки на использованные информационные источники. Должна быть соблюдена последовательность написания библиографического аппарата.

Задание со звездочкой повышенной сложности на оценку **«отлично».**

3 Организация и проведение практических работ:

Практическое занятие №1 «Анализ предметной области»

Цель: Изучить, описать и проанализировать предметную область, в которой будет создаваться информационная база.

Форма отчета:

1. Провести анализ предметной области в соответствии с выданным заданием.
2. Защитить практическую работу.

Время выполнения: 2 ч

Выделяются следующие шаги работы над проектом (системой):

1. Описание предметной области, под которой понимается та часть реального мира, которая имеет существенное значение или непосредственное отношение к процессу функционирования программы. Другими словами, предметная область включает в себя только те объекты и взаимосвязи между ними, которые необходимы для описания требований и условий решения некоторой задачи. Следовательно, разработчикам необходимо выделить основные объекты (компоненты), участвующие в функционировании системы, определить их наиболее существенные характеристики, взаимосвязи в рамках решаемой задачи, а также определить основные информационные потоки в системе. При этом отдельные компоненты выбираются таким образом, чтобы при последующей разработке их было удобно представить в форме классов и объектов. В этом случае немаловажное значение приобретает и сам язык представления информации о концептуальной схеме предметной области.

Сложность предметной области определяет количество объектов и связей между ними, поэтому описание должно включать в себя базовые *термины и определения*, сопровождаться различными примерами, в нем могут приводиться различного рода *классификации*, поясняющие различные свойства описываемых объектов. Если в системе используются математические модели, то они также должны быть описаны с учетом специфики применения.

2. Обзор существующих *систем-аналогов* – неотъемлемая часть описания предметной области, которая позволяет разработчику определить основные концепции, необходимые для реализации в системе. Описание должно приводиться с указанием отличительных особенностей разработанных систем, с перечислением их достоинств и недостатков, в отчете обязательно приводятся экранные формы этих систем.

3. Результатом последнего этапа является *диаграмма объектов предметной области* и краткое описание их свойств и функций. При построении данной диаграммы нужно помнить о том, что в данном случае объект – это «конкретная материализация абстракции», а не экземпляр класса. Диаграмма объектов представляет статическую составляющую взаимодействующих между собой объектов, она должна включить в себя только те объекты предметной области, которые потом преобразуются в диаграмму классов. Связи между объектами показывают отношения между ними, при необходимости в диаграмме можно привести и атрибуты (свойства) объектов.

Диаграммы объектов не позволяют полностью описать объектную структуру системы, поэтому при их использовании нужно сосредоточиться на изображении интересующих вас наборов конкретных объектов.

Для сбора, хранения, поиска и выдачи информации о предметной области и ее объектов настоящее время в информационных системах широко используются базы данных.

Анализ предметной области начинается с выделения сущностей и определения их свойств или атрибутов.

Видимые сущности представляют собой объекты предметной области, которые может распознать человек.

Поддерживаемые сущности или абстрактные сущности разрабатываются для физической поддержки общей логической модели.

Практическое занятие № 2 «Разработка и оформление технического задания»

Цель работы: Ознакомление с процедурой разработки технического задания на создание программного продукта с применением ГОСТ 19.102-77 «Стадии разработки программ и программной документации».

Форма отчета: выполнением данной лабораторной работы является правильно оформленное техническое задание

Время выполнения: 2 ч

Техническое задание

На данной стадии выполняются следующие работы:

1. Обоснование необходимости разработки программ:

- постановка задачи;
- сбор исходных материалов;
- выбор и обоснование критериев эффективности и качества;
- обоснование необходимости проведения научно-исследовательских работ.

2. Выполнение научно-исследовательских работ:

- определение структуры входных и выходных данных;
- предварительный выбор методов решения задач;
- обоснование целесообразности применения ранее разработанных программ;
- определение требований к техническим средствам;
- обоснование принципиальной возможности решения поставленных задач.

3. Разработка и утверждение технического задания:

- определение требований к программе;
- разработка технико-экономического обоснования разработки программы;
- определение стадий, этапов и сроков разработки программы и документации на нее;
- выбор языков программирования;

- определение необходимости проведения научно-исследовательской работы на последующих стадиях.

Практическое занятие №3 «Построение архитектуры программного средства»

Цель работы: Реализация начальных этапов процесса разработки программного средства в соответствии с ГОСТ Р ИСО/МЭК 12207.

Форма отчета:

С целью реализации начальных этапов разработки ПС в соответствии с техническим заданием:

−выполнить подготовительную работу;

−провести анализ требований к ПС;

−выполнить проектирование архитектуры ПС на высоком уровне.

Время выполнения: 2 ч

При возникновении потребностей в заказе, приобретении, разработке, эксплуатации и сопровождении программ перед всеми сторонами, вовлеченными в жизненный цикл программного средства (ПС), возникает целый ряд вопросов, связанных с определением и детальным структурированием жизненного цикла (ЖЦ) ПС, с организационными и техническими правами и обязанностями сторон, с управлением ЖЦ и контролем за его реализацией. Одним из действенных инструментов для решения данных вопросов является использование унифицированных подходов, закрепленных в современных международных и российских стандартах.

Понятия «жизненный цикл системы» или «жизненный цикл программного средства» часто появляются в статьях и звучат в разговорах разработчиков, по крайней мере, руководителей проектов и подразделений.

Всем понятно, что относятся они к тому, что и в какой последовательности должно делаться при создании и эксплуатации систем. Но прежде чем две организации или два специалиста договорятся о том, что конкретно входит или не входит в ЖЦ, проходит значительное время. А позже вполне может обнаружиться, что эти двое (две «стороны») все-таки по-разному понимают, какие работы будут входить в ЖЦ, а какие - нет, какие проверки будут планироваться, когда и т. д. Естественно, общие принципы организации работ описаны давно, но что делать сторонам в конкретном проекте — это каждый раз приходится решать заново.

В стандартах, регламентирующих жизненный цикл программных средств, обобщаются опыт и результаты исследований множества специалистов и рекомендуются наиболее эффективные современные методы и процессы создания и развития комплексов программ. В результате таких обобщений оттачиваются технологические процессы и приемы разработки, а также методическая база для их автоматизации.

ЖЦ ПС в стандартах представляет собой набор этапов, частных работ и операций в последовательности их выполнения и взаимосвязи, регламентирующих ведение работ от подготовки технического задания до завершения испытаний ряда версий и окончания эксплуатации ПС или информационной системы (ИС).

Стандарты включают правила описания исходной информации, способов и методов выполнения операций, устанавливают правила контроля технологических процессов, требования к оформлению их результатов, а также регламентируют содержание технологических и эксплуатационных документов на комплексы программ. Они определяют организационную структуру коллектива, обеспечивают распределение и планирование заданий, а также, контроль за ходом создания ПС.

Для того чтобы привнести порядок и понимание, общие для любых сторон, участвующих в ЖЦ систем и ПС, давно разрабатывались стандарты различных уровней утверждения - национальные и международные.

В России основы построения и использования профилей стандартов ЖЦ ПС заложены принятием в качестве базового стандарта ГОСТ Р ИСО/МЭК 12207. Данный документ введен в действие с 1 июля 2000 г., тесно взаимоувязан с рядом стандартов, принятых ранее, и с некоторыми стандартами, разрабатываемыми в данное время на основе прямого применения стандартов ИСО.

Актуальность стандарта ГОСТ Р ИСО/МЭК 12207 для современных условий настолько высока, что принятие в ISO его исходного, международного варианта вскоре вызвало самую положительную оценку российских экспертов. Был дан ряд рекомендаций по его использованию в реальных условиях.

В данном стандарте программное обеспечение (ПО) или программный продукт определяется как набор компьютерных программ, процедур исвязанной с ними документации и данных.

Процесс определяется как совокупность взаимосвязанных действий,преобразующих некоторые входные данные в выходные. Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными от других процессов, и результатами.

В соответствии с ГОСТ Р ИСО/МЭК 12207 все процессы ЖЦ ПО разделены на три группы:

1) Основные процессы:

−приобретение;

−поставка;

−разработка;

−эксплуатация;

−сопровождение.

2) Вспомогательные процессы:

−документирование;

−управление конфигурацией;

−обеспечение качества;

−верификация;
−аттестация;
−совместная оценка;
−аудит;
−разрешение проблем.
3) Организационные процессы:
−управление;
−усовершенствование;
−создание инфраструктуры;
−обучение.

Процесс разработки предусматривает действия и задачи, выполняемые разработчиком, и включает следующие действия:

А) Подготовительная работа, которая начинается с выбора модели ЖЦ ПО, соответствующей масштабу, значимости и сложности проекта. Действия и задачи процесса должны соответствовать выбранной модели. Разработчик должен выбрать, адаптировать к условиям проекта и использовать согласованные с заказчиком стандарты, методы и средства разработки, а также составить план выполнения работ.

Б) Анализ требований к системе подразумевает определение ее функциональных возможностей, пользовательских требований, требований к надежности и безопасности, требований к внешним интерфейсам и т.д. Требования к системе оцениваются исходя из критериев реализуемости и возможности проверки при тестировании.

Анализ требований к ПО предполагает определение следующих характеристик для каждого компонента ПО:

−функциональных возможностей, включая характеристики производительности и среды функционирования компонента;
−внешних интерфейсов;
−спецификаций надежности и безопасности;
−эргономических требований;
−требований к используемым данным;
−требований к установке и приемке;
−требований к пользовательской документации;
−требований к эксплуатации и сопровождению.

Требования к ПО оцениваются исходя из критериев соответствия требованиям к системе, реализуемости и возможности проверки при тестировании.

В) Проектирование архитектуры системы на высоком уровне заключается в определении компонентов ее оборудования, ПО и операций, выполняемых эксплуатирующим систему персоналом. Архитектура системы должна соответствовать требованиям, предъявляемым к системе, а также принятым проектным стандартам и методам.

Проектирование архитектуры ПО включает следующие задачи:

− трансформацию требований к ПО в архитектуру, определяющую на высоком уровне структуру ПО и состав ее компонентов;

− разработку и документирование программных интерфейсов ПО и баз данных;

− разработку предварительной версии пользовательской документации;

− разработку и документирование предварительных требований к тестам и планам интеграции ПО.

Архитектура компонентов ПО должна соответствовать требованиям, предъявляемым к ним, а также принятым проектным стандартам и методам.

Г) Детальное проектирование ПО включает следующие задачи:

− описание компонентов и интерфейсов между ними на более низком уровне, достаточном для их последующего самостоятельного кодирования и тестирования;

− разработку и документирование детального проекта базы данных;

− обновление (при необходимости) пользовательской документации;

Д) Кодирование и тестирование ПО охватывает задачи:

− разработку и документирование каждого компонента ПО и базы данных, а также совокупности тестовых процедур и данных для их тестирования;

− тестирование каждого компонента ПО и базы данных на соответствие предъявляемых к ним требованиям. Результаты тестирования компонентов должны быть документированы;

− обновление (при необходимости) пользовательской документации;

− обновление плана интеграции ПО.

