Министерство образования Белгородской области
Областное государственное автономное
профессиональное образовательное учреждение
«Белгородский индустриальный колледж»

Рассмотрено
цикловой комиссией
информатики и ПОВТ
Протокол заседания №__1__.
от «31» ___августа_____2022 г.
Председатель цикловой комиссии
_____/Третьяк И. Ю.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ
по **МДК 02.02 ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

по специальности

09.02.07 Информационные системы и программирование

Квалификация - Программист

Разработчик:
Преподаватель
Белгородский индустриальный
колледж
Нечаева В.В.

Белгород 2022 г.

Содержание

1. Пояснительная записка

1.1. Краткая характеристика МДК, ее цели и задачи. Место лабораторных работ в курсе МДК

МДК. 02.02 Инструментальные средства разработки программного обеспечения является частью рабочей основной образовательной программы в соответствии с ФГОС по специальности СПО 09.02.07 Информационные системы и программирование, квалификация – программист.

МДК изучается в VII семестре. В целом рабочей программой предусмотрено 24 часов на выполнение лабораторных работ, что составляет 44 % от обязательной аудиторной нагрузки, которая составляет 54 часов, при этом максимальная нагрузка составляет 56 часов, из них 2 часа приходится на самостоятельную работу обучающихся.

Цель настоящих методических рекомендаций: оказание помощи обучающимся в выполнении лабораторных работ по МДК. 02.02 Инструментальные средства разработки программного обеспечения, качественное выполнение которых поможет обучающимся освоить обязательный минимум содержания курса и подготовиться к промежуточной аттестации в форме дифференцированного зачета.

1.2. Организация и порядок проведения лабораторных работ

Лабораторные работы проводятся после изучения теоретического материала. Введение лабораторных работ в учебный процесс служит связующим звеном между теорией и практикой. Они необходимы для закрепления теоретических знаний, а также для получения практических навыков и умений. При проведении лабораторных работ задания, выполняются студентом самостоятельно, с применением знаний и умений, усвоенных на предыдущих занятиях, а также с использованием необходимых пояснений, полученных от преподавателя. Обучающиеся должны иметь методические рекомендации по выполнению лабораторных работ, конспекты лекций, измерительные и чертежные инструменты, средство для вычислений.

1.3. Общие указания по выполнению лабораторных работ

Курс лабораторных работ по МДК. 02.02 Инструментальные средства разработки программного обеспечения предусматривает проведение 12 работ, посвященных изучению:

- моделей процесса разработки программного обеспечения;
- основных принципов процесса разработки программного обеспечения;
- основных подходов к интегрированию программных модулей;
- основы верификации и аттестации программного обеспечения

При подготовке к проведению лабораторной работы необходимо:

- ознакомиться с лабораторным оборудованием;
- ознакомиться с порядком выполнения работы.

После выполнения лабораторной работы обучающийся к следующему занятию оформляет отчет, который должен содержать:

- название лабораторной работы, ее цель;
- краткие, общие сведения об изучаемом лабораторном оборудовании;
- необходимый графический материал, указанный преподавателем при выполнении лабораторной работы (принципиальная схема лабораторной установки, графики);
- данные, полученные непосредственно из проводимых опытов;
- результаты обработки данных опытов с необходимыми пояснениями;
- графический материал, отображающий полученные в ходе опытов значения измеряемых величин;
- оценку результатов испытаний.

При работе в лаборатории необходимо руководствоваться инструкциями по технике безопасности, учитывающими все специфические особенности лаборатории, такие как наличие высокого напряжения, легкодоступных для прикосновения токоведущих частей электрооборудования.

В лаборатории нельзя находиться в отсутствие преподавателя или лица, ответственного за технику безопасности.

При нахождении в лаборатории следует находиться в рабочей зоне, указанной преподавателем.

Перед выполнением лабораторной работы необходимо получить вводные инструкции преподавателя и внимательно ознакомиться с описанием лабораторного стенда и оборудованием.

Внимание! Включать моноблоки и выполнять какие-либо действия с приборами допускается ТОЛЬКО с разрешения преподавателя!

При обнаружении признаков неисправности, таких как: появление искрения, дыма, специфического запаха, аномальных показаний измерительных приборов, следует немедленно отключить все источники электроэнергии и сообщить о случившемся преподавателю.

При возникновении реальной опасности травматизма для одного или нескольких присутствующих, участники испытания должны произвести срочное отключение лаборатории от всех источников электроэнергии выключением вводного автомата. Лаборатории должны иметь средства пожаротушения и оказания первой медицинской помощи. На первом занятии изучаются правила техники безопасности и проводится вводный инструктаж с последующей проверкой его усвоения, о чем свидетельствует запись в журнале по технике безопасности кабинета/лаборатории, подписываемый преподавателем, проводившем инструктаж, и всеми обучающимися.

1.4. Критерии оценки результатов выполнения лабораторных работ

Критериями оценки результатов работы обучающихся являются:

- уровень усвоения обучающимся учебного материала;
- умение обучающегося использовать теоретические знания при выполнении практических задач;
- сформированность общеучебных и профессиональных 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
ОК 11 Планировать предпринимательскую деятельность в профессиональной сфере
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

- обоснованность и четкость изложения материала;
- уровень оформления работы.
- анализ результатов.

Критерии оценивания лабораторной работы

Оценка	**Критерии оценивания**
5	Работа выполнена в полном объеме с соблюдением необходимой последовательности проведения, содержит результаты и выводы, все записи, таблицы, рисунки, чертежи, графики выполнены аккуратно. Обучающийся владеет теоретическим материалом, формулирует собственные, самостоятельные, обоснованные, представляет полные и развернутые ответы на дополнительные вопросы.
4	Работа выполнена в полном объеме с соблюдением необходимой последовательности проведения, содержит результаты и выводы, все записи, таблицы, рисунки, чертежи, графики выполнены аккуратно. Обучающийся владеет теоретическим материалом, допуская незначительные ошибки на дополнительные вопросы.
3	Работа выполнена в полном объеме, содержит результаты и выводы, все записи, таблицы, рисунки, чертежи, графики выполнены аккуратно. Обучающийся владеет теоретическим материалом на минимально допустимом уровне, допуская ошибки на дополнительные вопросы.
2	Работа выполнена не полностью. Студент практически не владеет теоретическим материалом, допускает ошибки при ответе на дополнительные вопросы.

2. Тематическое планирование лабораторных работ

	Наименование тем	Вид и название работы студента	Количество часов на выполнение работы
Тема 1	**Современные технологии и инструменты интеграции.**		**10**
		Лабораторная работа № 1. Разработка структуры проекта	2
		Лабораторная работа № 2. Разработка модульной структуры проекта (диаграммы модулей)	2
		Лабораторная работа № 3. Разработка перечня артефактов и протоколов проекта	2
		Лабораторная работа № 4. Разработка и интеграция модулей проекта (командная работа)	2
		Лабораторная работа № 5. Отладка отдельных модулей программного проекта	2
Тема 2	**Инструментарий тестирования и анализа качества программных средств**		**14**
		Лабораторная работа № 6. Применение отладочных классов в проекте	2
		Лабораторная работа № 7. Отладка проекта	2
		Лабораторная работа № 8. Инспекция кода модулей проекта	2
		Лабораторная работа № 9 Тестирование интерфейса пользователя средствами инструментальной среды разработки	2
		Лабораторная работа № 10. Разработка тестовых модулей проекта для тестирования отдельных модулей	2
		Лабораторная работа № 11. Выполнение функционального тестирования	2
		Лабораторная работа № 12. Тестирование интеграции	2
		Итого:	**24**

3.Содержание лабораторных работ

Лабораторная работа № 1. Разработка структуры проекта

Цель: Научиться разрабатывать простейшие модули программ.

Задание: Решить задачу вашего варианта (в соответствии с номером по списку) с использование подпрограммы (процедуры или функции), оформить отчет, который должен содержать код вашей программы и блок-схему к ней.

1) Даны координаты вершин многоугольника (x1,y1,x2,y2,x3,y3,…x10,y10). Определить его периметр (вычисление расстояния между вершинами оформить подпрограммой).
2) Составить программу для вычисления суммы факториалов всех нечетных чисел от1 до 9.
3) Составить программу для нахождения наименьшего общего кратного двух натуральных чисел

$$\text{НОК}(A,B) = \frac{A * B}{\text{НОД}(A,B)}$$

4) Составить программу для нахождения наибольшего общего делителя четырех натуральных чисел.
5) Задан массив D. Определить следующие суммы: D[1]+D[2]+D[3]; D[3]+D[4]+D[5]; D[5]+D[6]+D[7].
6) На плоскости заданы своими координатами n точек. Составить программу, определяющую, между какими из пар точек самое большое расстояние. Координаты точек занести в массив.
7) Составить программу для вычисления суммы факториалов всех четных чисел от m до n.
8) Заменить отрицательные элементы линейного массива их модулями, не пользуясь стандартной функцией вычисления модуля. Подсчитать количество произведенных замен.
9) Дан массив A(N) (N-четное). Сформировать массив B(N), элементами которого являются большие из двух рядом стоящих в массиве A чисел. (Например, A=(1,3,5,-2,0,4,0). Элементами массива B будут 3,5,5,0,4,4)
10) Дано натуральное число N. Составить программу для формирования массива, элементами которого являются цифры числа N.
11) Составить программу, определяющую, в каком из данных двух чисел больше цифр.
12) Дан массив *A(N) (N*— четное). Сформировать массив *B(M),* элементами которого являются средние арифметические соседних пар рядом стоящих в массиве *A* чисел. (Например, массив *A* состоит из элементов 1; 3; 5; -2; 0; 4; 0; 3. Элементами массива *B* будут 2; 1,5; 2; 1,5).
13) Даны числа *a, b, c, d* (длины сторон прямоугольника) и число e (диагональ прямоугольника). Вычислить его площадь, разделив данный прямоугольник на 2 треугольника и используя формулу Герона для нахождения их площади.
14) Даны отрезки a, b, c, d. Для каждой тройки этих отрезков, из которых можно построить треугольник, напечатать площадь данного треугольника. Определить функцию Plo(x, y, z), печатающую площадь треугольника со сторонами x, y, z, если такой треугольник существует.
15). Даны действительные числа s, *t.* Получить g(1.2, s)+g(t, s)-g(2s-l, st), где

$$g(a,b) = (a^2+b^2) / (a^2+2ab+3b^2+4)$$

Лабораторная работа № 2. Разработка модульной структуры проекта (диаграммы модулей)

Цель работы: изучение среды программного инструмента моделирования StarUML, поддерживающего UML, и приобретение навыков по созданию диаграмм прецедентов (вариантов) использования.

Теоретическая часть

Визуальное моделирование и UML

Визуальным моделированием (visual modeling) называется способ представления идей и проблем реального мира с помощью моделей.

Модель – это абстракция, описывающая суть сложной проблемы или структуры без акцента на несущественных деталях, тем самым делая ее более понятной.

Разработка программного обеспечения - не исключение. При построении сложной системы строятся ее абстрактные визуальные модели. В настоящее время в области проектирования информационных систем успехом применяется визуальное моделирование с помощью унифицированного языка моделирования UML.

Унифицированный язык моделирования (Unified Modeling Language, UML) является графическим языком для визуализации, специфицирования, конструирования и документирования систем, в которых большая роль принадлежит программному обеспечению.

С помощью UML можно детально описать систему, начиная разработку с концептуальной модели с ее бизнес-функциями и процессами, а также описать особенности реализации системы, такие как классы программного обеспечения системы, схему базы данных. Используя UML, мы также можем разрабатывать сложные системы быстро и качественно.

Как язык графического визуального моделирования UML имеет свою **нотацию** – принятые обозначения. Нотация обеспечивает семантику языка, является способом унификации обозначений визуального моделирования, обеспечивает всестороннее представление системы, которое сравнительно легко и свободно воспринимается человеком.

Моделирование с помощью UML осуществляется поэтапным построением ряда диаграмм, каждая из которых отражает какую-то часть или сторону системы либо ее замысла.

Диаграмма - это графическое представление множества элементов. Обычно диаграмма изображается в виде графа с вершинами (сущностями) и ребрами (отношениями). Диаграммы подчиняются нотации UML и изображаются в соответствии с ней.

Определены следующие виды диаграмм: вариантов использования (use case diagram); классов (class diagram); кооперации (collaboration diagram);последовательности (sequence diagram); состояний (statechart diagram); деятельности (activity diagram); взаимодействия (interactiondiagrams); компонентов (component diagram); развертывания (deployment diagram).

Построения этих диаграмм достаточно для полного моделирования системы.

Особенности рабочего интерфейса программного комплекса StarUML

StarUML ориентирован на UML версии 1.4, поддерживает одиннадцать различных типов диаграмм, принятых в нотации UML 2.0, и представляет собой интегрированный инструментарий для проектирования архитектуры, анализа, моделирования и разработки программных систем.

Среда моделирования StarUML превосходно настраивается в соответствии с требованиями пользователя и имеет высокую степень расширяемости, особенно в области своих функциональных возможностей.

Создание нового проекта в StarUML

Основная структурная единица в StarUML – это проект. Проект сохраняется в одном

файле в формате XML с расширением «.UML». Проект может содержать одну или несколько моделей и различные представления этих моделей (View) – визуальные выражения информации, содержащейся в моделях. Каждое представление модели содержит диаграммы – визуальные образы, отображающие определенные аспекты модели.

Новый проект будет автоматически создан при запуске программы StarUML. При этом вам будет предложено в диалоговом окне выбрать один из подходов (Approaches), поддерживаемых StarUML (см. рис. 1).

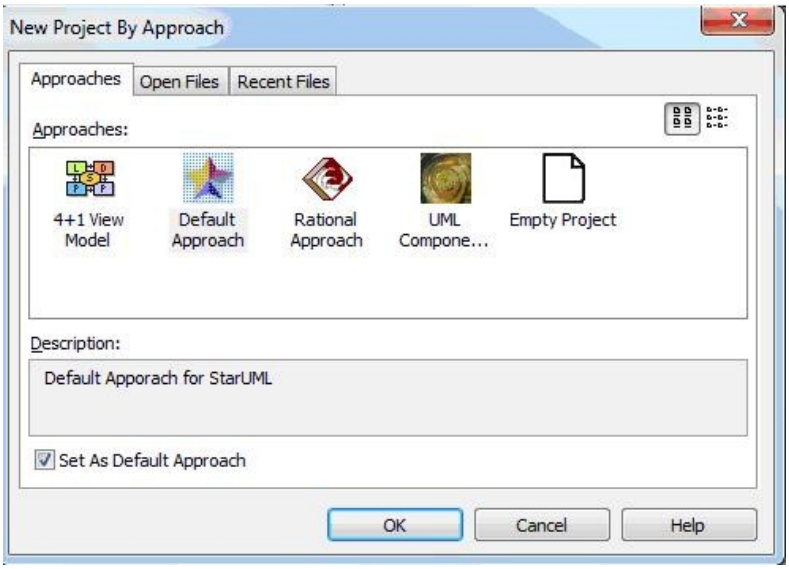
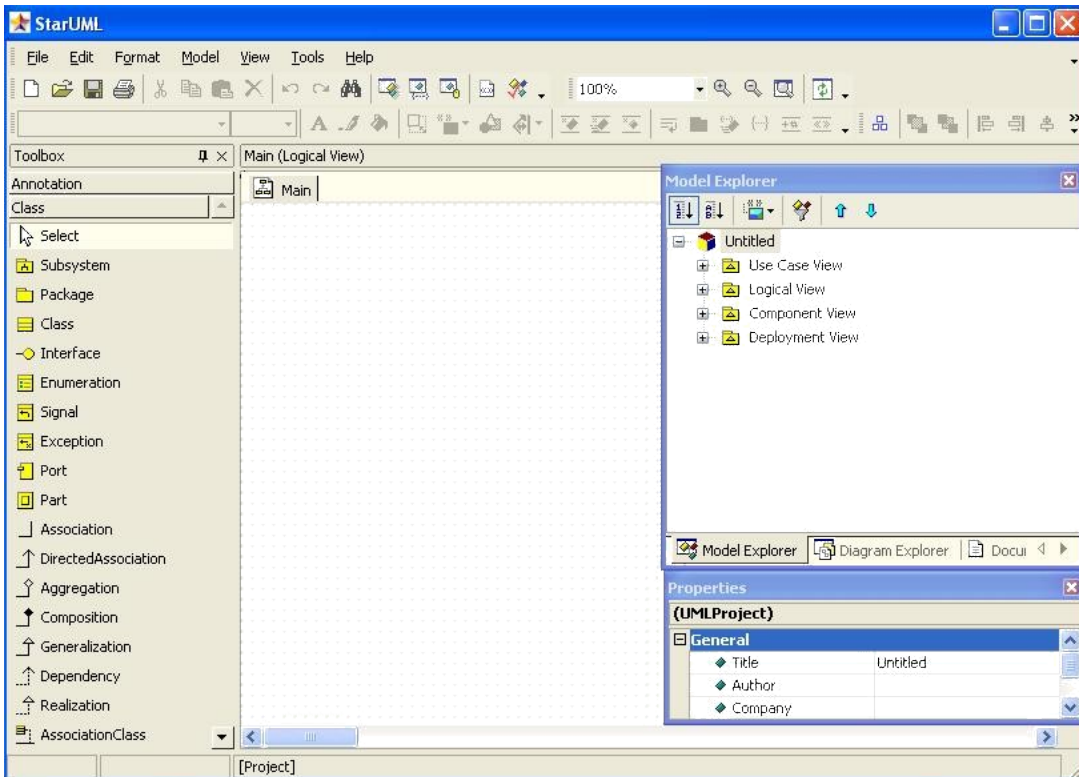


Рисунок 1. Выбор подхода

Существуют различные методологии моделирования информационных систем, компании-разработчики систем также могут разрабатывать свои методологии. Следовательно, на начальной стадии проектирования необходимо определить основные положения методологии или выбрать одну из уже существующих. Для того чтобы согласовать между собой различные элементы и этапы моделирования, StarUML предлагает концепцию подходов.



После выбора одного из предложенных подходов, появится основное окно программы (рис. 2).Главное окно StarUML состоит из следующих компонентов:

главное меню - находится наверху экрана. Большинство функции StarUML ™ доступны через главное меню;

инструментальные панели, которые расположены ниже главного меню. Их кнопки дублируют часто используемые пункты меню;

область браузера. Она расположена в верхнем правом углу экрана. Эта область содержит инструменты, облегчающие просмотр составляющих элементов проекта. Эта область включает [Навигатор модели] (*Model Explorer*), который отображает модельные элементы в виде иерархической структуры, и [Навигатор диаграмм], (*Diagram Explorer*), который отображает диаграммы модели, сгруппированные по типам;

область инспектора расположена в нижнем правом углу экрана. Эта область содержит инструменты, облегчающие редактирование детальной информации о модельных элементах. Эта область включает [Редактор свойств] (*Properties*), который позволяет редактировать свойства, [Документационный редактор] (Documentation), позволяющий водить детальные описания элементов, и [Редактор вложений] (Attachments), который позволяет присоединять к элементам дополнительные файлы и URL;

информационная область расположена в нижней части экрана. Эта область содержит инструменты, отображающие различные вспомогательные данные, касающиеся приложения StarUML ™. Эта область включает [Окно Вывода] (*Output*), которое показывает регистрационную информацию, и [Окно Сообщений] (*Message*), которое отображает результаты поиска и проверки модели;

область диаграммы расположена в центре экрана. Эта область содержит инструментарий, позволяющий редактировать и управлять диаграммами;

палитра элементов расположена на левой стороне экрана. Палитра содержит инструменты для быстрого создания модельных элементов.

В верхней части окна расположено главное меню, кнопки быстрого доступа. Слева расположена панель элементов (Toolbox) с изображениями элементов диаграммы. Элементы соответствуют типу выбранной диаграммы. В центре находится рабочее поле диаграммы, на котором она может быть построена с использованием соответствующих элементов панели инструментов.

Справа находится **инспектор модели**, на котором можно найти вкладки навигатора модели Model Exploer, навигатора диаграмм Diagram Exploer, окно редактора свойств Properties, окно документирования элементов модели Documentation и редактор вложений Attachments. Внешний вид инспектора модели с вкладками представлен ниже (рис. 3).

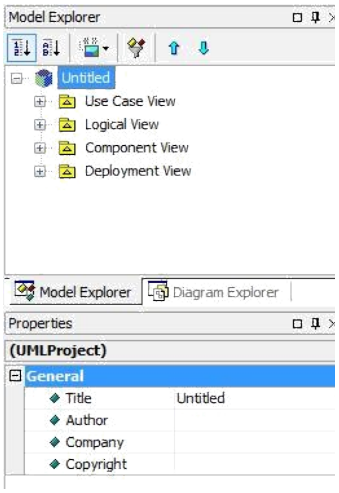


Рисунок 3. Инспектор модели

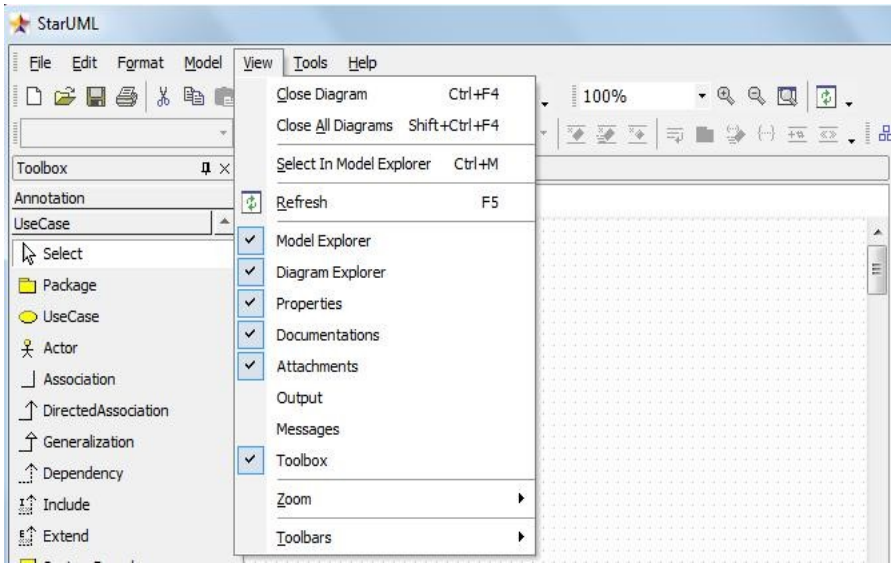


Рисунок 4. Пункт меню View. Управление видом инспектора модели

Управлять видом инспектора модели, панели элементов, закрывать и открывать редакторы инспектора можно с помощью пункта меню View (рис.4). Если рядом с пунктом меню стоит «галочка», этот элемент активен и его можно видеть в окне программы или открыть на доступных вкладках инспектора модели.

Иерархическая структура проекта отображается справа на навигаторе модели (Model Exploer). В зависимости от выбранного подхода на навигаторе модели будут отображены различные пакеты представлений модели. Каждый пакет представления будет содержать элементы моделей и диаграмм, которые мы создадим.

Если при создании нового проекта моделирования выбрать подход Rational Approach, то при таком подходе в навигаторе будут присутствовать четыре пакета представлений модели системы (см. рис. 5):

- Use Case View – представление требований к системе, описывает, что система должна делать;
- Logical View – логическое представление системы, описывает, как система должна быть построена;
- Component View – представление реализации, описывает зависимость между программными компонентами;
- Deployment View – представление развертывания, описывает аппаратные элементы, устройства и программные компоненты.

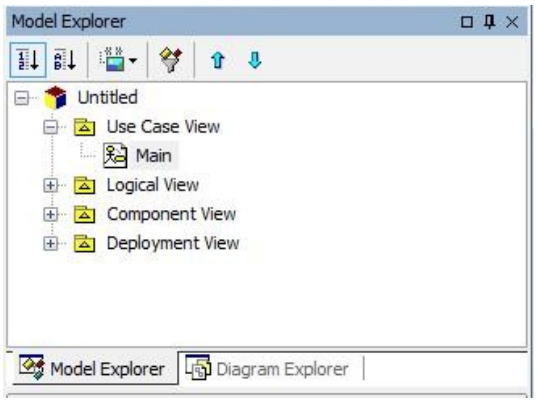


Рисунок 5. Навигатор модели

Сейчас каждое представление содержит одну диаграмму с именем Main. Если щелкнуть по ней два раза, то откроется рабочее поле этой диаграммы и соответствующая панель инструментов.

Пример. Если щелкнуть два раза по диаграмме Main представления Use Case View, то

откроется рабочее этой диаграммы и ее панель элементов (рис. 6).

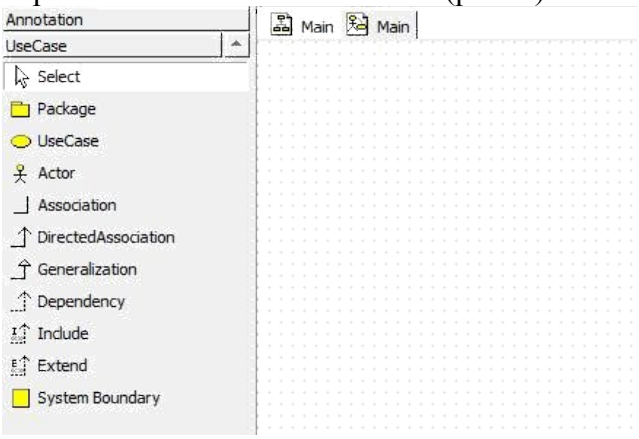


Рисунок 6. Рабочее поле диаграммы прецедентов Main и ее панель элементов

Ниже иерархии представлений отображаются свойства выделенного элемента модели или диаграммы (в данном случае свойства диаграммы Main, так как она выделена в навигаторе модели) (см. рис. 7).

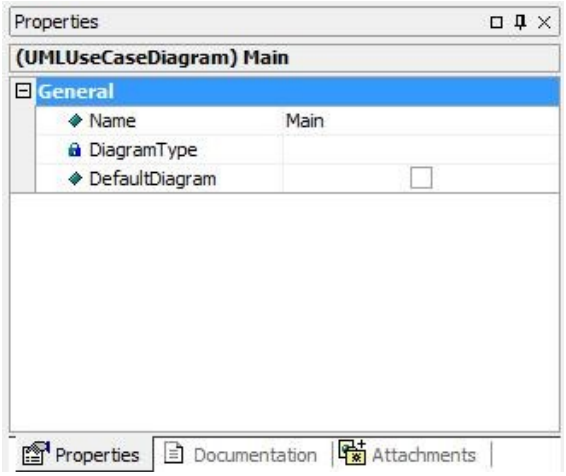


Рисунок 7. Редактор свойств

Назначение кнопок специальной панели инструментов для диаграммы прецендентов использования приведено в табл. 1.1.

Таблица 1.1 - Назначение кнопок специальной панели инструментов для диаграммы прецендентов использования

Графическое изображение	Всплывающая подсказка	Назначение кнопки
	Selection Tool	Превращает изображение курсора в форму стрелки для последующего выделения элементов на диаграмме
	Package	Добавляет на диаграмму пакет
	Use Case	Добавляет на диаграмму прецендент использования
	Actor	Добавляет на диаграмму актера
	Association	Проводит семантическую ассоциацию между двумя

		сущностями
	Unidirectional Association	Добавляет на диаграмму направленную ассоциацию
	Dependency	Соединяет два элемента отношением зависимости
	Include	Соединят два прецедента отношением включения
	Extend	Соединят два прецедента отношением расширения
	Generalization	Добавляет на диаграмму отношение обобщения
	SystemBoundary	Создаёт системную границу

Порядок выполнения работы

Необходимо разработать модель прецедентов (вариантов) использования, состоящую из одной или нескольких диаграмм прецедентов использования, для разрабатываемой информационной системы.

Постановка задачи (описание предметной области). Магазин осуществляет продажу товаров клиенту путем оформления документов «Заказ». Директор магазина, принял решение автоматизировать документооборот продаж товара и пригласил для выполнения работ программиста. Поговорив с директором, в соответствие с концепцией жизненного цикла (ЖЦ) программы программист приступил к описанию бизнес процессов, сопровождающих продажу товара. Взяв за основу язык UML, он начал с построения контекстной диаграммы процессов- Use Case diagram. Диаграмма должна ответить на вопрос-«что должно делаться в системе и кто участник этих процессов». Окончательный вид диаграммы показан на рис.8

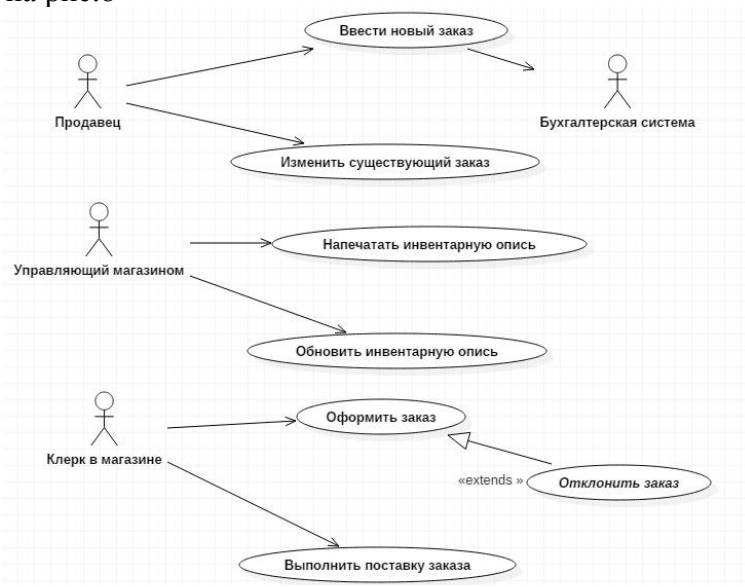


Рисунок 8 – Диаграмма вариантов использования задачи о заказе товара

Для этого выполните следующую последовательность действий:

1. Создайте новый проект и заполните его свойства в области Properties.
2. В навигаторе моделей выберите <<useCaseModel>>, и с помощью контекстного меню выберите действие «Add Diagram» и создайте диаграмму прецедентов использования (Use Case Diagram) для отображения границ системы. Присвойте диаграмме наименование (область Properties) (рис.9).

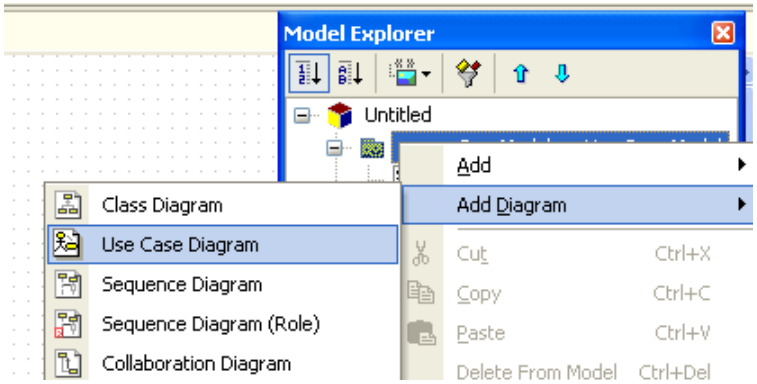


Рисунок 9 – Создание диаграммы прецедентов использования

3. С помощью окна «Toobox» и вкладки «Use Case» поместите на диаграмму новый вариант использования. Назовите его «Ввести новый заказ».
4. Повторив этапы 2 и 3, поместите на диаграмму остальные варианты использования:
   - Изменить существующий заказ;
   - Напечатать инвентарную опись;
   - Обновить инвентарную опись;
   - Оформить заказ;
   - Отклонить заказ;
   - Выполнить поставку заказа.
5. С помощью кнопки Actor (Действующее лицо) панели инструментов поместите на диаграмму новое действующее лицо.
6. Назовите его «Продавец».
7. Повторив шаги 4 и 5, поместите на диаграмму остальных действующих лиц:
   - Управляющий магазином;
   - Клерк магазина;
   - Бухгалтерская система.
8. Создание абстрактного варианта использования (не требующего дальнейшей декомпозиции).

