Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет
им. Г.И. Носова»
Многопрофильный колледж

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ И ЛАБОРАТОРНЫХ РАБОТ

по ПМ.02 ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ
МДК.02.02 Инструментальные средства разработки программного обеспечения

для студентов специальности
09.02.07 Информационные
системы и программирование
КВАЛИФИКАЦИЯ ПРОГРАММИСТ

Магнитогорск, 2018

ОДОБРЕНО:
Предметно-цикловой комиссией
«Информатика и вычислительная техника»
Председатель *И.Г.Зорина*
Протокол № 6 от 21.02.2018г.

Методической комиссией МпК

Протокол №4 от «01» марта 2018г

Составитель (и):
преподаватель ФГБОУ ВО «МГТУ им. Г.И. Носова», МпК
к.т.н., доцент В.Д. Тутарова

Методические указания по выполнению практических и лабораторных работ разработаны на основе рабочей программы *ПМ.02 ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ, МДК.02.02 ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.*

Содержание практических и лабораторных работ ориентировано на формирование общих и профессиональных компетенций по программе подготовки специалистов среднего звена по специальности 09.02.07 Информационные системы и программирование.

СОДЕРЖАНИЕ

1 ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Состав и содержание практических и лабораторных занятий направлены на реализацию Федерального государственного образовательного стандарта среднего профессионального образования.

Ведущей дидактической целью практических занятий является формирование профессиональных практических умений (умений выполнять определенные действия, операции, необходимые в последующем в профессиональной деятельности).

Ведущей дидактической целью лабораторных занятий является экспериментальное подтверждение и проверка существенных теоретических положений (законов, зависимостей).

В соответствии с рабочей программой ПМ.02 ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ, МДК.02.02 Инструментальные средства разработки программного обеспечения, предусмотрено проведение практических и лабораторных занятий. В рамках практического/лабораторного занятия обучающиеся могут выполнять одну или несколько практических/лабораторных работ.

В результате их выполнения, обучающийся должен:

уметь:

- У1 использовать выбранную систему контроля версий.
- У2 использовать методы для получения кода с заданной функциональностью и степенью качества.
- У3. анализировать проектную и техническую документацию.
- У4. использовать специализированные графические средства построения и анализа архитектуры программных продуктов.
- У5. организовывать заданную интеграцию модулей в программные средства на базе имеющейся архитектуры и автоматизации бизнес-процессов.
- У6. определять источники и приемники данных.
- У7. использовать приемы работы в системах контроля версий.
- У8. выполнять отладку, используя методы и инструменты условной компиляции (классы Debug и Trace)).
- У9. оценивать размер минимального набора тестов.
- У10. разрабатывать тестовые пакеты и тестовые сценарии.
- У11. выявлять ошибки в системных компонентах на основе спецификаций.
- У12 использовать различные транспортные протоколы и стандарты форматирования сообщений.
- У13 выполнять тестирование интеграции.
- У14 организовывать постобработку данных.
- У15 создавать классы- исключения на основе базовых классов.
- У16 выполнять ручное и автоматизированное тестирование программного модуля.
- У17 использовать инструментальные средства отладки программных продуктов.

- У01.1 распознавать задачу и/или проблему в профессиональном и/или социальном контексте
- У01.2 анализировать задачу и/или проблему и выделять её составные части
- У01.3 определять этапы решения задачи
- У01.4 выявлять и эффективно искать информацию, необходимую для решения задачи и/или проблемы
- У 02.1 определять задачи поиска информации;

- У 02.2 определять необходимые источники информации;
- У 02.3 планировать процесс поиска;
- У 02.4 структурировать получаемую информацию;
- У 02.5 выделять наиболее значимое в перечне информации;
- У 02.6 оценивать практическую значимость результатов поиска;
- У 02.7 оформлять результаты поиска
- У03.2 применять современную научную профессиональную терминологию
- У03.3 определять и выстраивать траектории профессионального развития и самообразования
- У03.4 применять исследовательские приемы и навыки, чтобы быть в курсе последних отраслевых решений
- У03.5 понимать и адаптироваться к изменяющимся потребностям смежных профессий
- У 04.1 организовывать работу коллектива и команды;
- У 04.2 взаимодействовать с коллегами, руководством, клиентами в ходе профессиональной деятельности
- У04.3 понимать требования и оправдывать ожидания клиентов/работодателя
- У04.4 реагировать на запросы клиентов/руководства лично и опосредованно
- У04.5 использовать коммуникационные навыки при работе в команде для успешной работы над групповым решением проблем
- У04.8 эффективно работать в команде
- У05.1 применять техники и приемы эффективного общения в профессиональной деятельности
- У05.2 использовать навыки устного общения в профессиональной деятельности
- У05.3 излагать свои мысли и оформлять документы по профессиональной тематике на государственном языке
- У06.2 описывать значимость своей специальности для развития экономики и среды жизнедеятельности граждан российского государства
- У07.1 соблюдать нормы экологической безопасности;
- У08.3 пользоваться средствами профилактики перенапряжения характерными для данной специальности.
- У 09.1 применять средства информационных технологий для решения профессиональных задач;
- У 09.2 использовать современное программное обеспечение
- У09.3 проявлять культуру информационной безопасности при использовании информационно-коммуникационных технологий
- У10.1 понимать общий смысл четко произнесенных высказываний на известные темы (профессиональные и бытовые);
- У10.2 участвовать в диалогах на знакомые общие и профессиональные темы
- У10.6 понимать тексты на базовые профессиональные темы
- У10.7 читать, понимать и находить необходимые технические данные и инструкции в руководствах в любом доступном формате
- У11.2 выявлять достоинства и недостатки коммерческой идеи
- У11.3 презентовать идеи открытия собственного дела в профессиональной деятельности

Содержание практических и лабораторных занятий ориентировано на формирование общих компетенций по профессиональному модулю программы подготовки специалистов среднего звена по специальности и овладению ***профессиональными компетенциями***:

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.

А также формированию ***общих 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

Выполнение обучающимися практических и лабораторных работ по ПМ.02 ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ, МДК.02.02 Инструментальные средства разработки программного обеспечения, направлено на:

- *обобщение, систематизацию, углубление, закрепление, развитие и детализацию полученных теоретических знаний по конкретным темам учебной дисциплины;*

- *формирование умений применять полученные знания на практике, реализацию единства интеллектуальной и практической деятельности;*

- *выработку при решении поставленных задач профессионально значимых качеств, таких как самостоятельность, ответственность, точность, творческая инициатива*.

Практические и лабораторные занятия проводятся после соответствующей темы, которая обеспечивает наличие знаний, необходимых для ее выполнения.

2 ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ И ЛАБОРАТОРНЫХ ЗАНЯТИЙ

МДК 02.02 ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Разделы/темы	Темы практических/лабораторных занятий	Количество часов	Требования ФГОС СПО (уметь)
Раздел 2. **Средства разработки программного обеспечения**			
Тема 2.2.1. Современные технологии и инструменты интеграции	Практическая работа № 1 «Разработка структуры проекта»	**1**	У3-У7, У1, У12, У14 У02.1 - У02.7, У03.2-У03.5, У05.1-У05.3, У10.6, У01.1-У01.5, У09.1-У09.3
	Практическая работа №2 «Разработка модульной структуры проекта (диаграммы модулей)»	**1**	
	Практическая работа №3 «Разработка перечня артефактов и протоколов проекта»	**1**	
	Лабораторная работа №1 «Настройка работы системы контроля версий (типов импортируемых файлов, путей, фильтров и др. параметров импорта в репозиторий)»	**1**	
	Лабораторная работа №2 «Разработка и интеграция модулей проекта (командная работа)»	**2**	
	Лабораторная работа №3 «Отладка отдельных модулей программного проекта»	**2**	
	Лабораторная работа №4 «Организация обработки исключений»	**2**	
Тема 2.2.2 Инструментарий тестирования и анализа качества программных средств	Лабораторная работа №5 «Применение отладочных классов в проекте»	**1**	У2, У3, У6, У8-У11, У13-У17 У02.1 - У02.7, У03.2-У03.5, У05.1-У05.3, У10.6, У01.1-У01.5, У09.1-У09.3
	Лабораторная работа №6 «Отладка проекта»	**1**	
	Лабораторная работа №7 «Инспекция кода модулей проекта»	**1**	
	Лабораторная работа №8 «Тестирование интерфейса пользователя средствами инструментальной среды разработки»	**1**	
	Лабораторная работа №9 «Разработка тестовых модулей проекта для тестирования отдельных модулей»	**1**	
	Практическая работа №4 «Выполнение функционального тестирования»	**1**	

Разделы/темы	Темы практических/лабораторных занятий	Количество часов	Требования ФГОС СПО (уметь)
	Практическая работа №5 «Тестирование интеграции»	**1**	
	Практическая работа №6 «Документирование результатов тестирования»	**1**	
ИТОГО		**18**	

3 МЕТОДИЧЕСКИЕ УКАЗАНИЯ
МДК 02.02 ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Тема «Современные технологии и инструменты интеграции»

Практическая работа № 1
«Разработка структуры проекта»

Цель: создание структуры проекта и заполнение базовой информации о проекте**.**

Выполнив работу, Вы будете:

уметь:

- У3. анализировать проектную и техническую документацию.
- У4. использовать специализированные графические средства построения и анализа архитектуры программных продуктов.

Материальное обеспечение:

Персональный компьютер, пакет Microsoft Office, доступ к Internet ресурсам, программа MS Project.