Е) Интеграция ПО предусматривает сборку разработанных компонентов ПО в соответствии с планом интеграции и тестирование агрегированных компонентов. Для каждого из агрегированных компонентов разрабатываются наборы тестов и тестовые процедуры, предназначенные для проверки каждого из квалификационных требований при последующем квалификационном тестировании.

Ж) Квалификационное тестирование - это набор критериев и условий,

которые необходимо выполнить, чтобы квалифицировать программный продукт как соответствующий своим спецификациям и готовый к использованию в условиях эксплуатации.

Квалификационное тестирование ПО проводится разработчиком, в присутствии заказчика (по возможности), для демонстрации того, что ПО удовлетворяет своим спецификациям и готово к использованию в условиях эксплуатации. Квалификационное тестирование выполняется для каждого компонента ПО по всем разделам требований при широком варьировании тестов.

З) Установка ПО осуществляется разработчиком в соответствии с планом в той среде и на том оборудовании, которые предусмотрены договором. В процессе установки проверяется работоспособность ПО и баз данных.

И) Приемка ПО предусматривает оценку результатов квалификационного тестирования ПО и системы и документирование результатов оценки, которые проводятся заказчиком с помощью разработчика.

Задание: разработать проект архитектуры программного средства в соответствии с ГОСТ Р ИСО/МЭК 12207.

Практическое занятие №4 «Изучение работы в системе контроля версий»

Цель работы: получение первоначальных навыков использования систем контроля версий исходного кода программ и первоначальных навыков организации коллективной разработки программного обеспечения.

Форма отчета:

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Software Configuration Management или Конфигурационное управление подразумевает под собой комплекс методов, направленных на то, чтобы систематизировать изменения, вносимые разработчиками в программный продукт в процессе его разработки и сопровождения, сохранить целостность системы после изменений, предотвратить нежелательные и непредсказуемые эффекты, а также сделать процесс внесения изменений более формальным.

К процедурам можно отнести создание резервных копий, контроль исходного кода, требований проекта, документации и т. д. Степень формальности выполнения данных процедур зависит от размеров проекта, и при правильной ее оценке данная концепция может быть очень полезна. Конфигурационное управление требует выполнения множества трудоемких рутинных операций. На практике, в большинстве случаев, для конфигурационного управления применяются специальные системы контроля версий исходного кода программ. В качестве примера рассмотрим информационную систему Subversion. Это бесплатная система управления версиями с открытым исходным кодом.

Subversion позволяет управлять файлами и каталогами, а так же сделанными в них изменениями во времени. Это позволяет восстановить более ранние версии данных, предоставляет возможность изучить историю всех изменений.

Задание: Создать тестовый проект на любом знакомом языке программирования и отработать навыки использования хранилища на локальном компьютере.

Практическое занятие № 5 «Построение диаграммы Вариантов использования и диаграммы Последовательности»

Цель работы: научиться строить диаграмму вариантов использования.

Форма отчета:

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая справка:

Визуальное моделирование в UML можно представить как некоторый процесс поуровневого спуска от наиболее обшей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме так называемой диаграммы вариантов использования (use case diagram), которая описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Разработка диаграммы вариантов использования преследует цели:

- Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.
- Сформулировать общие требования к функциональному поведению проектируемой системы.
- Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
- Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

Состав диаграммы Use Case

Диаграмма вариантов использования состоит из актеров, для которых система производит действие и собственно действия Use Case, которое описывает то, что актер хочет получить от системы. Актер обозначается значком человечка, а Use Case - овалом. Дополнительно в диаграммы могут быть добавлены комментарии.

Виды взаимодействий

Между актерами и вариантами использования могут быть различные виды взаимодействия. Основные виды взаимодействия следующие:

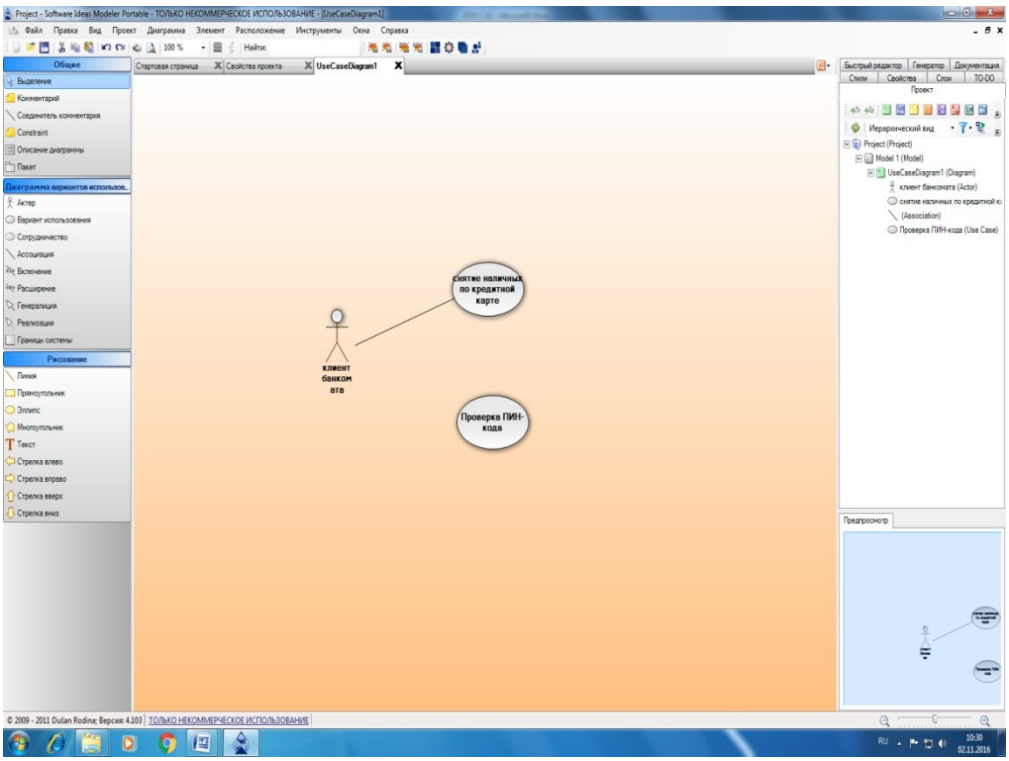
- **Простая ассоциация** - отражается линией между актером и вариантом использования (без стрелки). Отражает связь актера и варианта использования. На рисунке между актером администратор и вариантом использованияпросматривать заказ.
- **Направленная ассоциация** - то же что и простая ассоциация, но показывает, что вариант использования инициализируется актером. Обозначается стрелкой.
- **Наследование** - показывает, что потомок наследует атрибуты и поведение своего прямого предка. Может применяться как для актеров, так для вариантов использования.
- **Расширение** (extend) - показывает, что вариант использования расширяет базовую последовательность действий и вставляет собственную последовательность. При этом в отличие от типа отношений "включение" расширенная последовательность может осуществляться в зависимости от определенных условий.
- **Включение** (include) - показывает, что вариант использования включается в базовую последовательность и выполняется всегда (на рисунке не показан).

Существуют и другие виды взаимодействия, но они менее важны и реже применяются.

Задание 1. Построить диаграмму вариантов использования модели вариантов использования банкомата.

Выполните следующие действия:

1. Добавить актера с именем Клиент банкомата.
2. Добавить вариант использования Снятие наличных по кредитной карте
3. Добавить направленную ассоциацию от бизнес-актера Клиент Банкомата к варианту использования Снятие наличных по кредитной карте
4. Добавить вариант использования Проверка ПИН-кода.



5. Добавить актера с именем Банк.
6. Добавить вариант использования Получение справки о состоянии счета.
7. Добавить вариант использования Блокирование кредитной карточки.
8. Добавить направленную ассоциацию от бизнес-актера Клиент Банкомата к варианту использования Получение справки о состоянии счета.
9. Добавить направленную ассоциацию от варианта использования Снятие наличных по кредитной карточке к сервису Банк.
10. Добавить направленную ассоциацию от варианта использования Получение справки о состоянии счета к сервису Банк.
11. Добавить отношение зависимости со стереотипом «include», направленное от варианта использования Снятие наличных по кредитной карте к варианту использования Проверка Пин-кода.
12. Добавить отношение зависимости со стереотипом «include», направленное от варианта использования Получение справки о состоянии счета к варианту использования Проверка Пин-кода.
13. Добавить отношение зависимости со стереотипом «extend», направленное от варианта использования Блокирование кредитной карточки к варианту использования Проверка Пин-кода.

Задание 2. Построить диаграмму вариантов использования.

Имеются следующие данные:

- четыре действующих лица: Клиента банка, Банк, Кассира и Оператора,
- пять вариантов использования: Снять наличные, Перевести деньги со счета, Положить деньги на счет, Пополнить запас денег и Подтвердить пользователя,

- три зависимости, и отношения между действующими лицами и вариантами использования.

Варианты использования: Снять наличные, Перевести деньги со счета, Положить деньги на счет - требуют включения идентификации клиента в системе. Это поведение может быть выделено в новый вариант использования включения, называемый Подтвердить пользователя. Базовые варианты использования не зависимы от метода, используемого для идентификации. Поэтому он инкапсулируется (скрывается) в варианте использования включения. С точки зрения базовых вариантов использования не имеет значение производится ли идентификация с помощью магнитной карты или сканированием сетчатки глаза. Они только зависят от результата выполнения варианта использования Подтвердить клиента.

Задание для самостоятельной работы: Построить диаграмму вариантов использования на основе вербальной модели информационной системы «Компьютерный клуб»

Контрольные вопросы:

1. Какие цели преследует разработка диаграммы использования?
2. Для чего нужна диаграмма вариантов использования?
3. Из чего состоит диаграмма вариантов использования?
4. Виды взаимодействия используемые в диаграмме вариантов использования?
5. Из чего состоит созданная вами диаграмма?

Практическое занятие № 6 «Построение диаграммы Кооперации и диаграммы Развертывания»

Цель: ознакомиться с методологией моделирования информационных систем на основе языка UML. Теоретические вопросы Универсальный язык моделирования UML. Понятие диаграммы. Виды диаграмм. Основные элементы диаграммы кооперации. Основные элементы диаграммы развертывания.

Форма отчета:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Задание № 1. Ознакомиться с методологией построения диаграммы кооперации основе языка UML. Задание № 2. Проанализируйте пример построения диаграммы кооперации (рисунок 4).