   Щелкните мышью на варианте использования «Отклонить заказ» на диаграмме.

   В окне «Properties» установите флажок isAbstract (Абстрактный), чтобы сделать этот вариант использования абстрактным (рис.10).

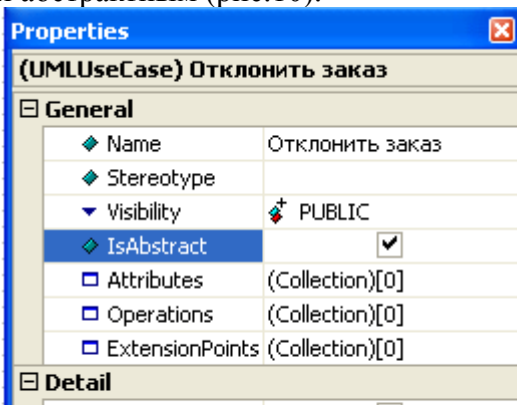


Рисунок 10 – Создание абстрактного варианта использования

9. Добавление ассоциаций. С помощью кнопки Directed Association (Однонаправленная ассоциация) в окне «Toolbox» нарисуйте ассоциацию между действующим лицом Продавец и вариантом использования «Ввести заказ».

   Повторив шаг 1, поместите на диаграмму остальные ассоциации, согласно рис.8.
10. Добавление связи расширения. С помощью кнопки Generalization (Обобщение) в окне «Toolbox» нарисуйте связь между вариантом использования «Отклонить заказ» и вариантом использования «Оформить заказ». Стрелка должна быть

направлена от первого варианта использования ко второму. Связь расширения означает, что вариант использования «Отклонить заказ» при необходимости дополняет функциональные возможности варианта использования «Оформить заказ».

11. Добавление описаний к вариантам использования. Выделите вариант использования «Ввести новый заказ». В окне документации введите следующее описание: «Этот вариант использования дает клиенту возможность ввести новый заказ в систему» (рис.11).

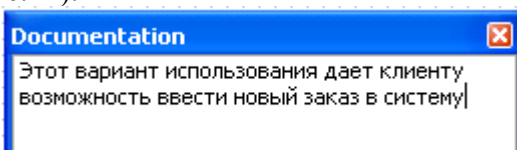


Рисунок 11 – Описание к вариантам использования

С помощью окна документации добавьте описания ко всем остальным вариантам использования.

12. Добавление описаний к действующему лицу. Выделите действующее лицо «Продавец». В окне документации введите следующее описание: «Продавец – это служащий, старающийся продать товар». С помощью окна документации добавьте описания к остальным действующим лицам.
13. Создайте еще одну или несколько диаграмм прецедентов использования, на которых отобразите все необходимые для системы прецеденты использования, актеров и все необходимые связи, например.

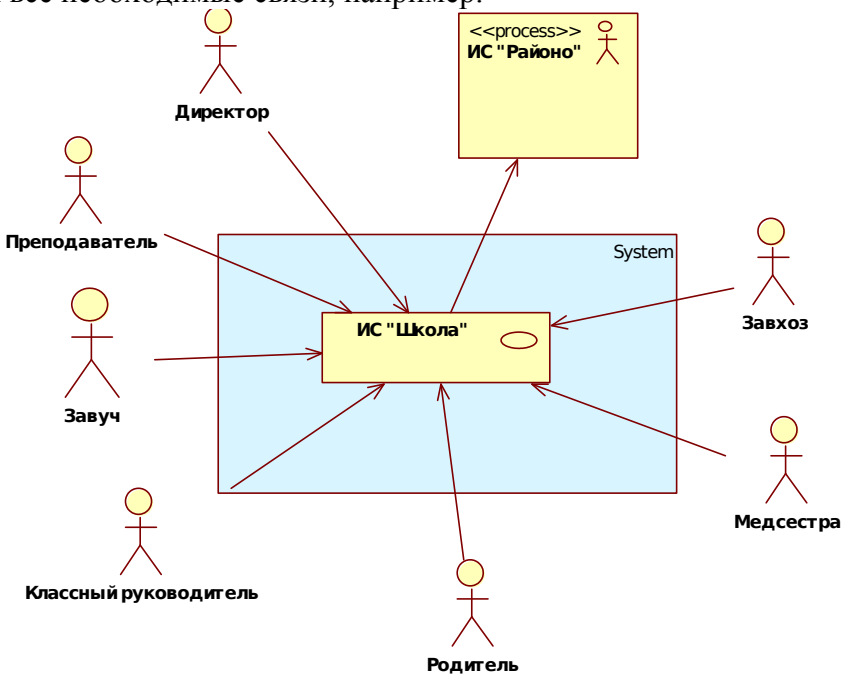


Рисунок 12 – Пример диаграммы описания границ системы

14. Диаграмма обязательно должна содержать включаемый и расширяющий прецеденты и соответствующие связи зависимости прецендентов (использования и расширения) между прецедентами. Для добавления связи использования необходимо нажать кнопку Include панели инструментов и направить связь от базового прецендента к включаемому. Для добавления связи расширения необходимо нажать кнопку Extend панели инструментов и направить связь к базовому прецеденту.

<<include>>

15. В окне свойств связи расширения необходимо указать в текстовом виде условие (свойство Condition секции General области Properties), при котором будет выполняться расширение.
16. В базовом прецеденте, имеющем расширение, необходимо создать точку расширения (свойство ExtenzionPoints секции General области Properties

прецедента). Пример фрагмента диаграммы с точкой расширения приведен на рис. 13

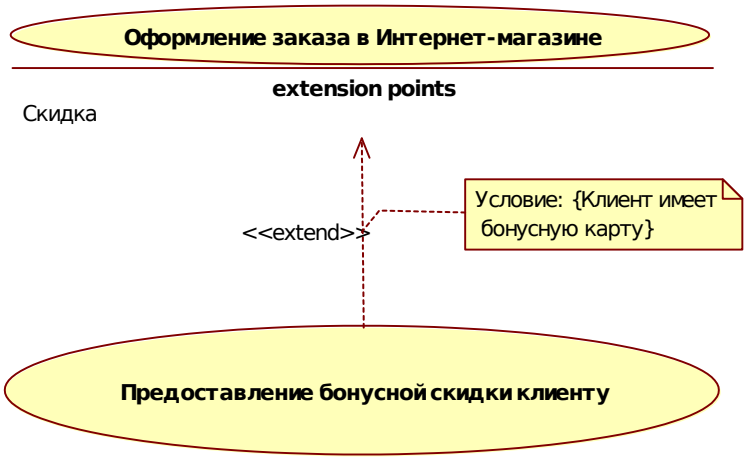


Рисунок 13 - Пример фрагмента диаграммы с точкой расширения

На рис.14 отображен пример полной диаграммы прецедентов использования.

17. Создайте для одного из прецендентов использования файл, содержащий спецификацию потока событий (описание сценария его выполнения) и прикрепите его к прецеденту (область Attachments).

Рисунок 14 – Диаграмма прецендентов использования для банкомата

18. Структура спецификации прецедента использования (use case document) может варьироваться, однако типичное описание должно содержать следующую информацию:

- Краткое описание.
- Участвующие действующие лица.
- Предусловия, необходимые для инициирования прецедента использования.
- Детализированное описание потока событий, включающее в себя:
- основной поток событий, который можно разбить на несколько частей, чтобы показать подчиненные потоки событий (подчиненные потоки могут быть разделены затем на еще более мелкие потоки, с целью улучшить читабельность документа);
- альтернативные потоки для определения исключительных ситуаций.

- ■Постусловия, определяющие состояние систем, по достижении которого прецедент использования завершается.

Пример спецификации потоков событий прецедента приведен в табл. 1.2.

Таблица 1.2 – Пример спецификации потоков событий

Действия актеров	Отклик системы
Основной поток событий	
1. Клиент вставляет кредитную карточку в устройство чтения банкоматаИсключение №1: Кредитная карточка недействительна	2. Банкомат проверяет кредитную карточку 3. Банкомат предлагает ввести ПИН-код
4. Клиент вводит персональный PIN-код Исключение №2: Клиент вводит неверный ПИН-код	5. Банкомат проверяет ПИН-код 6. Банкомат отображает опции меню
7. Клиент выбирает снятие наличных со своего счета	8. Система делает запрос в Банк и выясняет текущее состояние счета клиента 9. Банкомат предлагает ввести требуемую сумму
10. Клиент вводит требуемую сумму 11. Банк проверяет введенную сумму Исключение №3: Требуемая сумма превышает сумму на счете клиента	12. Банкомат изменяет состояние счета клиента, выдает наличные и чек
13. Клиент получает наличные и чек	14. Банкомат предлагает клиенту забрать кредитную карточку
15. Клиент получает свою кредитную карточку	16. Банкомат отображает сообщение о готовности к работе
Исключение №1. Кредитная карточка недействительна или неверно вставлена	
Действия актера	Отклик системы
	1. Банкомат отображает информацию о неверно вставленной кредитной карточке 2. Банкомат возвращает клиенту его кредитную карточку
3. Клиент получает свою кредитную карточку	
Исключение №2. Клиент вводит неверный ПИН-код	
	1. Банкомат отображает информацию о неверном ПИН-коде
2. Клиент вводит новый ПИН-код	
Исключение №3. Требуемая сумма превышает сумму на счете клиента	
	1. Банкомат отображает информацию о превышении кредита
2. Клиент вводит новую требуемую сумму	

19. После окончания сеанса работы над проектом выполненную работу необходимо сохранить в файле проекта с расширением .UMl.
20. Покажите разработанные диаграммы преподавателю.
21. Оформите отчет.
22. Сдайте выполненную и оформленную надлежащим образом работу преподавателю.

Содержание отчета

Индивидуальный отчет о выполненной лабораторной работе должен содержать:

1) цель работы;
2) наименование функционального комплекса задач или системы, для которых разработана диаграмма прецендентов использования;
3) разработанные диаграммы прецендентов использования;
4) спецификацию потоков событий для одного из прецендентов использования;
5) выводы о полученных знаниях и умениях.

Контрольные вопросы

1. Какие элементы может содержать диаграмма прецендентов использования?
2. Чем или кем могут быть представлены актеры (действующие лица) в системе?
3. В каких случаях используют отношения включения на диаграммах Прецендентов Использования?
4. В каких случаях используют отношения расширения на диаграммах прецендентов использования?
5. Как определяется направленность связи ассоциации между актером и прецендентом использования?
6. Что такое точка расширения прецедента?
7. С помощью чего описывается динамика прецендента использования.
8. На каком этапе разработки программной системы разрабатывается диаграмма прецедентов использования? Каково ее назначение?
9. Сколько диаграмм прецедентов использования необходимо разрабатывать для одной программной системы?

Лабораторная работа № 3. Разработка перечня артефактов и протоколов проекта

Цель работы:

- Составление диаграмм: взаимодействия ролей, кооперации и кооперации ролей с помощью программы StarUML
- Изучение методов и приемов объектно-ориентированнного проектирования , моделирования и программирования с помощью программы StarUML

Ход работы:

- Изучить основы языка UML.
- Изучить методы разработки диаграмм на языке UML.
- Для выбранной подсистемы разработать модель на языке UML.
- Составить диаграммы на языке UML с помощью программы StarUML.
- Разработать модель информационной системы АСУ ВУЗ.
- Изучить теоретические сведения о программе StarUML.
- Для выбранной подсистемы создать проект с помощью программы StarUML.
- Для выбранной подсистемы разработать диаграммы с помощью программы StarUML.
- Сохранить диаграммы в проекте.
- Оформить отчет о лабораторной работе

Теоретические сведения

Диаграмма взаимодействия ролей

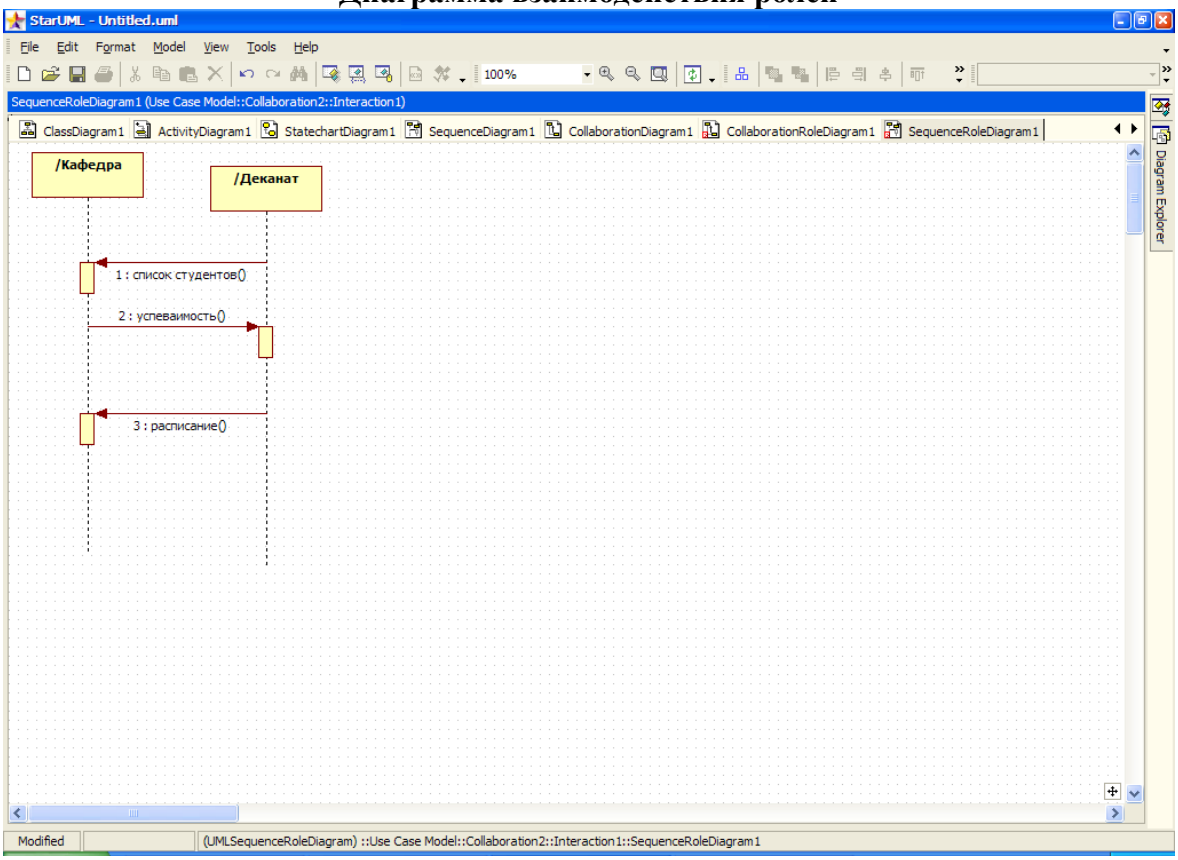


Рис.2.1. Диаграмма сообщений роли

Диаграмма сообщений роли отображает взаимодействия в концепции ролей. В нашем случае диаграмма показывает взаимодействие кафедры с деканатом.

Диаграмма кооперацій

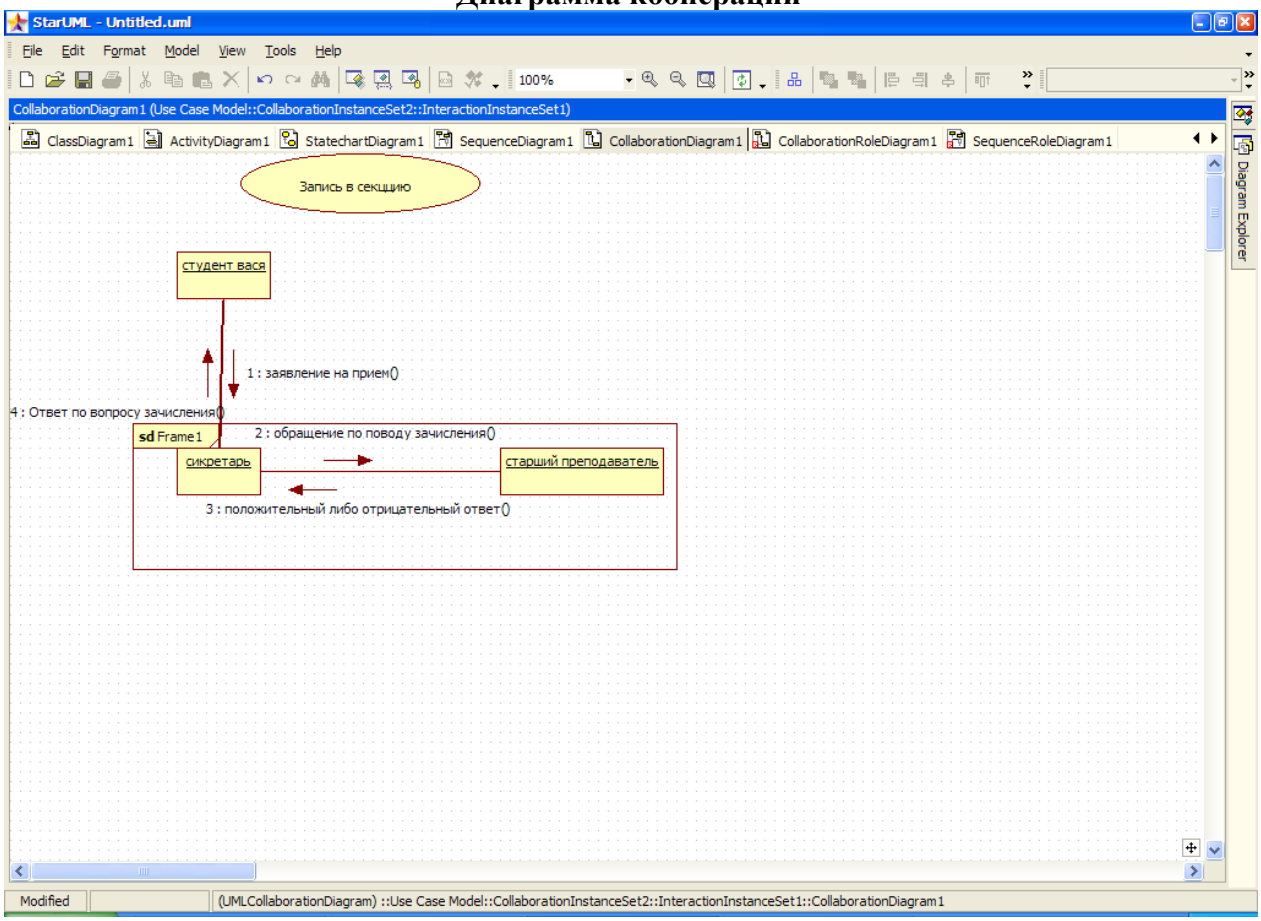


Рис.2.2. Диаграмма кооперацій

Диаграмма кооперацій отображает взаимодействие между объектами. Она является прямым отображением модели взаимодействия объектов.

В нашем случае диаграмма показывает взаимодействие кафедры во время обработки заявления на запись в секцию.

В ходе работы программы объекты обмениваются сообщениями.

Сообщение –это передача информации от одного объекта к другому.

Рассмотрим 2 аспекта: 1) временной – в какой очередности сообщения передаются между объектами; 2) структурный – как сообщения могут быть переданы между объектами. По сути передача сообщения означает, что объект одного класса вызывает метод объекта другого класса. Обычно на диаграмме указывают объекты, а не классы. Но если все объекты ведут себя идентично, то можно написать имя класса (:имя класса). Каждый объект обладает линией жизни. Если она заканчивается крестиком, то в этот момент времени объект уничтожается. Если на ней нарисован прямоугольник, то это значит, что объект в это время действует.

Диаграмма кооперацій – вариант диаграммы последовательности откуда исключено время.

Диаграмма кооперацій ролей

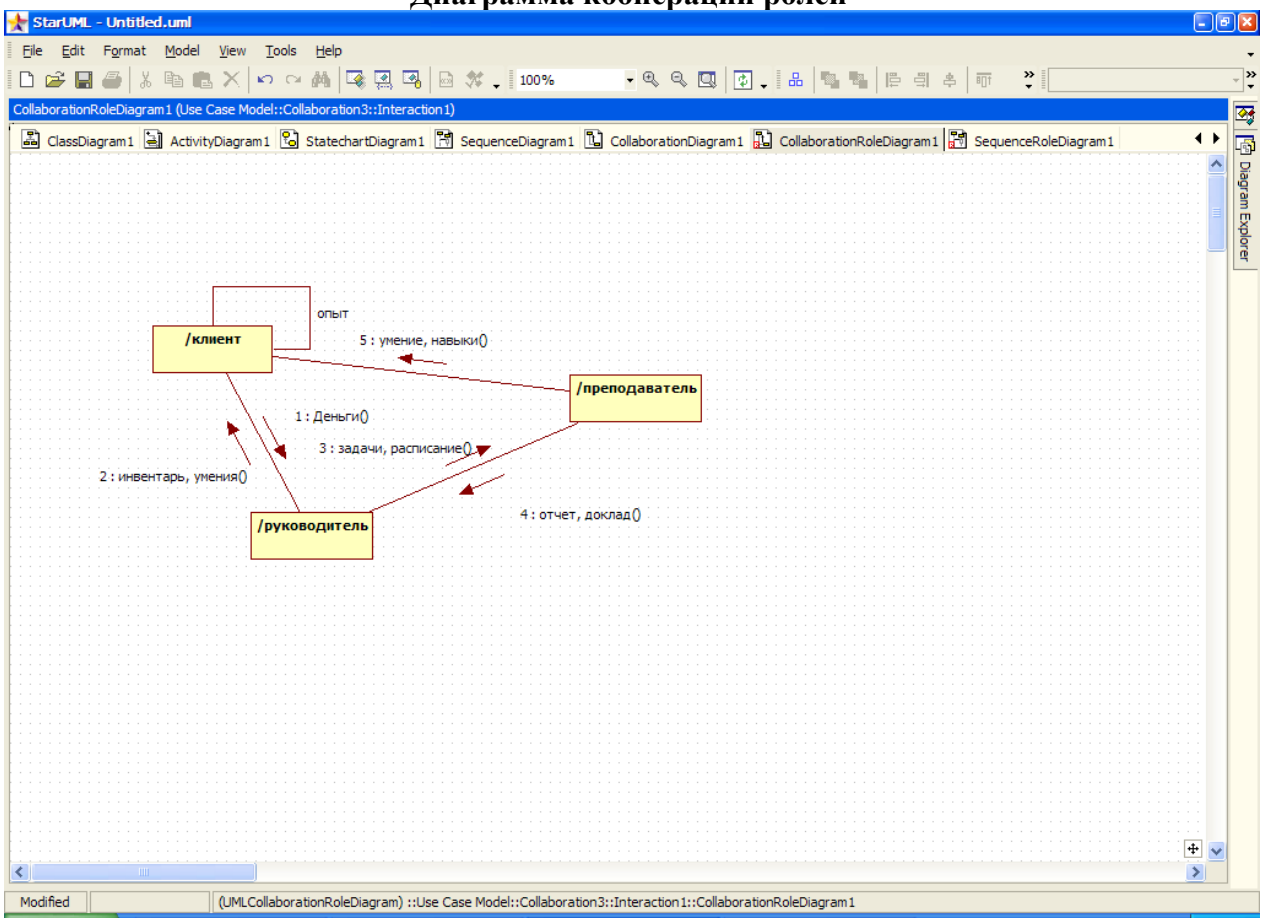


Рис.2.3.Диаграмма кооперацій ролей

Диаграмма кооперацій ролей отображает взаимодействия между ролями. Она является прямым отображением модели взаимодействия классификаторов-ролей внутри кооперации. В нашем случае диаграмма описывает взаимодействие и роли каждого участника спортивной секции, образованной кафедрой физ. воспитания.

Описание подсистем АСУ ВУЗ

Вариант задания	Название	Описание
1	Подсистема «Ректорат»	Модель работы ректора ВУЗа, проректора по учебной работе, проректора по научной работе, проректора по АХЧ, секретаря ректора, секретаря проректора
2	Подсистема «Ученый Совет ВУЗа»	Модель работы Ученого Совета ВУЗа, председателя (ректора), ученого секретаря, членов Ученого Совета ВУЗа
4	Подсистема «Деканат»	Модель работы декана, зам. декана, секретаря
5	Подсистема «Кафедра КСУ»	Модель работы зав. кафедрой, секретаря, преподавателей (профессоров, доцентов, старших преподавателей, преподавателей, ассистентов),Зав. учебной лабораторией, инженера, лаборанта, техника
6	Подсистема «Кафедра физвоспитания»	Модель работы зав. кафедрой, секретаря, преподавателей (профессоров, доцентов, старших преподавателей, преподавателей, ассистентов), инженера, лаборанта, техника

7	Подсистема «Дворец культуры»	Модель работы зав. Дворцом культуры, секретаря, руководителя кружка
8	Подсистема «Профком студентов»	Модель работы председателя профкома, бухгалтера, секретаря, членов профкома, профоргов студенческих групп, студентов
9	Подсистема «Учебная библиотека»	Модель работы зав. библиотекой, библиографов, сотрудников библиотеки, читателей (студентов и преподавателей)
10	Подсистема «Отдел кадров»	Модель работы начальника отдела кадров, секретаря, сотрудников отдела кадров, сотрудников, преподавателей и студентов ВУЗа
11	Подсистема «Канцелярия»	Модель работы зав. канцелярией, секретаря, сотрудников канцелярии, сотрудников, преподавателей и студентов ВУЗа
12	Подсистема «Отдел охраны труда и ТБ»	Модель работы начальника, секретаря, инженера, лаборанта, техника
13	Подсистема «Бухгалтерия»	Модель работы главного бухгалтера, бухгалтера, секретаря
14	Подсистема «Приемная комиссия»	Модель работы председателя приемной комиссии, секретаря, сотрудника приемной комиссии

Контрольные вопросы

1.Что означает название UML?

2.Что означает название MDA?

3. Перечислите разные типы диаграмм в языке UML.

4.В каком формате должны готовиться файлы для программы StarUML?

5.В каких областях науки и техники могут использоваться модели, созданные с помощью программы StarUML?

6.Перечислите основные концепции StarUML.

7.Что такое модель в программе StarUML?

8. Что такое представление?

9. Что такое диаграмма?

10. Что такое проект?

11. Какие элементы входят в структуру проекта?

12. Какое расширение имеют файлы проекта в StarUML?

14. Какие профили используются в StarUML?

15. Какие базовые модули использует StarUML?

16. Перечислите основные типы диаграмм в программе StarUML

Лабораторная работа № 4. Разработка и интеграция модулей проекта (командная работа)

Цель работы:

- Составление диаграмм: компонентов, развертывания и композиционная структурная диаграмма с помощью программы StarUML.
- Изучение методов и приемов объектно-ориентированнного проектирования, моделирования и программирования с помощью программы StarUML

Теоретические сведения

Диаграмма компонентов

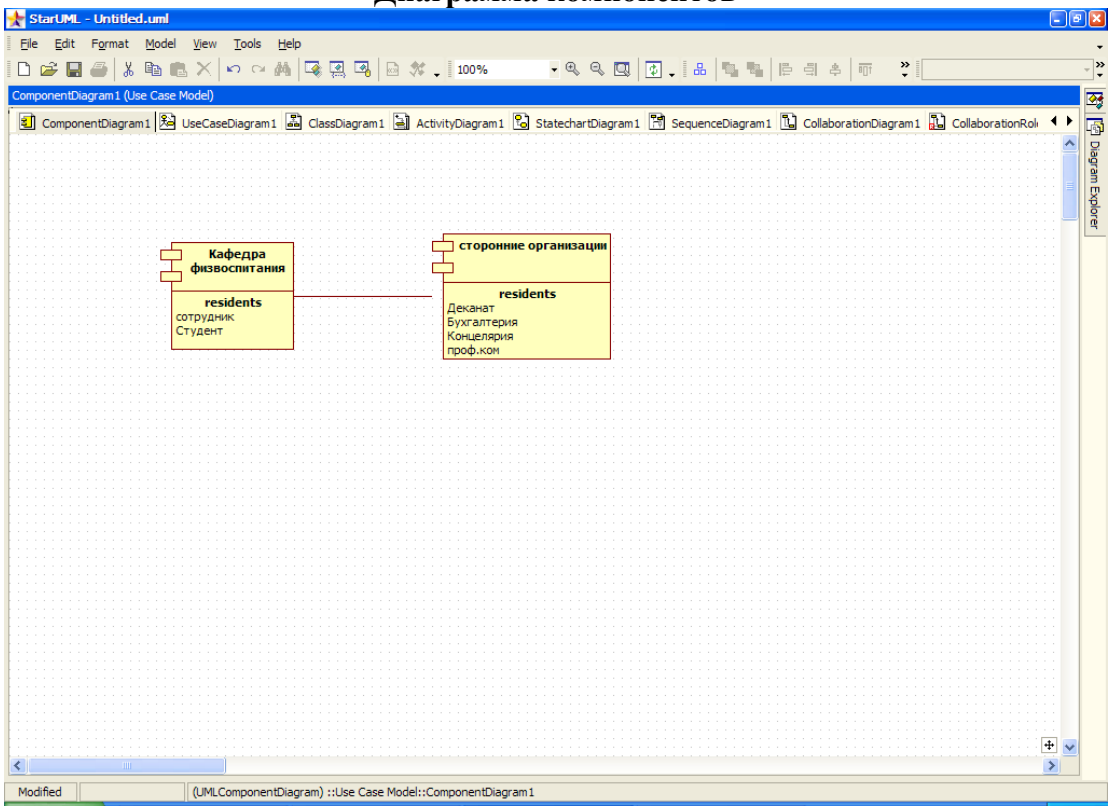


Рис.3.1. Диаграмма компонентов

Диаграмма компонентов отображает зависимость между программными компонентами. Элементы, которые составляют программные компоненты и элементы, которые реализуют эти компоненты, могут быть отображены на диаграмме компонентов.

В нашем случае диаграмма компонентов отображает взаимодействие кафедры физ. воспитания со сторонними организациями внутри университета.

Диаграмма компонентов описывает особенности физического представления системы, позволяет определить архитектуру разрабатываемой системы.

Компонента – единица физической реализации системы. Все классы нужно прикрепить к компонентам и все компоненты к узлам обработки. Компоненты взаимодействуют с собой через интерфейс (обозначается кружочком, присоединяется сплошной линией).

Диаграмма развертывания

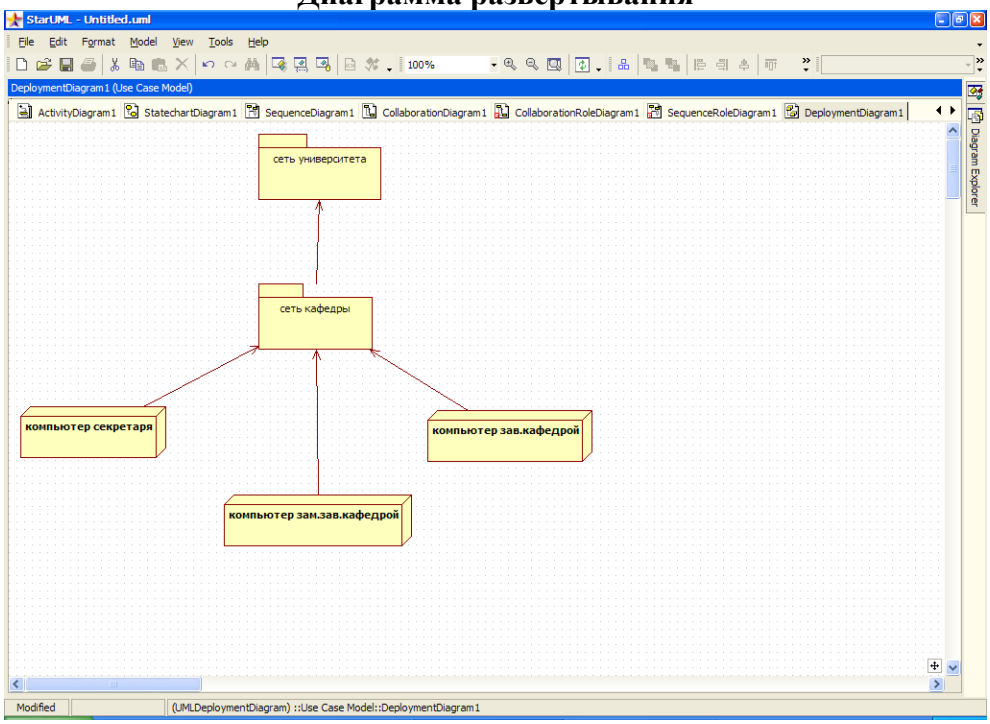


Рис.3.2. Диаграмма развертывания

Диаграмма развертывания отображает аппаратные элементы компьютера, другие устройства и программные компоненты, а также процессы и объекты, которые им назначены.