Задание:

1. Создание нового проекта любым способом
2. Заполнение сведений о проекте
3. Изменение базовых календарей проекта
4. Включение в проект дополнительной документации

Краткие теоретические сведения:

Новый проект в программе MS Project может быть создан как с нуля, так и используя один из предлагаемых стандартных шаблонов. Шаблон представляет собой особенный тип файла проекта, содержащий набор информации, призванной упростить работу над проектом. В состав шаблона обычно входит список заранее организованных и размещенных определенным образом задач, а также информация о ресурсах, пользовательские представления, календари, отчеты, макросы и т.д. Любая информация, предлагаемая шаблоном, может быть изменена в соответствии с требованиями конкретного проекта. В качестве шаблона также может быть использован созданный ранее проект. При создании проекта из шаблона необходимо выбрать на панели *Консультанта* ссылку *Общие шаблоны*. Далее на вкладке *Шаблоны проектов* выбирается необходимый шаблон.

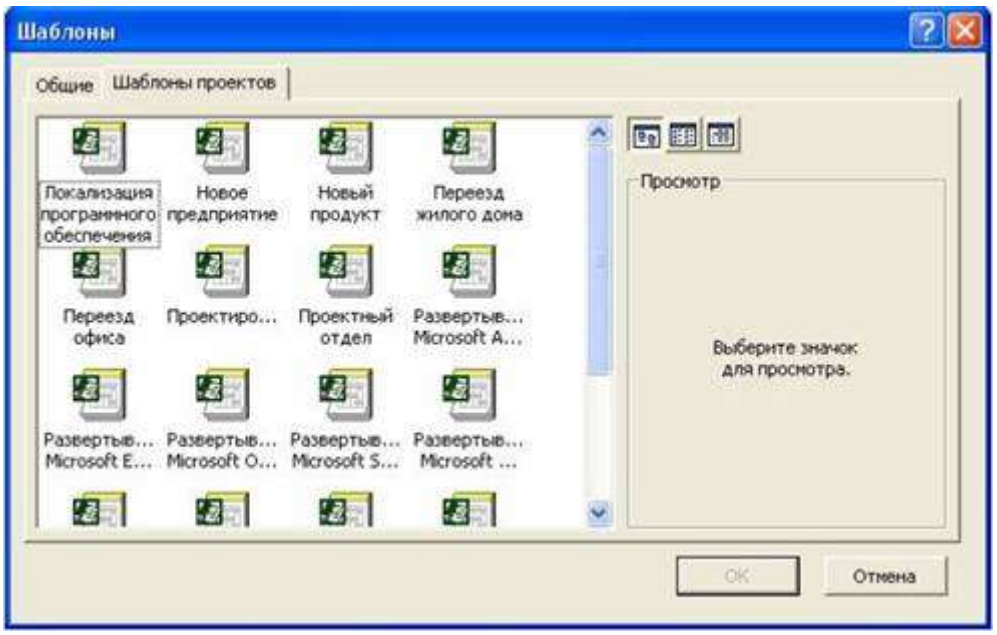


Рис.1. Выбор шаблона проекта

Рабочее пространство программы называется видом или представлением. По умолчанию после создания проекта активен вид *Диаграмма Ганта* (рис.2). Данная диаграмма служит для отображения последовательности задач проекта как в текстовом так и в графическом виде.

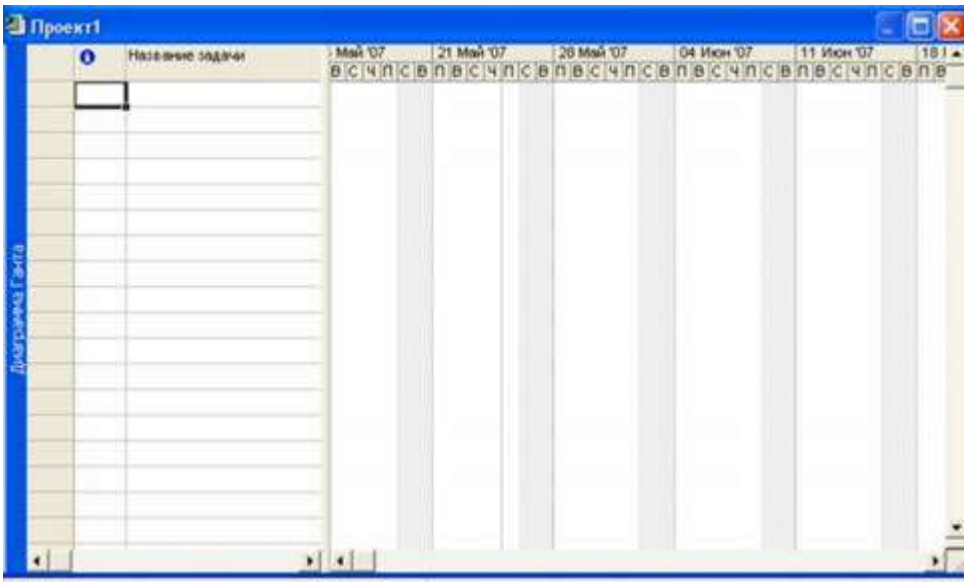


Рис.2. Окно диаграммы Ганта

После создания проекта необходимо настроить его основные параметры. Для этого

удобно использовать мастер *Новый проект.* Для этого нажимаем кнопку *Задачи* на панели *Консультанта* и выбираем ссылку *Определение проекта*. Ответив на вопросы о дате начала проекта и совместной работе над проектом и сохранив результат, выбираем ссылку *Определение рабочего времени проекта* для запуска мастера *Рабочее время проекта*. Таким образом мы можем настроить календарь проекта. Следующим решением, которое необходимо принять на стадии создания, является выбор исходной даты проекта. План проекта может быть составлен от даты начала или завершения проекта. Для настройки планирования от начальной даты выберите в меню Проект пункт сведения о проекте. В

появившемся окне (рис.3) выбираем планирование *От даты начала проекта* и ставим *Дату начала*. Да окончания будет рассчитана далее автоматически. В случае планирования от конечной даты выбираем *От даты окончания проекта* и ставим *Дату окончания*. В этом случае автоматически будет рассчитываться дата начала.

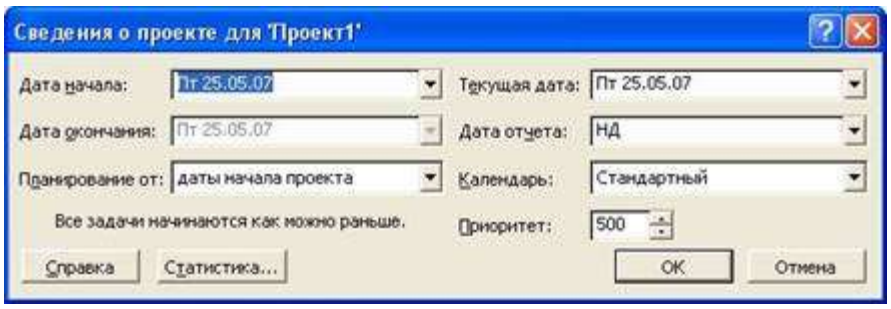


Рис.3. Настройка сведений о проекте

Также в этом окне мы можем выбрать календарь для проекта. В состав пакета MS Project входит три базовых календаря – стандартный, ночная смена и 24 часа. В *стандартном календаре* рабочий день начинается с 8:00 и заканчивается в 17:00 с обеденным перерывом с 12:00 до 13:00. Рабочая неделя начинается с понедельника и заканчивается в пятницу. Это календарь, применяемый по умолчанию. В *календаре ночной смены* рабочий день начинается с 23:00 и заканчивается в 8:00 с часовым перерывом с 03:00 до 04:00. В *календаре «24 часа»* рабочее время продолжается круглые сутки без выходных и обеденных перерывов. Базовые календари можно редактировать для этого в меню *Сервис* необходимо выбрать пункт *Изменение рабочего времени*. В появившемся окне (рис.4.) выбираем базовое расписание, которое мы хотим отредактировать. Для изменения рабочего времени одного дня необходимо выбрать этот день в календаре. Далее, если необходимо сделать этот день выходным, мы выбираем параметр *нерабочее время*, если же мы хотим только изменить временные рамки рабочего дня, то выбираем параметр *нестандартное рабочее время* и в полях ниже вводим время начала и завершения рабочего дня.

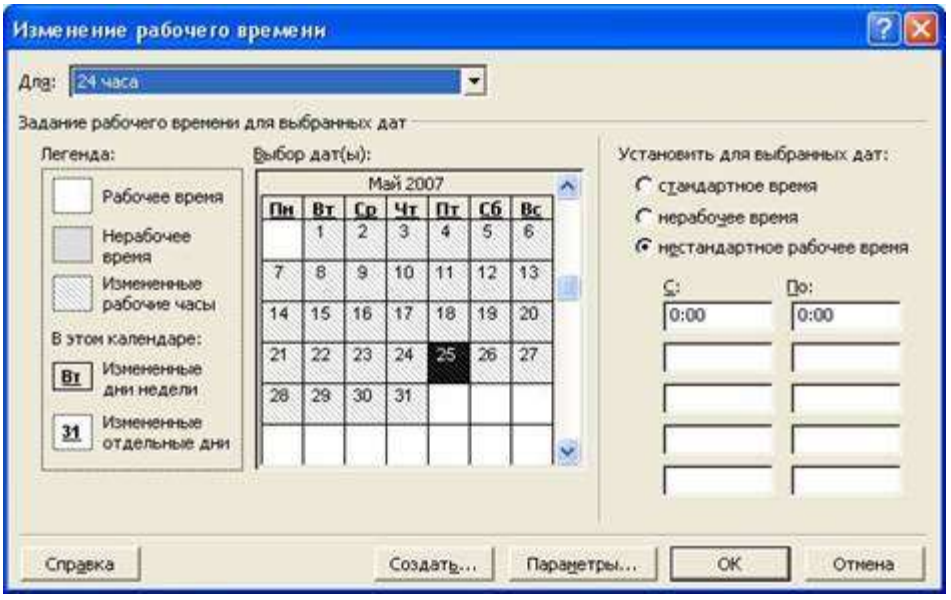


Рис.4. Изменение рабочего времени

Можно также создать новое базовое расписание. Для этого в окне *Изменение рабочего времени* нажимаем кнопку *Создать*. В появившемся окне (рис.5) выбираем создание нового календаря на основе стандартного или создание копии любого другого

календаря. Значения рабочего времени для вновь созданного календаря могут также быть отредактированы через окно *Изменение рабочего времени*.