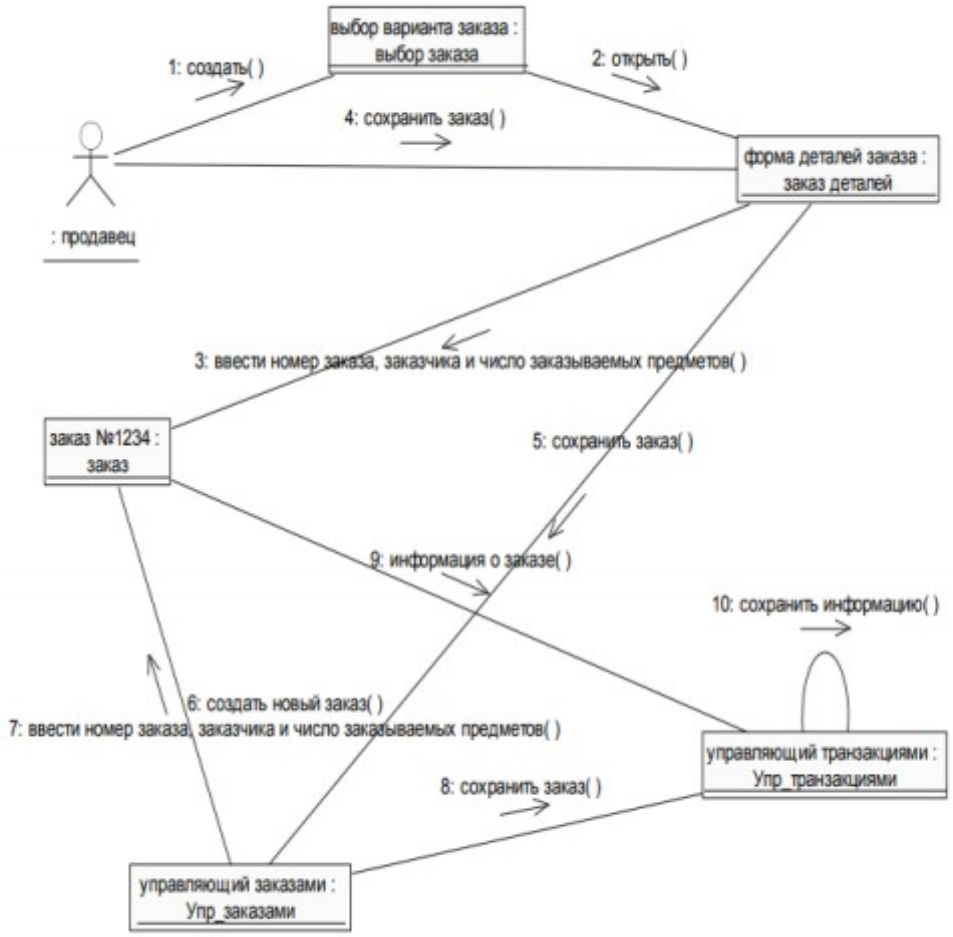


Рис. 4

Задание № 3. Постройте диаграмму кооперации для выбранной информационной системы (практическая работа № 11). Задание № 4. Ознакомиться с методологией построения диаграммы развертывания основе языка UML. Задание № 5. Проанализируйте пример построения диаграммы развертывания. Примеры построения диаграмм развертывания Фрагмент диаграммы развертывания с соединениями между узлами показан на рисунке 5.

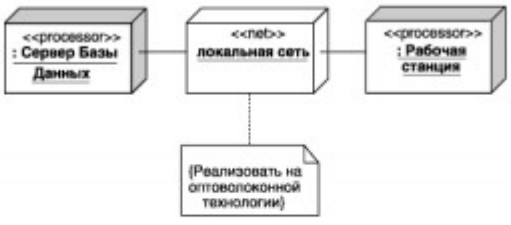


Рисунок 5

Диаграмма развертывания с отношением зависимости между узлом и развернутыми на нем компонентами приведена на рисунке 6.

Рисунок 6

Диаграмма развертывания для системы мобильного доступа к корпоративной базе данных изображена на рисунке 7.

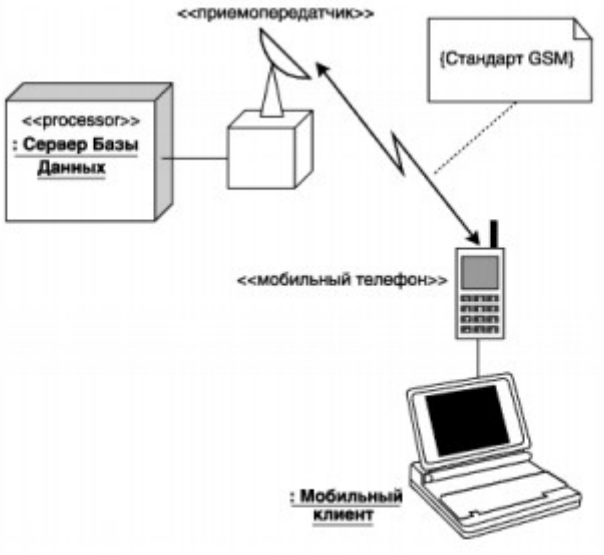


Рисунок 7

Задание № 6. Постройте диаграмму развертывания для выбранной информационной системы (практическая работа №11). Задание № 7. Оформите отчет.

Практическое занятие № 7 «Построение диаграммы Деятельности, диаграммы Состояний и диаграммы Классов»

Целью работы является изучение основ создания диаграмм деятельности на языке UML, получение навыков построения диаграмм деятельности, применение приобретенных навыков для построения объектноориентированных моделей определенной предметной области.

Форма отчета:

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

Время выполнения: 2 ч

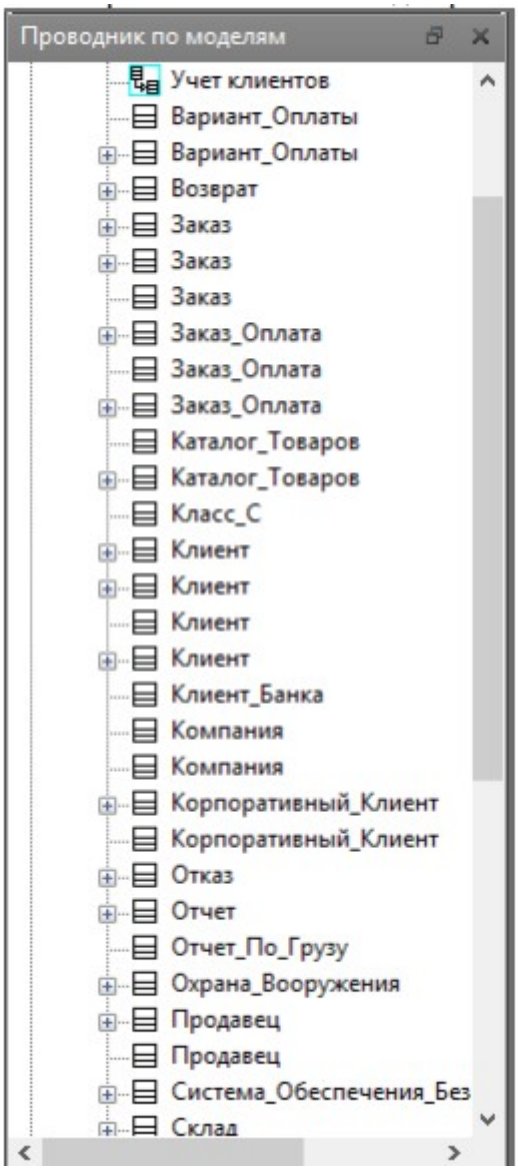
Задачи Основными задачами практической работы являются: – ознакомиться с теоретическими вопросами построения диаграмм деятельности на языке UML; – ознакомиться с теоретическими вопросами построения диаграмм деятельности с помощью MS Visio. 3. Краткие теоретические сведения При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Традиционно для этой цели использовались блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата. Алгоритмические и логические операции, требующие выполнения в определенной последовательности, окружают нас постоянно. Например, чтобы позвонить по телефону, нам предварительно нужно снять трубку или включить его. Для приготовления кофе или заваривания чая необходимо вначале вскипятить воду. Чтобы выполнить ремонт двигателя автомобиля, требуется осуществить целый ряд нетривиальных операций, таких как разборка силового агрегата, снятие генератора и некоторых других. С увеличением сложности системы строгое соблюдение последовательности выполняемых операций приобретает все более важное значение. Если попытаться заварить кофе холодной водой, то мы можем только испортить одну порцию напитка. Нарушение последовательности операций при ремонте двигателя может привести к его поломке или выходу из строя. Еще более катастрофические последствия могут произойти в случае отклонения от установленной последовательности действий при взлете или посадке авиалайнера, запуске ракеты, регламентных работах на АЭС.

Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому. Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может

являться выполнением операции некоторого класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы. В контексте языка UML деятельность (activity) представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом. При этом отдельные элементарные вычисления могут приводить к некоторому результату или действию (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения. Состояние действия и деятельности Состояние деятельности (activity state) – состояние в графе деятельности, которое служит для представления процедурной последовательности действий, требующих определенного времени. Переход из состояния деятельности происходит после выполнения специфицированной в нем ду-деятельности, при этом ключевое слово do в имени деятельности не указывается. Состояние деятельности не может иметь внутренних переходов, поскольку оно является элементарным.

Методика выполнения В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите MS Visio.
2. Откройте файл, созданный в практических занятиях №8-9 и содержащий диаграмму классов.
3. В проводнике по моделям должны отображаться все классы и диаграммы, созданные ранее (рис. 9).



Опишем с помощью диаграммы деятельности процесс формирования заказа и выдачу товара. В бизнеспроцессе участвуют 3 действующих лица: клиент, продавец и система оплаты. Следовательно, необходимо добавить 3 дорожки для распределения ответственности между этими лицами. Для этого, в файле с диаграммой классов, созданной в практическом занятии 8, необходимо проделать следующие действия:

6. Щелкнуть правой кнопкой мыши по классу Заказ.

7. В контекстном меню выбрать пункт Схемы.

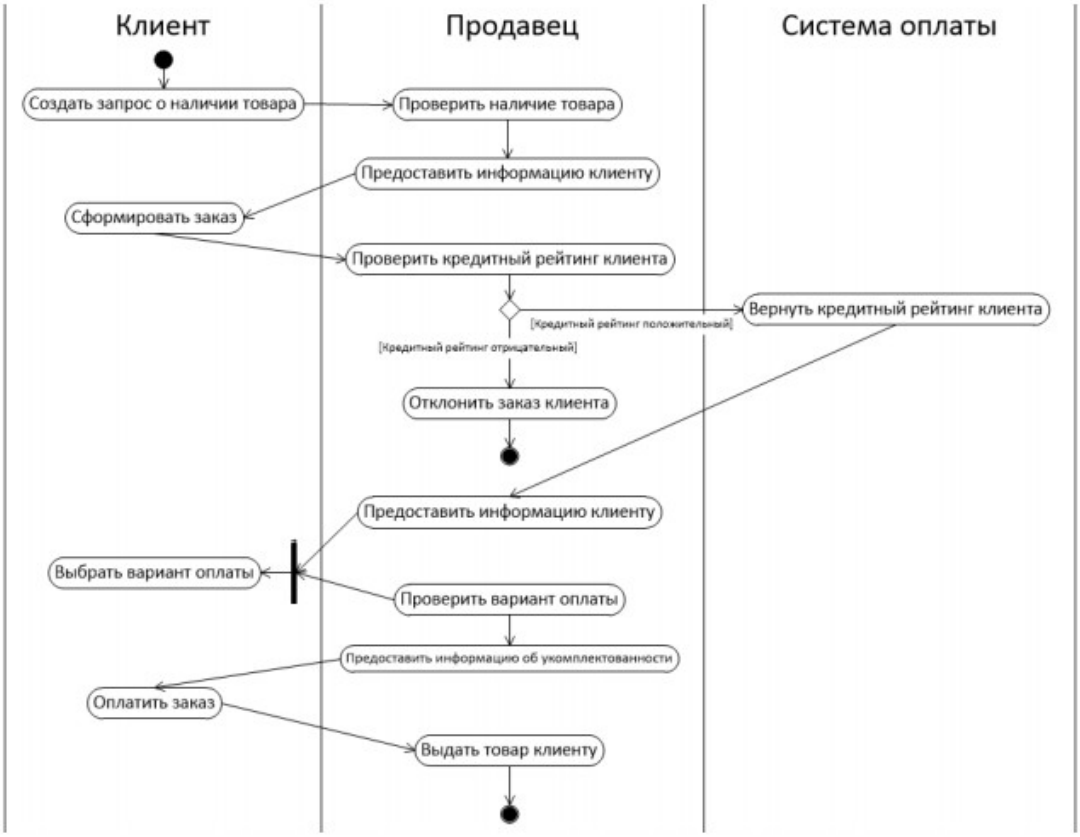
8. Нажать кнопку Создать и выбрать Деятельность.