В нашем случае данная диаграмма показывает связь имеющихся на кафедре компьютеров и их связь с общей сетью университета.

Диаграмма развертывания применяется для представления общей конфигурации системы и содержит распределение компонентов системы по отдельным узлам системы. Кроме того показывает наличие физических соединений.

Узел – некоторый физически существующий элемент системы, обладающий некоторым вычислительным ресурсом. Узел изображается в виде куба.

Композиционная структурная диаграмма

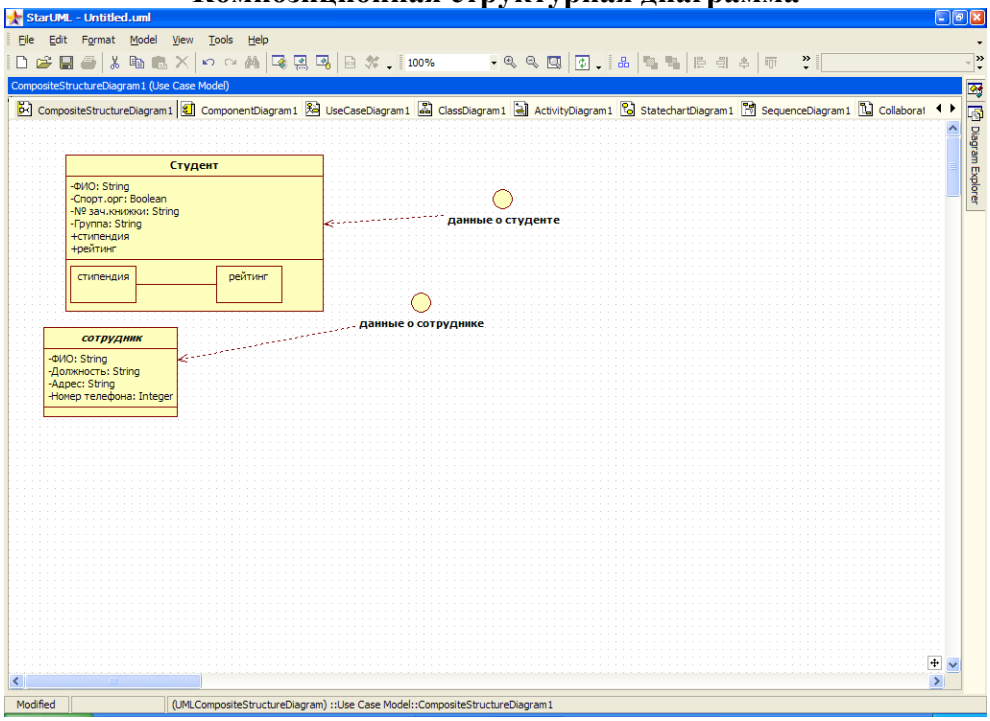


Рис.3.3. Композиционная структурная диаграмма

Композиционная структурная диаграмма - диаграмма, выражающая внутреннюю структуру классификатора. Она показывает его точки зрения взаимодействия с другими частями системы. В нашем случае диаграмма показывает структуру элементов и связь между ними.

Содержание отчета о лабораторной работе:

- Титульный лист (с указанием названия кафедры, названия дисциплины, ФИО и номера группы студента, ФИО преподавателя)
- Содержание с номерами стр.
- Название лабораторной работы.
- Вариант индивидуального задания.
- Скрин-шоты диаграмм, подготовленных с помощью программы StarUML
- Описание диаграмм
- Выводы

Описание подсистем АСУ ВУЗ

Вариант задания	**Название**	**Описание**
1	Подсистема «Ректорат»	Модель работы ректора ВУЗа, проректора по учебной работе, проректора по научной работе, проректора по АХЧ, секретаря ректора, секретаря проректора
2	Подсистема «Ученый Совет ВУЗа»	Модель работы Ученого Совета ВУЗа, председателя (ректора), ученого секретаря, членов Ученого Совета ВУЗа
4	Подсистема «Деканат»	Модель работы декана, зам. декана, секретаря
5	Подсистема «Кафедра КСУ»	Модель работы зав. кафедрой, секретаря, преподавателей (профессоров, доцентов, старших преподавателей, преподавателей, ассистентов),Зав. учебной лабораторией, инженера, лаборанта, техника
6	Подсистема «Кафедра физвоспитания»	Модель работы зав. кафедрой, секретаря, преподавателей (профессоров, доцентов, старших преподавателей, преподавателей, ассистентов), инженера, лаборанта, техника
7	Подсистема «Дворец культуры»	Модель работы зав. Дворцом культуры, секретаря, руководителя кружка
8	Подсистема «Профком студентов»	Модель работы председателя профкома, бухгалтера, секретаря, членов профкома, профоргов студенческих групп, студентов
9	Подсистема «Учебная библиотека»	Модель работы зав. библиотекой, библиографов, сотрудников библиотеки, читателей (студентов и преподавателей)
10	Подсистема «Отдел кадров»	Модель работы начальника отдела кадров, секретаря, сотрудников отдела кадров, сотрудников, преподавателей и студентов ВУЗа
11	Подсистема «Канцелярия»	Модель работы зав. канцелярией, секретаря, сотрудников канцелярии, сотрудников, преподавателей и студентов ВУЗа
12	Подсистема «Отдел охраны труда и ТБ»	Модель работы начальника, секретаря, инженера, лаборанта, техника

13	Подсистема «Бухгалтерия»	Модель работы главного бухгалтера, бухгалтера, секретаря
14	Подсистема «Приемная комиссия»	Модель работы председателя приемной комиссии, секретаря, сотрудника приемной комиссии

Контрольные вопросы

1. Перечислите разные типы диаграмм в языке UML.
2.В каком формате должны готовиться файлы для программы StarUML?
3.В каких областях науки и техники могут использоваться модели, созданные с помощью программы StarUML?
4. Что такое проект?
5. Какие профили используются в StarUML?

Лабораторная работа № 5. Отладка отдельных модулей программного проекта

Цель работы:

- Составление диаграмм прецендентов, классов, деятельности, взаимодействий и состояний с помощью программы STARUML
- Изучение методов и приемов объектно-ориентированного проектирования , моделирования и программирования с помощью программы StarUML

Порядок выполнения работы:

- Изучить основы языка UML.
- Изучить методы разработки диаграмм на языке UML.
- Для выбранной подсистемы разработать модель на языке UML.
- Составить диаграммы на языке UML с помощью программы StarUML.
- Разработать модель информационной системы АСУ ВУЗ.
- Изучить теоретические сведения о программе StarUML.
- Для выбранной подсистемы создать проект с помощью программы StarUML.
- Для выбранной подсистемы разработать диаграммы с помощью программы StarUML.
- Сохранить диаграммы в проекте.
- Оформить отчет о лабораторной работе

Содержание отчета о лабораторной работе:

- Название лабораторной работы.
- Вариант индивидуального задания.
- Скрин-шоты диаграмм, подготовленных с помощью программы StarUML
- Описание диаграмм
- Выводы

Теоретические сведения
Язык и методика объектно-ориентированного моделирования UML

UML- универсальный язык моделирования (universalmodelinglanguage). UML используется для создания моделей сложной системы.

Цель разработки UML – предоставить в распоряжение пользователей легко воспринимаемый и выразительный язык визуального моделирования, предназначенный для разработки и документирования моделей сложных систем. В основе UML лежат диаграммы.

Диаграммы UML

Определены следующие виды диаграмм:

1. Диаграмма вариантов использования, диаграмма прецедентов (usecasediagram),
2. Диаграмма классов (classdiagram)
3. Диаграммы поведения (behaviordiagrams)

а) диаграммы состояний (statechartdiagram)

б) диаграмма деятельности (activitydiagram)

в) диаграммы взаимодействия (interactiondiagrams)

диаграмма последовательности (sequencediagram)

диаграмма кооперации (collaborationdiagram)

4. Диаграммы реализации (implementation diagram)

а) диаграмма компонентов (componentdiagram)

б) диаграмма развертывания (deploymentdiagram).

Диаграмма прецедентов

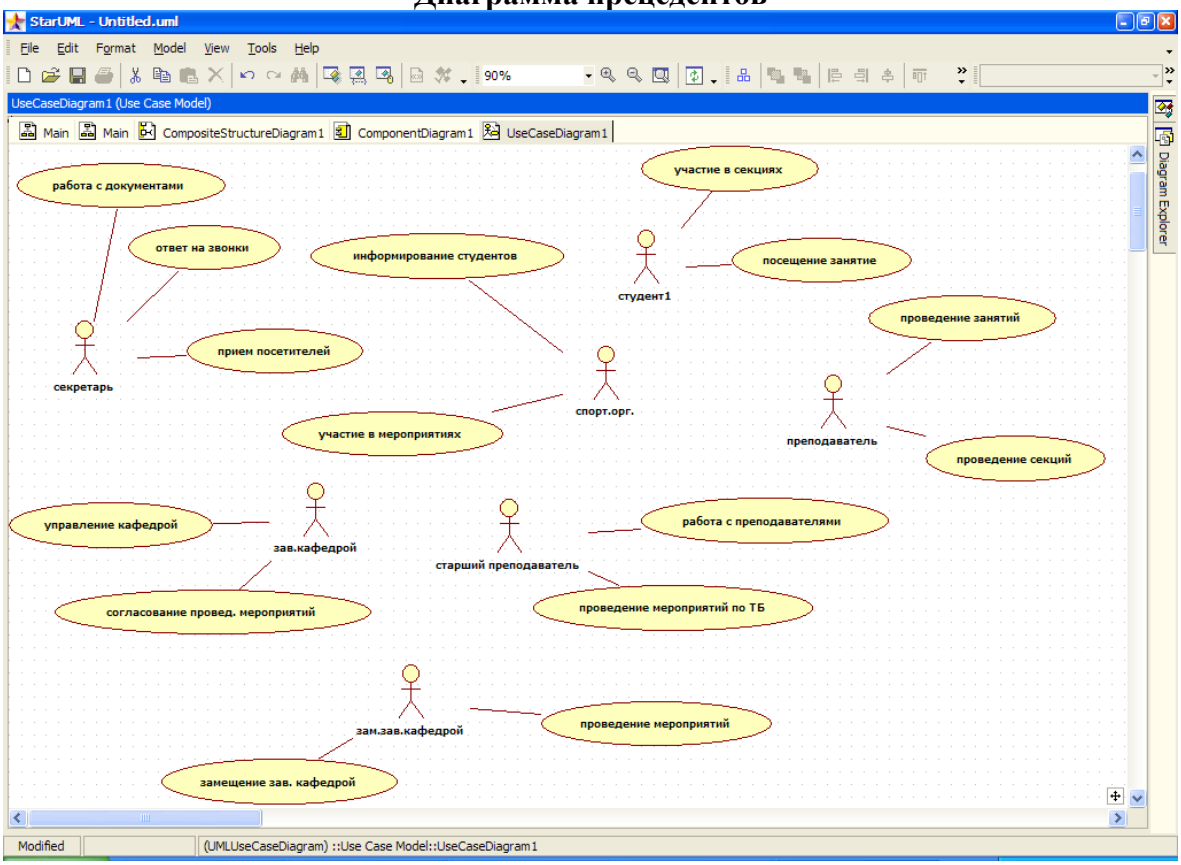


Рис.1.1.Диаграмма прецедентов

Диаграмма прецедентов включает актеров (действующих лиц) и действия (прецедент). Каждому актеру ставят в соответствие одно или несколько действий. Пара «актер-действие» образует роль.

Описание диаграммы прецедентов

Табл.1.1.

Актер	Краткое описание
Зав. кафедрой	Сотрудник, который занимается организацией работы кафедры. В том числе ее управлением и согласованием мероприятий, проводимых на кафедре
Зам. зав. кафедрой	Сотрудник, который замещает заведующего, проводит мероприятия.
Секретарь	Сотрудник, который отвечает на звонки по телефону, принимает посетителей, работает с документами, выполняет поручения Зав. кафедрой
Старший преподаватель	Сотрудник, который отвечает за технику безопасности и взаимодействие с преподавателями, проводит занятия со студентами, проводит занятия специализированных секций.
Преподаватель	Сотрудник, который проводит занятия со студентами, проводит занятия специализированных секций.
Спорт.орг.	Сотрудник, который отвечает за взаимодействие со студентами, помощь в проведение мероприятий и своевременное оповещение об их проведении.
Студент	Посещает занятия по общей физической подготовке, посещает спортивные секции, принимает участие в мероприятиях.

Цели:

- Определение общих принципов и контекста моделируемой предметной области на начальных этапах проектирования.
- Сформулировать общие требование к функциональному поведению проектируемой области.
- Разработать исходную концептуальную модель для ее следующей детализации.
- Подготовка исходной документации для взаимодействия с заказчиком.

Используются обозначения:

Действующее лицо (actor, актер) – тот кто со своим запросом обращается к проектируемому программному комплексу.

Действующее лицо имеет 3 разновидности: человек, какое-то техническое устройство, для управления которым создается ПК, другой ПК, внешний к данному.

Варианты использования описывают отношения:

Отношение ассоциации: служит для обозначения специфической роли актера в отдельном варианте использования, т.е. какую конкретную **роль** играет актер при взаимодействии с экземпляром варианта использования.

Отношение обобщения: служит для указания факта, что некоторый вариант использования может быть обобщен до другого варианта использования.

Отношение расширения: определяет связь одного варианта использования с более общим.

Отношение включения: показывает, что некоторый вариант использования включает в себя подварианты

Диаграмма классов

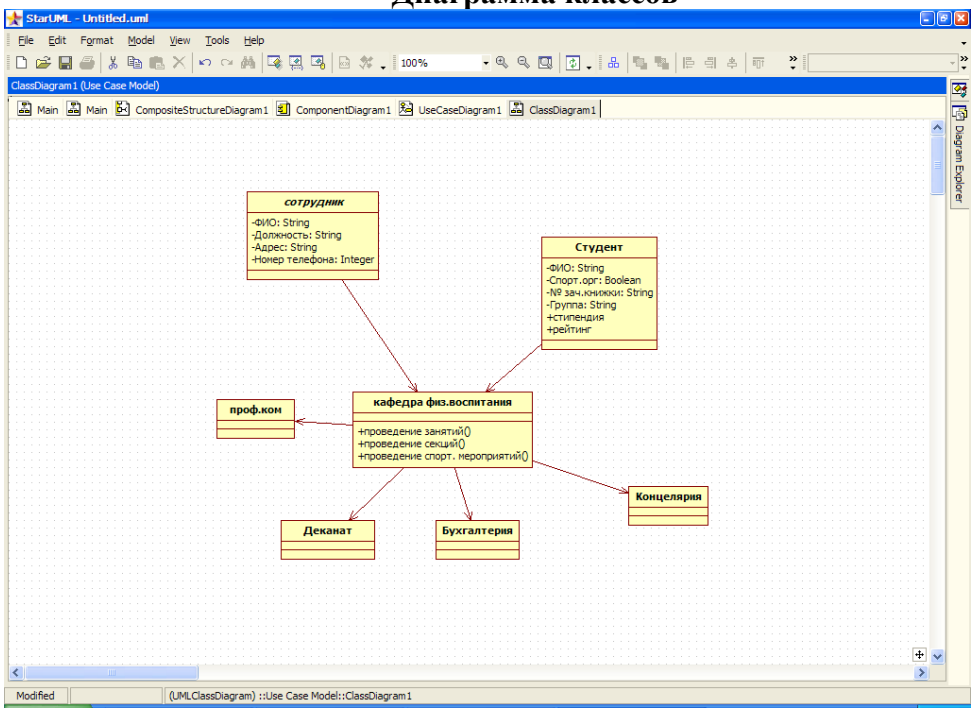


Рис. 1.2. Диаграмма классов

Класс - это абстрактное описание множества объектов с одинаковыми свойствами. Класс включает атрибуты (свойства, характеристики объектов) и методы (действия над объектами этого класса). Модификатор доступа для атрибутов и методов – public, privat.

Таблица описания класса «кафедра физ. воспитания»

Табл.1.2

Параметр	**Значение**
комментарий	Класс, который представляет собой описание работы кафедрой.
операции	Проведение занятий() – занесение

	информации о проводимом занятии. Проведение секций() - занесение информации о проводимой секции. Проведение спорт. мероприятий() – занесение информации о проводимом мероприятии. Все операции имеют модификатор доступа –public.

Класс «Сотрудник»

Параметр	**Значение**
Комментарий	Класс, представляющий собой сотрудника кафедры
Атрибуты	-ФИО: string - фамилия, имя отчество сотрудника кафедры. -Должность: string - должность сотрудника. -Адрес: string - домашний адрес сотрудника -номер телефона: string - телефон сотрудника для связи с ним. Все атрибуты имеют модификатор доступа -privat.

Класс «Студент»

Параметр	**Значение**
Комментарий	Класс, представляющий собой студента
Атрибуты	-ФИО: string- ФИО студента -спорт. орг: boolean- идентификатор спорт.орга -№ зачетной книжки: string- номер зачетной книги студента -група: string- Наименование группы студента -степендия – назначенная степендия -рейтинг – рейтинг студента Все атрибуты имеют модификатор доступа -privat.

Диаграмма деятельности

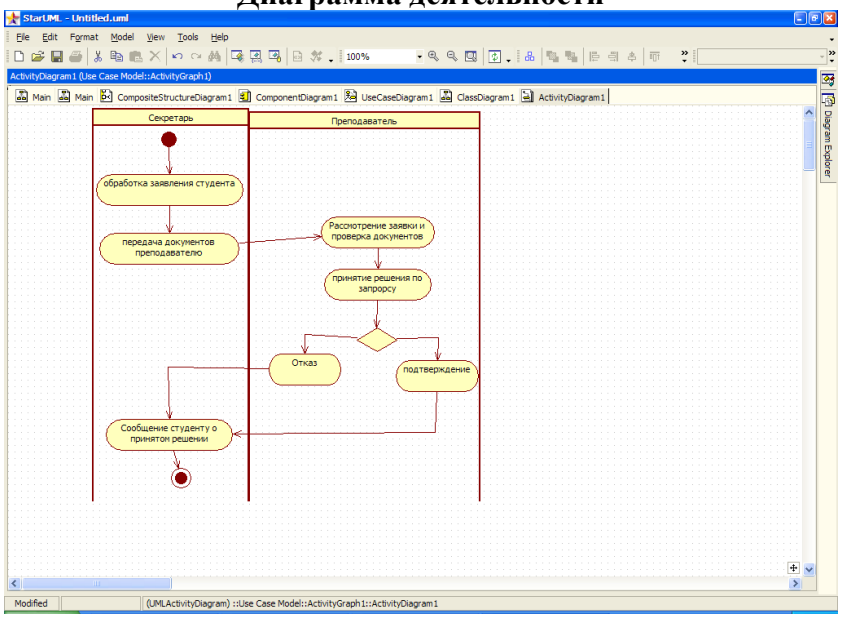


Рис.1.3. Диаграмма деятельности

Диаграмма действий - специальная форма диаграммы состояний, которая отображает последовательность выполнения действий во времени. Диаграмма действий в общем случае используется для отображения любых последовательных действий для обработки данных, но чаще всего применительно к объектам, классам, пакетам и операциям.

В нашем случае диаграмма действий показывает процесс обработки поданного студентом заявления на зачисление в спортивную секцию.

Диаграмма взаимодействий

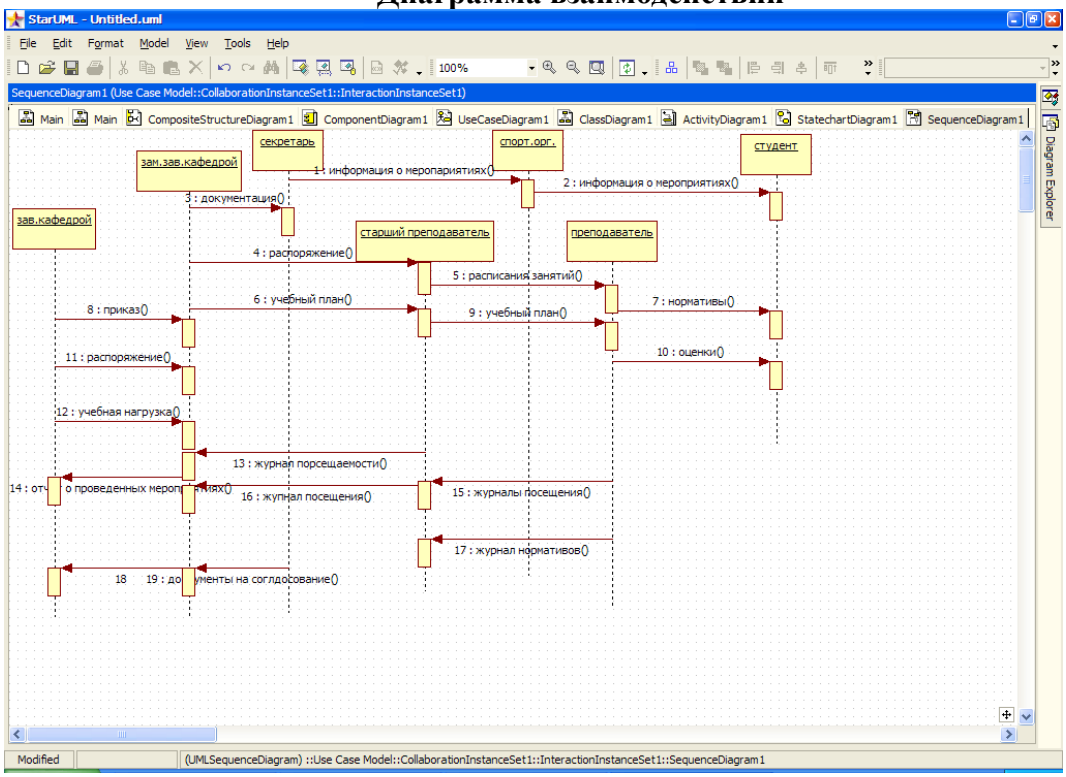


Рис. 1.4. Диаграмма взаимодействий

Диаграмма сообщений отображает взаимодействие объектов.

В нашем случае данная диаграмма отображает взаимодействие (передачу сообщений) между сотрудниками кафедры физического воспитания.

Диаграмма состояний

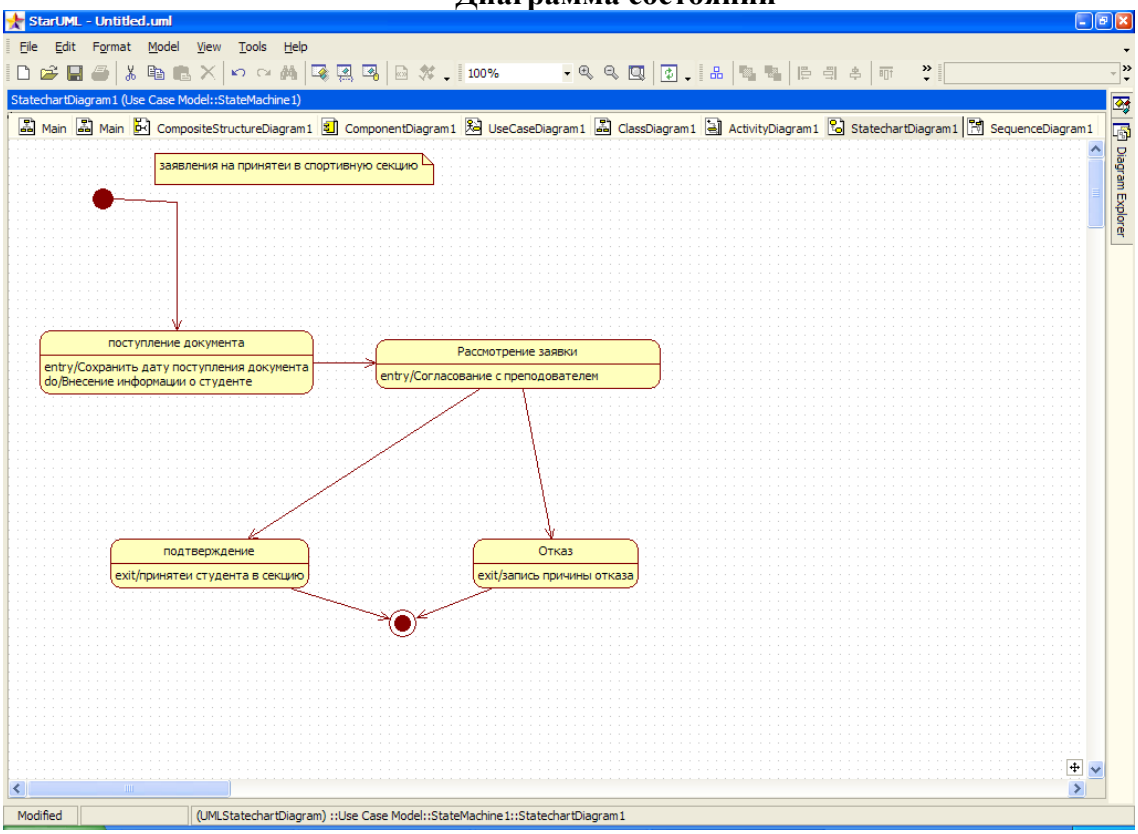


Рис. 1.5. Диаграмма состояний

Диаграмма состояний выражает поведение объекта через состояния и переходы состояний.

В нашем случае данная диаграмма показывает состояние заявления, начиная с его подачи и заканчивая заверением (подтверждение или отказ).

Диаграмма состояний необходима для описания динамического поведения классов, если эти классы могут находиться в разных состояниях. Одна диаграмма соответствует одному классу.

В любой момент времени класс может находиться в одном состоянии. Переход от одного состояния в другое - скачкообразный. Переход должен сопровождаться изменением значения хотя бы одной переменной из данных класса. Классы могут переходить из одного состояние в другое самостоятельно или под внешним воздействием. Переход класса из одного состояния в другое – событие (event).

Событие = условие возникновения + параметры.

Варианты индивидуального задания.

Описание подсистем АСУ ВУЗ

Вариант задания	Название	Описание
1	Подсистема «Ректорат»	Модель работы ректора ВУЗа, проректора по учебной работе, проректора по научной работе, проректора по АХЧ, секретаря ректора, секретаря проректора
2	Подсистема «Ученый Совет ВУЗа»	Модель работы Ученого Совета ВУЗа, председателя (ректора), ученого секретаря, членов Ученого Совета ВУЗа
4	Подсистема «Деканат»	Модель работы декана, зам. декана, секретаря
5	Подсистема «Кафедра КСУ»	Модель работы зав. кафедрой, секретаря, преподавателей (профессоров, доцентов, старших

		преподавателей, преподавателей, ассистентов),Зав. учебной лабораторией, инженера, лаборанта, техника
6	Подсистема «Кафедра физвоспитания»	Модель работы зав. кафедрой, секретаря, преподавателей (профессоров, доцентов, старших преподавателей, преподавателей, ассистентов), инженера, лаборанта, техника
7	Подсистема «Дворец культуры»	Модель работы зав. Дворцом культуры, секретаря, руководителя кружка
8	Подсистема «Профком студентов»	Модель работы председателя профкома, бухгалтера, секретаря, членов профкома, профоргов студенческих групп, студентов
9	Подсистема «Учебная библиотека»	Модель работы зав. библиотекой, библиографов, сотрудников библиотеки, читателей (студентов и преподавателей)
10	Подсистема «Отдел кадров»	Модель работы начальника отдела кадров, секретаря, сотрудников отдела кадров, сотрудников, преподавателей и студентов ВУЗа
11	Подсистема «Канцелярия»	Модель работы зав. канцелярией, секретаря, сотрудников канцелярии, сотрудников, преподавателей и студентов ВУЗа
12	Подсистема «Отдел охраны труда и ТБ»	Модель работы начальника, секретаря, инженера, лаборанта, техника
13	Подсистема «Бухгалтерия»	Модель работы главного бухгалтера, бухгалтера, секретаря
14	Подсистема «Приемная комиссия»	Модель работы председателя приемной комиссии, секретаря, сотрудника приемной комиссии

Контрольные вопросы

1.Что означает название UML и MDA?

2. Перечислите разные типы диаграмм в языке UML.

3.В каких областях науки и техники могут использоваться модели, созданные с помощью программы StarUML?

4.Перечислите основные концепции StarUML.

5.Что такое модель в программе StarUML?

6. Что такое представление, диаграмма, проект?

7. Какие элементы входят в структуру проекта?

8. Какое расширение имеют файлы проекта в StarUML?

9. Что такое фреймворк?

10. Какие профили используются в StarUML?

11. Какие базовые модули использует StarUML?

12. Перечислите основные типы диаграмм в программе StarUML.

Лабораторная работа № 6. Применение отладочных классов в проекте

Цель работы: изучение среды программного инструмента моделирования StarUML, поддерживающего UML, и приобретение навыков по созданию диаграмм классов.

Теоретическая часть

Диаграммы классов

Диаграмма классов является частью логической модели системы и представляет статическую картину системы.

Для каждой системы строится не одна, а несколько диаграмм классов: возможно, что для каждого прецедента или сценария своя. На одних показывают подмножества классов, объединенные в пакеты, и отношения между ними, на других – отображают те же подмножества, но с атрибутами и операциями классов. Для представления системы разрабатывается столько диаграмм классов, сколько потребуется.

Основные элементы диаграмм классов

Дадим некоторые определения и опишем основные элементы нотации диаграмм классов.

Объект – это некоторая сущность реального мира или концептуальная(абстрактная) сущность.

Пример. Примерами объектов могут служить дом №4 по улице Садовая, сотрудник фирмы Иван Петров, ваш компьютер. Или нечто абстрактное: химическая формула, торговый заказ номер 456789, банковский счет клиента Петра Иванова.

Объект имеет четко определенные границы и значение для системы и характеризуется состоянием, поведением и индивидуальностью.

Состояние объекта – это одно из условий, в котором он может находиться. Состояние обычно изменяется со временем и характеризуется набором свойств, которые называются атрибутами.

Пример. Покупатель определяется его именем, адресом, телефоном, датой рождения.

Поведение определяет, как объект реагирует на запросы других объектов и что может делать сам объект. Поведение характеризуется операциями объекта.

Пример. Покупатель может добавить товар в корзину, просматривать каталог, удалять товар из корзины.

Индивидуальность означает, что каждый объект уникален, даже если его состояние идентично состоянию другого объекта.

Пример. Объекты Ирина Петрова и Анна Седова уникальны, хотя каждый из них является покупателем магазина и имеет одинаковые поведение и состояния.

Как правило, в системе существует множество объектов имеющих одинаковое поведение, принимающих одинаковые состояния. Например, сотрудники фирмы, которых может быть несколько десятков, и данные о которых содержатся в базе данных, имеют одинаковые атрибуты – фамилию, имя, отчество, дату рождения, должность и др. – с разными значениями этих атрибутов, а также могут иметь схожее поведение – подать заявление на отпуск или перевод в другое подразделение. Для группировки объектов используются классы.