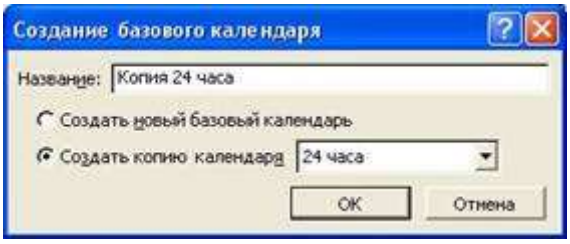


Рис.5. Создание базового календаря

Создаваемый проект может быть использован в качестве хранилища проектной документации, например, обзора проекта, результатов проведенных анализов или спецификации создаваемого продукта. Для присоединения такой документации целесообразно использовать т.н. суммарную задачу проекта, содержащую итоговую информацию о датах и стоимости проекта. Для отображения суммарной задачи на диаграмме Ганта необходимо в меню *Сервис* выбрать пункт *Параметры* и перейти на вкладку *Вид*. На данной вкладке необходимо выбрать параметр *Показать суммарную задачу проекта* под заголовком *Парметры структуры для проекта*. Суммарная задача появится в нулевом ряду диаграммы Ганта. Проектная документация может как включаться в файл проекта, так и быть доступной через гиперссылки. Для включения документов в файл проекта необходимо выбрать суммарную задачу проекта и нажать кнопку *Сведения о задаче,* расположенную на стандартной панели задач. В открывшемся окне (рис.6) выбираем вкладку Заметки. На вкладке нажимаем кнопку *Вставить объект.* В открывшемся окне необходимо выбрать опцию *Создать из файла*. После этого указываем путь к файлу документа, который предполагается включить в проект.

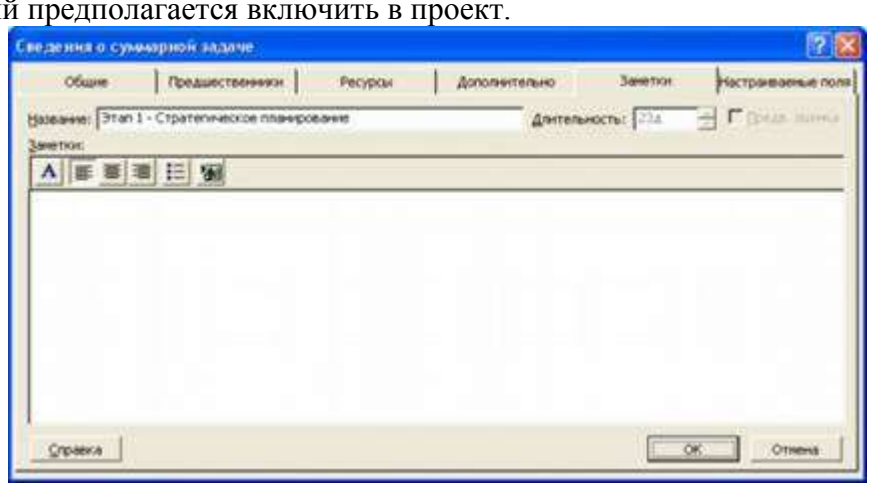


Рис.6. Сведения о задаче

После закрытия окна сведений о суммарной задаче в диаграмме Ганта появится индикатор примечаний. Для создания гиперссылки к документу необходимо нажать кнопку Гиперссылка на панели задач. В поле Текст открывшегося диалогового окна *Добавление гиперссылки* (рис.7) введите название связываемого документа, затем выберите документ в списке. В поле индикаторов диаграммы Ганта появится индикатор гиперссылок.

Рис.7. Добавление гиперссылки

Выполнение работы:

1. С помощью меню кнопки Пуск вызвать приложение Microsoft Project на рабочий стол.

2. Вызвать на поле рабочего окна приложения ранее подготовленный проект, используя библиотеку шаблонов: в области задач Создание проекта перейти на поле группы Создание с помощью шаблона и выбрать гиперссылку Общие шаблоны. На вкладке Шаблоны открыть лист Шаблоны проектов, выбрать шаблон Новый продукт и щелкнуть по кнопке ОК.

3. Сохранить шаблон проекта под новым именем: открыть меню Файл, активизировать команду Сохранить как и перейти на поле диалогового окна Сохранение документа; в диалоговом окне перейти на рабочую папку, записать в поле имени файла новое имя проекта, например FIO_Project, и щелкнуть по кнопке Сохранить.

4. Закрыть пустой проект, т.е. перейти на поле проекта Проект 1, активизировать команду Закрыть в меню Файл.

5. На поле текущего проекта убрать Область задач, для чего следует щелкнуть по кнопке Закрыть, размещенной в правом верхнем углу области.

6. Разместить на рабочем поле различные представления о состоянии проекта: вызвать Панель представлений, для чего в меню Вид щелкнуть по соответствующей команде. Панель представлений позволяет вызвать различные формы представления информации о проекте с помощью соответствующих кнопок; настроить комбинированное представление, используя команду Разделить в меню Окно и соответствующие кнопки панели инструментов, например Диаграмма Ганта и График ресурсов, Использование задач и Использование ресурсов, Диаграмма Ганта и Использование задач. Найти необходимую информацию об использовании ресурсов на Графике ресурсов, данные об их использовании (временной загрузке).

7. Настройка таблицы диаграммы Ганта: снять разделение на рабочей области с помощью команды Разделить из меню Окно и вызвать представление Диаграмма Ганта. Для вызова соответствующего столбца используется меню команды Таблица. Далее следует ознакомиться с основными опциями этого меню; перейти на поле меню Вид и раскрыть меню опций с помощью команды Таблица.

В исходном положении выбрана опция Ввод, которая устанавливает рядом с диаграммой первые два столбца таблицы: Наименование задачи (постоянный столбец) и столбец Длительность задачи; выбрать опцию Гиперссылка — рядом со столбцом задач появится столбец Гиперссылка. В ячейках этого столбца можно записать вспомогательные сведения о задачах путем составления заметок, вложения файлов или формирования гиперссылок на сопутствующую информацию, находящуюся в файле проекта или в других местах. Это позволяет подготовить библиотеки документов и связать их с проектами и задачами. После этого руководители проекта и другие заинтересованные стороны смогут

просматривать сопровождающие документы в своих веб-обозревателях; выбрать опцию Затраты. В этом случае появляются три столбца затрат: Фиксированные затраты, Начисления фактических затрат и Общие затраты. перейти на опцию Использование. На рабочем поле таблицы появятся два столбца: Трудозатраты и Длительность; просмотреть опцию Календарный план. Она вызывает четыре столбца: Начало, Окончание, Позднее начало, Позднее окончание; активировать опцию Отклонение и убедиться, что последние два столбца на поле будут заменены на столбцы Базовое начало и Базовое окончание; вызвать опцию Отслеживание, что позволяет вызвать другую группу из четырех столбцов: Фактическое начало, Фактическое окончание, % завершения и Физический % завершения; щелкнуть по опции Суммарные данные, что позволит установить такую последовательность столбцов: Длительность, Начало, Окончание и % завершения; использование опции Трудозатраты, чтобы одновременно увидеть следующие данные: Трудозатраты, Базовые, Отклонения, Фактические. 8. Настроить таблицу, добавляя необходимые и удаляя лишние столбцы: добавить новые столбцы в таблицу следует в меню Вставка, выбрать команду Столбец и в поле диалога Определение столбца с помощью раскрывающегося списка Имя поля выбрать новое поле, например, Трудозатраты; удалить установленный столбец с помощью контекстного меню, которое следует вызвать щелчком правой клавиши мыши по полю удаляемого столбца. В контекстном меню следует активизировать команду Скрыть столбец.

9. Выполнить фильтрацию данных диаграммы Ганта: выбрать кнопку Другие представления на панели представлений. На поле диалогового окна в списке Представления выбрать строку Подробная Диаграмма Ганта и щелкнуть по кнопке Применить; раскрыть список Фильтр, размещенный на панели форматирования, и щелкнуть по строке Вехи.

10. Выполнить сортировку задач проекта по длительности: в меню Проект выбрать команду Сортировка, в меню опций которой выбрать Сортировать по; в диалоговом окне Сортировка раскрыть список Сортировать по и выбрать в нем строку Длительность. Установить флажок По возрастанию и щелкнуть по кнопке Сортировать; отодвинув поле диаграммы так, чтобы видеть столбец Длительность, убедиться в правильности выполненной операции.

11. Настроить изображение диаграммы Ганта: для настройки формы и цвета отрезков в меню Формат выбрать команду Стили отрезков. В верхней части вкладки выбрать тип задачи и стиль отрезка, который следует изменить. На нижней части вкладки перейти на лист Отрезки, где выполнить операции по изменению стиля отрезка, его формы, узора и цвета; показать текст, который следует разместить рядом с отрезком (информация, отображаемая соответствующим элементом диаграммы). Для этого на вкладке Стили отрезков раскрыть лист Текст и показать, где (слева, справа, снизу, сверху или внутри отрезка) следует разместить текст; настроить шкалу времени диаграммы. Перевести курсор на поле шкалы времени и вызвать контекстное меню, где выбрать опцию Шкала времени. На соответствующей вкладке выбрать уровень шкалы времени (Верхний, Средний, Нижний) и в раскрывающемся списке Отображать выбрать строку Три уровня. Шкала времени может состоять из трех уровней (например, год, квартал, месяц), но обязателен только Средний уровень; перейти на лист Верхний уровень и установить Год в строке Единицы, показать текстовое обозначение года в раскрывающемся списке Надписи, выбрать необходимый стиль форматирования надписи и др.; перейти на лист Средний уровень и выбрать в качестве единицы измерения Квартал; перейти на лист Нижний уровень и установить в качестве единицы измерения времени Месяцы; проверить полученный результат настройки диаграммы Ганта.