9. Переименовать созданный лист в Деятельность-Заказ.

10.Построить диаграмму деятельности для класса Заказ. Для это выполните действия, описанные ниже.

a. Добавить 3 элемента Дорожка и изменить их названия на Клиент, Продавец и Система оплаты соответственно.

b. Добавить элементы Начальное состояние, Конечное состояние, Состояние действия, Решение, Переход (объединение), изменить их названия и задать расположение в соответствии с рисунком 10.



Задание Построить диаграмму деятельности в соответствии с вариантом. Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию. Создать диаграммы деятельности не менее чем для трех классов, описанных в практическом занятии №8.

Варианты 1. «Отдел кадров»;

2. «Агентство аренды»;

3. «Аптека»;

4. «Ателье»;

5. «Аэропорт»;

6. «Библиотека»;

7. «Кинотеатр»;

8. «Поликлиника»;

9. «Автосалон»;

10.«Таксопарк».

Практическое занятие № 8 «Построение диаграммы компонентов»

Цель и содержание работы: научиться создавать диаграммы Компонентов системы, добавлять компоненты к пакетам, изображать зависимости.

Постановка задачи

Разработаем диаграммы Компонентов и создадим для каждого класса соответствующие языку программирования C++ компоненты.

Теоретическое обоснование

Диаграмма компонентов предназначена для распределения классов и объектов по компонентам при физическом проектировании системы.

Компоненты на диаграмме компонентов представляют собой физические модули программного кода. Обычно они в точности соответствуют пакетам на диаграмме пакетов; таким образом, диаграмма компонентов отражает выполнение каждого пакета в системе.

Зависимости между компонентами должны совпадать с зависимостями между пакетами. Эти зависимости показывают, каким образом одни компоненты взаимодействуют с другими. Направление данной зависимости показывает уровень осведомленности о коммуникации.

Методика и порядок выполнения работы

На рис.34 показана главная диаграмма Компонентов всей системы. Внимание на ней уделяется пакетам создаваемых компонентов.

На рис.35 показаны компоненты пакета Entities. Эти компоненты содержат классы пакета Entities Логического представления системы.

На рис.36 показаны компоненты пакета Control. Они содержат классы пакета Control Логического представления системы.

Рис.34 - Главная диаграмма Компонентов системы

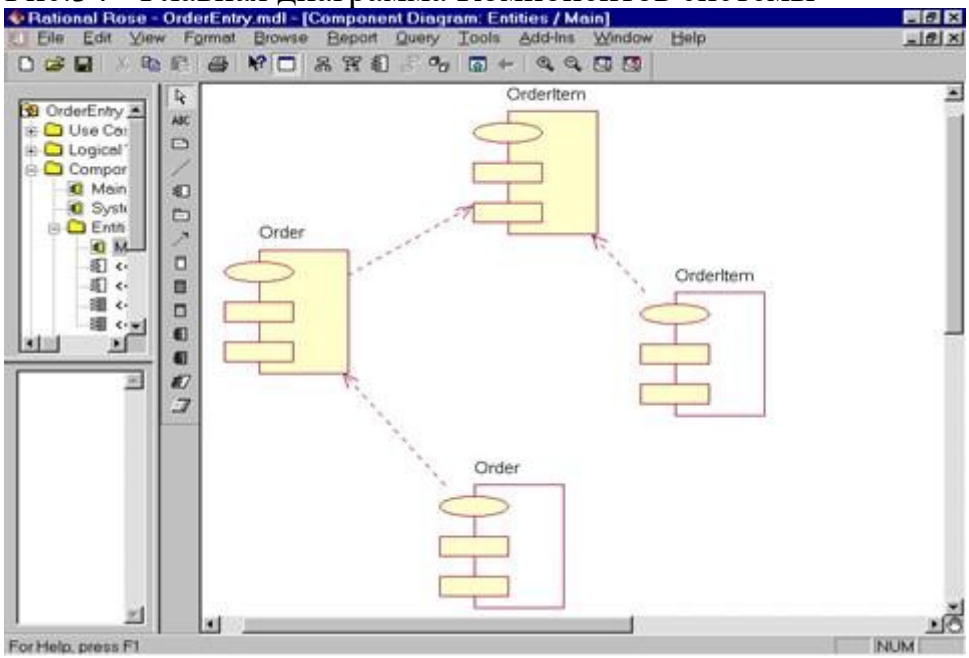


Рис.35 - Диаграмма Компонентов пакета Entities

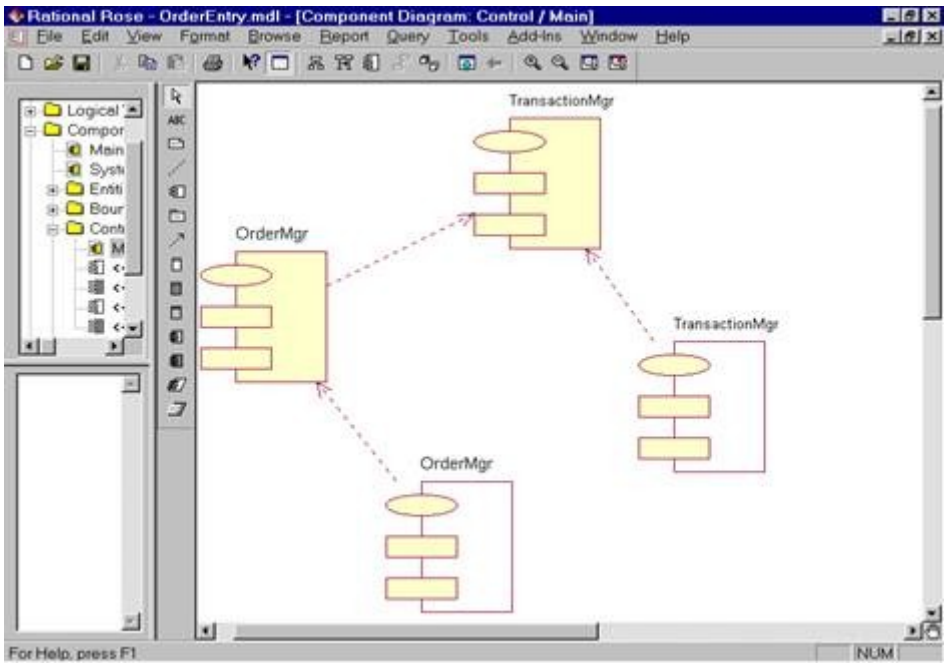


Рис.36 - Диаграмма Компонентов пакета Control

Наконец, на рис.37 показаны компоненты пакета Boundaries. Они также соответствуют классам одноименного пакета Логического представления системы.

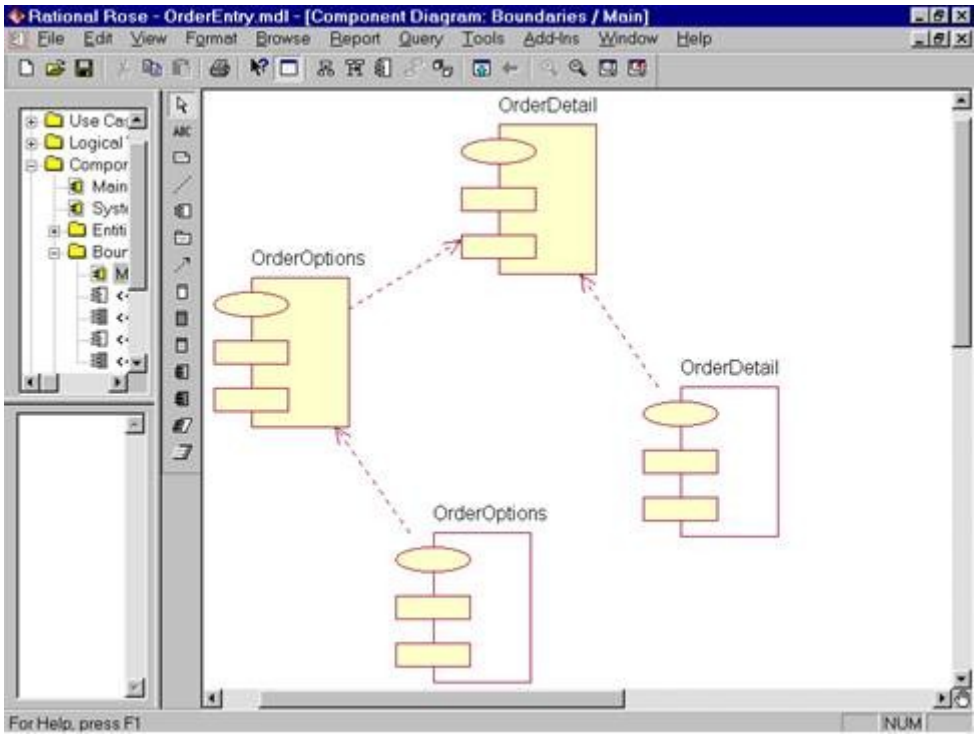


Рис.37 - Диаграмма Компонентов пакета Boundaries

На рис.38 показаны диаграмма Компонентов системы. На ней вы можете видеть все зависимости между всеми компонентами проектируемой системы.

Этапы выполнения упражнения

Создание пакетов компонентов

1. Щелкните правой кнопкой мыши на представлении компонентов в броузере.

2. В открывшемся меню выберите пункт New > Package (Создать > пакет).

3. Назовите этот пакет Entities (Сущности).

4. Повторите этапы с первого по третий, создав пакеты Boundaries (Границы) и Control (Управление).