Класс – это описание группы объектов с общими свойствами (атрибутами), поведением (операциями), отношениями с другими объектами и семантикой.

Каждый класс является шаблоном для создания объекта. А каждый объект – это экземпляр класса. Важно помнить, что ***каждый объект может быть экземпляром только одного класса***

Пример. Применительно к магазину мы можем сгруппировать сотрудников магазина, описав общий для них класс Сотрудник. Объект этого класса, например, Иван Петров, может включать в себя следующую информацию: имя, адрес, должность, размер заработной платы, кроме того этот объект может выйти в отпуск.

В нотации UML классы и объекты изображаются в виде прямоугольников (см. рис. 1).

Прямоугольник класса всегда делится на три секции (раздела), имя класса помещается в первую секцию, каждое слово в названии класса принято писать с большой буквы. Во второй и третьей секциях могут указываться атрибуты и операции класса соответственно, эти секции могут быть пустыми. Названия классов выбираются в соответствии с понятиями предметной области. Это должно быть существительное или словосочетание в единственном числе, наиболее точно характеризующее предмет. Класс должен описывать только одну сущность.

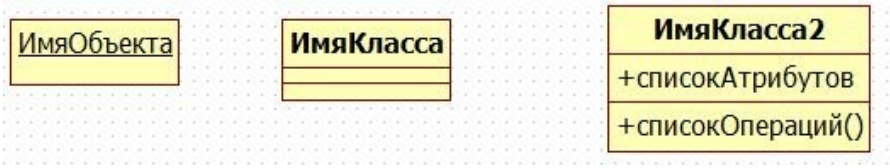


Рисунок 1- Изображение классов и объектов

Имя класса может быть простым, как это показано на рисунке 1, или составным (см. рис. 2). Составное имя класса состоит из самого имени класса и из имени пакета, которому принадлежит класс, разделенных двоеточием. Имя класса должно быть уникальным внутри пакета.

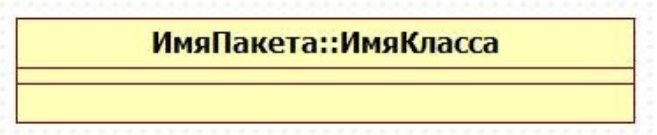


Рисунок 2 - Составное имя класса

Составное имя объекта также состоит из имени объекта и имени класса, разделенных двоеточием. Объект может быть анонимным, если неизвестно его настоящее имя. Тогда на диаграмме объект изображается с именем, которое состоит из двоеточия и имени класса, которому принадлежит объект. Если пока неизвестен класс, экземпляром которого является объект, то изображается имя объекта после которого идет двоеточие. Такой объект называется «сиротой» (см. рис. 3).

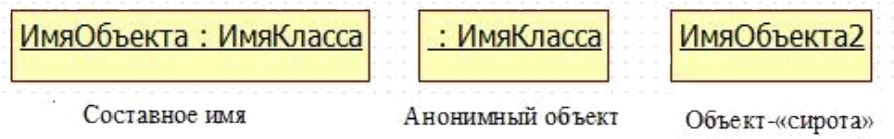


Рисунок 3 - Именование объектов

Пример. Класс Сотрудник и объект этого класса - некоторого сотрудника - можно изобразить так, как показано на рисунке 4.

Рисунок 4 - Класс и его объект

Мы дали объекту класса Сотрудник имя, совпадающее с именем класса.

Выявление классов

Выявление классов можно начать с изучения потока событий. Имена существительные в описании этого потока дадут понять, что может являться классом. В общем случае существительное может оказаться действующим лицом, классом, атрибутом класса или выражением, не являющимся ни действующим лицом, ни классом, ни атрибутом класса.

Если в ходе проектирования системы Вы уже построили диаграммы взаимодействия, перед тем, как приступать к построению диаграмм классов, то ищите на этих диаграммах похожие объекты. Например, у Вас может быть диаграмма последовательности, описывающая оформление заказа объектами Ивановым и Петровым. Обратите внимание на эти объекты: они имеют одинаковые свойства: имя, счет в банке и т.п. Значит, в системе должен появиться класс с именем Покупатель, который будет шаблоном объектов Иванов и Петров.

Некоторые возможные классы будут выявлены при рассмотрении трех стереотипов: сущность (entity), граница (boundary) и управление (control). Мы уже встречались со стереотипами отношений, когда говорили об отношениях на диаграммах прецедентов. Тот же принцип создания нового типа на основе уже существующего применим и для классов.

Стереотип – это механизм, позволяющий категоризировать классы. Он используется для создания нового типа элемента, в данном случае нового типа класса.

Например, Вы хотите выделить все экранные формы в модели. Для этого нужно создать стереотип Form (Форма).

Стереотипы помогают лучше понять ответственности каждого класса в модели, категоризировать выполняемые ими функции. В UML для этого применяют три основных стандартных вида стереотипов классов: классы-сущности, граничные классы и управляющие классы.

Класс-сущность содержит информацию, хранимую постоянно. Используется для моделирования данных и поведения с длинным жизненным циклом. Они могут представлять информацию о предметной области, а могут представлять элементы самой системы. Часто являясь абстракциями предметной области, они имеют наибольшее значение для пользователя, поэтому в их названиях применяются термины предметной области. Если существует проект базы данных, то можно обратиться к изучению названий таблиц, многие из них станут классами-сущностями. Обозначаются классы-сущности стереотипом <<entity>> либо специальной пиктограммой (рис. 5).

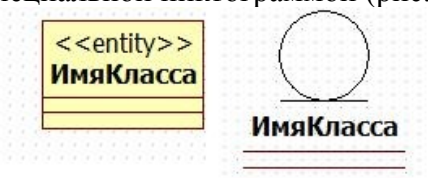


Рисунок 5 - Обозначение классов-сущностей

Граничными классами называются классы, расположенные на границе системы со всем остальным миром, и т.о. они обеспечивают взаимодействие между окружающей средой и внутренними элементами системы.

Для вычисления пограничных классов необходимо исследовать диаграммы вариантов использования. Для каждого взаимодействия между актером и прецедентом нужно создать хотя бы один граничный класс. Обратите внимание, что если два действующих лица инициируют один прецедент, то они могут применять один общий пограничный класс для взаимодействия с системой. Обозначаются граничные классы именем стереотипа <<boundary>> либо специальной пиктограммой (рис. 6).

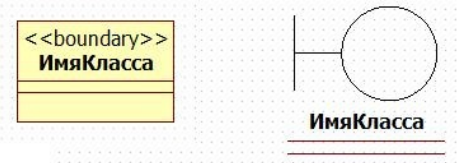


Рисунок 6 - Обозначение граничных классов

Управляющий класс отвечает за координацию действий других классов. Они служат для моделирования последовательного поведения одного или нескольких прецедентов и координации событий, реализующих заложенное в них поведение. Обозначаются управляющие классы именем стереотипа <<control>> либо специальной пиктограммой (рис. 7).

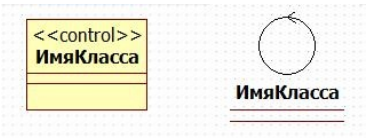


Рисунок 7 - Обозначение управляющих классов

Управляющие классы можно представить, как «исполняющие» прецедент, поэтому у каждого варианта использования обычно имеется один управляющий класс,

контролирующий последовательность событий этого прецедента. Они обычно зависят от приложения.

Управляющий класс делегирует ответственности другим классам. Сам он может получать мало сообщений, но отсылать множество. Его называют классом-менеджером. Он запускает альтернативные потоки и знает, как поступить в случае ошибки. На начальном этапе проектирования управляющие классы создаются для каждой пары актер/прецедент, в дальнейшем они могут объединяться, разделяться или исключаться.

Документирование классов

После того, как класс создан, информацию о нем необходимо документировать. Заметим, что документация предназначена для описания предназначения класса, а не его структуры.

Пример. Если в нашей модели присутствует класс Сотрудник, то хорошим описанием для него будет:

Сотрудник – это человек, работающий на фирме. Класс содержит информацию, необходимую для исполнения организацией своих обязанностей по отношению к сотруднику (начисление зарплаты, перевод на другую должность, увольнение и т.п.)

Плохим описанием будет описание структуры класса, которая может быть и так описана с помощью атрибутов. Например, плохое описание класса Сотрудник:

Имя, телефон, адрес, должность, зарплата.

В StarUML документирование классов выполняется также как и описанное ранее документирование прецедентов. Нужно выделить класс, который вы хотите описать, открыть окно документирования Documentation на инспекторе модели и ввести описание класса.

Пакеты в языке UML

Если в нашей модели немного классов, то нам легко ими управлять, однако многие системы содержат большое количество классов, поэтому необходим механизм, позволяющий классы группировать и облегчающий их повторное использование. Таким механизмом в UML являются пакеты.

Пакет (package) — общецелевой механизм для организации различных элементов модели в группы.

Подпакет (subpackage) — пакет, который является составной частью другого пакета.

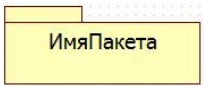
Пакет в логическом представлении модели – это объединение классов или других пакетов. С помощью объединения классов в пакеты мы можем получить представление о системе на более высоком уровне. Напротив, рассматривая пакет, мы получаем более детальное представление модели.

Объединять классы в пакеты можно как угодно, однако, существует несколько наиболее распространенных подходов.

1. можно группировать классы по стереотипам: классы-сущности, граничные и управляющие классы.
2. группировка классов по их функциональности: например, пакет классов, отвечающих за безопасность системы или пакет классов Работа с сотрудниками и т.п.
3. наконец, применяют комбинацию двух указанных методов.

 дальнейшем можно вкладывать пакеты друг в друга.

Чаще всего пакет на диаграмме изображается в виде папки с закладкой с именем пакета



Для того чтобы создать пакет на диаграмме, нужно открыть рабочее поле диаграммы, щелкнуть по элементу пакет Package на панели элементов слева, затем щелкнуть по рабочему полю диаграммы в том месте, где вы хотите поместить пакет. В окне редактора свойств можно задать новое имя пакета.

Чтобы разместить классы по пакетам, используют метод перетаскивания: на навигаторе модели нужно перетащить, удерживая левую кнопку мыши, классы в соответствующие

пакеты на навигаторе модели.

Постановка задачи: *Необходимо сгруппировать в пакеты классы, созданные при выполнении предыдущих работ. Затем нужно будет построить несколько диаграмм Классов и показать на них классы и пакеты системы.*

Порядок выполнения работы:

Создание диаграммы Классов

Объедините обнаруженные классы в пакеты. Создайте диаграмму Классов для отображения пакетов, диаграммы Классов, для представления классов в каждом пакете и диаграмму Классов для представления всех классов варианта использования "Ввести новый заказ".

Этапы выполнения упражнения Создание пакетов

1. В обозревателе моделей щелкните правой кнопкой мыши по разделу <<useCaseModel>> и выберите Add > Package (рисунок 8).

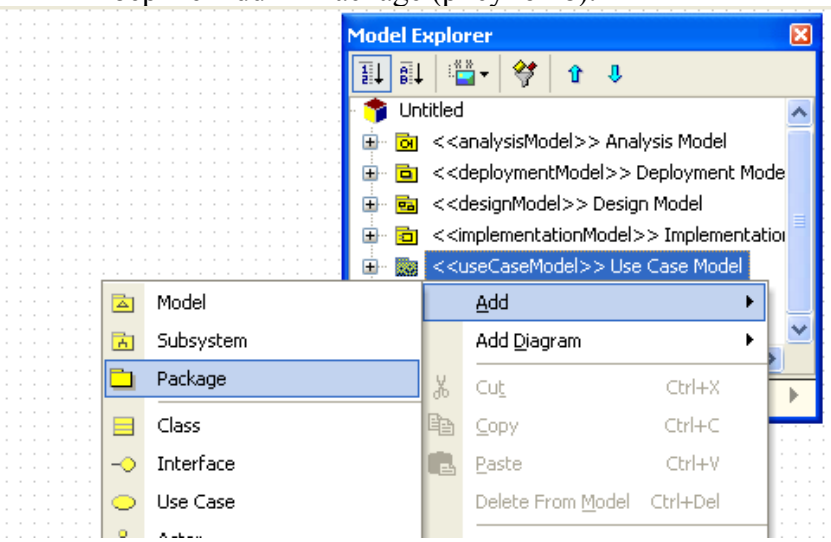


Рисунок 8 – Создание пакета

2. Назовите новый пакет Сущности.
3. Повторив шаги 1—2, создайте пакеты Границы и Управление.

Создание Главной диаграммы Классов

1. В навигаторе моделей выберите с помощью контекстного меню создайте новую диаграмму классов **Class Diagram**, откройте ее.
2. Перетащите пакет *Сущности* из браузера на диаграмму.
3. Перетащите пакеты *Границы* и *Управление* из браузера на диаграмму.

Главная диаграмма Классов должна выглядеть, как показано на рис. 9

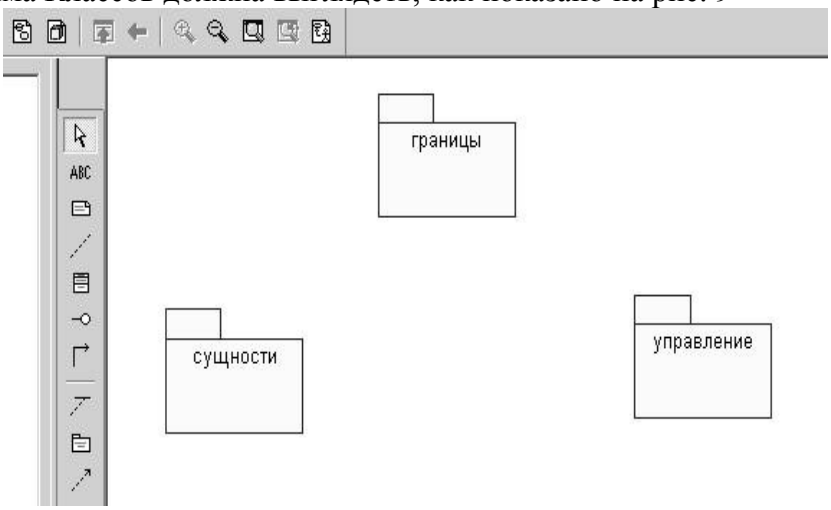


Рисунок 9 - Диаграмма классов в логическом представлении браузера.

Создание диаграммы Классов для сценария "Ввести новый заказ" с отображением всех классов

1. В навигаторе моделей выберите с помощью контекстного меню создайте новую диаграмму классов **Class Diagram**.
2. Назовите новую диаграмму Классов: Ввод нового заказа.
3. Дважды щелкнув мышью на этой диаграмме в браузере, откройте ее.
4. Перетащите из браузера все классы (*Выбор_заказа, Заказ_деталей, упр_заказами, Заказ, Упр_транзакциями*

Объединение классов в пакеты

1. В браузере перетащите класс *выбор_заказа* на пакет Границы.
2. Перетащите класс *заказ_деталей* на пакет Границы.
3. Перетащите классы *Упр_заказами* и *Упр-транзакциями* на пакет Управление.
4. Перетащите класс *Заказ* на пакет Сущности.

Классы и пакеты в браузере показаны на рис. 10

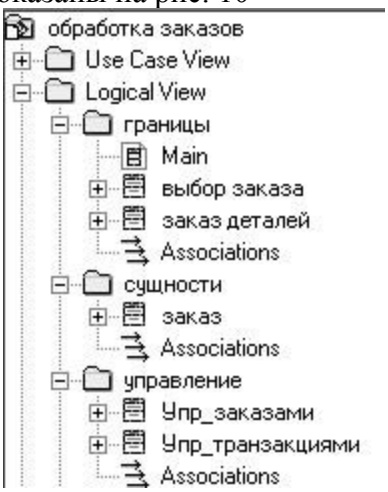


Рисунок 10 - Представление пакетов и классов Добавление диаграмм Классов к каждому пакету

1. В браузере щелкните правой кнопкой мыши на пакете *Границы*.
2. В открывшемся меню выберите пункт Add Diagram > Class Diagram.
3. Введите имя новой диаграммы — Main (Главная).
4. Дважды щелкнув мышью на этой диаграмме, откройте ее.
5. Перетащите на нее из браузера классы *выбор_заказа* и *заказ_деталей*.
6. Закройте диаграмму.

В браузере щелкните правой кнопкой мыши на пакете *Сущности.*

8. В открывшемся меню выберите пункт Add Diagram > Class Diagram.
9. Введите имя новой диаграммы — Main (Главная).
10. Дважды щелкнув мышью на этой диаграмме, откройте ее.
11. Перетащите на нее из браузера класс *Заказ*.
12. Закройте диаграмму
13. В браузере щелкните правой кнопкой мыши на пакете *Управление*
14. В открывшемся меню выберите пункт Add Diagram > Class Diagram.
15. Введите имя новой диаграммы — Main (Главная).
16. Дважды щелкнув мышью на этой диаграмме, откройте ее.
17. Перетащите на нее из браузера классы *Упр_заказами* и *Упр_транзакциями*
18. Закройте диаграмму

Уточнение методов и свойств классов

В этом упражнении к описаниям операций будут добавлены детали, включая параметры и типы возвращаемых значений, и определены атрибуты классов

Постановка проблемы

Для определения атрибутов классов был проанализирован поток событий. В

результате к классу Заказ диаграммы Классов были добавлены атрибуты Номер заказа и Имя клиента. Так как в одном заказе можно указать большое количество товаров и у каждого из них имеются свои собственные данные и поведение, было решено моделировать товары как самостоятельные классы, а не как атрибуты класса Заказ.

Добавление атрибутов и операций

Добавим атрибуты и операции к классам диаграммы Классов "Ввод нового заказа". При этом используем специфические для языка особенности. Установим параметры так, чтобы показывать все атрибуты, все операции и их сигнатуры. Применим нотацию UML.

Этапы выполнения упражнения

Настройка

1. В меню модели выберите пункт Tools > Options (Инструменты >Параметры).
2. На вкладках General, Diagram, General View выполните необходимые настройки:
3. Убедитесь, что флажок Show visibility (Показать видимость) установлен.
4. Убедитесь, что флажок Show stereotyps (Показать стереотипы) установлен.
5. Убедитесь, что флажок Show operation signatures (Показать сигнатуры операций) установлен.
6. Убедитесь, что флажки Show all attributes (Показать все атрибуты) и Show all operations (Показать все операции) установлены.
7. Убедитесь, что флажки Suppress attributes (Подавить атрибуты) и Suppress operations (Подавить операции) сброшены.
8. Убедитесь, что флажок Visibility as icons (Отображать пиктограммы) сброшен.

Добавление нового класса

1. Найдите в браузере диаграмму Классов варианта использования "Ввести новый заказ".
2. Дважды щелкнув мышью на диаграмме, откройте ее.
3. Нажмите кнопку C1ass панели инструментов.
4. Щелкните мышью внутри диаграммы, чтобы поместить туда новый класс.
5. Назовите его *Позиц_заказа*.
6. Назначьте этому классу стереотип Entity (рисунок 11).

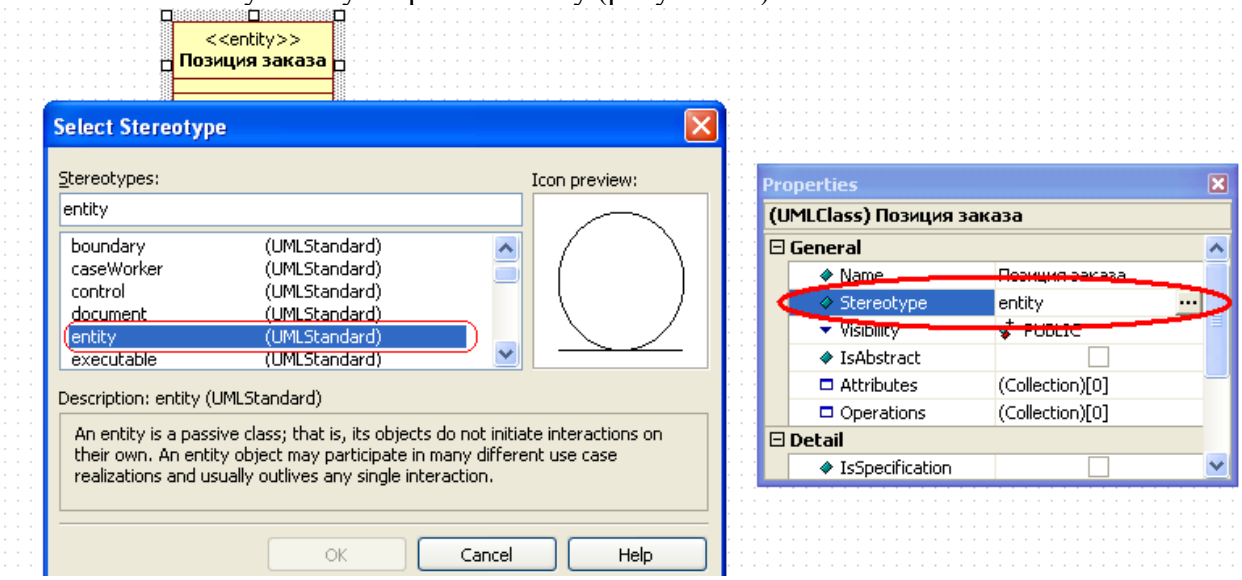


Рисунок 11 – Назначение классу Позиция заказа стереотипа Entity

7. В браузере перетащите класс в пакет *Сущности.*
8. Стереотипы классов можно отобразить с помощью пиктограмм. Для этого нужно выделить класс, щелкнуть по выделенной области правой кнопкой мыши, в контекстном меню выберите пункт Format, затем выберите пункт Stereotype Display, далее в списке выберите Iconic (рисунок 12)

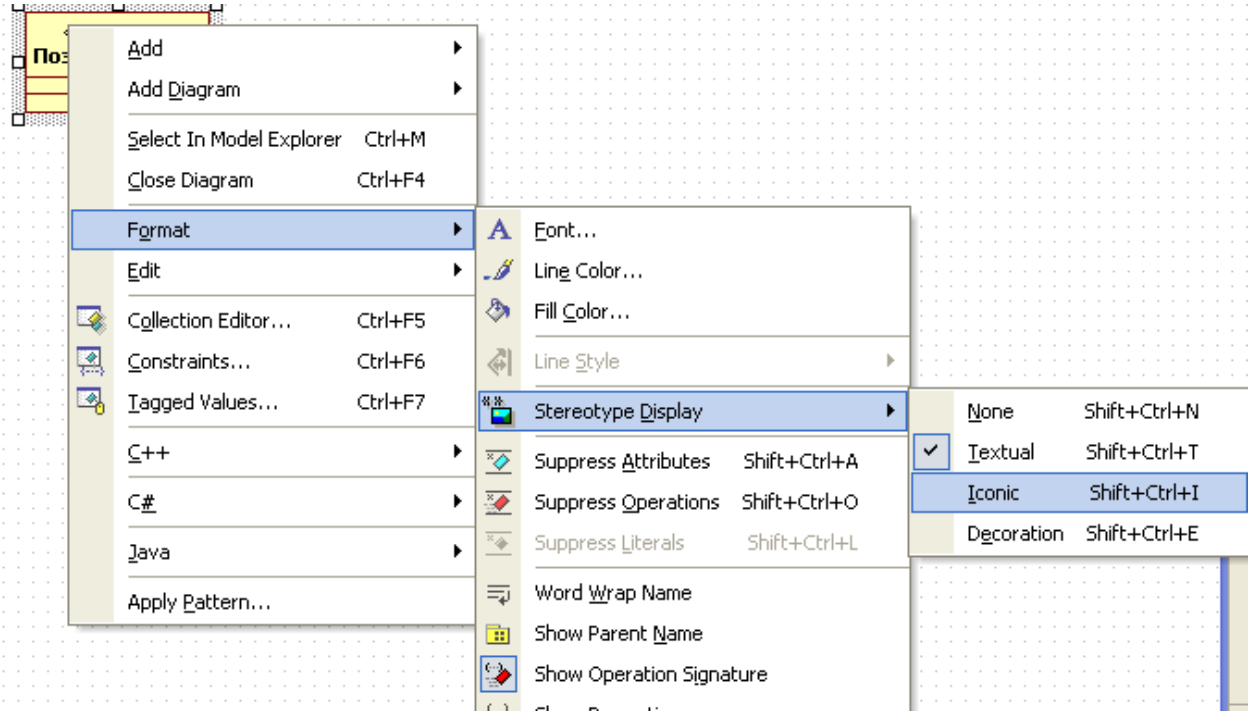


Рисунок 12 –Задание отображения стереотипов классов с помощью пиктограмм

Классы будут отображаться как пиктограммы (рисунок 13)

Рисунок 13 –Отображение классов с помощью пиктограмм стереотипов

Добавление атрибутов

1. Щелкните правой кнопкой мыши на классе *Заказ*.

2. В открывшемся меню выберите пункт New Attribute (Создать атрибут),

3. Введите новый атрибут: **OrderNumber** : Integer

4. Нажмите клавишу Enter

5. Введите следующий атрибут: **CustomerName** : String.

5. Повторив шаги 4 и 5, добавьте атрибуты: **OrderDate** : Date **OrderFillDate** : Date

Если тип атрибута не появляется в выпадающем списке, то введите его от руки и он далее будет появляться.

7. Щелкните правой кнопкой мыши на классе *Позиц_заказа*.

8. В открывшемся меню выберите пункт New Attribute (Создать атрибут).

9. Введите новый атрибут: ***ItemID*** **: Integer.**

10. Нажмите клавишу Enter.

11. Введите следующий атрибут: ***ItemDescription*** **: String.**

Добавление операций к классу *Позиц_заказа*

1. Щелкните правой кнопкой мыши на классе *Позиц_заказа*.

2. В открывшемся меню выберите пункт Collection Editor и перейдите на вкладку Operations, то же самое можно выполнить выбрав свойство Operations в окне Properties (рисунок 14).

3. Введите новую операцию: *Создать()*

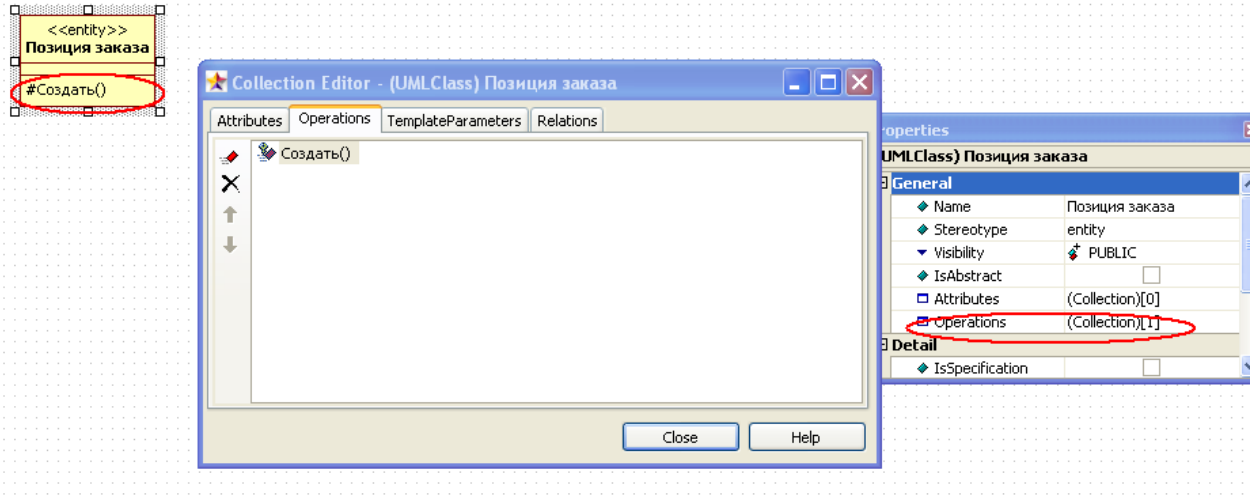


Рисунок 14 –Добавление операции к классу

Более быстрый способ создать атрибут или операцию – это щелкнуть два раза левой кнопкой мыши по классу (рисунок 15).

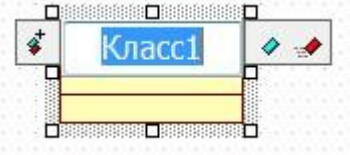


Рисунок 15. Быстрое создание атрибута

Нажав кнопку (для атрибута) или (для операции) вы получите атрибут или операцию соответственно.

Чтобы удалить атрибут или операцию щелкните по ней два раза левой кнопкой мыши и нажмите на значок справа. Чтобы добавить еще один атрибут или операцию нажмите на значок (рис. 16).

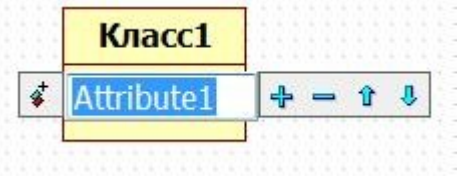


Рисунок 16. Быстрое добавление и удаление атрибута

4. Нажмите клавишу Enter.

5. Введите следующую **операцию:** *Взять_информацию()*

6. Нажмите клавишу Enter.

7. Введите операцию: *Дать_информацию()*

Подробное описание операций с помощью диаграммы Классов

1. Щелкнув мышью на классе *Заказ*, выделите его.

2. Щелкните на этом классе еще раз, чтобы переместить курсор внутрь.

3. Отредактируйте операцию *Создать()*, чтобы она выглядела следующим образом: **Создать() : Boolean**

4. Отредактируйте операцию *Взять_информацию*:

Взять_информацию (OrderNum : Integer, Customer : String, OrderDate : Date, FillDate : Date) : Boolean

5. Отредактируйте операцию *Дать_информацию*; *Дать_информацию()***: String**

Подробное описание операций с помощью браузера

1. Найдите в браузере класс *Позиц_заказа*.

2. Раскройте этот класс, щелкнув на значке "+" рядом с ним. В браузере появятся атрибуты и операции класса.

3. Дважды щелкнув мышью на операции *Дать_информацию()* , откройте окно свойств:

4. В свойстве Type укажите тип String (рисунок 17).

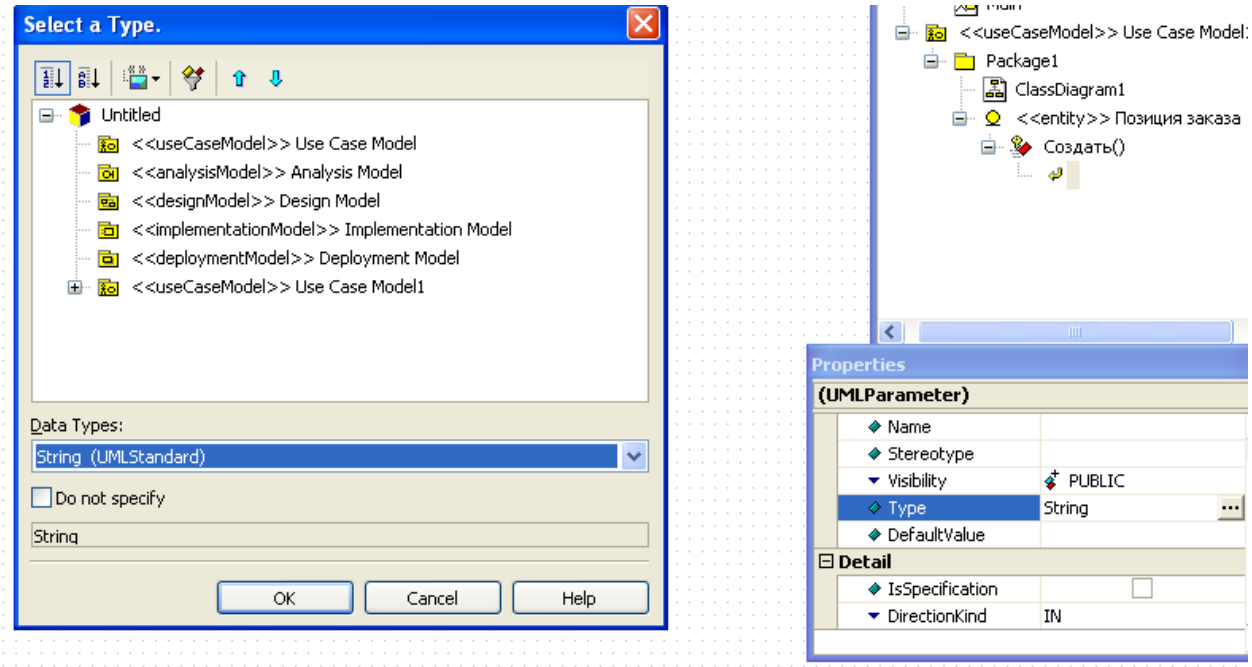


Рисунок 17. Определение типа атрибута

5. Щелкнув мышью на кнопке ОК, закройте окно спецификации операции.