12. Сохранить проект в рабочей папке и закрыть приложение.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине

страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.

2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Зачем необходимы шаблоны проектов?
2. В чем разница между планированием проекта от даты начала или даты его окончания?
3. Какие существуют базовые календари в программе MS Project?
4. Как внести изменения в базовый календарь?
5. Как включить в проект проектную документацию?

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Современные технологии и инструменты интеграции»
Практическая работа № 2. Разработка модульной структуры проекта (диаграммы модулей)

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

Выполнив работу, Вы будете:

уметь:

- У3. анализировать проектную и техническую документацию.
- У4. использовать специализированные графические средства построения и анализа архитектуры программных продуктов.
- У5. организовывать заданную интеграцию модулей в программные средства на базе имеющейся архитектуры и автоматизации бизнес-процессов.
- У6. определять источники и приемники данных.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание:

1. В соответствии с требованиями, предъявляемыми техническим заданием, и результатами внешнего проектирования разработаем модульную структуру подсистемы обслуживания клиента по его кредитной карте в банкомате.

Краткие теоретические сведения:

Чтобы добиться декомпозиции на модули максимальной прочности и минимального сцепления, необходимо спроектировать модульную структуру в виде дерева, в том числе и со сросшимися ветвями. В узлах такого дерева размещаются программные модули, а направленные дуги (стрелки) показывают статическую подчинённость модулей, т.е. каждая дуга показывает, что в тексте модуля, из которого она исходит, имеется ссылка на модуль, в который она входит.

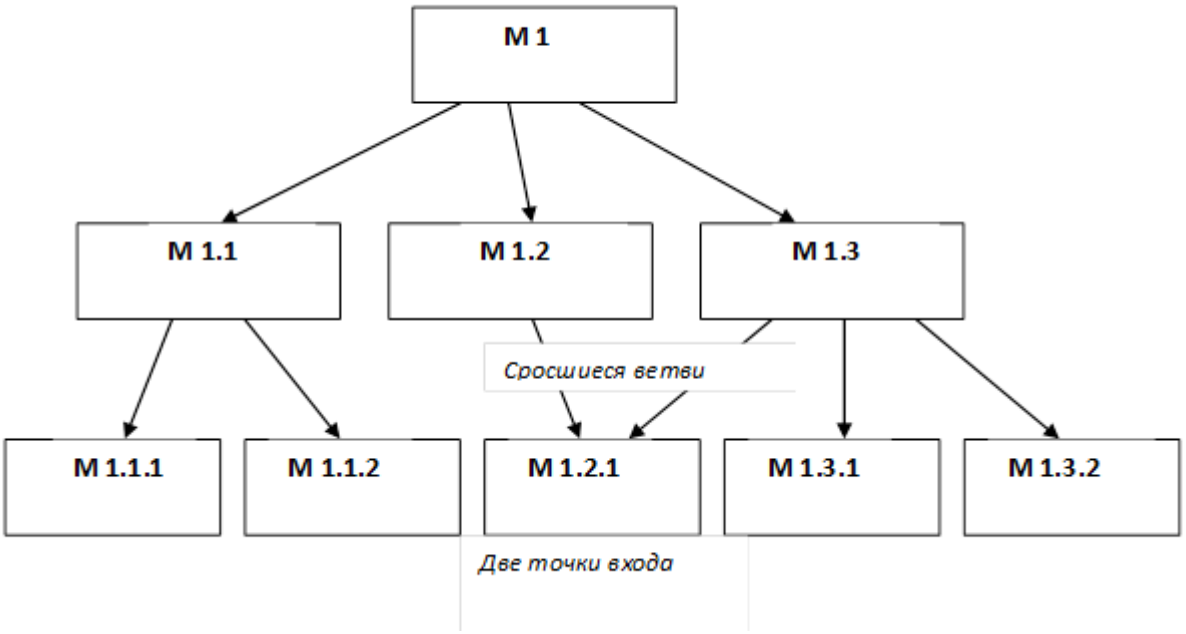


Рис.1. Пример иерархического дерева модулей

При этом модульная структура программы должна, помимо картинки, включать спецификацию программного модуля.

- спецификация программного модуля должна содержать:
- синтаксическую спецификацию его входов (имя модуля, типы передаваемых ему параметров, типы возвращаемых результатов, синтаксис обращения к любому ему входов)
- функциональную спецификацию (описание семантики функций, выполняемых модулем по каждому из его входов).

В процессе разработки модульная структура может по-разному использоваться для определения порядка программирования модулей — восходящая и нисходящая разработка.

В восходящей разработке модули программируются, начиная с нижних уровней, и сразу тестируются, исходя из функциональных спецификаций. Такой порядок представляется естественным, т.к. каждый новый модуль выражается через уже запрограммированные и проверенные модули. Однако современная технология не рекомендует этот прием, т.к. при этом трудно обеспечить концептуальную целостность ПС.

Концептуальная целостность предполагает общие принципы реализации, предположения, структуры данных, - а они могут быть ещё не ясны в начальных стадиях разработки.

Перепрограммирование же модулей нижних уровней связано с большими затратами, т.к. требует не только повторной разработки текстов, но и повторного тестирования.

Предпочтительной является нисходящая разработка. В этой технологии программирование начинается с модуля с самого верхнего уровня. При этом для тестирования модули нижних уровней заменяются простыми по конструкции имитаторами, которые либо моделируют работу нижних уровней (например, реализуют таблицы; вход-отклик), либо просто сообщают о своём вызове и завершаются признаком успеха. Такая реализация обеспечивает большую концептуальную целостность и меньший объём разрабатываемых тестов, каждый модуль здесь тестируется при т.н. «естественном» состоянии информационной среды, т.к. он вызывается реальным (оттестированным) модулем верхнего уровня.

Выполнение работы:

В составе программного обеспечения можно выделить следующие программные модули: Головной модуль (Main module), Модуль управления устройством считывания кредитной кары (Credit cart control module), Модуль аутентификации (Autentification module) и Модуль получения и обработки запроса на обслуживание (Reception and processing module). Кроме этого в состав ПО необходимо включить модуль данных кредитной карты (Credit cart data).

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

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

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

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

- обработка внутренней банковской документации по клиенту;
- распечатка баланса клиента;
- выдача наличных денег и информирование компьютера банка об изъятых из банка деньгах;
- распечатка операции клиента.

На рис. 2 приведена структурная карта, демонстрирующая отношения между указанными модулями системы.

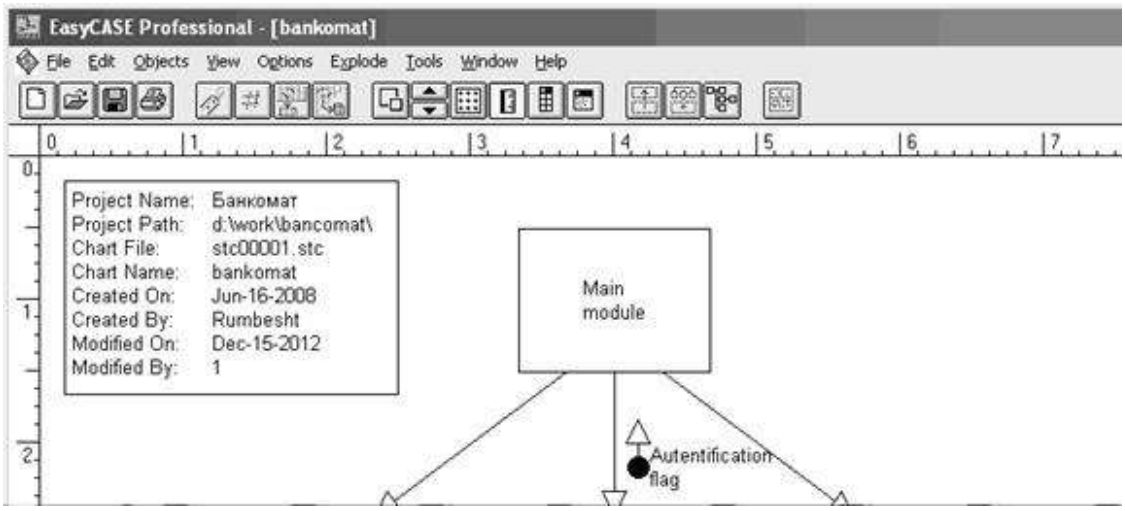


Рис. 2. Модульная структура программного обеспечения

Согласно этой диаграмме головной модуль обращается к модулям управления устройством считывания кредитной карты, аутентификации и получения и обработки запроса на обслуживание. Вызов указанных модулей осуществляется согласно внутренней логики головного модуля, реализующей следующий сценарий: При инициации действий со стороны клиента головной модуль, вызывает модуль управления устройством считывания кредитной карты для ее ввода и считывания с нее информации. После завершения считывания управление возвращается головному модулю, который затем обращается к модулю аутентификации. Модуль аутентификации проверяет подлинность клиента и вместе с результатом этой проверки возвращает управление головному модулю. В зависимости от результатов аутентификации головной модуль либо вызывает модуль управления устройством считывания для удаления кредитной карты, либо обращается к модулю получения и обработки запроса на обслуживание для предоставления требуемого сервиса. Если осуществляется вызов получения и обработки запроса на обслуживание, то после завершения его работы головной модуль обращается к модулю управления устройством считывания для удаления кредитной карты.