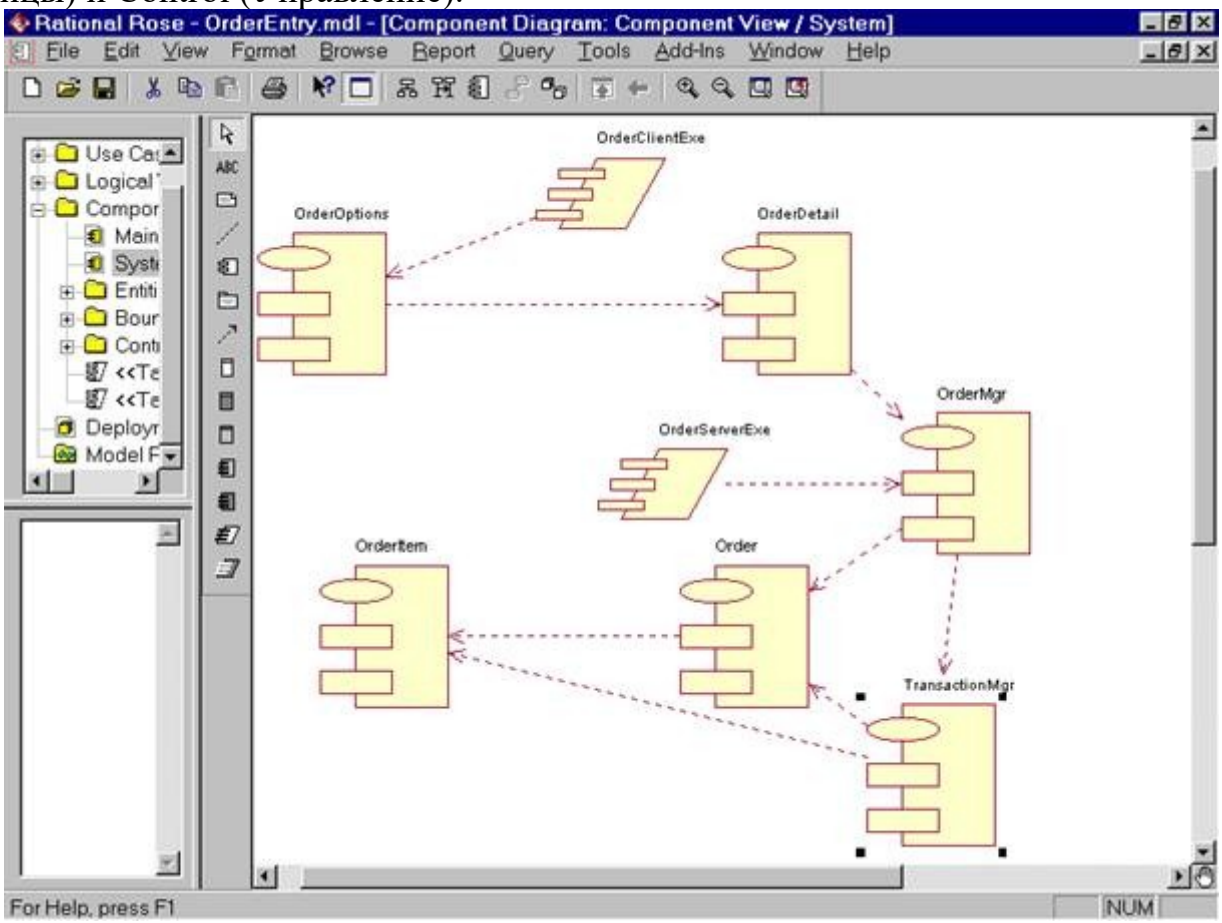


Рис.38 - Диаграмма Компонентов системы.

Добавление пакетов на Главную диаграмму Компонентов

1. Откройте Главную диаграмму Компонентов, дважды щелкнув на ней.

2. Перетащите пакеты Entities, Boundary и Control из броузера на Главную диаграмму.

Рисование зависимостей между пакетами

1. На панели инструментов нажмите кнопку Dependency (Зависимость).

2. Щелкните мышью на упаковке Boundaries Главной диаграммы Компонентов.

3. Проведите линию зависимости до упаковки Control.

4. Повторите этапы 1 - 3, проведя еще зависимость от пакета Control до пакета Entities.

Добавление компонентов к пакетам и рисование зависимостей

1. Дважды щелкните мышью на пакете Entities Главной диаграммы Компонентов, открыв Главную диаграмму Компонентов этого пакета.

2. На панели инструментов нажмите кнопку Package Specification (Спецификация пакета).

3. Поместите спецификацию пакета на диаграмму.

4. Введите имя спецификации пакета OrderItem.

5. Повторите этапы 2 - 4, добавив спецификацию пакета Order.

6. На панели инструментов нажмите кнопку Package Body (Тело пакета).
7. Поместите его на диаграмму.
8. Введите имя тела пакета OrderItem.
9. Повторите этапы 6 - 8, добавив тело пакета Order.
10. На панели инструментов нажмите кнопку Dependency (Зависимость).
11. Щелкните мышью на теле пакета OrderItem.
12. Проведите линию зависимости от него к спецификации пакета OrderItem.
13. Повторите этапы, добавив линию зависимости между телом пакета Order и спецификацией пакета Order.
14. Повторите этапы, добавив линию зависимости от спецификации пакета Order к спецификации пакета OrderItem.
15. С помощью описанного метода создайте следующие компоненты и зависимости:

Для пакета Boundaries:

Спецификацию пакета OrderOptions
Тело пакета OrderOptions
Спецификацию пакета OrderDetail
Тело пакета OrderDetail

Зависимости в пакете Boundaries:

От тела пакета OrderOptions до спецификации пакета OrderOptions
От тела пакета OrderDetail до спецификации пакета OrderDetail
От спецификации пакета OrderOptions до спецификации пакета OrderDetail

Создание диаграммы Компонентов системы

1. Щелкните правой кнопкой мыши на представлении Компонентов в броузере.
2. В открывшемся меню выберите пункт New→Component Diagram
3. Назовите новую диаграмму System.
4. Дважды щелкните на этой диаграмме.

Размещение компонентов на диаграмме Компонентов системы

1. Если это еще не было сделано, разверните в броузере пакет компонентов Entities, чтобы открыть его.
2. Щелкните мышью на спецификации пакета Order в пакете компонентов Entities.
3. Перетащите эту спецификацию на диаграмму.
4. Повторите этапы 2 и 3, поместив на диаграмму спецификацию пакета OrderItem.
5. С помощью этого метода поместите на диаграмму следующие компоненты:

Из пакета компонентов Boundaries:

Спецификацию пакета OrderOptions
Спецификацию пакета OrderDetail

Из пакета компонентов Control:

Спецификацию пакета OrderMgr

Спецификацию пакета TransactionMgr

6. На панели инструментов нажмите кнопку Task Specification (Спецификация задачи).

7. Поместите спецификацию задачи на диаграмму и назовите ее OrderClientExe.

8. Повторите этапы 6 и 7 для спецификации задачи OrderServerExe.

Оформите отчет в виде скриншотов в ворде.

Практическое занятие № 9 «Построение диаграмм потоков данных»

Цель работы

Ознакомиться с методологией построения диаграмм потоков данных с использованием программного продукта BpWin.

Форма отчета:

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Содержание работы и методические указания к ее выполнению

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.

<=""

p="">

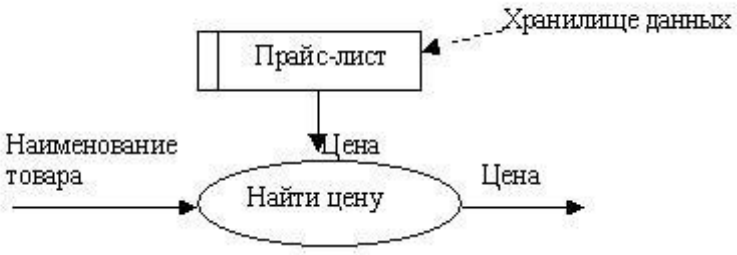
Поток данных соединяет выход объекта (или процесса) с входом другого объекта (или процесса) и представляет собой промежуточные данные вычислений. Поток данных изображается в виде стрелки между производителем и потребителем данных, помеченной именами соответствующих данных. Дуги могут разветвляться или 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.

<=""

p="">

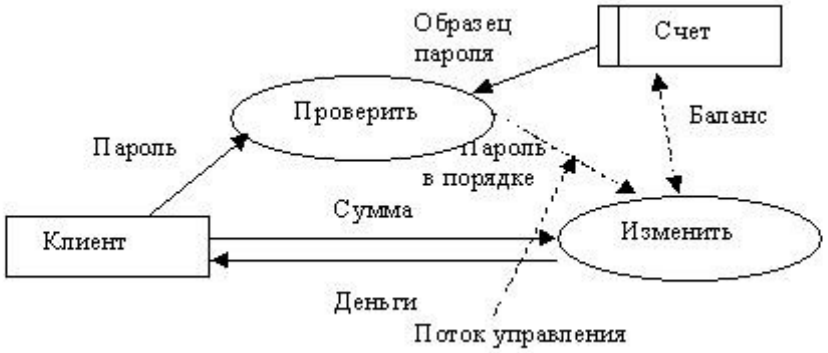
Хранилища данных. Хранилище данных - это пассивный объект в составе DFD, в котором данные сохраняются для последующего доступа. Хранилище данных допускает доступ к хранимым в нем данным в порядке, отличном от того, в котором они были туда помещены. Агрегатные хранилища данных, как, например, списки и таблицы, обеспечивают доступ к данным в порядке их поступления, либо по ключам.



<=""

p="">

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.



<=""

p="">

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.

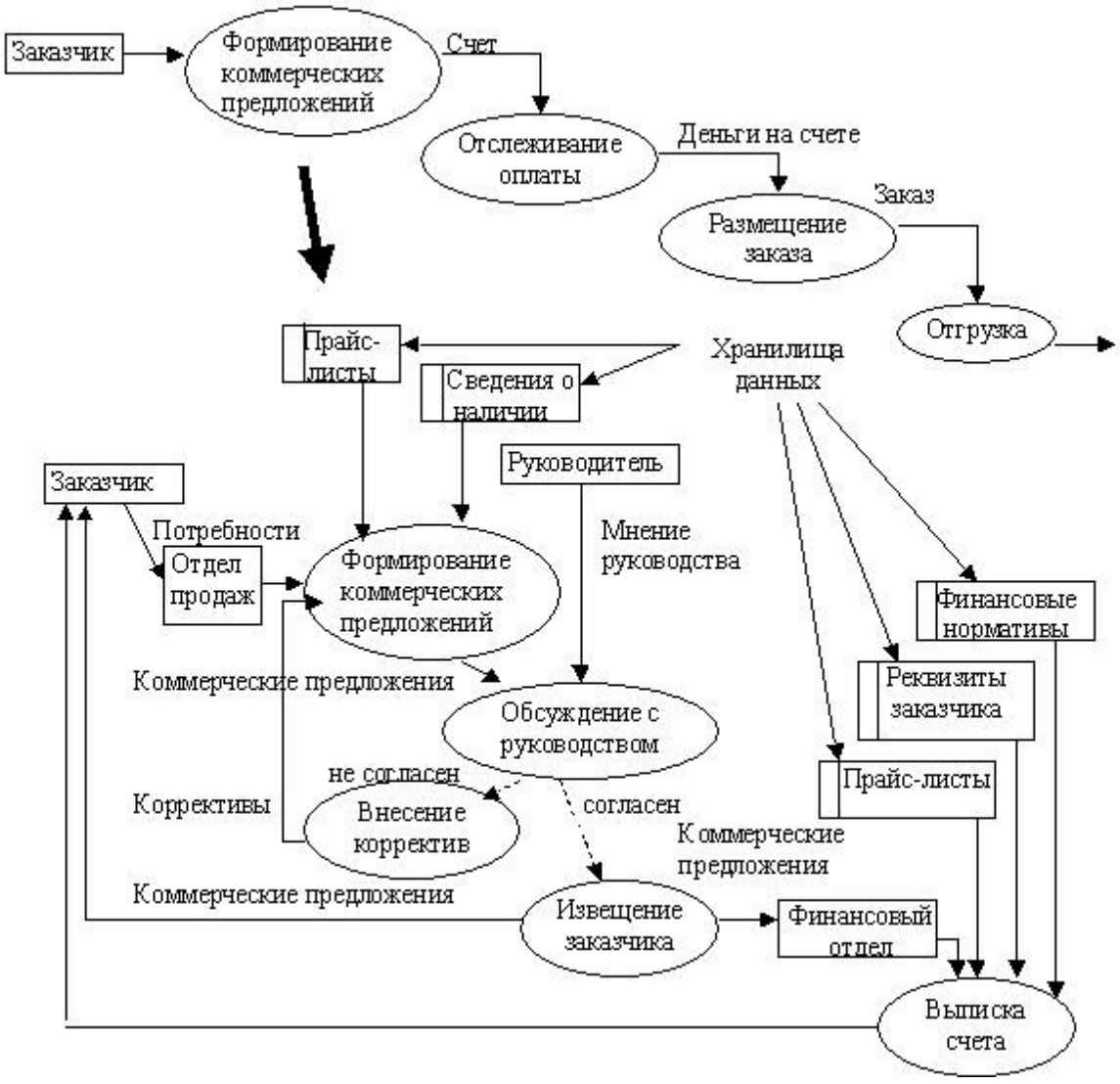
Если же для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и, кроме того, главный единственный процесс не раскрывает структуры распределенной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