6. Дважды щелкните в браузере на операции *Дать_информацию* () класса *Позиц_заказа*, чтобы открыть окно Properties.

7. В раскрывающемся списке Data Types окна Select a type, которое вызывается свойством Types укажите Boolean.

8. Перейдите в обозревателей моделей.

9. Щелкните правой кнопкой мыши в области операции, чтобы добавить туда новый параметр.

10. В открывшемся меню выберите пункт Add > Parametr. Будет добавлен параметр под названием Parametr1 (рисунок 18).

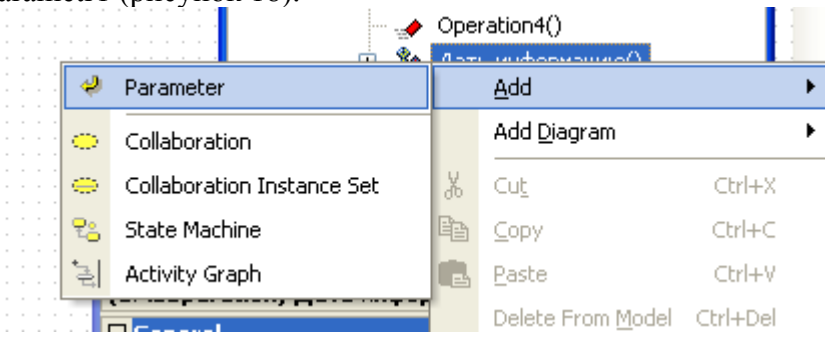


Рисунок 18 – Добавление параметра операции

11. В окне Properties измените имя аргумента на ID.

12. Щелкните на свойстве Type (Тип). В раскрывающемся списке типов выберите Integer (Если этого либо иного необходимого типа не будет- введите его вручную).

13. Щелкните на свойстве DefaultValue (По умолчанию), чтобы добавить значение аргумента по умолчанию. Введите число 0.

14. Дважды щелкните на операции *Создать()* класса *Позиц_заказа*, чтобы открыть окно ее свойств.

15. В раскрывающемся списке Data Types окна Select a type, которое вызывается свойством Types укажите Boolean.

16. Нажав на кнопку ОК, закройте окно.

Подробное описание операций

1. Используя браузер или диаграмму Классов, введите следующие сигнатуры операций класса *Заказ_деталей*: *Открыть()* : Boolean, *Сохранить заказ()* : Boolean

2. Используя браузер или диаграмму Классов, введите сигнатуру операций класса *Выбор_заказа*: *Создать()* : Boolean

3. Используя браузер или диаграмму Классов, введите сигнатуру операций класса *Упр_заказами*: *Сохранить заказ*(OrderID : Integer) : Boolean

4. Используя браузер или диаграмму Классов, введите сигнатуры операций класса *Упр_транзакциями*: *Сохранить заказ* (OrderID : Integer) : Boolean, *Сохранить информацию*() : Integer

Описание связей между классами

В этом упражнении определяются связи между классами, участвующими в варианте использования "Ввести новый заказ".

Постановка задачи

Чтобы найти связи, были просмотрены диаграммы Последовательности. Все взаимодействующие там классы нуждались в определении соответствующих связей на диаграммах Классов. После обнаружения связи были добавлены на диаграммы классов.

Добавление связей

Добавим связи к классам, принимающим участие в варианте использования "Ввести новый заказ".

Этапы выполнения упражнения

Добавление ассоциаций

1. Найдите в браузере диаграмму Классов "Ввод нового заказа",

2. Дважды щелкнув на диаграмме, откройте ее.

1. Нажмите кнопку DirectedAssociation панели инструментов.

2. Проведите ассоциацию от класса *выбор_заказа* к классу *заказ_деталей*.

3. Повторите шаги 1 и 2, создав ассоциации:

- От класса *заказ_деталей* к классу *упр_заказами*
- От класса *упр_заказами* к классу *Заказ*
- От класса *упр_заказами* к классу *упр_транзакциями*
- От класса *упр_транзакциями* к классу Заказ
- От класса *упр_транзакциями* к классу *Позиц_заказа*
- От класса *Заказ* к классу *Позиц_заказа*

4. Щелкните правой кнопкой мыши на однонаправленной ассоциации между классами *выбор_заказа* и *заказ_деталей* класса *выбор_заказа*.

5. В открывшемся меню выберите пункт Multiplicity > (Множественность >- Нуль или один). На рисунке 19 наглядно показано это действие.

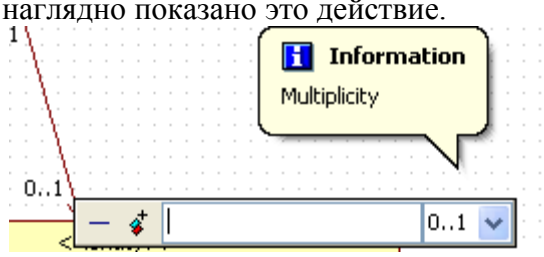


Рисунок 19 – Задание множественности

Задать это значение можно также пользуясь свойствами End.Multiplicity, End2.Multiplicity окна Properties (рисунок 20)

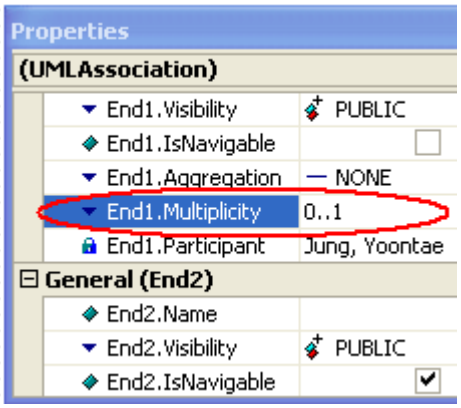


Рисунок 20 - Задание свойства множественности в окне Properties

6. Щелкните правой кнопкой мыши на другом конце однонаправленной ассоциации.
7. В открывшемся меню выберите пункт Multiplicity > (Множественность > Нуль или один),
8. Повторите шаги 4—7, добавив на диаграмму значения множественности для остальных ассоциаций, как показано на рис. 21

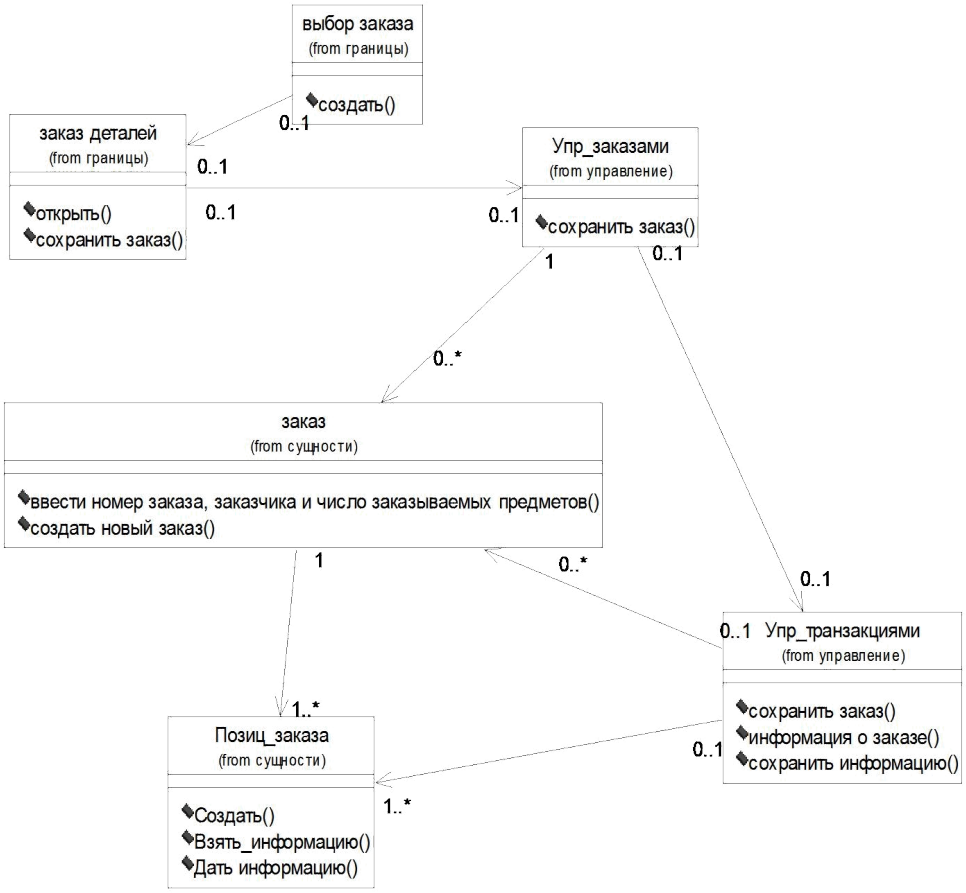


Рисунок 21 - Ассоциации сценария "Ввести новый заказ"

Задание+: Диаграмму классов сценария Оформление заказа отобразить с помощью пиктограмм стереотипов

Содержание отчета

Индивидуальный отчет о выполненной лабораторной работе должен содержать:

1 *цель* работы;

2 наименование функционального комплекса задач или задачи, для которых разработана диаграмма прецедентов использования;
3 разработанную диаграмму классов;
4 *выводы* о полученных знаниях и умениях.

Контрольные вопросы

1. Какие элементы может содержать диаграмма классов?
2. Какие свойства спецификации имеет класс?
3. Что называют сообщением в UML?
4. Какого типа отношения существуют между классами?
5. В каком случае устанавливается связь агрегации? Как определяется ее направленность?
6. Чем отличается отношение композиции от отношения агрегации?
7. В каком случае устанавливается связь обобщения? Как определяется ее направленность?
8. Каким образом может быть использована диаграмма классов?
9. В каком случае необходимо именовать отношение ассоциации или ее полюсов?
10. Прообразом чего в программном коде является операция класса?
11. Что такое интерфейс? В каких случаях в диаграмму классов включают интерфейсы?
12. Как определить, является ли класс классом сущности?

Лабораторная работа № 7. Отладка проекта

Цель работы состоит в приобретении навыков применения правил и последовательности отладки программного продукта

Краткие теоретические и учебно-методические материалы по теме практической работы

Процесс исправления ошибок называется отладкой. Отладка программ и обработка ошибок всегда выступает как часть процесса разработки. В большинстве систем разработки имеются инструменты, с помощью которых можно решить проблемы, возникающие в процессе программирования. В VBA также есть средства, которые позволяют либо исключить ошибки при разработке, либо задать отклик на ошибки при выполнении программ.

Отладка программ и обработка ошибок - это не одно и то же, но они тесно связаны друг с другом.

Отладка программ - это проверка и внесение исправлений в программу при ее разработке. Отладка позволяет идентифицировать ошибки, допущенные при программировании. Например, синтаксические ошибки в тексте программы, именах функций и переменных или логические ошибки.

Обработка ошибок - это задание реакции на ошибки, которые возникают во время выполнения программы. Причиной ошибок могут быть как ошибки в самой программе, так и другие обстоятельства, находящиеся вне сферы влияния программиста. Например, отсутствие файлов, к которым происходит программное обращение, отказ аппаратных средств или неправильные действия пользователя.

Невозможно предотвратить возникновение всех ошибок, но следует стремиться к уменьшению их числа. В маленькой программе довольно просто выявить ошибку. Однако по мере увеличения размеров и сложности программ находить их становится все труднее. В таких случаях необходимо пользоваться средствами отладки VBA.

Среда разработки программ на VBA предоставляет пользователю современные удобные средства отладки программы: предположим, что уже написан код вашей процедуры. Следующий этап в создании любой процедуры - тестирование написанного кода.

Тестирование - это процесс выполнения процедуры и исследование всех аспектов ее работы.

Цель тестирования - проверить правильность результатов выполнения процедуры и ее реакцию на разнообразные действия пользователя.

Если во время работы процедуры получены неверные результаты вычислений, непредвиденная реакция на те или иные действия пользователя, либо вообще произошла остановка выполнения, то это говорит о том, что в тексте программы имеются ошибки.

Все возможные ошибки можно разделить на три вида:

1. *Ошибки компиляции*. Возникают, если VBA не может интерпретировать введенный текст, например, при использовании неправильного синтаксиса инструкции или задании неверного имени метода или свойства. Некоторые ошибки компиляции обнаруживаются при вводе инструкции, а другие - только перед выполнением программы. Данный тип ошибок обычно просто идентифицировать и исправить, поскольку VBA выявляет их автоматически, а сами ошибки очевидны.

2. *Ошибки выполнения*. Возникают при выполнении программы, т.е. после успешной компиляции. Причиной таких ошибок может быть отсутствие данных или неправильная информация (например, данные, введенные пользователем). Ошибки выполнения, как и ошибки компиляции, легко идентифицируются VBA. При этом выводится инструкция, при выполнении которой произошла ошибка. Ошибки данного

типа тяжелее устранить: может понадобиться вывести значения переменных или свойств, а также другие данные, которые влияют на успешное выполнение программы.

3. *Логические ошибки* труднее всего заметить и устранить. Логические ошибки не приводят к прекращению компиляции или выполнения. Однако они являются причиной того, что программа не выдает желаемых результатов. Ошибки данного типа идентифицируются путем тщательной проверки с помощью средств отладки VBA.

Компиляция — это процесс преобразования программы, написанной на алгоритмическом языке, в язык машинных кодов. Если в программе есть синтаксические ошибки, то процесс компиляции прекращается, строки с ошибкой закрашиваются желтым цветом и выдается соответствующее сообщение. Для продолжения выполнения программы необходимо исправить ошибку и нажать кнопку «Continue»  на стандартной панели редактора VBA. Или прервать выполнение программы, нажав на кнопку «Reset»  , исправить ошибку в программе, а затем заново запустить ее. При обнаружении ошибки компилятор выдает сообщение с указанием номера ошибки. В этом случае полезно, воспользовавшись справочной системой редактора VBA, определить характер ошибки и исправить ее.

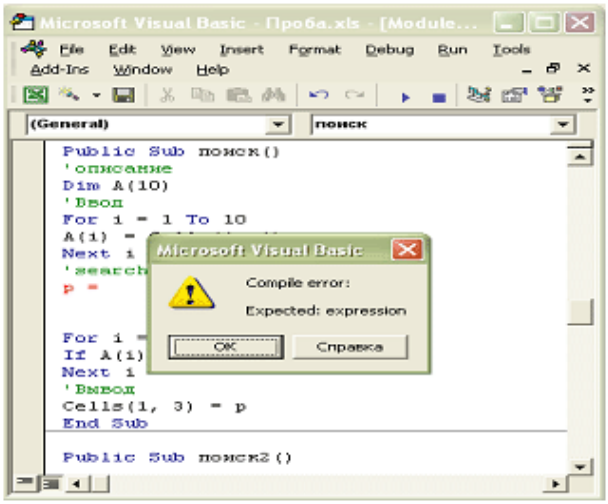


Рис. 1. Редактор Visual Basic немедленно реагирует на синтаксические ошибки

Чтобы исследовать процесс отладки на практике, нам необходима какая-нибудь программа, содержащая ошибку. В последующем примере написана такая программа «Отладка программ» рассматривается устранение ошибки при написании процедуры для объекта Image.

Задания для практического занятия:

1. Выполнить упражнения
2. Оформить работу в соответствии с ГОСТ 19.106-78. При оформлении использовать MS Office.
3. Сдать и защитить работу

Упражнения

Задание 1.

1. Откройте новую рабочую книгу.
2. Подготовьте экранную форму, представленную на рис.2. Внедрите в созданную форму с помощью панели Toolbox объект Image  . Рисунок лучше внедрить небольшой.

ВНИМАНИЕ!!! Правильно описывайте путь к графическим файлам, которые внедряются программно в форму.

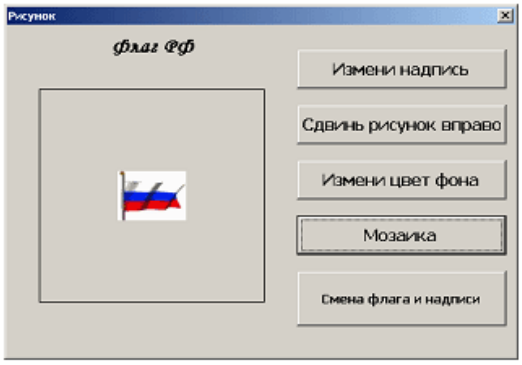


Рис. 2. Форма для выполнения задания

3. Создайте новую процедуру для кнопки «Измени надпись».

4. Введите текст процедуры. В тексте намеренно сделаем ошибку в свойстве Size (напишем Sie):

```
Private Sub CommandButton1_Click()
Label1.Caption = "Флаг России"
UserForm2.Image1.Picture = LoadPicture("C:\FlgRUS.gif")
Label1.Font.Sie = 14
End Sub
```

5. Вернемся в редакторе к созданной форме и выведем форму для работы, нажав клавишу.

6. После появления формы на экране нажмем на кнопку «Измени надпись». Так как в программе заложена ошибка, появится окно сообщения об ошибке (рис. 3), и открывается редактор VBA.

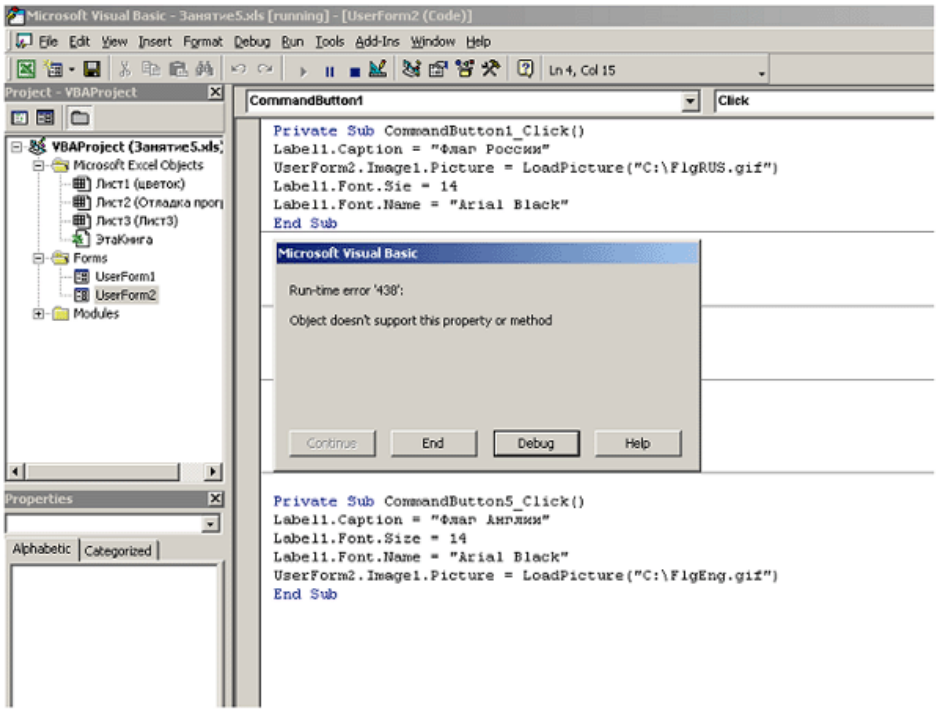


Рис. 3. Окно редактирования кода с окном сообщения об ошибке

7. Нажмите на кнопку «Debug» (отладка), и отладчик укажет, в какой строке у вас ошибка (рис. 4).

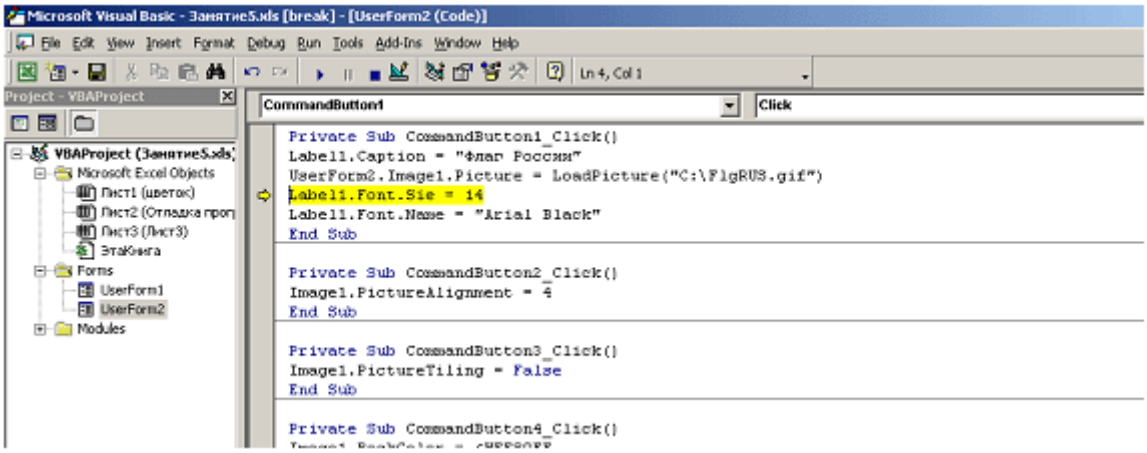


Рис. 4. Окно редактирования кода с указанной ошибкой

8. Исправьте ошибку и нажмите на стандартной панели инструментов на кнопку  («Продолжение»).

Тексты программ для кнопок CommandButton2, CommandButton3, CommandButton4, CommandButton5 представлены в таблице:

Объект	Программа
CommandButton2 (сдвинь рисунок вправо)	Private Sub CommandButton2_Click() Image1.PictureAlignment = 4 End Sub
CommandButton4 (измени цвет фона и формы)	Private Sub CommandButton4_Click() Image1.BackColor = &HFF80FF UserForm2.BackColor = RGB(64, 0, 0) End Sub
CommandButton3 (мозаика)	Private Sub CommandButton3_Click() Image1.PictureTiling = True End Sub
CommandButton5 (измени рисунок флага и надпись)	Private Sub CommandButton5_Click() Label1.Caption = "Флаг Англии" Label1.Font.Size = 14 Label1.Font.Name = "Arial Black" UserForm2.Image1.Picture = LoadPicture("C:\FlgEng.gif") End Sub

9. После щелчка по кнопке «Измени надпись» форма приобретет вид, представленный на рис. 5.

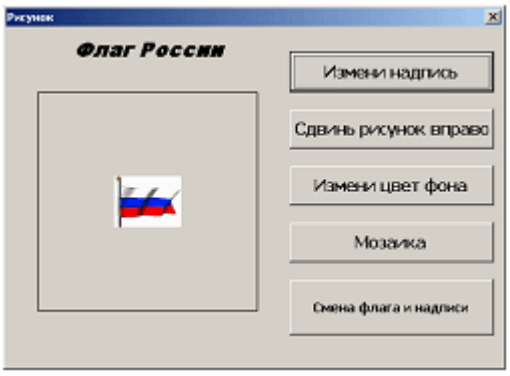


Рис. 5. Работа кнопки «Измени надпись»

10. После щелчка по кнопке «Сдвинь рисунок вправо» форма приобретет вид, представленный на рис. 6.

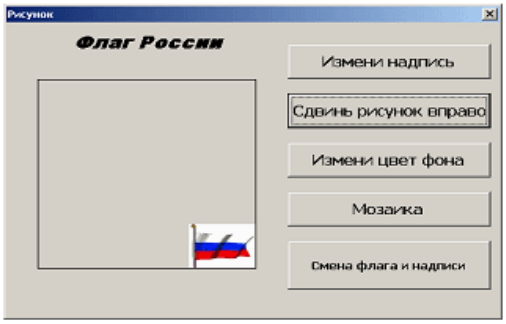


Рис. 6. Работа кнопки «Сдвинь рисунок вправо»

11. После щелчка по кнопке «Мозаика» форма приобретет вид, представленный на рис. 7.

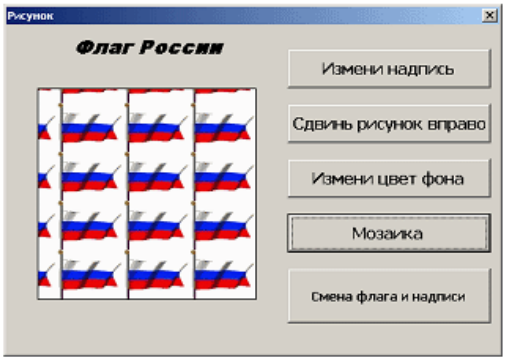


Рис. 7. Работа кнопки «Мозаика»

12. После щелчка по кнопке «Смена флага и надписи» форма приобретет вид, представленный на рис. 8.

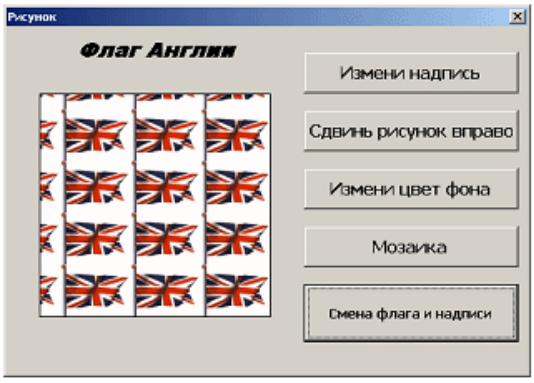


Рис. 8. Работа кнопки «Смена флага и надписи»

Можно предусмотреть разные комбинации рисунков и надписей.

13. Сохраните свою работу.

Задание 2.

1.Написать код на программный продукт с использованием редактора кода VBA, содержащий ошибку и показать преподавателю (см. пример).

2. Провести отладку программного продукта.

Контрольные вопросы:

1. Какие ошибки в программах существуют?
2. Что понимают под отладкой программы?
3. Чем отладка отличается от тестирования?

Лабораторная работа № 8. Инспекция кода модулей проекта

Цель работы: изучение среды программного инструмента моделирования StarUML, поддерживающего UML, приобретение навыков по созданию кооперативной диаграммы.

Теоретическая часть

Диаграммы взаимодействия (interactiondiagrams)

Диаграммы взаимодействия (interactiondiagrams) описывают поведение взаимодействующих групп объектов. Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой.

Сообщение (message) – это средство, с помощью которого объект-отправитель запрашивает у объекта получателя выполнение одной из его операций.

Информационное (informative) сообщение – это сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

Сообщение-запрос (interrogative) – это сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

Императивное (imperative) сообщение – это сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

Существует два вида диаграмм взаимодействия: диаграммы последовательности (sequencediagrams) и кооперативные диаграммы (collaborationdiagrams).

Диаграмма последовательности (sequencediagrams)

Диаграмма последовательности отражает поток событий, происходящих в рамках варианта использования.

Все действующие лица показаны в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций.

На диаграмме последовательности объект изображается в виде прямоугольника, от которого вниз проведена пунктирная вертикальная линия. Эта линия называется линией жизни (lifeline) объекта. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

Каждое сообщение представляется в виде стрелки между линиями жизни двух объектов. Сообщения появляются в том порядке, как они показаны на странице сверху вниз. Каждое сообщение помечается как минимум именем сообщения. При желании можно добавить также аргументы и некоторую управляющую информацию. Можно показать самоделегирование (self-delegation) – сообщение, которое объект посылает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни.

Диаграмма кооперации (collaborationdiagram)

Диаграммы кооперации отображают поток событий через конкретный сценарий варианта использования, упорядочены по времени, а кооперативные диаграммы больше внимания заостряют на связях между объектами.

Диаграмма кооперации – это альтернативный способ изображения сценария варианта использования. Этот тип диаграмм заостряет внимание на связях между объектами, отображая обмен данными в системе. А диаграммы последовательности отображают взаимодействие объектов во времени, поэтому ее следует читать сверху вниз и слева направо.

Диаграммы кооперации содержат все те же элементы, что и диаграммы последовательности: объекты, действующие лица, связи между ними и сообщения, которыми они обмениваются, но они уже не упорядочены во времени.

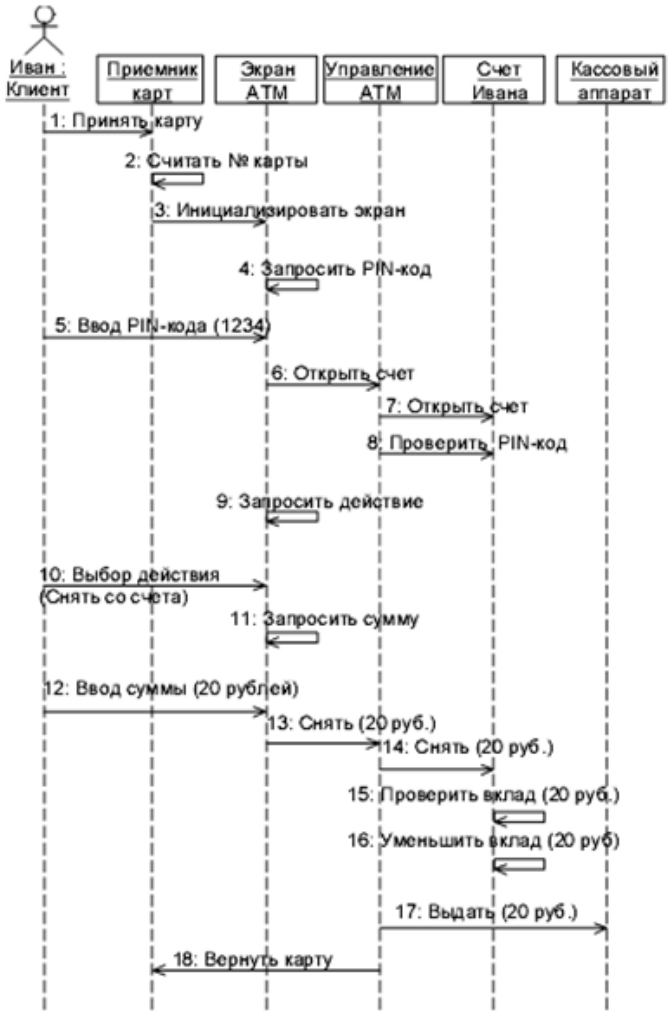


Рис. 1. Пример диаграммы последовательности

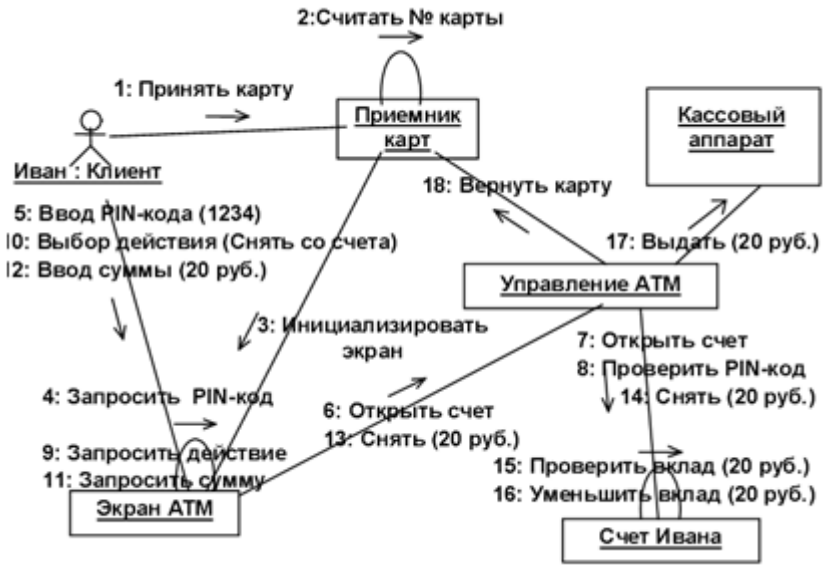


Рис. 2. Пример диаграммы кооперации

Порядок выполнения работы:

Постановка задачи. Создать диаграмму кооперации. Конечный вид диаграммы представлен на рис. 4.