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

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Цель разработки модульной структуры.

2. Понятие программного модуля, передачи управления, организации связи по управлению и по данным.

3. Виды связности модулей.

4. Виды целостности модулей.

5. Типовые модульные структуры.

6. Проектирование модульной структуры с помощью структурных карт.

7. Построение структурных карт с помощью программного продукта EasyCASE Professional Version 4.21.016.

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Современные технологии и инструменты интеграции»

Практическая работа № 3. Разработка перечня артефактов и протоколов проекта

Цель: Получение первичных навыков планирования работ по разработке и внедрению автоматизированных информационных систем, разработка протоколов проекта.

Выполнив работу, Вы будете:

уметь:

- У3. анализировать проектную и техническую документацию.
- У4. использовать специализированные графические средства построения и анализа архитектуры программных продуктов.
- У5. организовывать заданную интеграцию модулей в программные средства на базе имеющейся архитектуры и автоматизации бизнес-процессов.
- У6. определять источники и приемники данных.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019, Microsoft Project.

Задание:

На основе технического задания на разработку и внедрение автоматизированной информационной системы сформировать календарный план выполнения работ по проекту.

Краткие теоретические сведения:

Для проведения успешного проекта нужно оценить объем предстоящих работ, возможный риск, требуемые ресурсы, предстоящие задачи, определить контрольные точки, стоимость и план работ, которому желательно следовать. Процесс руководство программным проектом включает решение вышеперечисленных задач. Этот процесс начинается перед технической работой, продолжается по мере развития ПО от идеи к реальности и достигает наибольшей интенсивности к концу работ. Основной задачей при планировании является определение структуры распределения работ WBS (Work Breakdown Structure) с помощью средств планирования работ (например, MS Project). План выполнения работ составляется на основе декомпозиции проекта вплоть до постановки элементарных задач, которые могут быть выяснены по результатам предварительного анализа. При этом возможно применение содержательных моделей системного анализа. Например, использование модели декомпозиции типа «жизненный цикл» позволит разбивать отдельные задачи на подзадачи путем определения последовательности действий.

Процесс декомпозиции будет определяться принятой моделью жизненного цикла разработки программного обеспечения.

Рис. 1. Декомпозиция задач, которые необходимо решить в процессе выполнения проекта по разработке программного обеспечения

Для каждой элементарной задачи должны быть определены:

- ресурсы, необходимые для решения задачи (в том числе трудовые);
- объем работ, выраженный в принятой системе метрик;
- стоимость работ (может быть вычислена на основе объема работ и стоимости привлекаемых ресурсов);
- длительность работ (может быть вычислена на основе объема работ, количества привлекаемых рудовых ресурсов и принятых нормативов производительности).

Между отдельными элементарными задачами могут быть определенные зависимости, заключающиеся в том, что одни задачи могут выполняться параллельно, другие – в строгой последовательности (для выполнения одних задач могут требоваться результаты выполнения других).

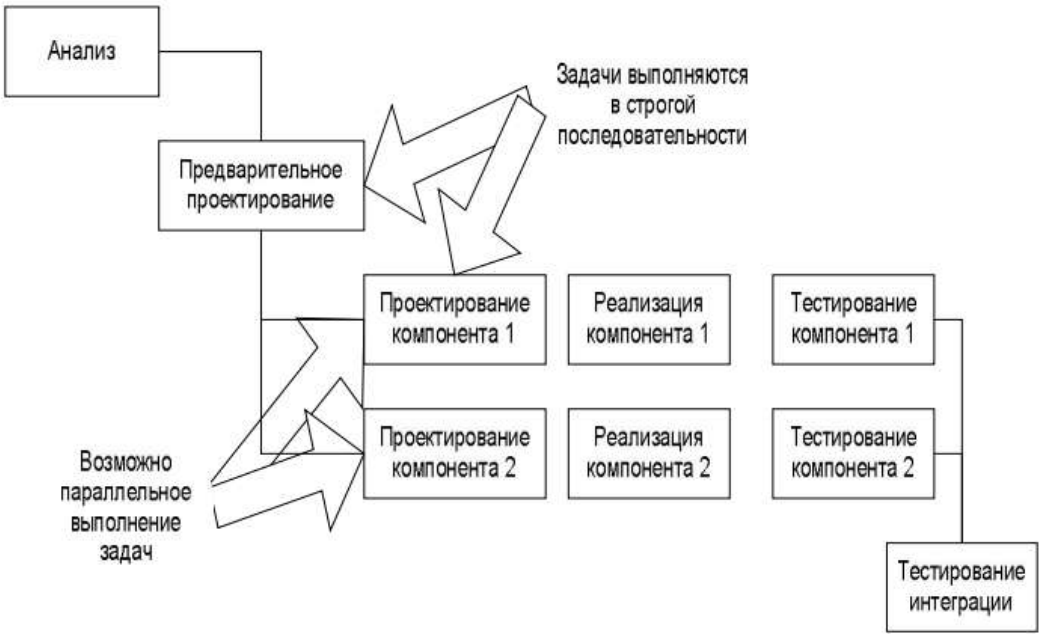


Рис. 2. Параллельное и последовательное выполнение задач

После определения зависимостей можно приступать к распределению элементарных задач по времени. При этом особое внимание следует остановить на задачах, выполняемых параллельно. Параллельность действий повышает требования к планированию.

Необходимо четко отследить наличие ресурсов, необходимых для выполнения каждой задачи. Если план предусматривает использование ресурса 1 для выполнения задач А и Б, то эти задачи не могут выполняться параллельно, даже если между ними нет концептуальной зависимости. Кроме того, руководитель проекта должен знать задачи, лежащие на критическом пути. Для того чтобы весь проект был выполнен в срок, необходимо выполнять в срок все критические задачи.

Календарный план помимо распределения задач и ресурсов по времени должен предусматривать процедуры контроля промежуточных результатов. Такие процедуры обычно называют вехами. Очень важно, чтобы вехи были расставлены через регулярные интервалы вдоль всего процесса разработки программного обеспечения. Кроме того, желательно чтобы вехи совпадали со сроком выполнения критических задач. Это даст руководителю возможность не только регулярно получать информацию о текущем положении дел, но и объективно оценивать риски срыва сроков выполнения проекта,

принимать оперативные решения по снижению этих рисков. В первую очередь определите основные фазы выполнения проекта.

В основу может быть положена принятая модель жизненного цикла процесса разработки программного обеспечения. Например, при использовании каскадной модели основными фазами будут являться анализ, проектирование, реализация, тестирование, внедрение. Далее определите состав работ внутри каждой фазы, в соответствии с сутью разработки.

Таким образом будет определен состав работ по проекту. Каждая фаза должна заканчиваться вехой – специальной единицей работы, подразумевающей контроль выполнения работ по проекту и достижение некоторого промежуточного или окончательного результата. Далее определите длительность каждой работы, входящей в план работ.

Для определения длительности могут быть использованы различные регрессионные модели (например, COCOMO II), или же может применяться прямой метод оценки. Следующим этапом является определение связей между задачами. В большинстве средств планирования (например, в MS Project), существует четыре вида связей: Связь типа окончание – начало означает, что задача Б не может начаться раньше окончания задачи А (например, если в процессе выполнения задачи Б используются результаты, получаемые при решении задачи А).

Связь типа начало – начало означает, что задача Б не может начаться до тех пор, пока не началось выполнение задачи А. Например, тестирование программного компонента не может начинаться до того, как была начата его разработка, но, в то же время, для написания тестов не обязательно дожидаться окончания разработки этого компонента. Связь типа окончание – окончание означает, что работа Б не может окончиться до тех пор, пока не завершится выполнение работы А. Например, проектирование базы данных не может быть закончено до того, как будет завершено семантическое моделирование предметной области.

Связь окончание – начало означает, что задача Б не может закончиться до того, как началась задача А. Обычно такая связь используется в том случае, когда А является задачей с фиксированной датой начала, которую нельзя изменить. После того, как определен состав работ, нужно определить, кто эти задачи будет выполнять и какое оборудование должно использоваться.

Сформируйте список ресурсов, для каждого ресурса определите название и стоимость его использования. Далее назначьте ресурсы на выполнение конкретных задач. При первом назначении ресурса будут автоматически рассчитаны трудозатраты. В тех случаях, когда необходимо ускорить выполнение тех или иных задач, на них могут быть назначены дополнительные ресурсы. После распределения ресурсов необходимо выполнить выравнивание их нагрузки. В тех случаях, когда на параллельно выполняемые задачи назначается один и тот же ресурс, нагрузка на него может превысить максимально допустимую. Для выравнивания нагрузки установите дополнительные связи между задачами таким образом, чтобы задачи, использующие один и тот же ресурс, выполнялись последовательно.

Выполнение работы:

1. Ознакомьтесь с техническим заданием
2. Выберите модель жизненного цикла процесса разработки и внедрения ПО, которая, по вашему мнению, в наибольшей степени соответствует рассматриваемой ситуации.
3. Выделите основные этапы работ.
4. Выделите основные задачи внутри отдельных этапов
работ.
5. Определите зависимости между задачами.
6. Определите порядок выполнения отдельных задач.

7. Назначьте исполнителей на решаемые задачи.
8. Сбалансируйте нагрузку исполнителей.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Опишите понятие техническое задание.
2. Что такое жизненный цикл ПО?
3. В чем разница между параллельным и последовательным выполнением задач?