<
="" p="">

Выполнение лабораторной работы производится с использованием программного продукта Enterprise Architect (методические указания по использованию Enterprise Architect)

Последовательность выполнения лабораторной работы:

1. Ознакомиться с методологией диаграмм потоков данных.

2. Ознакомиться с программным продуктом Enterprise Architect в части средств работы с диаграммами потоков данных.

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

4. Оформить отчет.

Практическое занятие 10 «Разработка тестового сценария»

Цель: получить навыки разработки тестовых сценариев. Теоретические вопросы Оценка стоимости и причины ошибок в программном обеспечении. Виды и методы тестирования. Понятие теста. Требования к разработке тестовых сценариев. Правила разработки тестовых сценариев.

Форма отчета:
−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.
Время выполнения: 2 ч

Задание № 1. Написать программу решения квадратного уравнения ax2 + bx + c = 0. Задание № 2. Найти минимальный набор тестов для программы нахождения веще-ственных корней квадратного уравнения ax2 + bx + c = 0. Решение представлено в таблице.

Номер теста	a	b	c	Ожидаемый результат	Что проверяется
1	2	-5	2	$x_1=2, x_2=0,5$	Случай вещественных корней
2	3	2	5	Сообщение	Случай комплексных корней
3	3	-12	0	$x_1=4, x_2=0$	Нулевой корень
4	0	0	10	Сообщение	Неразрешимое уравнение
5	0	0	0	Сообщение	Неразрешимое уравнение
6	0	5	17	Сообщение	Неквадратное уравнение
7	9	0	0	$x_1=x_2=0$	Нулевые корни

Таким образом, для этой программы предлагается минимальный набор функциональных тестов, исходя из 7 классов выходных данных. Заповеди по отладки программного средства, предложенные Г. Майерсом.

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

Заповедь 2. Хорош тот тест, для которого высока вероятность обнаружить ошибку, а не тот, который демонстрирует правильную работу программы.

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

Заповедь 4. Документируйте пропуск тестов через компьютер, детально изучайте результаты каждого теста, избегайте тестов, пропуск которых нельзя повторить.

Заповедь 5. Каждый модуль подключайте к программе только один раз, никогда не изменяйте программу, чтобы облегчить ее тестирование.

Заповедь 6. Пропускайте заново все тесты, связанные с проверкой работы какой-либо программы ПС или ее взаимодействия с другими программами, если в нее были внесены изменения (например, в результате устранения ошибки).

Задание № 6. Разработайте набор тестовых сценариев (как позитивных, так и негативных) для следующей программы: Имеется консольное приложение (разработайте самостоятельно). Ему на вход подается 2 строки. На выходе приложение выдает число вхождений второй строки в первую. Например:

Строка 1	Строка 2	Вывод
абвгабвг	аб	2
стстсап	стс	2

Набор тестовых сценариев запишите в виде таблицы, приведенной выше.

Задание № 3. Оформите отчет.

Практическое занятие № 12 «Оценка необходимого количества тестов»

Цель: Научиться проводить оценку качества программного средства по различным показателям.

Форма отчета:

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Основные теоретические сведения

Все программы по характеру использования и категориям пользователей можно разделить на два класса - утилитарные программы и программные продукты (изделия). Утилитарные программы («программы для себя») предназначены для удовлетворения нужд их разработчиков. Чаще всего утилитарные программы выполняют роль сервиса в технологии обработки данных либо являются программами решения функциональных задач, не предназначенных для широкого распространения. Программные продукты (изделия) предназначены для удовлетворения потребностей пользователей, широкого распространения и продажи. Существуют и другие варианты легального распространения программных продуктов: - freeware – бесплатные программы, свободно распространяемые, поддерживаются самим пользователем,

который правомочен вносить в них необходимые изменения; - shareware – некоммерческие (условно-бесплатные) программы, которые могут использоваться, как правило, бесплатно. Ряд производителей использует OEM - программы (Original Equipment Manufacturer), т.е. встроенные программы, устанавливаемые на компьютеры или поставляемые вместе с вычислительной техникой. Программные продукты (ПП) могут создаваться как: - индивидуальная разработка под заказ; - разработка для массового распространения среди пользователей.

Основными характеристиками программ являются:

- алгоритмическая сложность (логика алгоритмов обработки информации);
- состав и глубина проработки реализованных функций обработки;
- полнота и системность функций обработки;
- объём файлов программ;
- требования к операционной системе и техническим средствам обработки со стороны программного средства;
- объём дисковой памяти;
- размер оперативной памяти для запуска программ;
- тип процессора;
- версия операционной системы;
- наличие вычислительной сети и др. Программные продукты имеют многообразие показателей качества, которые отражают различные аспекты. Основная характеристика программного продукта – это его общая полезность, которая включает в себя мобильность, исходную полезность и удобство эксплуатации. Мобильность ПП означает их независимость от технического комплекса системы обработки данных, операционной среды, сетевой технологии обработки данных, специфики предметной области и т.п. Мобильный (многоплатформный) программный продукт может быть установлен на различных моделях компьютеров и операционных систем, без ограничений на его эксплуатацию в условиях вычислительной сети. Функции обработки такого программного продукта для массового использования без каких-либо изменений. Исходная полезность характеризуется следующими показателями: - надежность;

- эффективность;
- учет человеческого фактора;

Надежность работы ПП определяется бессбойностью и устойчивостью в работе программ, точностью выполнения предписанных функций обработки, возможностью диагностики возникающих в процессе работы программ ошибок. Эффективность ПП оценивается как с позиций прямого его назначения – требований пользователя, так и сточки зрения расхода вычислительных ресурсов, необходимых для его эксплуатации. Расход вычислительных ресурсов оценивается через объем внешней памяти для размещения программ и объём оперативной памяти для запуска программ. Учёт человеческого фактора означает обеспечение дружественного интерфейса для работы конечного пользователя, наличие котекстно- зависимой подсказки или обучающей системы в составе программного средства, хорошей документации для освоения и

использования, заложенных в программном средстве функциональных возможностей, анализ и диагностику возникших ошибок и др. Удобство эксплуатации включает следующие показатели качества: - модифицируемость; - коммуникативность. Модифицируемость ПП означает способность к внесению изменений, например расширение функций обработки, переход на другую техническую базу обработки и т.п. Коммуникативность ПП основана на максимально возможной их интеграции с другими программами, обеспечении обмене данными в общих форматах представления (экспорт/импорт баз данных, внедрение или связывание объектов обработки и др.). Естественно, что в условиях существования рынка программных продуктов важными характеристиками являются: стоимость; количество продаж; длительность продаж (время нахождения на рынке); известность фирмы-разработчика и программы; наличие программных продуктов аналогического назначения. Для оценки качества программного средства (ПС) используются различные способы получения информации о нём: - измерительный – основан на получении информации о свойствах и характеристиках ПС с использованием инструментальных средств (например, объём ПС); - регистрационный – получение информации во время испытаний или функционирования ПС, когда регистрируется и подсчитываются определённые события (число сбоев и отказов и др.);

Задание на лабораторную работу:

1. Скачать калькулятор любого производителя или взять разработанный студентами.

2. Сравнить два программных продукта: калькулятор фирмы Microsoft и калькулятор, написанный студентами (скачанный). Сравнение проводить по следующим оценочным элементам: надежность ПС, сопровождаемость, корректность. Критерии оценки (1 или 0)

3. Все сравнение занести в следующую таблицу.

Практическое занятие №13 «Разработка тестовых пакетов»

Цель: получить навыки разработки тестовых пакетов.

Форма отчета:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретические вопросы

Системные основы разработки требований к сложным комплексам программ. Формализация эталонов требований и характеристик комплекса программ. Формирование требований компонентов и модулей путем

декомпозиции функций комплексов программ. Тестирование по принципу «белого ящика».

Задание № 1. В Древней Греции (II в. до н.э.) был известен шифр, называемый "квадрат Полибия". Шифровальная таблица представляла собой квадрат с пятью столбцами и пятью строками, которые нумеровались цифрами от 1 до 5. В каждую клетку такого квадрата записывалась одна буква. В результате каждой букве соответствовала пара чисел, и шифрование сводилось к замене буквы парой чисел. Для латинского алфавита квадрат Полибия имеет вид:

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I, J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Пользуясь изложенным способом создать программу, которая: а) зашифрует введенный текст и сохранит его в файл; б) считает зашифрованный текст из файла и расшифрует данный текст. Задание № 2. Спроектировать тесты по принципу «белого ящика» для программы, разработанной в задании № 1. Выбрать несколько алгоритмов для тестирования и обозначить буквами или цифрами ветви этих алгоритмов. Выписать пути алгоритма, которые должны быть проверены тестами для выбранного метода тестирования. Записать тесты, которые позволят пройти по путям алгоритма. Протестировать разработанную вами программу. Результаты оформить в виде таблиц:

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
…	…	…	…

Задание № 3. Проверить все виды тестов и сделать выводы об их эффективности..

Задание № 4. Оформить отчет.

Практическое занятие № 14 «Оценка программных средств с помощью метрик»

Цель: Получить навыки по разработке метрик для моделей качества интерфейсов пользователя программных систем

Форма отчета:

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Теоретическая часть

1. Метрики качества программных систем

Атрибуты (характеристики) программной системы, характеризующие ее качество, измеряются с использованием метрик качества.