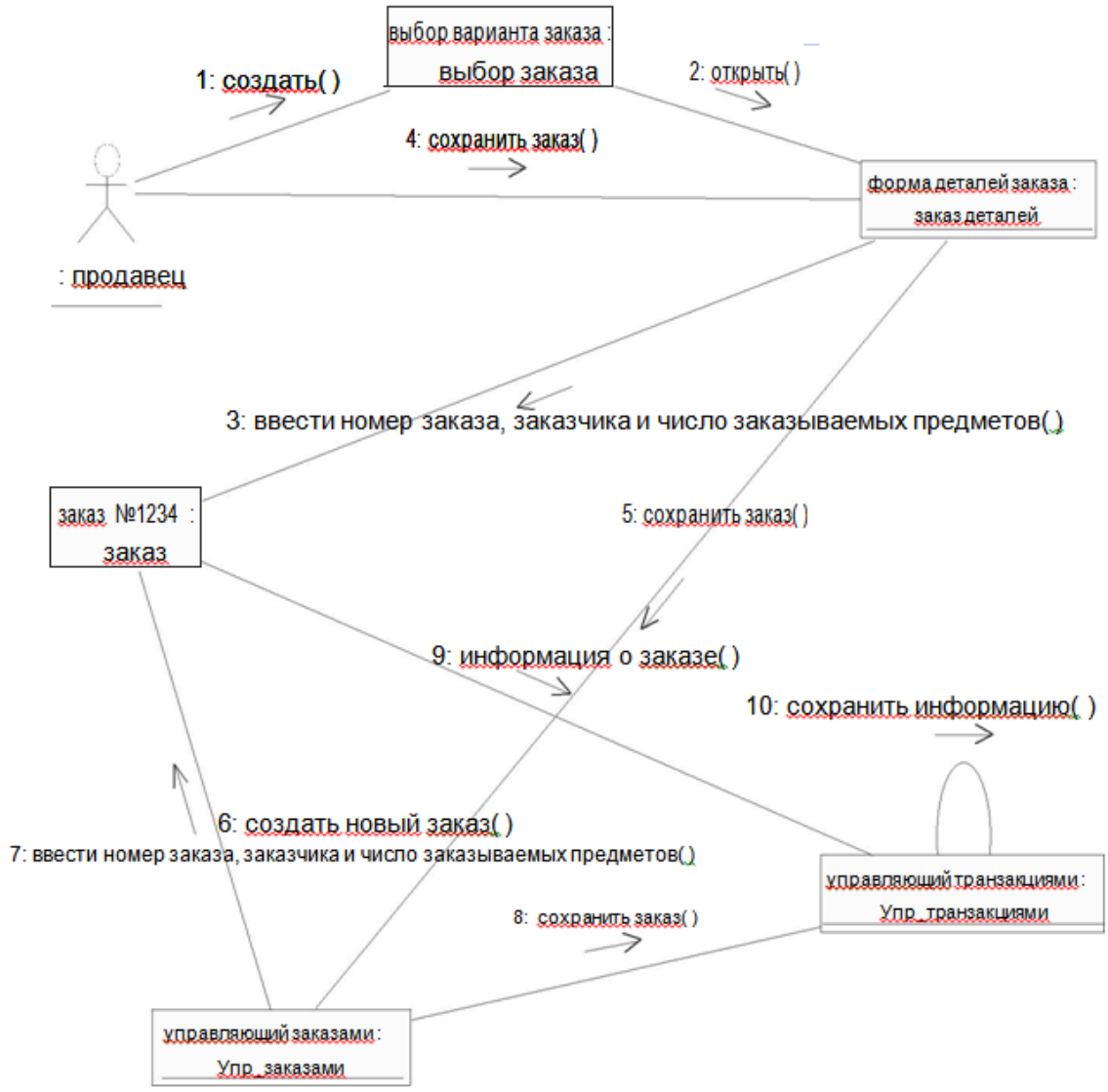


Рис. 3. Окончательный вид кооперативной диаграммы.

1. Для того чтобы добавить диаграмму кооперации в представление Logical View, щелкните правой кнопкой мыши по папке содержащей диаграмму последовательности (если вы ее не переименовывали, то она носит имя CollaborationInstanceSet1), в контекстном меню выберите пункт Add Diagram, в списке выберите диаграмму кооперации Collaboration diagram.
2. Назовите эту диаграмму *Ввод заказа*.
3. Дважды щелкнув мышью на диаграмме, откройте ее.

Добавление действующего лица и объектов на диаграмму

1. Перетащите действующее лицо *Продавец* из браузера на диаграмму.
2. Нажмите кнопку Object (Объект) панели инструментов.
3. Щелкните мышью где-нибудь внутри диаграммы, чтобы поместить туда новый объект.
4. Назовите объект *Выбор варианта заказа*.
5. Повторив шаги 3 и 4, поместите на диаграмму объекты:

- *Форма деталей заказа*
- *Заказ №1234*

Добавление сообщений на диаграмму

1. На панели инструментов нажмите кнопку Link (Связь объекта).
2. Проведите мышью от действующего лица *Продавец* к объекту *Выбор варианта заказа*.
3. Повторите шаги 1 и 2, соединив связями следующие объекты:

- Действующее лицо *Продавец* и объект *Форма деталей Заказа*
- Объект *Форма деталей Заказа* и объект *Выбор Варианта Заказа*
- Объект *Форма деталей Заказа* объект *Заказ N1234*

4. На панели инструментов нажмите кнопку Link Message (Сообщение связи).
5. Щелкните мышью на связи между *Продавец* и *Форма деталей Заказа.*
6. Выделив сообщение, введите его имя —*Создать новый заказ*;
7. Повторив шаги с 4 по 6, поместите на диаграмму сообщения:

- *Открыть форму* — между *Выбор Варианта Заказа* и *Форма Деталей Заказа*.
- *Ввести номер заказа, заказчика и число заказываемых предметов* — между *Продавец* и *Форма Деталей Заказа*
- *Сохранить заказ* — между *Продавец* и *Форма деталей Заказа*
- *Создать пустой заказ* — между *Форма деталей Заказа* и *Заказ №1234*
- Ввести номер заказа, заказчика и число заказываемых предметов — между *Форма деталей Заказа* и *Заказ №1234*
- Сохранить заказ — между *Форма деталей Заказа* и *Заказ №1234*

Теперь нужно поместить на диаграмму дополнительные элементы, а также рассмотреть ответственности объектов.

Добавление на диаграмму дополнительных объектов

1. Нажмите кнопку Object панели инструментов.
2. Щелкните мышью где-нибудь на диаграмме, чтобы поместить туда новый объект.
3. Введите имя объекта — *Управляющий заказами*.
4. На панели инструментов нажмите кнопку Object.
5. Поместите на диаграмму еще один объект.
6. Введите его имя — *Управляющий транзакциями*.

Назначение ответственностей объектам

1. Выделите сообщение 5: *Создать пустой заказ*. Выделяйте слова, а не стрелку.
2. Нажав комбинацию клавиш CTRL+D, удалите это сообщение.
3. Повторите шаги 1 и 2 для удаления сообщений 6 и 7:

- *Ввести номер заказа, заказчика и число заказываемых предметов*
- *Сохранить заказ*

4. Выделите связь между объектами *Форма деталей Заказа* и *Заказ №1234*
5. Нажав комбинацию клавиш CTRL+D, удалите эту связь
6. На панели инструментов нажмите кнопку Object Link (Связь объекта).
7. Нарисуйте связь между *Форма деталей Заказа* и *Управляющий Заказами*.
8. На панели инструментов нажмите кнопку Object Link (Связь объекта).
9. Нарисуйте связь между *Управляющий Заказами* и *Заказ №1234*
10. На панели инструментов нажмите кнопку Object Link (Связь объекта).
11. Нарисуйте связь между *Заказ №1234* и *Управляющий Транзакцией*.
12. На панели инструментов нажмите кнопку Object Link (Связь объекта).
13. Нарисуйте связь между *Управляющий Заказами* и *Управляющий Транзакцией*.
14. На панели инструментов нажмите кнопку Link Message (Сообщение связи).
15. Щелкните мышью на связи между объектами *Форма деталей Заказа* и *Управляющий Заказами*, чтобы ввести новое сообщение.
16. Назовите это сообщение Сохранить заказ.

17. Повторите шаги 14 — 16, добавив сообщения с шестого по девятое, и назвав их:

- *Создать новый заказ* — между *Управляющий Заказами* и *Заказ №1234* - *Ввести номер заказа, заказчика и число заказываемых предметов* — между *Управляющий Заказами* и *Заказ №1234* - *Сохранить заказ* — между *Управляющий Заказами* и *Управляющий Транзакцией* - *Информация о заказе* — между *Управляющий Транзакцией* и *Заказ №1234*

18. На панели инструментов нажмите кнопку Link to Self (Связь с собой).

19. Щелкнув на объекте *Управляющий Транзакцией*, добавьте к нему рефлексивное сообщение.

20. На панели инструментов нажмите кнопку Link Message (Сообщение связи).

21. Щелкните мышью на рефлексивной связи *Управляющий Транзакциями*, чтобы ввести туда сообщение.

22. Назовите новое *Сохранить информацию о заказе в базе данных*.

Соотнесение объектов с классами (если классы были созданы при разработке описанной выше диаграммы Последовательности)

1. Найдите в браузере класс *Выбор Заказа*.
2. Перетащите его на объект *Выбор варианта заказа* на диаграмме.
3. Повторите шаги 1 и 2 соотнеся остальные объекты и соответствующие им классы:

- Класс *заказ деталей* соотнесите с объектом *Форма деталей заказа* - Класс *Упр_заказами* — с объектом *Управляющий Заказами*
- Класс *Заказ* — с объектом *Заказ №1234*
- Класс *Упр_транзакциями* — с объектом *Управляющий транзакциями*

Соотнесение объектов с классами (если вы не создавали описанную выше диаграмму Последовательности)

1. Щелкните правой кнопкой мыши на объекте *Форма деталей Заказа*.
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
3. В раскрывающемся списке классов выберите пункт <New> (Создать). Появится окно спецификации классов.
4. В поле имени введите *Выбор заказа*.
5. Щелкните на кнопке OK. Вы вернетесь в окно спецификации объекта.
6. В списке классов выберите класс *Выбор заказа.*
7. Щелкните на кнопке OK, чтобы вернуться к диаграмме. Теперь объект называется *Выбор варианта заказа: Выбор Заказа*
8. Для соотнесения остальных объектов с классами повторите шаги с 1 по 7:

– Класс *Детали заказа* соотнесите с объектом *Форма деталей заказа*
– Класс *Упр_заказами* — с объектом *Управляющий заказами*
– Класс *Заказ* — с объектом *Заказ N 1234*
– Класс *Упр_ транзакциями* — с объектом *Управляющий транзакциями*

Соотнесение сообщений с операциями (если операции были созданы при разработке описанной выше диаграммы Последовательности)

1. Щелкните правой кнопкой мыши на сообщении 1: Создать новый заказ.
2. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
3. В раскрывающемся списке имен укажите имя операции — *Создать()*.
4. Нажмите на кнопку OK.
5. Повторите шаги 1—4 для соотнесения с операциями остальных сообщений:

– Сообщение 2: *Открыть форму* соотнесите с операцией *Открыть()*
– Сообщение 3: *Ввести номер заказа, заказчика и число заказываемых предметов* — с операцией *Ввести номер заказа, заказчика и число заказываемых предметов()*
– Сообщение 4: *Сохранить заказ* — с операцией *Сохранить заказ()*
– Сообщение 5: *Сохранить заказ* — с операцией Сохранить заказ()
– Сообщение 6: *Создать пустой заказ* – с операцией Создать пустой заказ()
– Сообщение 7: *Ввести номер заказа, заказчика и число заказоваемых предметов* с

одноименной операцией.

– Сообщение 8 Сохранить заказ – с операцией Сохранить заказ()
– Сообщение 9 *Информация о заказе* – с одноименной операцией
– Сообщение 10 *Сохранить информацию о заказе* с одноименной операцией

Соотнесение сообщений с операциями (если вы не создавали описанную выше диаграмму Последовательности)

1. Щелкните правой кнопкой мыши на сообщении 1: *Создать новый заказ()*.
2. В открывшемся меню выберите пункт <new operation> (создать операцию). Появится окно спецификации операции.
3. В поле имени введите имя операции —*Создать().*
4. Нажмите на кнопку ОК, чтобы закрыть окно спецификации операции и вернуться к диаграмме.
5. Еще раз щелкните правой кнопкой мыши на сообщении 1.
6. В открывшемся меню выберите пункт Open Specification (Открыть спецификацию).
7. В раскрывающемся списке Name (Имя) укажите имя новой операции.
8. Нажмите на кнопку ОК.
9. Повторите шаги 1—8, чтобы создать новые операции и соотнести с ними остальные сообщения:

– Сообщение 2: *Открыть форму* соотнесите с операцией *Открыть()*
– Сообщение 3: *Ввести номер заказа, заказчика и число заказываемых предметов* - с операцией *Ввести номер заказа, заказчика и число заказываемых предметов()*
– Сообщение 4: *Сохранить заказ* — с операцией *Сохранить заказ()*
– Сообщение 5: *Сохранить заказ* — с операцией *Сохранить заказ()*
– Сообщение 6: *Создать пустой заказ* – с операцией *Создать пустой заказ()*
– Сообщение 7: *Ввести номер заказа, заказчика и число заказоваемых предметов*- одноименной операцией
– Сообщение 8 Сохранить заказ – с операцией *Сохранить заказ()*
– Сообщение 9 *Информация о заказе* – с одноименной операцией
– Сообщение 10 *Сохранить информацию о заказе* с одноименной операцией Ваша диаграмма должна выглядеть, как показано на рис. 3

Контрольные вопросы

1. Какие элементы может содержать диаграмма последовательности и диаграмма кооперaций?
2. На каком этапе разработки программной системы разрабатывается диаграмма прецедентов взаимодействия? Каково ее назначение?
3. Сколько диаграмм взаимодействия необходимо разрабатывать для одной программной системы?

Лабораторная работа № 9 Тестирование интерфейса пользователя средствами инструментальной среды разработки

Цель работы состоит в приобретении навыков методики автоматизированного тестирования логики программы, формализованного описания результатов тестирования и применению стандартов по составлению схем программ

Краткие теоретические и учебно-методические материалы по теме практической работы

Автоматизированное тестирование программного обеспечения (**Software Automation Testing**) - это процесс верификации программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования.

Специалист по автоматизированному тестированию программного обеспечения (**Software Automation Tester**) - это технический специалист (тестировщик или разработчик программного обеспечения), обеспечивающий создание, отладку и поддержку работоспособного состояния тест скриптов, тестовых наборов и инструментов для автоматизированного тестирования.

Инструмент для автоматизированного тестирования (**Automation Test Tool**) - это программное обеспечение, по средствам которого специалист по автоматизированному тестированию осуществляет создание, отладку, выполнение и анализ результатов прогона тест скриптов.

Тест Скрипт (**Test Script**) - это набор инструкций, для автоматической проверки определенной части программного обеспечения.

Тестовый набор (**Test Suite**) - это комбинация тест скриптов, для проверки определенной части программного обеспечения, объединенной общей функциональностью или целями, преследуемыми запуском данного набора.

Тесты для запуска (**Test Run**) - это комбинация тест скриптов или тестовых наборов для последующего совместного запуска (последовательного или параллельного, в зависимости от преследуемых целей и возможностей инструмента для автоматизированного тестирования).

Для оценки качества ПО всегда применяется целый комплекс мер, среди которых тестирование ПО на предмет обнаружения ошибок - один из важнейших этапов.

1. Для его проведения необходимы объект тестирования - в данном случае ПО - и эталон, с которым этот объект сравнивается. Тестирование ПО проводится на соответствие заранее определенным требованиям (по функциональности, производительности, безопасности и пр.). Поскольку объект тестирования сложный, необходим системный подход к тестированию, его организации и проведению.

Требования к ПО подразделяются на функциональные (какие функции и с каким качеством должно реализовывать ПО) и нефункциональные (ограничения на время решения задачи, скорость доступа к данным, требования к занимаемым ресурсам и т. п.). У заказчика и разработчика должна быть возможность сравнить текущее функционирование системы с ее эталонным (ожидаемым) поведением. При этом рекомендуется использовать итеративный подход, так как раннее тестирование критичных областей значительно снижает риск неудачи и стоимость исправлений для всего проекта разработки ПО.

2. Эффективнее поэтапно осуществлять контроль над ходом отработки ПО. Рекомендуется использовать шаблоны документов, в том числе плана тестирования, разработанный в соответствии с требованиями международного стандарта IEEE 829-1983 Standard for Software Test Documentation.

Обсуждение технического задания, технического проекта, архитектуры системы с

заказчиком - это тоже часть процесса обнаружения ошибок и уточнения эталона.

Принято разделять тестирование по уровням задач и объектов на разных стадиях и этапах разработки ПО (см. таблицу):

- тестирование частей ПО (модулей, компонентов) с целью проверки правильности реализации алгоритмов -- выполняется разработчиками;
- функциональное тестирование подсистем и ПО в целом с целью проверки степени выполнения функциональных требований к ПО - рекомендуется проводить отдельной группой тестировщиков, не подчиненной руководителю разработки;
- нагрузочное тестирование (в том числе стрессовое) для выявления характеристик функционирования ПО при изменении нагрузки (интенсивности обращений к нему, наполнения базы данных и т. п.) - для выполнения этой работы требуются высококвалифицированные тестировщики и дорогостоящие средства автоматизации экспериментов.

Этапы тестирования

Вид тестирования	Стадия, этап	Объект	Критерий
Структурное, надежности	Разработка	Компоненты	Покрытие ветвлений, функции
Сборочное	Разработка	Подсистемы	Функциональность, степень проверки компонентов
Функциональное	Разработка	Система в целом	Соответствие функциональным требованиям ТЗ
Регрессионное	Разработка, сопровождение	Система в целом	Проверка качества внесения изменений
Нагрузочное	Разработка, сопровождение	Система в целом	Оценка статистических характеристик системы, соответствие ТЗ, ТТХ, подбор конфигурации оборудования
Стрессовое	Разработка, сопровождение	Система в целом	Корректность работы системы при предельных нагрузках

3. Для того чтобы увеличить объем проверок и повысить качество тестирования, обеспечить возможность повторного использования тестов при внесении изменений в ПО применяют средства автоматизации тестирования. К их числу относятся средства автоматизации функционального и нагрузочного тестирования клиент-серверных и Web-приложений: Rational TestStudio, Mercury LoadRunner, Segue SilkPerformer, а также менее популярные продукты фирм Compuware, CA, Empirix, RadView Software и др.

Тестированием надо заниматься не только постоянно, но и систематично. Если не забывать, что это процесс обнаружения ошибок, то стоит потребовать от разработчика, чтобы он периодически силами специально созданных групп проводил так называемые "review", или "структурные просмотры" проектных материалов и аудит исходных кодов программ. Заказчик вправе договориться с разработчиком о предъявлении подобных материалов или о не очень глубоком контроле хода такого процесса. Он заинтересован в том, чтобы в организации разработчика были поставлены процессы. В этом случае заказчик может быть уверен, что качество разрабатываемого ПО контролируется и обеспечивается в ходе разработки. Именно на это направлены известная модель технологической зрелости CMM (Capability Maturity Model, www.sei.cmu.edu/cmmi/orgdocs/conops.html) и стандарт ISO 15504.

Желательно, чтобы на этапах сборки, комплексной отладки и опытной эксплуатации разработчик фиксировал интенсивность обнаружения ошибок, тогда по характеру изменения этой интенсивности можно будет судить об изменении качества ПО и, например, о целесообразности его передачи в опытную или постоянную эксплуатацию. Наконец, необходимо проведение комплекса испытаний ПО на соответствие требованиям ТЗ или других нормативных документов, на возможность эффективно работать с ПО на основе использования программной документации, приемосдаточных и других видов испытаний, обеспечивающих заказчику уверенность в работоспособности созданного для

него ПО.

4.Между испытаниями, когда ПО еще только создается очень важно, чтобы деятельность по тестированию велась планомерно. Это значит, что на каждом этапе работ должны быть выбраны критерий качества и критерий завершения тестирования. Первый нужен для того, чтобы тестировщик или группа тестировщиков понимали, что и на соответствие чему они проверяют. То есть, каковы объект и эталон и какие свойства объекта проверяются. Второй критерий помогает принять решение в случае, когда исчерпываются ресурсы, отведенные на тестирование, он отвечает на вопрос, при каких условиях тестирование может быть завершено.

Для проверки сложного объекта можно и нужно применять разные стратегии, позволяющие добиваться максимального результата при существующих ограничениях на ресурсы тестирования.

План тестирования определяется международным стандартом IEEE 829-1983. В нем должны быть предусмотрены как минимум три раздела содержащие, следующие описания:

- что будет тестироваться (тестовые требования, тестовые варианты);
- какими методами и насколько подробно будет тестироваться система;
- план-график работ и требуемые ресурсы (персонал, техника).

Дополнительно описываются критерии удачного/неудачного завершения тестов, критерии окончания тестирования, риски, непредвиденные ситуации, приводятся ссылки на соответствующие разделы в основных документах проекта - план управления требованиями, план конфигурационного управления и т. п.

5.Как подготовиться к тестированию, что именно нужно для его проведения?

Необходимы проверяемые требования, затем каждое из них связывается с одним или несколькими тестовыми требованиями. Далее логический набор тестовых требований группируется в тестовые сценарии, проверка которых позволит дать односложный ответ на вопрос, корректно ли осуществляется ввод, правильно ли работает та или иная бизнес-функция в системе. Этот результат понятен не только специалистам, но и конечному пользователю. Основные объекты автоматизации тестирования - системы, реализующие работу клиентской части. Ключевой особенностью тестирования клиент-серверных систем является возможность проверки корректности функционирования и удовлетворительного быстродействия системы через работу клиентской части. Таким образом, тщательно и всесторонне проверив эти возможности, мы получаем гарантию работоспособности системы для конечного пользователя.

Еще один важный аспект организации работ - хранение созданных тестов. Рекомендуется относиться к тестам так же, как и к исходному коду, т. е. нужно использовать версионные хранилища для возможности восстановления тестов предыдущих версий системы (MS SourceSafe, Rational ClearCase,...). Это пригодится на этапе сопровождения ПО и даст возможность повторного использования готовых тестов на нескольких версиях системы. Аналогично нужно относиться и к тестовым данным, создавая архивы, копии и версии содержимого БД.

6.Тестирование - это всегда эксперимент. Для его проведения нужна база. Как в любом эксперименте, при тестировании нужно где-то собирать накопленную информацию, обрабатывать результаты. Есть самое крупное разделение видов тестирования: статическое и динамическое. При статическом тестировании ПО не исполняется, а происходит анализ кода, структур данных. Динамическое тестирование, напротив, требует выполнения тестируемого ПО. Для этого нужны не только средства автоматизации тестирования, но и вспомогательные средства. Известно очень много средства построения и автоматической генерации тестов, средств мониторинга ресурсов во время выполнения тестов, средств измерения и визуализации результатов тестирования, средств статистической обработки результатов и т. п.

Нагрузочное тестирование

Нагрузочное тестирование (Load Testing) или **тестирование производительности** (Performance Testing) - это **автоматизированное тестирование**, имитирующее работу определенного количества бизнес пользователей на каком либо общем (разделяемом ими) ресурсе. Современное программное обеспечение просто обязано бесперебойно работать под колоссальными нагрузками. Любого рода проблемы, связанные с плохой производительностью, могут стать причиной отказа клиентов от использования вашего ПО. В связи с этим, проведение качественного нагрузочного тестирования должно стать обязательным, для обеспечения стабильности работы ваших приложений.

Начиная работу в области нагрузочного тестирования, следует четко понимать, что это не просто запись и прогон (Record and Playback) скриптов, а более сложный процесс:

Во-первых, нагрузочное тестирование - это серьезная исследовательская и аналитическая работа

Во-вторых - это реальное автоматизированное тестирование, требующее серьезных навыков программирования, а также знания сетевых протоколов и различных серверов приложений и баз данных

В-третьих - существуют разные **виды нагрузочного тестирования,** ставящие перед собой разные цели

В качестве примера можно привести работу сотрудников современного банка, в котором все работают с одними и теми же программными приложениями, установленными на банковских серверах. Или использование программного приложения веб магазин, в данном случае посетителями, нагружающими сервера, будут пользователи интернета.

Моделирование нагрузки происходит с помощью специальных продуктов и техник.

Терминология:

Виртуальный пользователь *(Virtual User)* - программный процесс, циклически выполняющий моделируемые операции

Итерация *(Iteration)* – это один повтор выполняемой в цикле операции

Интенсивность выполнения операции *(Operation Intensity)* - частота выполнения операции в единицу времени, в тестовом скрипте задается интервалом времени между итерациями

Нагрузка *(Loading)* - совокупное выполнение операций на общем ресурсе (тр./сек, хитов/сек)

Производительность *(Performance*) - количество выполняемых операций за период времени (N операций за M часов)

Масштабируемость приложения *(Application Scalability)* - пропорциональный рост производительности при увеличении нагрузки

Профиль нагрузки *(Performance Profile)* - это набор операций с заданными интенсивностями, полученный на основе сбора статистических данных либо определенный путем анализа требований к тестируемой системе

Нагрузочной точкой называется рассчитанное (либо заданное Заказчиком) количество виртуальных пользователей в группах, выполняющих операции с определенными интенсивностями

Теперь рассмотрим как эти сущности связаны между собой. Выразив интенсивность через интервал времени между итерациями, видим что рост интенсивности выполняемых операций это сокращение интервала времени. Рост нагрузки пропорционален росту интенсивности. Естественно также, что при увеличении интенсивности растет производительность. При этом увеличивается степень использования (загруженности) ресурсов. С какого-то момента рост производительности прекращается (а нагрузка может продолжать расти), происходит насыщение и затем деградация системы. В дополнение можно заметить что при тестировании изменение интенсивности операций может подчиняться какому либо закону (например, Пуассона) либо быть равномерным в течении

всего теста.

Этапы проведения нагрузочного тестирования

- Анализ требований и сбор информации о тестируемой системе.
- Разработка модели нагрузки
- Выбор инструмента для нагрузочного тестирования
- Создание и отладка тестовых скриптов
- Проведение тестирования
- Анализ результатов
- Подготовка, отправка и публикация отчета по проведенному нагрузочному тестированию

Разработка модели нагрузки

Определившись с видами нагрузочного тестирования, целями и терминологией, у вас появился определенный фундамент, чтобы узнать, что основная задача в нагрузочном тестировании - разработка модели нагрузки.

Для решения этой задачи необходимо определить следующее:

- список тестируемых операций
- интенсивность выполнения операций
- зависимость изменения интенсивности выполнения операций от времени

В список таких задач должны войти операции, которые критичны с точки зрения бизнеса и с технической точки зрения. Критичностью с точки зрения бизнеса (и это основной критерий) является реальное влияние ухудшения производительности таких операций на бизнес процессы. Например, увеличение длительности обслуживания клиентов в банке, невозможность выполнить необходимое количество операций в течение дня и так далее. С технической точки зрения - это операции максимально потребляющие ресурсы серверов. Как правило - это операции выполняемые большим количество бизнес пользователей одновременно или создание сложных отчетов, в которые входят так называемые «тяжелые» запросы к базе данных. Хотим еще раз подчеркнуть, что степенью критичности является влияние на бизнес и работоспособность системы, например, создание отчета полностью загружающего сервер базы данных, в ночное время, не будет носить высокий приоритет для оптимизации, а в рабочие часы это будет иметь максимальный приоритет.

Изучение Приложения (тестируемое прикладное программное обеспечение)

Чтобы выделить части приложения, а именно операции, которые будут тестироваться, необходимо провести работу связанную с изучением приложения. Очень большую пользу при этом должны оказать разработчики приложения, если речь идет о тестировании в процессе разработки, либо бизнес пользователи и системные администраторы, если приложение находится в процессе эксплуатации. В ходе этой работы разумно сделать такие шаги:

Описать компоненты приложения и составить схемы взаимодействия между ними

Выделить критические с точки зрения предполагаемого тестирования операции. В качестве таковых могут быть выбраны:

- Операции с «тяжелыми» запросами к базе данных, процессы генерации отчетов
- Операции, выполняемые большим количеством пользователей или с высокой интенсивностью
- Операции критичные с точки зрения бизнеса, и к тому же удовлетворяющие условиям двух верхних пунктов

Еще раз хочется заметить, что опрос бизнес пользователей или совместное исследование с разработчиками и администраторами системы может значительно облегчить задачу. Если приложение находится в эксплуатации, то можно провести мониторинг загрузки компонентов аппаратных серверов (процессора, память, диски) и проанализировать системные журналы веб серверов (снять stats pack, если в качестве

сервера базы данных, например, используется Oracle). Системные журналы могут показать пики высокой активности пользователей в течение дня и дать количественное оценки того сколько транзакций (хитов) выполняется в единицу времени. Согласно закону Паретто или принципу 20/80, 20% операций приложения генерируют 80% нагрузки в системе, поэтому нужно стараться выбрать для моделирования именно эти 20% операций.

Определение профиля нагрузки

Ключевым моментом в модели нагрузки являются выбранные для тестирования операции или профиль нагрузки. Естественно выполняться эти операции в тесте должны одновременно. Профилей нагрузки для приложения может быть несколько и это оправдано. Ведь бизнес пользователи могут выполнять разные наборы операций в разное время. Например, начало операционного дня и конец дня, начало месяца (квартала) и соответственно завершение могут отличаться. Таким образом получаем различные наборы операций приложения, выполняющиеся одновременно и соответственно создающие различную нагрузку. Меняться могут не только сами операции но и их интенсивности. В первом приближении моделью нагрузки является набор профилей нагрузки, где каждый профиль отличается от другого или набором операций или интенсивностями выполнения этих операций.

Пример профиля нагрузки, в который входит 5 операций, значение n может быть различным для каждой операции:

<Профиль нагрузки>
Операция_1 - интенсивность выполнения n раз / ед. времени
Операция_2 - интенсивность выполнения n раз / ед. времени
Операция_3 - интенсивность выполнения n раз / ед. времени
Операция_4 - интенсивность выполнения n раз / ед. времени
Операция_5 - интенсивность выполнения n раз / ед. времени

Расчет нагрузочных точек

Поскольку в профиле нагрузки как правило присутствует несколько операций - это означает, что у нас будет несколько групп пользователей. Желательно моделировать каждую операцию отдельной группой виртуальных пользователей (хотя в жизни часто бывает наоборот, один бизнес пользователь может отвечать за выполнение нескольких операций). Тем не менее, если назначить одному виртуальному пользователю выполнение одной операции, то так легче выдержать определенную интенсивность (и соответственно производительность) для этой операции в тесте, чем в случае, когда виртуальному пользователю назначается последовательная цепочка операций. Зная интенсивность выполнения операции нужно определить количество виртуальных пользователей в группе, выполняющих эту операцию. Идеальный случай, когда работа с приложением аналогична работе заводского конвейера и есть точные оценки сколько операций в день делает один пользователь. Чаще всего бывает не так и известно только общее количество операций выполняемое в течение дня. Так же может оказаться, что интенсивность выполнения операции каждым пользователем очень низкая, например, один пользователь выполняет операцию раз в день или раз в два дня.