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Современные технологии и инструменты интеграции»

Лабораторная работа № 1. Настройка работы системы контроля версий (типов импортируемых файлов, путей, фильтров и др. параметров импорта в репозиторий)

Выполнив работу, Вы будете:

уметь:

- У1 использовать выбранную систему контроля версий.
- У7. использовать приемы работы в системах контроля версий.
- У12 использовать различные транспортные протоколы и стандарты форматирования сообщений.
- У14 организовывать постобработку данных.

Цель: Изучить на практике понятия и компоненты систем контроля версий (СКВ), приемы работы с ними.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание:

1. Установите TortoiseSVN на компьютере.
2. Создайте новый проект.
3. Создайте локальный репозиторий для своего проекта.
4. Удалите созданный проект на своем компьютере и обновите проект из репозитория.
5. Внесите изменения в файлах с исходными кодами и сохраните изменения в репозитории. Обновите файлы с исходными кодами из репозитория.
6. Внесите изменения в файлах с исходными кодами таким образом, чтобы у двух участников проекта изменения были в одном и том же файле. Попытайтесь сохранить изменения в репозитории. Устраните обнаруженные конфликты версий. Повторно сохраните изменения в репозитории.
7. Создайте отдельную ветку проекта. Внесите изменения в файлы с исходными кодами. Сохраните изменения в репозитории.
8. Объедините созданную на предыдущем шаге ветку с основной веткой проекта.
9. Выведите на экран лог изменений файла, в котором было наибольшее количество изменений.
10. Отобразите на экране сравнение фала до и после внесения одного из изменений.
11. Создайте репозитарий в сети Интернет. Повторите шаги 4 – 6.

Краткие теоретические сведения:

Управление версиями - это искусство работы с изменениями информации. Долгое время оно было жизненно важным инструментом программистов, которым необходимо произвести небольшие изменения в программе, или же сделать ―откат‖ изменений, возвращаясь к предыдущей версии. Однако полезность систем управления версиями выходит далеко за пределы мира разработчиков программного обеспечения. Управление версиями требуется повсюду, где можно встретить людей, использующих компьютер для работы с постоянно изменяющейся информацией

Software Configuration Management или **Конфигурационное управление** подразумевает под собой комплекс методов, направленных на то, чтобы систематизировать изменения, вносимые разработчиками в программный продукт в процессе его разработки и сопровождения, сохранить целостность системы после изменений, предотвратить нежелательные и непредсказуемые эффекты, а также сделать процесс внесения изменений более формальным. Изначально управление конфигурацией применялось не в программировании, но в связи с высокой динамичностью сферы разработки ПО, в ней она особенно полезна. К процедурам можно отнести создание резервных копий, контроль исходного кода, требований проекта, документации и т. д. Степень формальности

выполнения данных процедур зависит от размеров проекта, и при правильной ее оценке данная концепция может быть очень полезна. Конфигурационное управление требует выполнения множества трудоемких рутинных операций. На практике, в большинстве случаев, для конфигурационного управления применяются специальные системы контроля версий исходного кода программ. В качестве примера такой системы рассмотрим самую распространенную на сегодняшний день – Subversion.

Subversion – это бесплатная система управления версиями с открытым исходным кодом. Subversion позволяет управлять файлами и каталогами, а так же сделанными в них изменениями во времени. Это позволяет восстановить более ранние версии данных, дает возможность изучить историю всех изменений. Благодаря этому многие считают систему управления версиями своего рода «машиной времени».

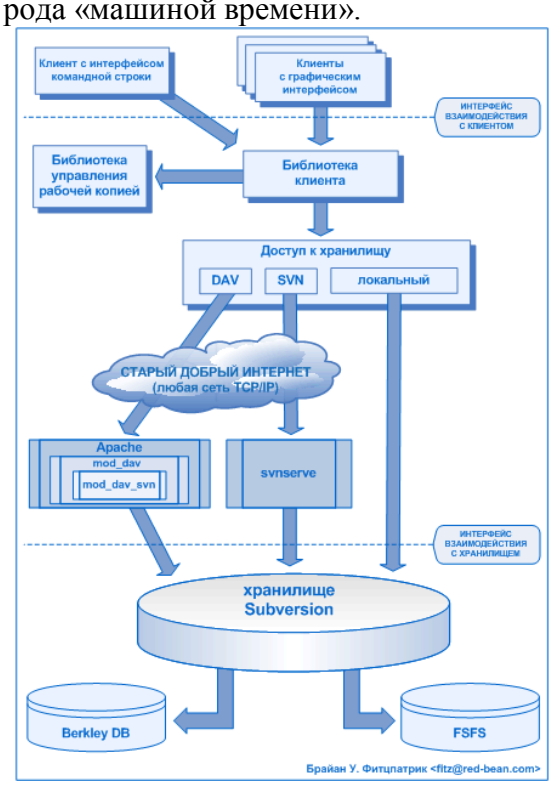


Схема общей архитектуры Subversion. В нижней части схемы изображено хранилище Subversion, в котором хранится информация с версиями. В верхней части схемы показана программа-клиент Subversion, которая управляет локальными отражениями различных фрагментов этих данных (также называемыми «рабочими копиями»). Между этими сторонами проложены различные маршруты, проходящие через разные слои доступа к хранилищу. Некоторые из этих маршрутов используют компьютерные сети и сетевые сервера, чтобы достичь хранилища, в то время как другие маршруты в сети не нуждаются и ведут к хранилищу напрямую.

Subversion может работать через сеть, что позволяет использовать ее на разных компьютерах. В какой-то степени, возможность большого количества людей не зависимо от их местоположения совместно работать над единым комплектом данных поощряет сотрудничество. Когда нет того ответственного звена цепи, того контролирующего элемента, который утверждает все изменения, работа становится более эффективной. При этом не нужно опасаться, что отказ от контролирующего элемента повлияет на качество, ведь благодаря сохранению истории изменений, даже если при изменении данных будут допущены ошибки, всегда можно сделать откат изменений к прежнему состоянию.

Некоторые системы управления версиями выступают также в качестве систем управления конфигурацией программного обеспечения. Такие системы специально созданы для управления деревьями исходного кода и имеют множество особенностей, непосредственно относящихся к разработке программ: они понимают языки программирования и предоставляют инструменты для сборки программ. Subversion не является такой системой, она представляет собой систему общего назначения, которую можно использовать для управления *любым* набором файлов. Для Вас это будут исходники Ваших программ, а для кого-то другого это будет список продуктов или сведённое цифровое видео.

TortoiseSVN

TortoiseSVN – это бесплатный Windows-клиент с открытыми исходным кодом для системы управления версиями *Apache™ Subversion®*. То есть TortoiseSVN управляет файлами и директориями во времени. Файлы хранятся в центральном *хранилище*. Хранилище больше похоже на обычный файловый сервер, кроме того он запоминает каждое изменение, когда-либо сделанное в ваших файлах и директориях. Это позволяет вам восстановить старые версии ваших файлов и проверить историю изменений — как, когда и кто изменял ваши данные. Вот почему многие думают о Subversion, и вообще о системах управления версиями, как своего рода «машине времени».

Некоторые системы контроля версий являются также и системами управления конфигурацией программ (software configuration management - SCM). Такие системы специально созданы для управления деревьями исходного кода, и имеют множество возможностей, специфичных для разработки программ, таких как непосредственное понимание языков программирования, или предоставление инструментов для сборки программ. Однако Subversion не является такой системой, она является системой общего назначения, которая может быть использована для управления *любым* набором файлов, включая и исходные коды программ.

Возможности TortoiseSVN

Что делает TortoiseSVN таким хорошим клиентом Subversion? Вот краткий список возможностей:

Интеграция с оболочкой

TortoiseSVN интегрируется непосредственно в оболочку Windows (т.е. в Проводник). Это значит, что вы можете работать с уже знакомыми инструментами, и вам не надо переключаться на другое приложение каждый раз, когда вам необходимы функции для управления версиями!

И вам даже не обязательно использовать именно Проводник. Контекстные меню TortoiseSVN работают во многих других файловых менеджерах, и в диалогах для открытия файлов, используемых в большинстве стандартных Windows-приложений. Однако вы должны учитывать, что TortoiseSVN изначально разработан как расширение для Проводника Windows, и, возможно, в других приложениях интеграция будет не полной, например, могут не отображаться пометки на значках.

Пометки на значках

Статус каждого версированного файла и папки отображается при помощи маленькой пометки поверх основного значка. Таким образом, вы сразу можете видеть состояние вашей рабочей копии.

Графический интерфейс пользователя

При просмотре списка изменений файла или папки вы можете кликнуть на ревизию, чтобы увидеть комментарии для этой фиксации. Также доступен список измененных файлов - всего лишь сделайте двойной клик на файле, чтобы увидеть, какие конкретно изменений были внесены.

Диалог фиксации – это список, в котором перечислены все файлы и папки, которые будут включены в фиксацию. У каждого элемента списка имеется флажок, чтобы вы могли выбрать именно то, что вы хотите включить в фиксацию. Неверсированные файлы также

могут быть представлены в этом списке, чтобы вы не забыли добавить в фиксацию новый файл или папку.

Простой доступ к командам Subversion

Все команды Subversion доступны из контекстного меню Проводника. TortoiseSVN добавляет туда собственное подменю.

Поскольку TortoiseSVN является клиентом Subversion, мы хотели бы показать и некоторые из возможностей самой Subversion:

Версирование папок

CVS отслеживает только историю отдельных файлов, тогда как Subversion реализует «виртуальную» версионную файловую систему, которая отслеживает изменения в целых деревьях папок во времени. Файлы *и* папки являются версированными. В результате, есть команды **переместить** и **копировать**, реально выполняемые на стороне клиента и работающие непосредственно с файлами и папками.

Атомарные фиксации

Фиксация сохраняется в хранилище либо полностью, либо не сохраняется вообще. Это позволяет разработчикам фиксировать изменения, собранные в логически связанные части.