Метрика – это комбинация конкретного метода измерения (способа получения значений) атрибута сущности и шкалы измерения (средства, используемого для структурирования получаемых значений).

Метрика определяет (вычисляет) меру атрибута – переменную, которой присваивается значение в результате измерения.

Мера атрибута может быть непосредственной, если она не зависит от мер других атрибутов, либокосвенной, полученной по мерам других атрибутов.

По определению стандарта ISO/IEC 9126-2 метрика качества программной системы представляет собой «модель измерения атрибута, связываемого с характеристикой качества ПС. Служит индикатором одного или многих атрибутов. Ее можно увидеть, например, в левой части большинства уравнений X = A*B, где X имеет не ту же шкалу, что A или B».

Метрика называется базовой, если в ее основе лежит элементарный метод (примитив) измерения атрибута. По определению того же стандарта «базовая метрика сама по себе не является индикатором характеристики или подхарактеристики качества. Ее можно увидеть, например, в правой части большинства уравнений X =A*B. A и B – базовые метрики». То есть базовые метрики используются только в составе модели измерения атрибута.

Для того чтобы правильно пользоваться результатами измерений, для каждой меры нужно идентифицировать шкалу измерения.

Стандарт ISO/IEC 9126-2 рекомендует применять 5 видов шкалы измерения значений (упорядоченных от менее строгих к более строгим):

· номинальная шкала. Это классификационная шкала, выполняющая категоризацию свойств оцениваемого объекта. Категории не упорядочены.

Например, дефекты могут классифицироваться на дефекты интерфейса, логики, объявления данных и др. Языки – Fortran, C++, Java и др.;

· порядковая шкала. Позволяет упорядочивать характеристики по возрастанию или убыванию путем сравнения их с базовыми значениями. Например, для уровня серьезности последствий события шкала может включать значения «низкий», «средний», «высокий», «критический». Для уровней CMM – 1, 2, 3, 4, 5. Расстояние между значениями по шкале не играет роли. Характеристики, имеющие номинальную или порядковую шкалу измерения, называются качественными (или категорийными). Все остальные - количественными.

· интервальная шкала. Отмечает существенные различия свойств объекта, «дистанцию» между ними (например, календарные даты или значения плотности дефектов - 1.5 дефекта/KSLOC, 3.5 дефекта/KSLOC и т.д.). Используется в арифметических операциях и операциях сравнения (в данном примере разница равна 2 дефекта/KSLOC). Нулевое значение не допустимо;

· относительная шкала. Значения по этой шкале различаются по отношению к выбранной единице (например, времени, изменяющемся от 0 до бесконечности, или стоимости). Применяя эту шкалу можно рассчитать, например, время между отказами, размер программного компонента в SLOC и др. Считается наиболее предпочтительной шкалой измерений. Позволяет применять широкий спектр инструментов измерения (гистограмм, диаграмм Парето и др.);

· абсолютная шкала. Это специальный случай относительной шкалы. В

этой шкале указывается абсолютное значение величины. Например: «размер программы равен 2К», «число обнаруженных ошибок равно 20».

Измеренное значение метрики само по себе не несет информации об уровне удовлетворения требований к качеству. Для этих целей шкала должна быть разделена на области (ранги), соответствующие различным степеням удовлетворения требований. Примеры деления шкалы:

· деление значений по двум категориям - удовлетворительные и неудовлетворительные значения;

· деление шкалы по четырем категориям.

По мере накопления практики измерений и знаний об измеряемых атрибутах шкалы их измерения могут эволюционировать от менее информативных (номинальной и порядковой) к более информативным (относительной или абсолютной).

1. Познакомьтесь с метриками, которые приведены в примерах теоретической части.

2. Познакомьтесь с характеристиками и требованиями к ним, которые предъявляются к человеко-машинным интерфейсам программно-технических комплексов а атомной энергетике.

3. Выберите вариант задания

4. Определите метрики для оценки заданной характеристики качества. Для этого выберите метрики из моделей 9126, QUIM и т.л. или разработайте собственные оригинальные метрики для следующих характеристик пользовательских программных интерфейсов.

5. Задайте метрику в форме спецификации табл. 7

6. Обоснуйте адекватность выбранных метрик.

7. Составьте отчет

Варианты заданий

Таблица 11 Характеристики и требования к ним

№	Характеристика	Описание
1.	Безопасность персонала (Personnel Safety)	Дизайн интерфейса должен обеспечивать минималь ную возможность травм и воздействия вредных материалов

2.	Когнитивная совместимость (Cognitive Compatibility)	Роль оператора должна состоять из целенаправленных и значимых задач, которые позволяют персоналу поддерживать хорошую осведомленность с АЭС и поддерживать уровень нагрузки, который не настолько высокий, чтобы негативно повлиять на производительность, но достаточной для поддержания бдительности
3.	Физиологическая совместимость (Physiological Compatibility)	Дизайн интерфейса должен отражать рассмотрениефизиологических характеристик человека, включая визуальное / слуховое восприятие, биомеханику(достижения и движения), характеристики управления, и антропометрии
4.	Простота конструкции (Simplicity of Design)	ЧМИ должны представлять простой дизайн в соответствии с функциональными требованиями и требованиями задачи
5.	Согласованность (Consistency)	Должна быть высокая степень согласованности между ЧМИ, процедурами и обучающими системами. В ЧМИ пути системных функций и деятельности бригады всегда должны быть согласованы, отражать высокую степень стандартизации, и быть в полном соответствии с процедурами и подготовкой кадров
6.	Понимание ситуации (Situation Awareness)	Информация, представленная пользователям ЧМИ должна быть правильно, быстро и легко понята (например, "непосредственное восприятие"или "определение состояния с одного взгляда" надисплее) и поддерживаться на высоком уровне с целью осведомленности

		пользователей о статусе системы
		Система должна отвечать требованиям пользователей для выполнения своих задач (в том числе, безопасное завершение работы, осмотр, техническое обслуживание и ремонт).
7.	Целевая совместимость (Task Compatibility)	Данные должны быть представлены в формах и форматах, соответствующих задач (включая, необходимость доступа к подтверждающим данным или необработанным данным в случае отображения более высокого уровня).
		Возможность контроля должна охватывать ряд потенциальных действий. Не должно быть ненужной информации или вариантов контроля
8.	Пользовательская модель совместимости (User Model Compatibility)	Все аспекты системы должны быть совместимы с ментальными (психическими) моделями пользователей(понимание и ожидание поведения системы осуществляется путем подготовки кадров, использования процедур и опыта).Все аспекты системы должны быть совместимы с установленными допущениями, т.е. должны быть выражены в обычной, привычной, пригодной и функциональной точки зрения, а не абстрактно
9.	Структура элементов ЧМИ (Organization of HSI Elements)	Структура всех аспектов ЧМИ (от элементов вотдельных дисплеях до отдельных рабочих станций и всейкомнаты управления) должна быть основана на требованияхпользователя и должна отражать общие принципы организации по важности, частоте и порядке использования. Информация критических функций безопасности должна быть доступна всем,

		работающим в команде, для обеспечения ее распознавания и сведения к минимуму поиска данных и ответных мер
10.	Логическая/Явная структура (Logical/Explicit Structure)	Все аспекты системы (форматы, терминология, последовательность, группировка, и поддержка принятия решений оператора) должна отражать очевидную логику, основанную на требованиях задачи или других непроизвольных обоснованиях. Отношения каждого отображения, управления и обработки данных для общей задачи/функции должны быть ясными. Структура интерфейса и связанная с ней навигация должны быть сделаны легкой для пользователей, чтобы было понятно, где они находятся в пространстве данных. Структура интерфейса должна позволить пользователям получить быстрый доступ к данным, не видимым в настоящее время (например, на других страницах дисплей). Ход работы системы и структурированность должны быть ясными для пользователя
11.	Своевременность (Timeliness)	Проектирование системы должны принимать во внимание когнитивные возможности пользователей, а также связанные с процессом ограничения времени для обеспечения того, чтобы задачи были выполнены в срок. Скорость информационного потока и требования контроля за исполнением, которые являются слишком быстрыми или слишком медленными могут привести к снижению производительности
12.	Совместимость управления/отображения	Отображения должны быть совместимы с вводимыми данных и требованиями управления

	(Controls/Displays Compatibility)	
13.	Обратная связь (Feedback)	Система должна давать полезную информацию о состоянии системы, допустимых операциях, ошибках и восстановлении после ошибки, опасных операциях, и достоверности данных
14.	Когнитивная нагрузка (Cognitive Workload)	Информация, представленная системой должна быстровоспрниматься и пониматься. Поэтомусистема должна минимизировать требования для вычислений или преобразований в уме и использовать напоминания (ссылаясь на длинные списки кодов, сложные команды, информацию с одного экрана на другой,или длительные последовательности действий). Исходные данные должны быть обработаны и представлены внепосредственно удобной форме. Исходные данные должны быть доступны для подтверждения
15.	Нагрузка ответа (реакции) (Response Workload)	Система должна требовать минимальное количество действий для получения результата. Например,одну команду ввода вместо нескольких команд, меню выбора вместо многократных команд, один режим ввода (клавиатура, мышь) вместо смешанного режима.Система не должна требовать ввода избыточных данных, повторного ввода информации, имеющейся уже в системе,или информации, которую система может генерировать по уже поступившим данным
16.	Гибкость	Система должна предоставить

6.	(Flexibility)	пользователю несколько способов для совершения действий ипроверить автоматические действия. Отображение и контроль должен быть отформатирован в конфигурации наиболее удобной для задачи.Однако, гибкость должна быть ограничена ситуациями, когда она предлагает преимущества в выполнении задачи (например, для приспособления к различным уровням опыта пользователей)
7.	Руководства и поддержка пользователя (User Guidance and Support)	Система должна обеспечить эффективную "Помощь".Информативные, легкие в использовании рекомендации должны быть предоставлены в он-лайн и офф-лайн режимах, чтобы помочь пользователю понять, как работать с системой
8.	Толерантность и управление ошибками (Error Tolerance and Control)	Отказоустойчивый дизайн должен предоставляться везде, где сбой может привести к повреждению оборудования, травмам персонала, или непреднамеренной работе критически важного оборудования.Таким образом, система должна вообще быть сконструирована таким образом, чтобы ошибки пользователя не имели серьезных последствий. Надо управлять негативными последствиями ошибок, и сводить их к минимуму. Система должна предлагать простые,понятные уведомления об ошибке, и простые, эффективные методы для восстановления

Практическое занятие № 15 «Разработка структуры проекта»

Цели:

На базе лекционного материала по структурам проекта, где рассматривались типовые структуры проекта в общем виде и типовые структуры

для проекта «Капитальный ремонт дома», студенты разрабатывают типовые структуры для индивидуального проекта.