Автоматизированное тестирование использует программные средства для выполнения тестов и проверки результатов выполнения, что помогает сократить время тестирования и упростить его процесс.

Существует два основных подхода к автоматизации тестирования: тестирование на уровне кода и GUI-тестирование. К первому типу относится, в частности, модульное тестирование. Ко второму - имитация действий пользователя с помощью специальных тестовых «фреймворков».

Наиболее распространенной формой автоматизации является тестирование приложений через графический пользовательский интерфейс. Популярность такого вида тестирования объясняется двумя факторами: во-первых, приложение тестируется тем же способом, которым его будет использовать человек, во-вторых, можно тестировать

приложение, не имея при этом доступа к исходному коду.

Одной из главных проблем автоматизированного тестирования является его трудоемкость: несмотря на то, что оно позволяет устранить часть рутинных операций и ускорить выполнение тестов, большие ресурсы могут тратиться на обновление самих тестов. Это относится к обоим видам автоматизации. При рефакторинге часто бывает необходимо обновить и модульные тесты, и изменение кода тестов может занять столько же времени, сколько и изменение основного кода. С другой стороны, при изменении интерфейса приложения необходимо заново переписать все тесты, которые связаны с обновленными окнами, что при большом количестве тестов может отнять значительные ресурсы.

Для автоматизации тестирования существует большое количество приложений. Наиболее популярные из них (по итогам 2007 года):

- HP LoadRunner, HP QuickTest Professional, HP Quality Center
- Segue SilkPerformer
- IBM Rational FunctionalTester, IBM Rational PerformanceTester, IBM Rational TestStudio
- AutomatedQA TestComplete

Использование этих инструментов помогает тестировщикам автоматизировать следующие задачи:

- установка продукта
- создание тестовых данных
- GUI взаимодействие
- определение проблемы

Однако автоматические тесты не могут полностью заменить ручное тестирование. Автоматизация всех испытаний — очень дорогой процесс, и потому автоматическое тестирование является лишь дополнением ручного тестирования. Наилучший вариант использования автоматических тестов - регрессионное тестирование.

Сравнение ручного и автоматизированного тестирования

Результаты сравнения приведены в таблице 1. Сравнение показывает тенденцию современного тестирования, ориентирующую на максимальную автоматизацию процесса тестирования и генерацию тестового кода, что позволяет справляться с большими объемами данных и тестов, необходимых для обеспечения качества при производстве программных продуктов.

Таблица 1. Сравнение ручного и автоматизированного подхода

	Ручное	Автоматизированное
Задание входных значений	Гибкость в задании данных. Позволяет использовать разные значения на разных циклах прогона тестов, расширяя покрытие	Входные значения строго заданы
Проверка результата	Гибкая, позволяет тестировщику оценивать нечетко сформулированные критерии	Строгая. Нечетко сформулированные критерии могут быть проверены только путем сравнения с эталоном
Повторяемость	Низкая. Человеческий фактор и нечеткое определение данных приводят к неповторяемости тестирования	Высокая
Надежность	Низкая. Длительные тестовые	Высокая, не зависит от длины

	циклы приводят к снижению внимания тестировщика	тестового цикла
Чувствительность к незначительным изменениям в продукте	Зависит от детальности описания процедуры. Обычно тестировщик в состоянии выполнить тест, если внешний вид продукта и текст сообщений несколько изменились	Высокая. Незначительные изменения в интерфейсе часто ведут к коррекции эталонов
Скорость выполнения тестового набора	Низкая	Высокая
Возможность генерации тестов	Отсутствует. Низкая скорость выполнения обычно не позволяет исполнить сгенерированный набор тестов	Поддерживается

Тестовый отчет

Тестовый отчет обновляется после каждого цикла тестирования и должен содержать следующую информацию для каждого цикла:

1. Перечень функциональности в соответствии с пунктами требований, запланированный для тестирования на данном цикле, и реальные данные по нему.
2. Количество выполненных тестов – запланированное и реально исполненное.
3. Время, затраченное на тестирование каждой функции, и общее время тестирования.
4. Количество найденных дефектов.
5. Количество повторно открытых дефектов.
6. Отклонения от запланированной последовательности действий, если таковые имели место.
7. Выводы о необходимых корректировках в системе тестов, которые должны быть сделаны до следующего тестового цикла.

Оценка качества тестов

Тесты нуждаются в контроле качества так же, как и тестируемый продукт. Поскольку тесты для продукта являются своего рода эталоном его структурных и поведенческих характеристик, закономерен вопрос о том, насколько адекватен эталон. Для оценки качества тестов используются различные методы, наиболее популярные из которых кратко рассмотрены ниже.

Тестовые метрики

Существует устоявшийся набор тестовых метрик, который помогает определить эффективность тестирования и текущее состояние продукта. К таким метрикам относятся следующие:

1. Покрытие функциональных требований.
2. Покрытие кода продукта. Наиболее применимо для модульного уровня тестирования.
3. Покрытие множества сценариев.
4. Количество или плотность найденных дефектов. Текущее количество дефектов сравнивается со средним для данного типа продуктов с целью установить, находится ли оно в пределах допустимого статистического отклонения. При этом обнаруженные отклонения как в большую, так и в меньшую сторону приводят к анализу причин их появления и, если необходимо, к выработке корректирующих действий.
5. Соотношение количества найденных дефектов с количеством тестов на данную функцию продукта. Сильное расхождение этих двух величин говорит либо о неэффективности тестов (когда большое количество тестов находит мало дефектов) либо о плохом качестве данного участка кода (когда найдено большое количество дефектов на не очень большом количестве тестов).

6. Количество найденных дефектов, соотнесенное по времени, или скорость поиска дефектов. Если производная такой функции близка к нулю, то продукт обладает качеством, достаточным для окончания тестирования и поставки заказчику.

Обзоры тестов и стратегии

Тестовый код и стратегия тестирования, зафиксированные в виде документов, заметно улучшаются, если подвергаются коллективному обсуждению. Такие обсуждения называются обзорами (review). Существует принятая в организации процедура проведения и оценки результатов обзора. Обзоры наряду с тестированием образуют мощный набор методов борьбы с ошибками с целью повышения качества продукта. Цели обзоров тестовой стратегии и тестового кода различны.

Цели обзора тестовой стратегии:

1. Установить достаточность проверок, обеспечиваемых тестированием.
2. Проанализировать оптимальность покрытия или адекватность распределения количества планируемых тестов по функциональности продукта.
3. Проанализировать оптимальность подхода к разработке кода, генерации кода, автоматизации тестирования.

Задания для практического занятия:

- Протестировать разработанный программный продукт (по вариантам).
- Написать отчет.

Вопросы для закрепления теоретического материала к практическому занятию:

1. Что такое автоматизированное тестирование?
2. Чем отличается автоматизированное тестирование от «ручного»?
3. Тестовый отчет. Состав тестового отчета.
4. Два основных подхода к автоматизации тестирования.
5. Метрики. Виды метрик.

Порядок выполнения отчета по практической работе

1. Сохранить работу на диске.
2. Ответить на контрольные вопросы.

Лабораторная работа № 10. Разработка тестовых модулей проекта для тестирования отдельных модулей

Цель работы: состоит в приобретении навыков методов тестирования логики программы, формализованного описания результатов тестирования и применении стандартов по составлению схем программ

Краткие теоретические и учебно-методические материалы по теме практической работы

Тестирование методом «черного ящика»

Тестирование программного обеспечения включает в себя целый комплекс действий, аналогичных последовательности процессов разработки программного обеспечения. В него входят:

- постановка задачи для теста;
- проектирование теста;
- написание тестов;
- тестирование тестов;
- выполнение тестов;
- изучение результатов тестирования.

Один из подходов проектирования тестов состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей либо спецификаций сопряжения модуля с другими модулями, программа при этом рассматривается как «черный ящик». Смысл теста заключается в том, чтобы проверить, соответствует ли программа внешним спецификациям. При этом содержание модуля не имеет значения. Такой подход получил название - **стратегия «черного ящика».**

Реализация тестирования методом **«черного ящика»** сводится к проверке всех возможных комбинаций входных данных. Невозможно протестировать программу, подавая на вход бесконечное множество значений, поэтому ограничиваются определенным набором данных. При этом исходят из максимальной отдачи теста по сравнению с затратами на его создание. Она измеряется вероятностью того, что тест выявит ошибки, если они имеются и программе. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста.

Существуют следующие критерии черного ящика:

- тестирование функций;
- тестирование классов входных данных;
- тестирование классов выходных данных;
- тестирование области допустимых значений (тестирование границ класса);
- тестирование длины набора данных;
- тестирование упорядоченности набора данных.

Критерий **тестирования функций** актуален для многофункциональных программ. Он требует подобрать такой набор тестов, чтобы был выполнен хотя бы один тест для каждой из функций, реализуемых программой.

Критерий **тестирования классов входных данных** требует классифицировать входные данные, разделить их на классы таким образом, чтобы все данные из одного класса были равнозначны с точки зрения проверки правильности программы. Считается, что если программа работает правильно на одном наборе входных данных из этого класса, то она будет правильно работать на любом другом наборе данных из этого же класса. Критерий требует выполнения хотя бы одного теста для каждого класса входных данных.

Критерий **тестирования классов выходных данных** выглядит аналогично предыдущему критерию, только проверяются не входные данные, а выходные.

Часто эти три критерия хорошо согласуются друг с другом. При применении

одного из них остальные будут удовлетворены автоматически. Если программа реализует несколько функций, то вполне естественно, что каждой из этих функций будет соответствовать свой класс входных и свой класс выходных данных. Часто существует соответствие между классами входных и выходных данных.

Рассмотрим программу для учета кадров предприятия.

Она будет иметь следующие функции:

- принять на работу,
- уволить с работы,
- перевести с одной должности на другую,
- выдать кадровую сводку.
- Классы входных данных:
- приказ о приеме,
- приказ об увольнении,
- приказ о переводе,
- заявка на кадровую сводку.

Классы выходных данных:

- запись о приеме,
- запись об увольнении,
- запись о переводе,
- кадровая сводка.

Этот пример хорошо демонстрирует соответствие между функциями, классами входных и выходных данных.

Тестирование области допустимых значений (тестирование границ класса). Если область допустимых значений переменной представляет собой простое перечисление (например, ноты, цвет, пол, диагноз и т. п.), надо проверить, что программа правильно понимает все эти значения и не принимает вместо них никаких иных значений. Например, как программа отреагирует на попытку ввести несуществующую ноту или пол.

Если класс допустимых значений представляет собой числовой диапазон, то понадобится более серьезная проверка. В этом случае выделяются:

- нормальные условия (в середине класса);
- граничные (экстремальные) условия;
- исключительные условия (выход за границу класса).

Тестирование длины набора данных можно считать частным случаем тестирования области допустимых значений. В данном случае речь пойдет о допустимом количестве элементов в наборе. Если программа последовательно обрабатывает элементы некоторого набора данных, имеет смысл проверить следующие ситуации:

- пустой набор (не содержит ни одного элемента);
- единичный набор (состоит из одного-единственного элемента);
- слишком короткий набор (если предусмотрена минимально допустимая длина)
- набор минимально возможной длины (если такая предусмотрена);
- нормальный набор (состоит из нескольких элементов);
- набор из нескольких частей (если такое возможно.
- набор максимально возможной длины (если такая предусмотрена);
- слишком длинный набор (с длиной больше максимально допустимой).

Тестирование упорядоченности входных данных важно для задач сортировки и поиска экстремумов. В этом случае имеет смысл проверить следующие ситуации (классы входных данных):

- данные не упорядочены;
- данные упорядочены в прямом порядке;

- данные упорядочены в обратном порядке;
- в наборе имеются повторяющиеся значения;
- экстремальное значение находится в середине набора;
- экстремальное значение находится в начале набора;
- экстремальное значение находится в конце набора;
- в наборе несколько совпадающих экстремальных значений.

Порядок работы над программой

1. Тестирование начинают с критериев черного ящика. Соответствующие тесты составляются *до* написания текста программы. Тесты заносятся в таблицу тестов, в которой кроме входных данных и ожидаемых результатов предусмотрена графа для реальных результатов и отметки о совпадении их с ожидаемыми.
2. Пишется текст программы.
3. Проверяется текст программы, исходя из списка ошибкоопасных конструкций и ситуаций.
4. Составляется таблица МГТ (минимально грубое тестирование).
5. Прогоняются тесты, составленные исходя из критериев черного ящика. В процессе прогона строятся трассировочные таблицы.
6. При необходимости корректируется программа. Проверяются места корректировок на ошибкоопасность.
7. После корректировки программы проводится повторное тестирование: повторяются заново все ранее прогнанные тесты.
8. Заполняются последние графы таблицы тестов и таблицу МГТ.
9. Если таблица МГТ еще неполна, добавляются тесты для покрытия незаполненных строк таблицы МГТ. Заносятся в таблицу тестов.
10. Тесты прогоняются через программу. Строятся трассировки.
11. При необходимости корректируется программа. Проверяются места корректировок на ошибкоопасность.
12. Заполняются последние графы таблицы тестов и таблица МГТ.
13. После корректировки программы проводится повторное тестирование: повторяются заново все ранее прогнанные тесты.

Задания для практического занятия:

1. Выполнить упражнения
2. Описать стратегию тестирования методом «Черного ящика»
3. Описать порядок работы над программой
4. Выписать порядок тестирования классов входных и выходных данных, областей допустимых значений, длины последовательности, упорядоченности набора данных, которые должны быть проверены тестами для выбранного метода тестирования.
5. Протестировать программу. Результаты оформить в виде таблиц (см. методическое пособие).
6. Проверить все виды тестов и сделать выводы об их эффективности.
7. Оформить отчет. Сдать и защитить работу.

Упражнения

Пусть требуется написать программу, которая считает среднее арифметическое последовательности целых чисел, заканчивающейся нулем.

Пусть программа выглядит следующим образом:

```
Program p;
var a, b, c: integer;
begin
   repeat
      read(a);
      b:= b + 1;
      c:= c + a
   until a=0;
   writeln(c/b)
end.
```

Перед началом тестирования надо сформулировать цели тестирования, выбрать критерии полноты. Зафиксируем в качестве цели: провести тестирование, полное с точки зрения критериев черного ящика и критерия МГТ (минимально грубого тестирования).

Разработка тестов должна предшествовать написанию программы и первоначально должна быть направлена на уяснение поставленной задачи. Начинают с критериев черного ящика.

Первый критерий - тестирование функций - в данном случае неинтересен, поскольку программа однофункциональная. Эту единственную функцию и будем постоянно тестировать.

Второе - тестирование классов входных и выходных данных, областей допустимых значений, длины последовательности, упорядоченности набора данных. Выделение классов входных и выходных данных в этой задаче не очевидно. Областей допустимых значений - тоже. Упорядоченность здесь не важна.

В этой задаче важна проверка длины последовательности. Поэтому проверяем в такой последовательности:

1. Пустой набор (не содержит ни одного элемента).

Пустая последовательность - это последовательность, состоящая из одного нуля. Нуль элементом последовательности не является. Количество элементов в такой последовательности считается равным нулю.

Входные данные - 0.

Выходные данные. Что должна выдавать программа в ответ на пустую последовательность? В случае пустой последовательности результатом должна быть фраза «Последовательность пуста».

Вот теперь мы можем сформулировать первый тест.

Вход: 0.

Ожидаемый выход: Последовательность пуста.

2. Единичный набор (состоит из одного-единственного элемента).

Вход: 4, 0.

Ожидаемый выход: 4.

3. Слишком короткий набор.

Для нас такого понятия нет.

4. Набор минимально возможной длины.

Для нас такого понятия нет.

5. Нормальный набор (состоит из нескольких элементов).

Для определенности возьмем набор из 3 элементов.

Вход: 1, 3, 4, 0.

Среднее арифметическое выдается как вещественное число с той точностью, которую обеспечит ваш компьютер. Для определенности, пусть будет 6 знаков после запятой. Итак:

Вход: 1, 3, 4, 0.

Ожидаемый выход: 2.666667.

6. Набор из нескольких частей (если такое возможно).

В данном случае это невозможно.

7. Набор максимально возможной длины (если такая предусмотрена).

Не предусмотрена.

8. Слишком длинный набор (с длиной больше максимально допустимой).

В данном случае это невозможно.

Итоги:

1.Построено 3 теста.

2.Уточнены несколько важных деталей (что такое пустая последовательность, как на нее реагировать, какого типа должен быть результат, с какой точностью он должен выдаваться). Тесты покрывают классы выходных данных («Последовательность пуста» и число, являющееся средним арифметическим) и классы входных (3 вида последовательностей, различающихся длиной). Запишем 3 наших теста в таблицу следующего формата:

Тест	Вход	Ожидаемый результат	Результат «сухой прокрутки»	+/-	Результат выполнения на компьютере	+/-
T1	0	Последовательность пуста				
T2	4, 0	4				
T3	1,3,4,0	2,666667				

3.Заполним только первые строки и в каждой из них - первые графы. Строки, возможно, придется добавить, исходя из критериев белого ящика и анализа ошибкоопасных мест. Две средние графы заполним по результатам безмашинного тестирования, две последние - по результатам выполнения на компьютере. В графах «+/-» будем отмечать совпадение или несовпадение реального результата с ожидаемым.

4.Переходим к составлению текста программы. Среднее арифметическое - это частное от деления суммы элементов на их количество. Значит надо ввести элементы, просуммировать их, посчитать их количество, после чего напечатать частное. Например, так:

Программа на псевдокоде	Примечание

begin
выдать начальное приветствие;
ввести 1-й элемент (tek);
while последовательность не кончена do begin
увеличить сумму (sum);
увеличить количество (kol)
end;
вывести среднее арифметическое (sum/kol)
end.
	var tek: integer;
	
	var sum: integer;
	var kol: integer;

Упомянутые в проекте фрагменты раскроем следующим образом:

```
Ввести 1-й элемент
  writeln('Введите, пожалуйста, 1-й элемент');
  read(tek);

Последовательность не кончена
  tek<>0

Увеличить сумму
  sum:= sum + tek;

Увеличить количество
  kol:= kol + 1;
```

Первый вариант программы будет выглядеть так:

```
program SrednArifm;
var tek, kol, sum: integer;
begin
   writeln('Я считаю среднее арифметическое' +
      'последовательности чисел, кончающейся нулем');
```

```
  writeln('Введите, пожалуйста, 1-й элемент');
  read(tek);
  while tek<>0 {последовательность не кончена} do begin
    sum:= sum + tek;  {увеличить сумму элементов}
    kol:= kol + 1;    {увеличить кол-во элементов}
  end;
  if kol=0 then writeln('Последовательность пуста')
       else writeln('Среднее арифметическое=',sum/kol)
end.
```

В данном примере показана функциональность программы, учтены моменты для удобства пользователя: минимальный интерфейс.

Для упрощения чтения текста программы могут использоваться осмысленные имена переменных, абзацные отступы (запись лесенкой), комментарии.

5.Строим таблицу минимально грубого тестирования. В этой программе всего две развилки: цикл с предусловием и ветвление. В обоих случаях используются простые условия. Значит, МГТ-таблица будет содержать всего 5 строк:

			T1	T2	T3
Y1	while tek <>0	=0			
		=1			
		>1			
Y2	if col=0	+			
		-			

Имеются 3 теста построенные, исходя из критериев черного ящика.

Проверим, как они будут работать в данной программе и как отразятся в таблице МГТ (минимально грубое тестирование). Для этого вручную выполним первый тест и заполним трассировочную таблицу. Вход первого теста состоит из единственного числа - нуль. Оно и будет введено как значение переменной tek. Занесем его в трассировочную таблицу:

tek	kol	sum
0		

Тело цикла while не выполняется ни разу. Сразу переходим на условный оператор. Равна ли переменная kol нулю? Моделью оперативной памяти в данном случае является трассировочная таблица. Переменную kol необходимо инициализировать. kol — это количество элементов последовательности. Значит, изначально, пока никакой последовательности еще нет, kol должна быть равна нулю. Вставим соответствующий оператор перед вводом первого элемента последовательности. Тело программы приобретет вид:

```
begin
  writeln('Я считаю …');
  kol:= 0;
  writeln('Введите, пожалуйста, 1-й элемент');
  read(tek);
  while tek<>0 do begin
    sum:= sum + tek;
    kol:= kol + 1;
  end;
  if kol=0 then writeln('Последовательность пуста')
       else writeln('Среднее арифметическое = ', sum/kol)
end.
```

Повторим первый тест. Теперь его удастся выполнить до конца. Трассировочная таблица будет иметь вид:

tek	kol	sum
	0	
0		

В таблице тестов отметим результат выполнения первого теста:

Тест	**Вход**	**Ожидаемый результат**	**Результат «сухой**	**+/-**	**Результат выполнения**	**+/-**

			прокрутки»		**на компьютере**	
T1	0	Последовательность пуста		+		
T2	4, 0	4				
T3	1,3,4,0	2,666667				

В таблице МГТ отметим проверенные строки:

			T1	T2	T3
Y1	while tek <>0	=0	+		
		=1			
		>1			
Y2	if col=0	+	+		
		-			

Перейдем ко второму тесту. Выполняем программу и заполняем трассировочную таблицу.

tek	kol	sum
	0	
4		

После входа в цикл обнаруживаем, что переменная sum не имеет начального значения. Ее, так же как и переменную kol, необходимо инициализировать нулем перед входом в цикл. Получаем следующий текст тела программы:

```
begin
  writeln('Я считаю …');
  kol:= 0;
  sum:= 0;
  writeln('Введите, пожалуйста, 1-й элемент');
  read(tek);
  while tek<>0 do begin
    sum:= sum + tek;
    kol:= kol + 1;
  end;
  if kol=0 then writeln('Последовательность пуста')
          else writeln('Среднее арифметическое=',sum/kol)
end.
```

Результаты выполнения второго теста.

tek	kol	sum
	0	
		0
4		
		4
	1	
		8
	2	
		12
	3	

Воспользуемся перечнем ошибкоопасных ситуаций. Возникли проблемы с циклом: произошло зацикливание. Чтобы его избежать, в теле цикла, как минимум, должна меняться хотя бы одна переменная, входящая в условие цикла. Такая переменная всего одна, и она в теле цикла не меняется. Необходимо ввести остальные элементы последовательности. Тело программы приобретет вид:

```
begin
  writeln('Я считаю …');
  kol:= 0;
  sum:= 0;
  writeln('Введите, пожалуйста, 1-й элемент');
  read(tek);
  while tek<>0 do begin
    sum:= sum + tek;
    kol:= kol + 1;
    writeln('Введите, пожалуйста, элемент № ',kol);
    read(tek);
  end;
  if kol=0 then writeln('Последовательность пуста')
    else writeln('Среднее арифметическое = ',sum/kol)
end.
```

Выполняем еще раз второй тест и доходим до конца.

tek	kol	sum
	0	
		0
4		
		4
	1	
0		

По ходу выполнения выясняется, что допущена еще одна небольшая ошибка в интерфейсе. Программа второй раз попросила ввести элемент № 1. Приглашение в теле цикла должно выглядеть так:

```
writeln('Введите, пожалуйста, элемент № ',kol+1);
```

Внесем в текст соответствующие изменения, на результатах выполнения это не скажется.

Сделаем отметку в таблице тестов.

Тест	Вход	Ожидаемый результат	Результат «сухой прокрутки»	+/–	Результат выполнения на компьютере	+/–
T1	0	Последователь-ность пуста	Последователь-ность пуста	+		
T2	4, 0	4	4	+		
T3	1, 3, 4, 0	2.666667				

Заполним соответствующий столбец в таблице МГТ

			T1	T2	T3
У1	while tek<>0	=0	+		
		=1		+	
		>1			
У2	if kol=0	+	+		
		–		+	

Так как в программу были внесены исправления, необходимо повторить все ранее прогнанные тесты.

При его повторе результаты получим прежние, хотя трассировочная таблица несколько изменится:

tek	kol	sum
	0	
		0
0		

Выполним тест 3. Строим трассировку.

tek	kol	sum
	0	
		0
1		
		1
	1	
3		
		4
	2	
4		
		8
	3	
0		

Заполняем таблицы тестов и МГТ.

Тест	Вход	Ожидаемый результат	Результат «сухой прокрутки»	+/−	Результат выполнения на компьютере	+/−
T1	0	Последователь-ность пуста	Последователь-ность пуста	+		
T2	4, 0	4	4	+		
T3	1, 3, 4, 0	2.666667	2.666667	+		

			T1	T2	T3
У1	while tek<>0	=0	+		
		=1		+	
		>1			+
У2	if kol=0	+	+		
		-		+	+

Третий тест оказался неудачен - никакой ошибки не выявил. Но все строки таблицы МГТ уже заполнены. Значит, был построен и выполнен прогон набора тестов, полный и с точки зрения критериев черного ящика, и с точки зрения минимально грубого тестирования.

Ручное тестирование выполнено.

Теперь можно выходить на компьютер и прогнать еще раз те же тесты и заполнить крайние правые графы в таблице тестов:

Тест	Вход	Ожидаемый результат	Результат «сухой прокрутки»	+/-	Результат выполнения на компьютере	+/-
T1	0	Последовательность пуста	Последовательность пуста	+	Последовательность пуста	+
T2	4, 0	4	4	+	4	+
T3	1,3,4,0	2,666667	2,666667	+	2,666667	+

В итоге сделаем 3 замечания:

1. Тестов, построенных исходя из критериев черного ящика, оказалось достаточно, чтобы удовлетворить критерию МГТ.
2. При этом тесты эти (построенные в тот момент, когда текста программы еще не существовало) оказались настолько хороши, что первые 2 из них обнаружили в нашей программе 3 ошибки. И только последний третий тест оказался неудачен, ошибок не выявил.
3. Легко заметить, что все допущенные нами ошибки упоминались в перечне ошибкоопасных ситуаций. Значит, если бы вместо того, чтобы сразу же начинать тестовые прогоны, сначала проанализировали программу с точки зрения этого перечня, ошибки удалось бы обнаружить гораздо быстрей и с меньшими усилиями.

Контрольные вопросы:

1. Какие виды ошибок существуют?
2. Что такое тест? Какими свойствами должен обладать тест?

3. Дайте краткую характеристику методики тестирования «черным ящиком».
4. Перечислите свойства тестов.

Лабораторная работа № 11. Выполнение функционального тестирования

Цель работы состоит в приобретении навыков тестирования логики программы, формализованного описания результатов тестирования и применению стандартов по составлению схем программ

Краткие теоретические и учебно-методические материалы по теме практической работы

Тестирование программного обеспечения включает в себя целый комплекс действий, аналогичных последовательности процессов разработки программного обеспечения. В него входят:

- постановка задачи для теста;
- проектирование теста;
- написание тестов;
- тестирование тестов;
- выполнение тестов;
- изучение результатов тестирования.

Наиболее важным является проектирование тестов. Существуют разные подходы к проектированию тестов.

Первый состоит в том, что тесты проектируются на основе внешних спецификаций программ и модулей либо спецификаций сопряжения модуля с другими модулями, программа при этом рассматривается как «черный ящик». Смысл теста заключается в том, чтобы проверить, соответствует ли программа внешним спецификациям. При этом содержание модуля не имеет значения. Такой подход получил название - **стратегия «черного ящика».**

Второй подход - **стратегия «белого ящика»,** основан на анализе логики программы. При таком подходе тестирование заключается в проверке каждого пути, каждой ветви алгоритма. При этом внешняя спецификация во внимание не принимается.

Ни один из этих подходов не является оптимальным. Реализация тестирования методом **«черного ящика»** сводится к проверке всех возможных комбинаций входных данных. Невозможно протестировать программу, подавая на вход бесконечное множество значений, поэтому ограничиваются определенным набором данных. При этом исходят из максимальной отдачи теста по сравнению с затратами на его создание. Она измеряется вероятностью того, что тест выявит ошибки, если они имеются и программе. Затраты измеряются временем и стоимостью подготовки, выполнения и проверки результатов теста.

Тестирование методом **«белого ящика»** также не даст 100%-ной гарантии того, что модуль не содержит ошибок. Даже если предположить, что выполнены тесты для всех ветвей алгоритма, нельзя с полной уверенностью утверждать, что программа соответствует ее спецификациям. Например, если требовалось написать программу для вычисления кубического корня, а программа фактически вычисляет корень квадратный, то реализации будет совершенно неправильной, даже если проверить все пути. Вторая проблема — отсутствующие пути. Если программа реализует спецификации не полностью (например, отсутствует такая специализированная функция, как проверка на отрицательное значение входных данных программы вычисления квадратного корня), никакое тестирование существующих путей не выявит такой ошибки. И наконец, проблема зависимости результатом тестирования от входных данных. Одни данные будут давать правильные результаты, а другие нет. Например, если для определения равенства трех чисел программируется выражение вида:

IF (A + B+ C)/3 = A,

то оно будет верным не для всех значений *A, B* и *C* (ошибка возникает в том случае, когда из двух значений *B* или *C* одно больше, а другое на столько же меньше *A)*. Если концентрировать внимание только на тестировании путей, нет гарантии, что эта ошибка будет выявлена.

Таким образом, полное тестирование программы невозможно, т. е. никакое тестирование не гарантирует полное отсутствие ошибок в программе. Поэтому необходимо проектировать тесты таким образом, чтобы увеличить вероятность обнаружения ошибки в программе.

Стратегия «белого ящика»

Существуют следующие методы тестирования по принципу «белого ящика»:

- покрытие операторов;
- покрытие решений;
- покрытие условий;
- покрытие решений/условий;
- комбинаторное покрытие условий.

Метод покрытия операторов

Целью этого метода тестирования является выполнение каждого оператора программы хотя бы один раз.

Пример.

Если для тестирования задать значения переменных A = 2, B=0, X= 3, будет реализован путь *ace,* т. е. каждый оператор программы выполнится один раз (рис.1, *а).* Но если внести в алгоритм ошибки - заменить в первом условии ***and*** на ***or,*** а во втором *X*> 1 на *X*< 1 (рис.1, *б),* ни одна ошибка не будет обнаружена (табл.1). Кроме того, путь *abd* вообще не будет охвачен тестом, и если в нем есть ошибка, она также не будет обнаружена. В табл.1 ожидаемый результат определяется по блок-схеме на рис.1, *а,* а фактический - по рис.1, *б.*

Как видно из этой таблицы, ни одна из внесенных в алгоритм ошибок не будет обнаружена.

Таблица 1 - Результат тестирования методом покрытия операторов

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A = 2, B = 0, X=3	X=2,5	X=2,5	Неуспешно

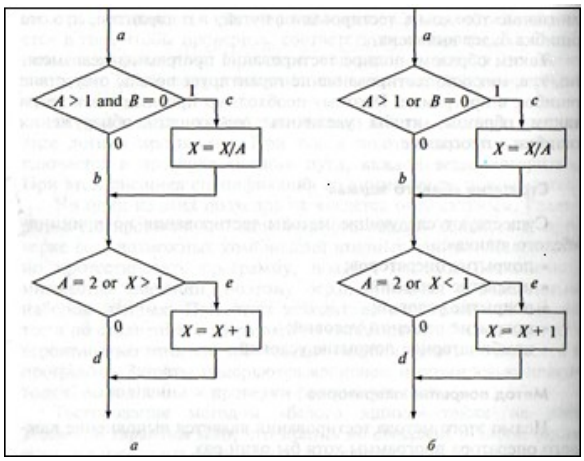


Рис. 1. Пример алгоритма программы: *а* - правильный; *б* - с ошибкой

Метод покрытия решений (покрытия переходов)

Согласно методу покрытия решений каждое направление перехода должно быть реализовано, по крайней мере, один раз. Этот метод включает в себя критерий покрытия операторов, так как при выполнении всех направлений переходов выполнятся все операторы, находящиеся на этих направлениях.