Версированные метаданные

Каждый файл и папка имеет прикреплённый невидимый набор «свойств». Вы можете создавать и сохранять произвольные пары ключ/значение для собственных нужд. Свойства тоже версируются во времени, как и содержимое файла.

Возможность выбора сетевого уровня

В Subversion есть абстрагируемое понятие доступа к хранилищу, которое упрощает реализацию новых сетевых механизмов. «Усовершенствованный» сетевой сервер Subversion является модулем для веб-сервера Apache, который использует для взаимодействия диалект HTTP под названием WebDAV/DeltaV. Это даèт Subversion большие преимущества в стабильности и совместимости, и предоставляет различные ключевые возможности без дополнительных затрат: проверка личности (аутентификация), проверка прав доступа (авторизация), сжатие потока данных при передаче, просмотр хранилища. Также доступна меньшая, автономная версия сервера Subversion, взаимодействующая по собственному протоколу, который легко может быть туннелирован через ssh.

Единый способ обработки данных

Subversion получает различия между файлами при помощи бинарного разностного алгоритма, который работает одинаково как с текстовыми (читаемыми человеком), так и с бинарными (не читаемыми человеком) файлами. Оба типа файлов содержатся в хранилище в сжатом виде, а различия передаются по сети в обоих направлениях.

Эффективные ветки и метки

Стоимость создания веток и меток не обязательно должна быть пропорциональна размеру проекта. Subversion создаёт ветки и метки, просто копируя проект с использованием механизма, похожего на жёсткие ссылки в файловых системах. Благодаря этому, операции по созданию веток и меток происходят за одинаковое, очень малое время и занимают очень мало места в хранилище.

Выполнение работы:

Установка

TortoiseSVN поставляется в виде простого в использовании установочного файла (на странице загрузки http://tortoisesvn.net/downloads.html необходимо выбрать дистрибутив, соответствующий разрядности системы). Сделайте на нем двойной клик и следуйте инструкциям - остальное он сделает за вас. Также на странице загрузки необходимо скачать языковой пакет и установить после установки основной программы. Не забудьте перезагрузить компьютер после установки.

Основная концепция

Перед тем, как мы погрузимся в работу с настоящими файлами, важно получить представление о том, как работает Subversion и какие термины используются.

Хранилище

Subversion использует центральную базу данных, которая содержит все ваши версированные файлы с их полной историей. Эта база данных называется *хранилищем*. Хранилище обычно находится на файловом сервере, на котором установлен Subversion, по запросу поставляющий данные клиентам Subversion (например, TortoiseSVN). Если вы делаете резервное копирование, то копируйте ваше хранилище, так как это оригинал всех ваших данных.

Рабочая копия

Это именно то место, где вы работаете. Каждый разработчик имеет собственную рабочую копию, иногда называемую песочницей, на своем локальном компьютере. Вы можете получить из хранилища последнюю версию файлов, поработать над ней локально, никак не взаимодействуя с кем- либо еще, а когда вы будете уверены в изменениях вы можете зафиксировать эти файлы обратно в хранилище.

Рабочая копия не содержит историю проекта, но содержит копию всех файлов, которые были в хранилище до того, как вы начали делать изменения. Это обозначает, что можно легко узнать, какие конкретно изменения вы сделали.

Вам также нужно знать, где найти TortoiseSVN, потому что в меню "Пуск" его нет. TortoiseSVN - расширение проводника Windows, поэтому для начала нужно запустить проводник. Сделайте правый клик на папке в проводнике, и вы увидите новые пункты в контекстном меню, такие как эти:

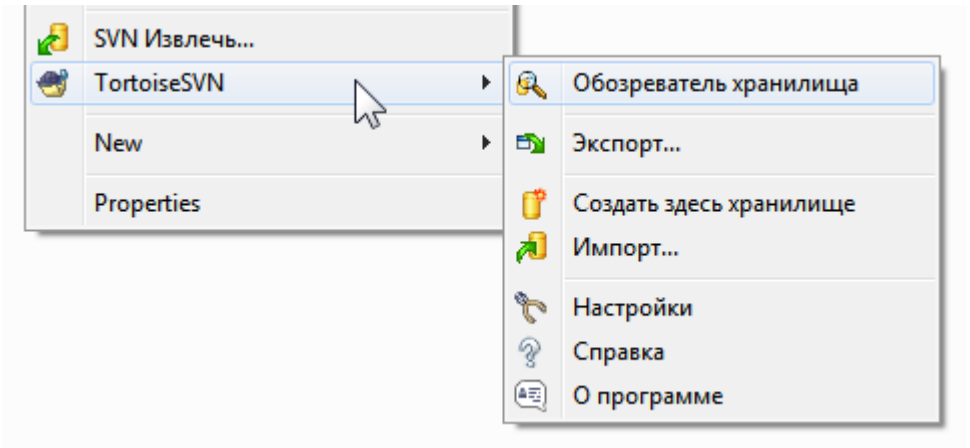


Рис.1. Меню TortoiseSVN для неверсированных папок

Создание хранилища

Для настоящего проекта вам понадобится хранилище, созданное в безопасном месте, и сервер Subversion, чтобы управлять им. Для обучения мы будем использовать функцию локального хранилища Subversion, которая разрешает прямой доступ к хранилищу, созданного на вашем жестком диске, и не требует наличия сервера.

Сначала создайте новую пустую директорию на вашем ПК. Она может быть где угодно, но в этом руководстве мы собираемся назвать её C:\svn_repos. Теперь сделайте правый клик на новой папке и в контекстном меню выберите **TortoiseSVN → Создать здесь хранилище...**. Хранилище, созданное внутри папки, готово к использованию. Также мы создадим внутреннюю структуру папок нажав кнопку. **Создать структуру каталогов**

Импорт проекта

Сейчас у нас есть хранилище, но оно совершенно пустое в данный момент. Давайте

предположим, что у меня есть набор файлов вC:\Projects\Widget1, который я хотел бы добавить. Перейдите к папке Widget1 в Проводнике и сделайте правый клик на ней. Теперь выберите пункт **TortoiseSVN → Импорт...**, который вызовет диалог

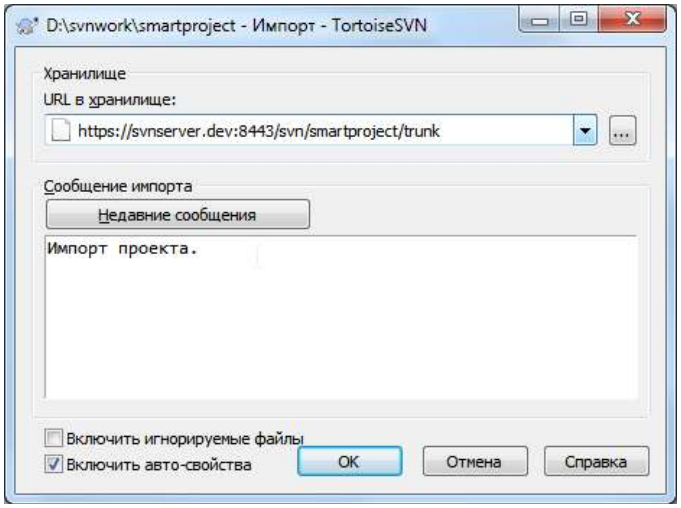


Рис. 2. Диалог импорта

К хранилищу Subversion обращаются по URL-адресу, который позволяет нам указать хранилище где угодно в Интернете. В данном случае нам нужно указать на наше локальное хранилище, которое имеет URL-адрес file:///c:/svn_repos/trunk, и к которому мы добавляем имя нашего проекта Widget1. Обратите внимание, что после file: есть 3 слэша и везде используются прямые слэши.

Другая важная функция данного диалога - это окно Сообщение импорта, в которое вы можете добавить сообщение о том, что вы делаете. Когда вам понадобится просмотреть историю проекта, эти сообщения будут ценным подспорьем для просмотра какие изменения и когда были сделаны. В нашем случае мы напишем что-нибудь простое как «Импорт проекта Виджет1». Нажмите, чтобы добавить папку в ваше хранилище.

Извлечение рабочей копии

Сейчас у нас есть проект в нашем хранилище, и нам надо создать рабочую копию для повседневной работы. Заметьте, что имортирование папки не превращает автоматически эту папку в рабочую копию. Для создания свежей рабочей копии в Subversion используется термин Извлечь. Мы собираемся извлечь папку Widget1 из нашего хранилища в папку для разработки называемуюC:\Projects\Widget1-Dev. Создайте эту папку, затем сделате правый клик на ней и выберите пункт **TortoiseSVN → Извлечь...**. Введите URL-адрес для извлечение, в данном случае file:///c:/svn_repos/trunk/Widget1, и кликните на . Наша папка для разработки заполнится файлами из хранилища.

Вы заметите что внешний вид этой папки отличается от обычной папки. У каждого файла появился зелёный флажок в левом углу. Это значки статуса TortoiseSVN, которые присутствуют только в рабочей копии. Зеленый статус означает, что файл не отличается от версии файла, находящегося в хранилище.

Внесение изменений

Можно приступать к работе. В папке Виджет1-Дев мы начинаем изменять файлы - предположим, мы вносим изменения в файлыВиджет1.c и ПрочтиМеня.txt. Обратите внимание, что значки на этих файлах теперь стали красными и показывают, что изменения были сделаны локально.

Но какие были изменения? Нажмите правой кнопкой на одном из изменённых файлов и выберите команду **TortoiseSVN → Различия**. Запустится инструмент TortoiseSVN для сравнения файлов и покажет какие точно строки в файлах были изменены.