Форма отчета:

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

Время выполнения: 2 ч

ЗАДАНИЕ

1. Разработать дерево целей проекта.
2. Разработать структуру продукции проекта.
3. Разработать структуру разбиения работ проекта.
4. Разработать сетевую модель проекта.
5. Разработать организационную структуру проекта.
6. Разработать матрицу ответственности по работам проекта.
7. Разработать структуру затрат проекта.
8. Разработать структуру трудовых ресурсов.
9. Разработать структуру материальных ресурсов.
10. Разработать таблицу основных рисков по проекту и оценить вероятность их наступления. Определить мероприятия по снижению рисков в проекте.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЗАДАНИЯ:

1. Практическое занятие проводится в форме деловой игры.
2. Учебная группа разбивается на бригады по 3-4 человека, образуя команду проекта.
3. Определяется менеджер проекта.
4. Проводится обсуждение темы в команде.
5. На основе коллегиального мнения разрабатываются структуры проекта по выбранной теме.
6. Разработанные структуры менеджер проекта защищает перед всей группой.

Темы проектов:

1. Проект внедрения в компании информационной системы электронного документооборота.
2. Проект создания локальной сети компании.
3. Проект разработки программного продукта.
4. Проект внедрения информационной системы в производственном отделе.
5. Проект создания сайта компании.
6. Проект создания модели бизнес-процессов компании для информационной системы.
7. Проект продвижения на рынок информационных услуг компании.

8. Проект продвижения на рынок нового программного продукта.

9. Проект внедрения информационной библиотечной системы в высшем учебном заведении.

10. Проект организации международной конференции по информационным технологиям.

Практическое занятие № 16«Разработка модульной структуры проекта (диаграммы модулей)»

Целью : является изучение процесса разработки модульной структуры программного обеспечения, осуществляемого с помощью структурных карт Константайна.

Форма отчета:

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Требования к содержанию, оформлению и порядку выполнения

Отчет по выполнению лабораторной работы должен содержать: титульный лист, название работы, цель работы и содержательную часть.

В содержательной части отчета по выполнению лабораторной работы для своего варианта требуется привести структурные карты с подробными комментариями к принятым решениям и построенным диаграммам.

Теоретическая часть

Теоретические сведения для выполнения лабораторной работы приведены в разделе 3.4 учебно-методического пособия.

Общая постановка задачи

Осуществите разработку модульной структуры программного обеспечения задачи, выбранной в первой лабораторной работе, и оформите результат в виде структурной карты, при этом используйте программный продукт EasyCASE Professional Version 4.21.016. Прокомментируйте принятые решения.

Список индивидуальных данных

Продолжается работа над задачей, выбранной в первой лабораторной работе.

Пример выполнения работы

В соответствии с требованиями, предъявляемыми техническим заданием, и результатами внешнего проектирования (см. предыдущие лабораторные работы) разработаем модульную структуру подсистемы обслуживания клиента по его кредитной карте в банкомате.

В составе программного обеспечения можно выделить следующие программные модули: Головной модуль (Main module), Модуль управления устройством считывания кредитной кары (Credit cart control module), Модуль аутентификации (Autentification module) и Модуль получения и обработки

запроса на обслуживание (Reception and processing module). Кроме этого в состав ПО необходимо включить модуль данных кредитной карты (Credit cart data).

Основной функцией Головного модуля является организация общего управления поведением подсистемы и выполняет вызов всех остальных программных модулей.

Модуль управления устройством считывания кредитной карты выполняет функции связанные с обработкой кредитной карты: ввод, считывание хранящейся на ней информации, удаление.

Модуль аутентификации выдает сообщение клиенту на ввод ключевых данных, выполняет получение пароля и проверку его правильности.

Модуль получения и обработки запроса на обслуживание выполняет следующие функции: Получение запроса на обслуживание и проверка возможности его исполнения, Обработка запроса на обслуживание, включающая такие действия как:

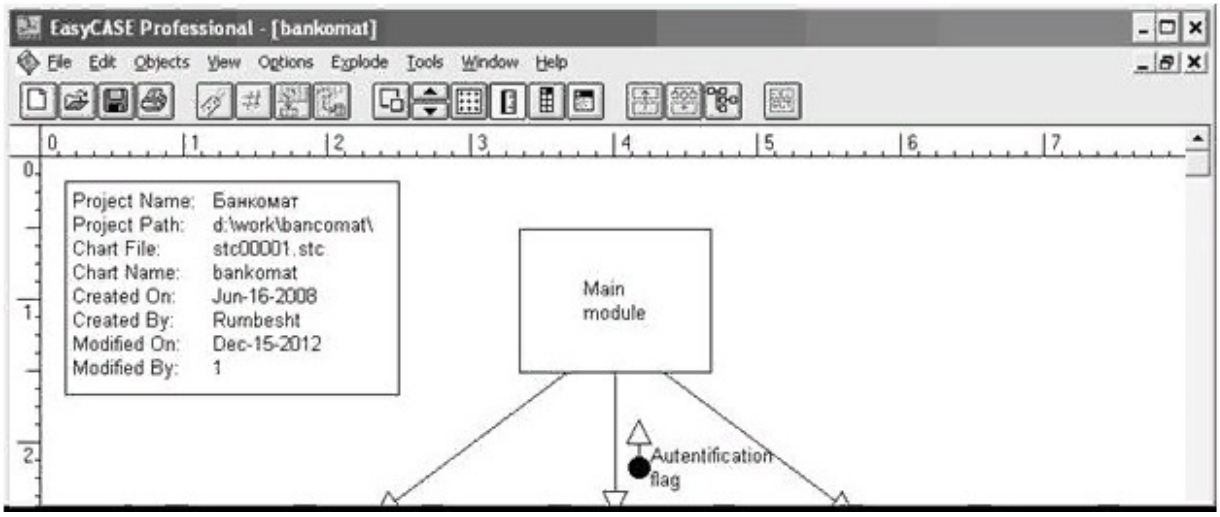
· обработка внутренней банковской документации по клиенту;

· распечатка баланса клиента;

· выдача наличных денег и информирование компьютера банка об изъятых из банка деньгах;

· распечатка операции клиента.

На рис. Л5.1 приведена структурная карта, демонстрирующая отношения между указанными модулями системы.



Согласно этой диаграмме головной модуль обращается к модулям управления устройством считывания кредитной карты, аутентификации и получения и обработки запроса на обслуживание. Вызов указанных модулей осуществляется согласно внутренней логики головного модуля, реализующей следующий сценарий: При инициации действий со стороны клиента головной модуль вызывает модуль управления устройством считывания кредитной карты для ее ввода и считывания с нее информации. После завершения считывания

управление возвращается головному модулю, который затем обращается к модулю аутентификации. Модуль аутентификации проверяет подлинность клиента и вместе с результатом этой проверки возвращает управление головному модулю. В зависимости от результатов аутентификации головной модуль либо вызывает модуль управления устройством считывания для удаления кредитной карты, либо обращается к модулю получения и обработки запроса на обслуживание для предоставления требуемого сервиса. Если осуществляется вызов получения и обработки запроса на обслуживание, то после завершения его работы головной модуль обращается к модулю управления устройством считывания для удаления кредитной карты.

Обмен данными между программными модулями осуществляется через общую область памяти, в которую модуль управления устройством считывания помещает данные о пароле (Parol), атрибуты клиента (Client Attributes) и лимит денег на счету (Limit of money). Модуль аутентификации получает из этой общей области памяти сведения о пароле и возвращает в головной модуль управляющий параметр Autentification flag, содержащий результат аутентификации. Модуль получения и обработки запроса на обслуживание для своей работы получает из общей области памяти атрибуты клиента и лимит денег на счету.

Контрольные вопросы к защите

1. Цель разработки модульной структуры.
2. Понятие программного модуля, передачи управления, организации связи по управлению и по данным.
3. Виды связности модулей.
4. Виды целостности модулей.
5. Типовые модульные структуры.
6. Проектирование модульной структуры с помощью структурных карт.

Практическое занятие №17 «Разработка перечня артефактов и протоколов проекта»

Цель работы: ознакомление с процедурой разработки эскизного проекта на программный продукт, с применением ГОСТ 19.105 -78 «Пояснительная записка к техническому проекту» и ГОСТ 19.404 – 79 «Пояснительная записка. Требования к содержанию и оформлению». Конкретное содержание работ на стадии эскизного проекта и их объем определяет степень сложности разрабатываемого ПП.

Форма отчета:

−выполнить задание;
−показать преподавателю;
−ответить на вопросы преподавателя.

Время выполнения: 2 ч

Результатом выполнения данной стадии является полное описание архитектуры ПП. Как правило, это описание делается на нескольких уровнях иерархии. На верхнем уровне детализации выделяются основные подсистемы, которым присваиваются имена, устанавливаются связи между подсистемами, их функции, получаемые путем декомпозиции предполагаемых функций ПП. Затем процедура декомпозиции выполняется для каждой подсистемы, выделяются модули, составляющие данную подсистему. В конечном итоге, получается иерархически организованная система, состоящая из уровней, каждый из которых представляет собой совокупность взаимосвязанных модулей. В качестве примера ниже приводится фрагмент расширенного описания работ стадии эскизного проекта:

• разработка плана совместных работ на разработку ПП;

• разработка и обоснование математической модели системы и описание результатов моделирования;

• разработка и обоснование алгоритмов и временных графиков функционирования ПП по всем режимам работы;

• разработка и обоснование ресурсов памяти для реализации алгоритмов;

• разработка перечня документов на ПП;

• разработка и обоснование структуры БД, внешних входных и выходных данных;

• разработка и обоснование алгоритмов информационного обеспечения;

• разработка и обоснование набора тестов для проверки ПП;

• разработка и обоснования организации работ по развитию ПП;

Результатом выполнения данной работы является эскизный проект, оформленный в соответствии с ОС ТУСУР.

Перечень использованных информационных ресурсов

1. *Лаврищева, Е. М.* Программная инженерия и технологии программирования сложных систем : учебник для вузов / Е. М. Лаврищева. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 432 с. — (Бакалавр. Академический курс). — ISBN 978-5-534-07604-2. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/436514
2. *Казарин, О. В.* Основы информационной безопасности: надежность и безопасность программного обеспечения : учебное пособие для среднего профессионального образования / О. В. Казарин, И. Б. Шубинский. — Москва : Издательство Юрайт, 2019. — 342 с. — (Профессиональное образование). — ISBN 978-5-534-10671-8. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/431080.
3. *Черткова, Е. А.* Программная инженерия. Визуальное моделирование программных систем : учебник для среднего профессионального образования / Е. А. Черткова. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 147 с. — (Профессиональное образование). — ISBN 978-5-534-09823-5. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/441255.
4. *Черткова, Е. А.* Программная инженерия. Визуальное моделирование программных систем : учебник для среднего профессионального образования / Е. А. Черткова. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 147 с. — (Профессиональное образование). — ISBN 978-5-534-09823-5. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/441255
5. *Куприянов, Д. В.* Информационное обеспечение профессиональной деятельности : учебник и практикум для среднего профессионального образования / Д. В. Куприянов. — Москва : Издательство Юрайт, 2019. — 255 с. — (Профессиональное образование). — ISBN 978-5-534-00973-6. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/434578
6. *Казарин, О. В.* Программно-аппаратные средства защиты информации. Защита программного обеспечения : учебник и практикум для вузов / О. В. Казарин, А. С. Забабурин. — Москва : Издательство Юрайт, 2019. — 312 с. — (Специалист). — ISBN 978-5-9916-9043-0. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/437163

Приложения

Приложение А