Для программы, приведенной на рис.1, покрытие решений может быть выполнено

двумя тестами, покрывающими пути *{ace, abd},* либо *{acd, abe}.* Для этого выберем следующие исходные данные: *{A = 3, B*=0, *X*= 3} - в первом случае и {*A=2, B =1, X= 1*} - во втором. Однако путь, где *X* не меняется, будет проверен с вероятностью 50%: если во втором условии вместо условия *X*> 1 записано *X*< 1, то ошибка не будет обнаружена двумя тестами.

Результаты тестирования приведены в табл.2.

Таблица 2 - Результат тестирования методом покрытия решений

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A=3,\ B=0,\ X=3$	$X=1$	$X=1$	Неуспешно
$A=2,\ B=1,\ X=1$	$X=2$	$X=1{,}5$	Успешно

Метод покрытия условий

Этот метод может дать лучшие результаты по сравнению с предыдущими. В соответствии с методом покрытия условий записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись, по крайней мере, один раз.

В рассматриваемом примере имеем четыре условия: *{A > 1, B = 0}, {A* = 2, *X*> 1}. Следовательно, требуется достаточное число тестов, такое, чтобы реализовать ситуации, где *A> 1, A≤ 1, B=0* и B≠ 0 в точке *a* и *A* = 2, *A* ≠ 2, *X*> 1 и *X*≤ 1 в точке *b.* Тесты, удовлетворяющие

критерию покрытия условий (табл.3), и соответствующие им пути:

а) *A = 2, B=0, X=4 ace;*

б)A = 1, *B= 1, X=0 abd.*

Таблица 3 - Результаты тестирования методом покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A=2,\ B=0,\ X=4$	$X=3$	$X=3$	Неуспешно
$A=1,\ B=1,\ X=0$	$X=0$	$X=1$	Успешно

Метод покрытия решений (покрытия переходов)

Критерий покрытия решений/условий требует такого достаточного набора тестов, чтобы все возможные результаты каждого условия выполнялись по крайней мере один раз, все результаты каждого решения выполнялись по крайней мере один раз и, кроме того, каждой точке входа передавалось управление по крайней мере один раз.

Недостатки метода:

- не всегда можно проверить все условия;
- невозможно проверить условия, которые скрыты другими условиями;
- недостаточная чувствительность к ошибкам в логических выражениях.

Так, в рассматриваемом примере два теста метода покрытия условий

а) *A* = 2, *B* =0, *X =4 ace;*

б) *A = 1, B =1, X =0 abd*

отвечают и критерию покрытия решений/условий. Это является следствием того, что одни условия приведенных решений скрывают другие условия в этих решениях. Так, если условие *A* > 1 будет ложным, транслятор может не проверять условия *B* = 0, поскольку при любом результате условия *B*=0 результат решения *((A* > 1)&(*B*=0)) примет значение *ложь.* То есть в варианте на рис.1 не все результаты всех условий выполнятся в процессе тестирования.

Рассмотрим реализацию того же примера на рис.2. Наиболее полное покрытие тестами в этом случае осуществляется

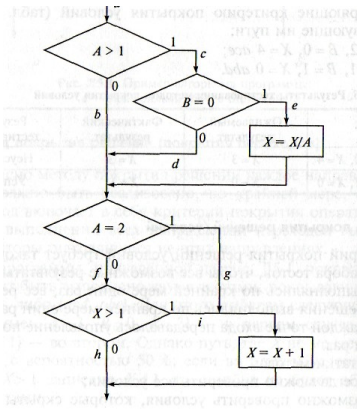


Рис.2. Пример алгоритма программы

так, чтобы выполнялись все возможные результаты каждого простого решения. Для этого нужно покрыть пути *aceg* (тест *A = 2, B=0, X*= 4), *acdfh* (тест *A* = 3, *B* = 1, *X*= 0), *abfh* (тест *A = 0,B=0, X*= 0), *abfi* (тест *A* = 0, *B*= 0, *X*= 2).

Протестировав алгоритм на рис.2, нетрудно убедиться в том, что критерии покрытия условий и критерии покрытия решений/условий недостаточно чувствительны к ошибкам в логических выражениях.

Метод комбинаторного покрытия условий

Критерий комбинаторного покрытия условий удовлетворяет также и критериям покрытия решений, покрытия условий и покрытия решений/условий.

Этот метод требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении выполнялись по крайней мере один раз. По этому критерию в рассматриваемом примере должны быть покрыты тестами следующие восемь комбинаций:

1. A>1, B=0
2. A > 1, B≠0
3. A ≤1, B=0
4. A ≤ 1, B≠0
5. A = 2, X>1
6. A = 2, X≤1
7. A≠2, X>1
8. A ≠2,X≤1

Для того чтобы протестировать эти комбинации, необязательно использовать все 8 тестов. Фактически они могут быть покрыты четырьмя тестами (табл.4):

- *A* = 2, B=0, *X*= 4 {покрывает 1, 5};
- *A* = 2, B = 1, *X*= 1 {покрывает 2, 6};
- *A* = 0,5, B= 0, *X*= *2* {покрывает 3, 7};
- *A* = 1, B = 0, *X*= 1 {покрывает 4, 8}.

Таблица 4 - Результаты тестирования методом комбинаторного покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
A = 2, *B* = 0, *X* = 4	*X* = 3	*X* = 3	Неуспешно
A = 2, *B* = 1, *X* = 1	*X* = 2	*X* = 1,5	Успешно
A = 0,5, *B* = 0, *X* = 2	*X* = 3	*X* = 4	Успешно
A = 1, *B* = 0, *X* = 1	*X* = 1	*X* = 1	Неуспешно

Задания для практического занятия:

1. Выполнить упражнения.
2.Оформить отчет по лабораторной работе.
3. Сдать и защитить работу.

Упражнения

Задание 1.

Первый из критериев белого ящика - критерий покрытия операторов. Он требует подобрать такой набор тестов, чтобы каждый оператор в программе был выполнен хотя

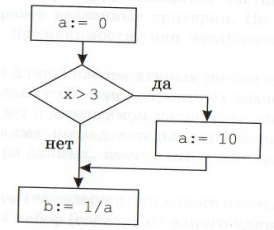
бы один раз. В качестве примера рассмотрим следующий фрагмент Паскаль-программы:

Пример 1

```
a:= 0;
if x>3 then a:= 10;
b:= 1/a;
```

Для того чтобы удовлетворить критерию покрытия операторов, достаточно одного выполнения. Такого, чтобы *x* был больше 3. Очевидно, что ошибка в программе этим тестом обнаружена, не будет. Она проявится как раз в том случае, когда $x \le 3$. Но такого теста критерий покрытия операторов от нас не требует.

Итак, мы имеем программу, оттестированную с точки зрения критерия покрытия операторов и при этом содержащую ошибку. Попробуем разобраться, в чем дело. Для наглядности перейдем с Паскаля на язык блок-схем.



Теперь причина видна сразу. Следуя критерию покрытия операторов, мы проверили только положительную ветвь раз вилки, но не затронули отрицательную. Сокращенная форма условного оператора в Паскале этому весьма способствует.

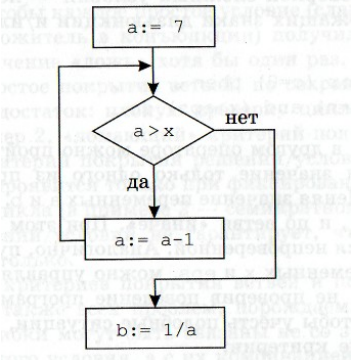
Чтобы избавиться от указанного недостатка, введем второй критерий белого ящика - **критерий покрытия ветвей** (иначе его называют критерием покрытия решений). Он требует подобрать такой набор тестов, чтобы каждая ветвь в программе была выполнена хотя бы один раз. Тестирование с точки зрения этого критерия обнаружит ошибку в предыдущем примере.

Рассмотрим другой пример. На Паскале он будет выглядеть так:

Пример 2

```
a:= 7;
while a>x do a:= a-1;
b:= 1/a;
```

Мы уже знаем, что паскалевская запись может служить провокатором ошибок. Поэтому сразу составим блок-схему:



Для того чтобы удовлетворить критерию покрытия ветвей, в данном случае достаточно одного теста. Например, такого, чтобы *x* был равен 6 или 5. Все ветви программы будут пройдены (при $x=5$ одна из ветвей - тело цикла - даже 2 раза). Но ошибка в программе обнаружена так и не будет! Она проявится в одном-единственном случае, когда $x = 0$. Но такого теста от нас критерий покрытия ветвей не потребовал.

Итак, мы имеем программу, оттестированную с точки зрения критерия покрытия ветвей и при этом содержащую ошибку. Причина в том, что некоторые ветви в программе могут быть пройдены несколько раз, и результат выполнения зависит от количества

проходов. Для того чтобы учесть этот факт, введем третий критерий белого ящика - критерий **покрытия путей.** Он требует подобрать такой набор тестов, чтобы каждый путь в программе был выполнен хотя бы один раз. Тестирование с точки зрения этого критерия обнаружило бы ошибку в примере 2. Но из этого же примера виден принципиальный недостаток данного критерия. Сколько всего путей возможно в примере 2? Бесконечно много! Проверить их все невозможно. Значит, как только в программе появляются циклы с пред- или постусловием или цикл со счетчиком, но с вычисляемыми границами, количество путей в программе становится потенциально бесконечным, и критерий покрытия путей становится неприменимым. Необходим какой-то компромиссный критерий. Более жесткий, чем покрытие ветвей, но менее жесткий, чем покрытие путей. О нем мы поговорим далее.

Кроме проблем с проверкой циклов существенные проблемы связаны с проверкой сложных условий - логических выражений, содержащих знаки дизъюнкции и/или конъюнкции. Например:

```
if (a<b) or (c=0) then …
while (i<=n) and (x>eps) do …
```

И в том, и в другом операторе можно пройти по обеим ветвям, изменяя значение только одного из простых условий. Пусть $c \neq 0$. Меняя значение переменных a и b, можно пройти и по ветви «то», и по ветви «иначе». При этом ситуация, когда $c = 0$, останется непроверенной. Аналогично, пусть $i \leq n$. Меняя значения переменных x и eps, можно управлять выполнением цикла *while* не проверив поведение программы при $i > n$.

Для того чтобы учесть подобные ситуации, были предложены следующие критерии:

- критерий покрытия условий;
- критерий покрытия решений/условий;
- критерий комбинаторного покрытия условий.

Критерий **покрытия условий** требует подобрать такой набор тестов, чтобы каждое простое условие (слагаемое в дизъюнкции и сомножитель в конъюнкции) получило и значение «истина», и значение «ложь» хотя бы один раз.

Критерий пытается «в лоб» исправить вышеуказанный недостаток в тестировании сложных условий. Однако сам оказывается весьма слаб. Дело в том, что выполнение критерия покрытия условий не гарантирует покрытие ветвей. Пусть сложное условие представляет собой дизъюнкцию двух слагаемых. Например,

```
if (a<b) or (c=0) then d:= 1 else d:= 1/c;
```

При первом выполнении первое слагаемое истинно, второе ложно, вся дизъюнкция в целом истинна. При втором выполнении первое слагаемое ложно, второе истинно, вся дизъюнкция в целом истинна. Критерий покрытия условий выполнен, критерий покрытия ветвей – нет. Ошибка в программе не обнаружена.

Аналогичная ситуация возможна для конъюнкции. Например,

```
if (a<b) and (c=0) then d:= 1/c else d:= 1;
```

Чтобы исправить этот недостаток, критерии покрытия ветвей (решений) и условий объединяют в единый критерий **покрытия решений/условий.** Он требует подобрать такой набор тестов, чтобы каждая ветвь в программе была пройдена хотя бы один раз и чтобы каждое простое условие (слагаемое в дизъюнкции и сомножитель в конъюнкции) получило и значение «истина», и значение «ложь» хотя бы один раз. Критерий надежнее, чем простое покрытие ветвей, но сохраняет его принципиальный недостаток: плохую проверку циклов. Приведенный выше пример 2, «ломающий» критерий покрытия ветвей, «сломает» и критерий покрытия решений/условий. Ошибка в данном случае проявится только при фиксированном количестве повторений цикла (в примере 2 - семикратном), а критерий покрытия решений/условий не гарантирует, что повторений будет именно столько.

Совмещение критериев покрытия ветвей и покрытия условий не решает также всех

проблем, порождаемых сложными условиями. Ошибки могут быть связаны не со значением того или иного простого условия, а с их комбинацией. Например, в фрагменте:

Пример 3

```
if (a=0) or (b=0) or (c=0)
  then d:= 1/(a+b)
  else d:= 1;
```

ошибка будет выявлена только при одновременном равенстве нулю двух переменных: *a* и *b*.

Для решения этой проблемы был предложен критерий **комбинаторного покрытия условий,** который требует подобрать такой набор тестов, чтобы хотя бы один раз выполнялась любая комбинация простых условий. Критерий значительно более надежен, чем покрытие решений/условий, но обладает двумя существенными недостатками. Во-первых, он может потребовать очень большого числа тестов. Количество тестов, необходимых для проверки комбинации *n* простых условий, равно 2". Комбинация двух условий потребует четырех тестов, трех условий - восьми, четырех условий - шестнадцати тестов и т. д. Во-вторых, даже комбинаторное покрытие условий не гарантирует надежную проверку циклов. Тот же самый пример 2, который демонстрировал недостатки покрытия ветвей и покрытия решений/условий, покажет и недостаток комбинаторного покрытия условий.

Критерии белого ящика требуют знания текста программы. Текста программы, анализирующей треугольники, у нас нет. Тем не менее предлагается еще раз вернуться к сформированному ранее набору тестов. Не возникнет ли желания его модифицировать?

Критерий покрытия решений/условий не гарантирует проверки такой ситуации.

Задание 2.

1. Спроектировать тесты по принципу «белого ящика» Выбрать несколько алгоритмов для тестирования и обозначить буквами или цифрами ветви этих алгоритмов.
2. Выписать пути алгоритма, которые должны быть проверены тестами для выбранного метода тестирования.
3. Записать тесты, которые позволят пройти по путям алгоритма.
4. Протестировать разработанную вами программу. Результаты оформить в виде таблиц (см. табл.1 - 4).
5. Проверить все виды тестов и сделать выводы об их эффективности.

Контрольные вопросы:

1. Охарактеризуйте этап реализации и тестирования программного продукта.
2. Какие существуют виды тестирования?
3. Назовите критерии выбора тестов.
4. Перечислите свойства тестов.
5. Приведите критерии надежности программ.
6. В чем заключается оценка надежности программ?

Лабораторная работа № 12. Тестирование интеграции

Цель. Научиться создавать инсталляционные файлы; выполнять оценочное тестирование программного продукта.

Оборудование. ПК

Ход работы

1. Ознакомиться с теоретической частью.
2. Выполнить практическое задание.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Теоретическая часть

После завершения комплексного тестирования приступают к оценочному тестированию, целью которого является тестирование программы на соответствие основным требованиям. Эта стадия тестирования особенно важна для программных продуктов, предназначенных для продажи на рынке.

Оценочное тестирование, которое также называют «тестированием системы в целом», включает следующие виды:

- *тестирование удобства использования* - последовательная проверка соответствия программного продукта и документации на него основным положениям технического задания;
- *тестирование на предельных объемах* - проверка работоспособности программы на максимально больших объемах данных, например, объемах текстов, таблиц, большом количестве файлов и т. п.;
- *тестирование на предельных нагрузках* - проверка выполнения программы на возможность обработки большого объема данных, поступивших в течение короткого времени;
- *тестирование удобства эксплуатации* - анализ психологических факторов, возникающих при работе с программным обеспечением; это тестирование позволяет определить, удобен ли интерфейс, не раздражает ли цветовое или звуковое сопровождение и т. п.;
- *тестирование защиты* - проверка защиты, например, от несанкционированного доступа к информации;
- *тестирование производительности* - определение пропускной способности при заданной конфигурации и нагрузке;
- *тестирование требований к памяти* - определение реальных потребностей в оперативной и внешней памяти;
- *тестирование конфигурации оборудования* - проверка работоспособности программного обеспечения на разном оборудовании;
- *тестирование совместимости* - проверка преемственности версий: в тех случаях, если очередная версия системы меняет форматы данных, она должна предусматривать специальные конвекторы, обеспечивающие возможность работы с файлами, созданными предыдущей версией системы;
- *тестирование удобства установки* - проверка удобства установки;
- *тестирование надежности* - проверка надежности с использованием соответствующих математических моделей;
- *тестирование восстановления* - проверка восстановления программного обеспечения, например системы, включающей базу данных, после сбоев оборудования и программы;
- *тестирование удобства обслуживания* - проверка средств обслуживания, включенных в программное обеспечение;
- *тестирование документации* - тщательная проверка документации, например, если документация содержит примеры, то их все необходимо попробовать;

• *тестирование процедуры* - проверка ручных процессов, предполагаемых в системе и др.

Естественно, целью всех этих проверок является поиск несоответствий техническому заданию. Считают, что только после выполнения всех видов тестирования программный продукт может быть представлен пользователю или к реализации. Однако на практике обычно выполняют не все виды оценочного тестирования, так как это очень дорого и трудоемко. Как правило, для каждого типа программного обеспечения выполняют те виды тестирования, которые являются для него наиболее важными. Так базы данных обязательно тестируют на предельных объемах, а системы реального времени - на предельных нагрузках.

Системы для создания инсталляторов

Практика разработки коммерческого программного обеспечения показывает, что далеко не все пользователи умеют работать с архивами. Поэтому программы рекомендуется поставлять в виде исполняемых файлов, которые автоматически создают необходимые папки в файловой системе, копируют туда файлы программы, создают необходимые файлы настроек или ключи в реестре, а так же пункты меню запуска программы и ярлыки на рабочем столе. Для упрощения создания инсталляторов существует много специализированных программных продуктов.

Знакомство пользователя с программой чаще всего начинается с запуска инсталлятора. Внешний вид («упаковка») и функциональность продукта определяется разработчиком. Пользователю нужно иметь возможность проконтролировать процесс, выставив нужные параметры установки. Для разработчика же важно, чтобы, как минимум, его программа была установлена корректно, а инсталлятор был совместим с необходимыми платформами.

Решений для создания инсталляторов достаточно много. Чаще всего используется подсистема Windows Installer, которая уже входит в инструментарий операционной системы. Но существуют и альтернативные решения – как платные, так и бесплатные, различной функциональности. Зачастую с их помощью можно создавать пакеты с инсталлятором, не зависящим от Windows Installer.

Основные критерии выбора системы создания инсталлятора следующие:

• среда разработки, интерфейс, поддержка сценариев;

• работа с проектом, типы создаваемых пакетов, возможности импорта проектов из других сред разработки;

• пользовательские опции инсталлятора: поддержка языков, профилей и другие опции;

• поддержка расширений.

Свободные программы для создания инсталляторов:

• NSIS (Nullsoft Scriptable Install System) – один из самых популярных инсталляторов. Обладает богатыми возможностями, которые присутствуют в большинстве коммерческих продуктов. Позволяет устанавливать различные параметры сжатия при создании дистрибутива;

• IzPack – java инсталлятор. Это универсальный инсталлятор, способен создавать дистрибутивы для Unix, Linux, FreeBSD, Mac OS X и Windows 2000, XP. Позволяет создавать как обычные пакеты инсталляции, так и Web инсталляторы, которые подгружают необходимые файлы по мере необходимости. Данная возможность позволяет свести к минимуму количество загружаемых файлов в зависимости от требуемой конфигурации установки;

• Inno Setup –довольно популярный простой инталлятор. Содержит встроенный скриптовой язык;

• WiX (Windows Installer XML) – специализированный продукт от Microsoft для создания MSI и MSM инсталляционных пакетов.

Коммерческие программы для создания инсталляторов:

• InstallShield – один из самых известных продуктов в ряду инсталляторов;

• WISE – простой в освоении с богатыми возможностями генератор инсталляторов;

• VISE - профессиональный инсталлятор для Windows, MacOS X и Macintosh;

• CreateInstall это универсальный, гибкий и мощный инсталлятор как для профессиональных разработчиков, так и для начинающих. С помощью этой программы Вы можете создать полнофункциональные инсталляционные программы для Ваших приложений, а также самораспаковывающиеся архивы с высокой степенью сжатия и многое другое;

• Advanced Installer – позволяет создавать инсталляторы для java приложений. Создает дополнительный исполняемый файл.

CreateInstall

Домашняя страница: http://www.createinstall.ru/

CreateInstall – инструментарий для создания установщиков. В его основу заложено две особенности – контроль над процессом установки и неограниченная расширяемость. Обе возможности реализованы благодаря языку программирования Gentee, применяемому для написания сценариев.

Интерфейс CreateInstall разбит на 3 вкладки – «Проект», «Скрипт установки» и «Скрипт деинсталляции». Первый раздел позволяет задать общие настройки инсталлятора: информация о продукте, поддерживаемые языки, пути, внешний вид. Дополнительно, инсталлятор можно защитить цифровой подписью и установить пароль.

«Проект» – не равноценная замена двух последующих разделов, т. е. для создания дистрибутива нужно тщательно настроить скрипты установки и деинсталляции. Соответствующие параметры отображаются в виде групп, можно отобразить их единым списком.

Дополнением для CreateInstall служит утилита Quick CreateInstall (рисунок 1). Она значительно упрощает создание инсталлятора, предоставляя только базовые настройки проекта. Из Quick CreateInstall в дальнейшем проект можно импортировать в CreateInstall.

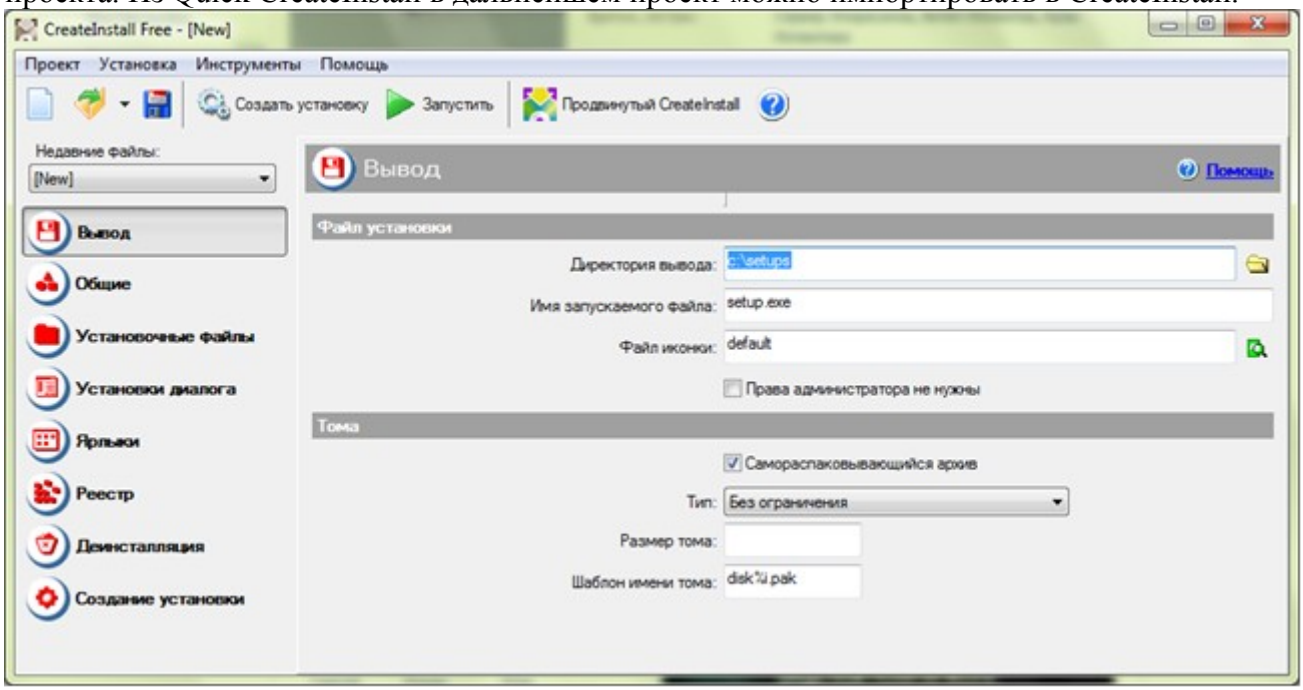


Рисунок 1 – Окно Quick CreateInstall

Код проекта не предназначен для самостоятельного редактирования, переноса в IDE-среду, экспорта. Хотя язык Gentee имеет отличный потенциал: как минимум, это переменные и функции, условные выражения и синтаксис, базирующийся на C, C++ и Java.

Существует 3 редакции программы – полная, light (простая) и бесплатная.

Интерфейс и справка доступны на русском языке.

Advanced Installer

Advanced Installer основывается на технологии Windows Inslaller, позволяя создавать msi-, exe- и других видов дистрибутивов. Этому способствует продуманный интерфейс и работа с проектами. В Advanced Installer можно обнаружить немало возможностей, которых нет в других подобных комплексах.

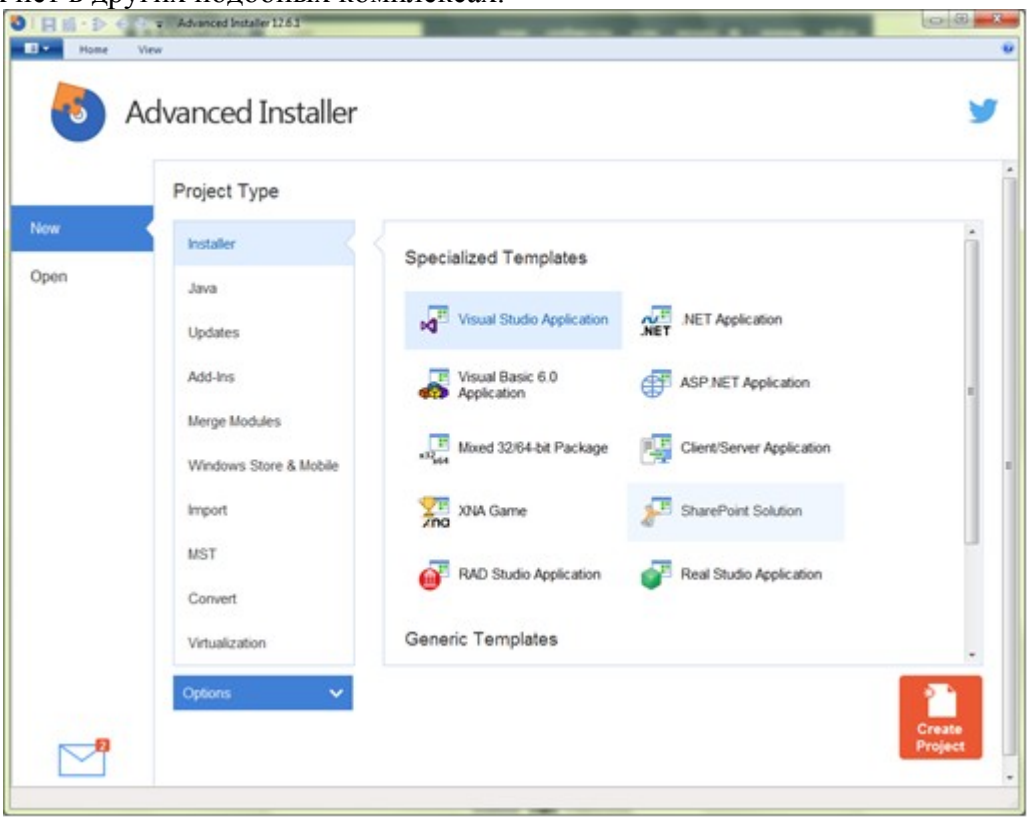


Рисунок 2 – Окно Advanced Installer

Примечательно, прежде всего, разнообразие проектов: сюда входят инсталляторы, Java-установщики, обновления, дополнения, модули слияния и другие. В разделе меню Installer собраны команды импорта проектов из Visual Studio, RAD Studio, Real Studio, Visual Basic. Здесь раскрывается потенциал Advanced Installer во взаимодействии с IDE-средами.

Для каждого из выбранных типов проекта предусмотрен детальный мастер настройки. Есть общие шаблоны – Simple, Enterprise, Architect или Professional. Большая часть проектов доступна только для определенных типов лицензии, общедоступные проекты обозначены как None в графе License Required.

Как уже сказано, при создании проекта можно воспользоваться пошаговым мастером, где, в частности, доступен выбор способа распространения пакета, языков локализации, настройка пользовательского интерфейса, ввод текста лицензии и другие опции. Advanced Installer позволяет выбрать вариант распространения программы – оставить данные без компрессии, разделить на CAB-архивы, сохранить в MSI и др., добавить цифровую подпись, потребовать ввод серийного номера и т. д.

Главное окно Advanced Installer (редактор проекта), в простом режиме отображения (Simple), содержит несколько секций:

• Product Information (Информация о продукте) – ввод сведений о продукте, параметры установки.

• Requirements (Требования) – указание аппаратных и системных требований, зависимостей ПО. Также имеется возможность создания пользовательских условий.

• Resources (Ресурсы) – редактор ресурсов (файлов и ключей реестра).

• Deployment (Развертывание) – выбор типа распространения продукта. Это может быть MSI, EXE или веб-инсталлятор. Для MSI, EXE ресурсы можно поместить отдельно от инсталлятора.

• System Changes – переменные среды.

При выборе ресурсов могут использоваться файлы, ключи реестра, переменные окружения, конфигурационные ini, драйверы, базы данных и переводы. С помощью модулей объединения можно добавить и другие ресурсы, такие как сервисы, разрешения, ассоциации и др.

Для выполнения более сложных задач позволяется использовать пользовательские действия, EXE, DLL или скрипты, написанные на C, C++, VBS или JS. Для создания сценариев предусмотрен удобный редактор.

Однако следует отметить, что в режиме Simple доступна лишь малая часть разделов. Работая с Advanced Installer в ознакомительном режиме, есть смысл зайти в настройки и переключиться в другой режим работы с проектом. После этих действий становятся доступны новые подразделы редактора.

Практическая часть

1. С помощью системы создания инсталляторов создайте из программы, созданной на лабораторной работе № 6, установочный файл.
2. Выполните тестирование удобства установки.
3. Выполните тестирование конфигурации оборудования.
4. Выполните тестирование восстановления.
5. Выполните тестирование удобства эксплуатации при помощи соседа.
6. Результаты выполнения практического задания запишите в отчет.

Контрольные вопросы

1. Что является целью тестирования программ?
2. Какие подходы к тестированию вы знаете? В чем они заключаются?
3. Обоснуйте необходимость создания инсталляторов программ.

4. Информационное обеспечение обучения

Перечень рекомендуемых учебных изданий, Интернет-ресурсов, дополнительной литературы

Основные источники:

1. Орлов В.В. Технологии разработки программных продуктов. – СПб.: Питер, 2013. – 437 с.
2. Рудаков А.В. Технология разработки программных продуктов: учебник /8-е изд., стер. – М.: Издательский центр «Академия», 2013. – 208 с.
3. Федорова Г.Н. Участие в интеграции программных модулей: учеб. пособие для студ. учреждений сред. проф. образования/Г.Н. Федорова. – М. :Издательский центр «Академия», 2016. – 304 с.

Дополнительные источники:

1. Гагарина, Л. Г. Технология разработки программного обеспечения: учеб. пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул; Под ред. Л. Г. Гагариной. - М.: ФОРУМ: ИНФРА-М, 2017.-400 с.
2. Калянов Г.Н. CASE – технологии: Консалтинг в автоматизации бизнес-процессов/ Г.Н. Калянов. - М.: Горячая линия-Телеком, 2012.

Электронные издания (электронные ресурсы)

1. От модели объектов - к модели классов. Единое окно доступа к образовательным ресурсам. http://real.tepkom.ru/Real_OM-CM_A.asp
2. Образовательный портал: http\\www.edu. ru
3. Интрернет университет информационных технологий - http://www.intuit.ru