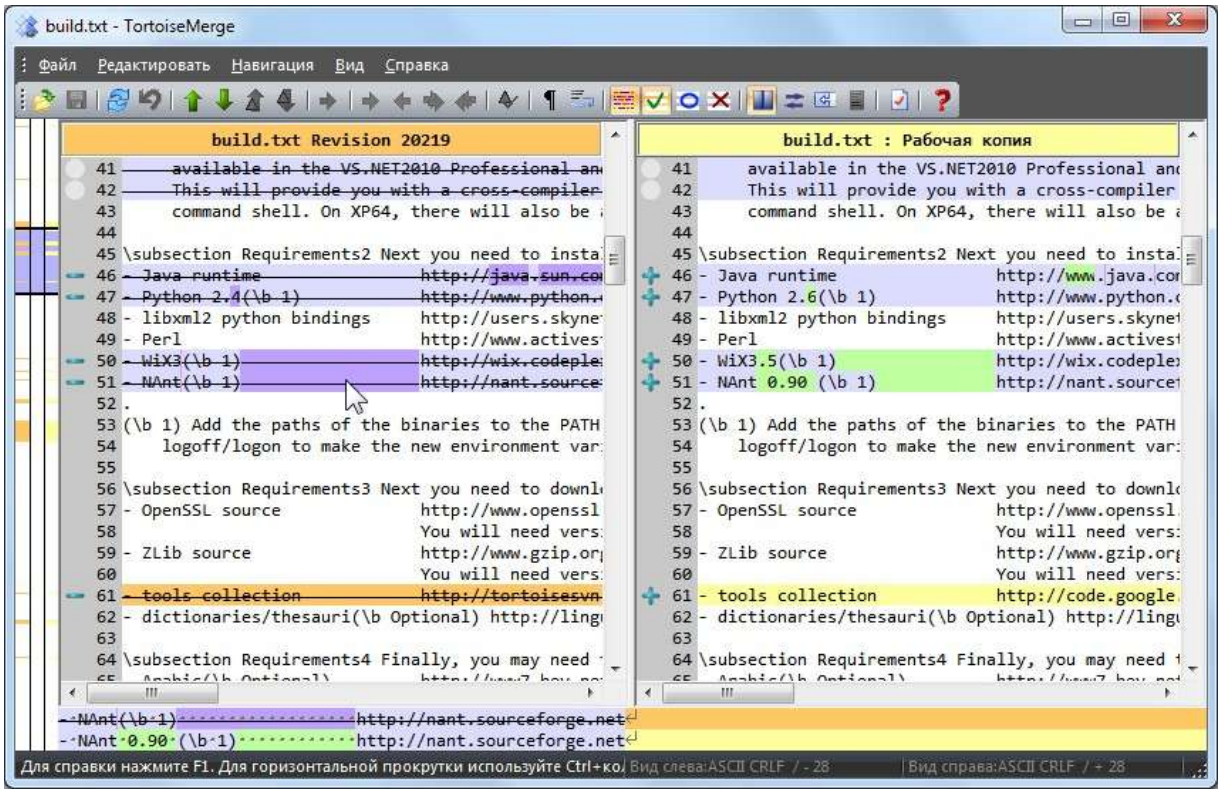


Рис. 3. Просмотрщик изменений в файлах

Ок**ОК**ас устраивают изменения, поэтому давайте обновим хранилище. Это действие называется Фиксировать изменения. Нажмите правой кнопкой на папке Виджет1-Дев и выберите команду **TortoiseSVN → Фиксировать**. Появится диалог фиксации со списком изменённых файлов и напротив каждого будет галочка. Вы можете выбрать лишь несколько файлов из списка для фиксации, но в нашем случае мы будем фиксировать изменения в обоих файлах. Введите сообщение с описанием сделанных изменений и нажмите ОК. Появится диалог с прогрессом процесса фиксации файлов в хранилище и мы закончили фиксацию.

Добавление новых файлов

Во время работы над проектом, вам понадобится добавлять новые файлы - предположим вы добавили новые функции в файлеЭкстра.с и добавили справку в существующем файле Создатьфайл. Нажмите правой кнопкой на папке и выберите команду**TortoiseSVN → Добавить**. Диалог добавления показывает все неверсированные файлы, и вы можете выбрать те файлы, которые вы хотите добавить. Другой способ добавления файлов - это нажать правой кнопкой на самом файле и выбрать команду**TortoiseSVN → Добавить**.

Теперь, если вы откроете папку для фиксации, новый файл будет отображаться как *Добавлен* и существующий файл как *Изменён*. Обратите внимание, что вы можете дважды нажать на изменённый файл, чтобы просмотреть какие именно изменения были сделаны.

Просмотр истории проекта

Одной из самых полезных функций TortoiseSVN является диалоговое окно журнала. Оно показывает список всех фиксаций изменений в файле или папке, а также все те детальные сообщения, которые вы вводили при фиксации изменений ;-)

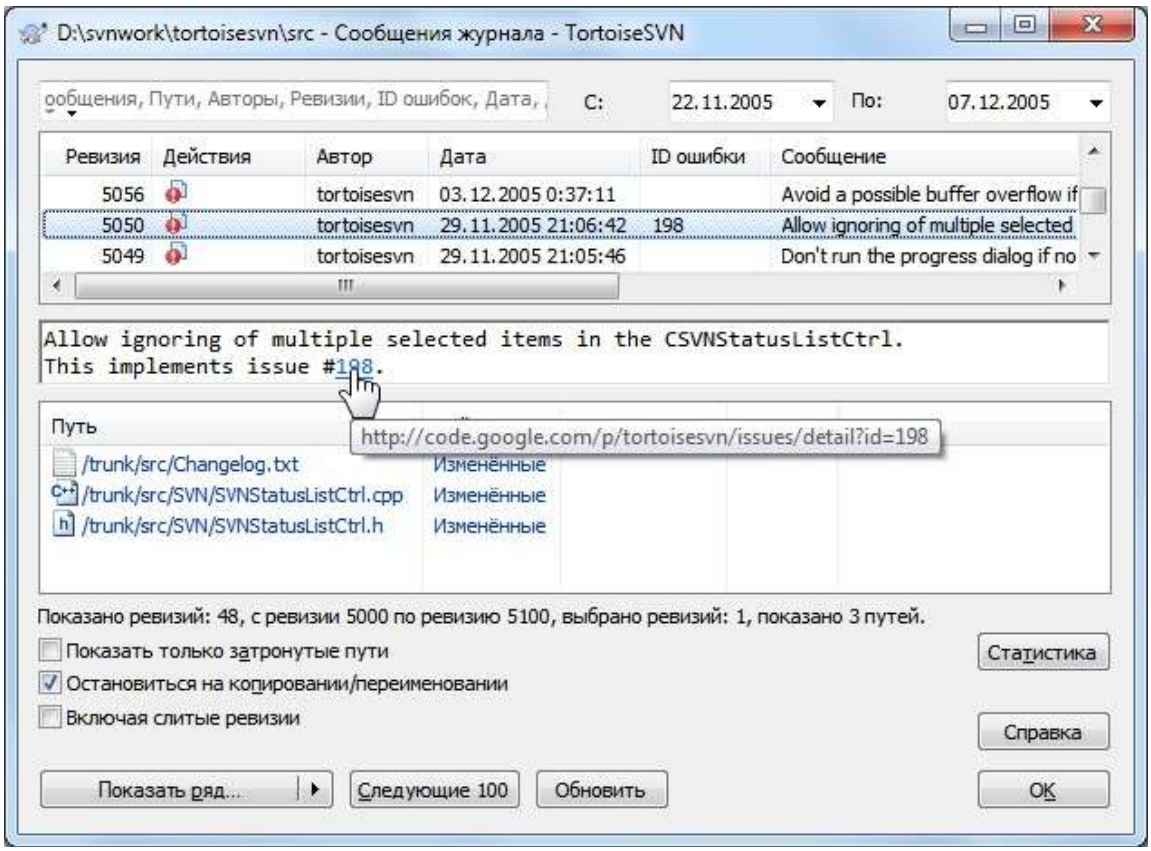


Рис. 4. Диалоговое окно журнала

Верхняя панель показывает список всех фиксированных ревизий вместе с началом сообщения фиксации. Если вы выберите одну из этих ревизий, то средняя панель отобразит полное сообщение журнала для той ревизии и нижняя панель покажет список измененных файлов и папок.

У каждой из этих панелей есть контекстное меню, которое предоставляет много других способов использования информации. В нижней панели вы можете **дважды нажать** на файл, чтобы просмотреть какие именно изменения были внесены в той ревизии. Прочтите «Диалоговое окно журнала ревизий», чтобы узнать больше.

Отмена изменений

Одной общей функцией всех систем управления ревизиями является функция, которая позволяет вам отменить изменения, которые вы внесли ранее. Как вы и догадались, в TortoiseSVN это легко сделать.

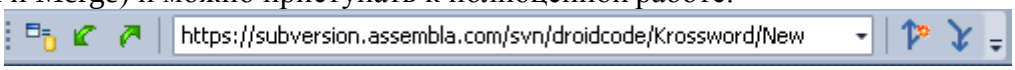
Если вы хотите избавиться от изменений, которые вы еще не успели фиксировать и восстановить нужный файл в том виде, в котором он был перед началом изменений, то выберите команду **TortoiseSVN → Убрать изменения**. Это действие отменит ваши изменения (в Корзину) и вернет фиксированную версию файла, с которой вы начинали. Если же вы хотите убрать лишь некоторых изменения, то вы можете использовать инструмент TortoiseMerge для просмотра изменений и выборочного удаления измененных строк.

Если вы хотите отменить действия определѐнной ревизии, то начните с диалогового окна журнала и найдите проблемную ревизию. Выберите команду **Контекстное меню → Отменить изменения из этой ревизии** и те изменения будут отменены.

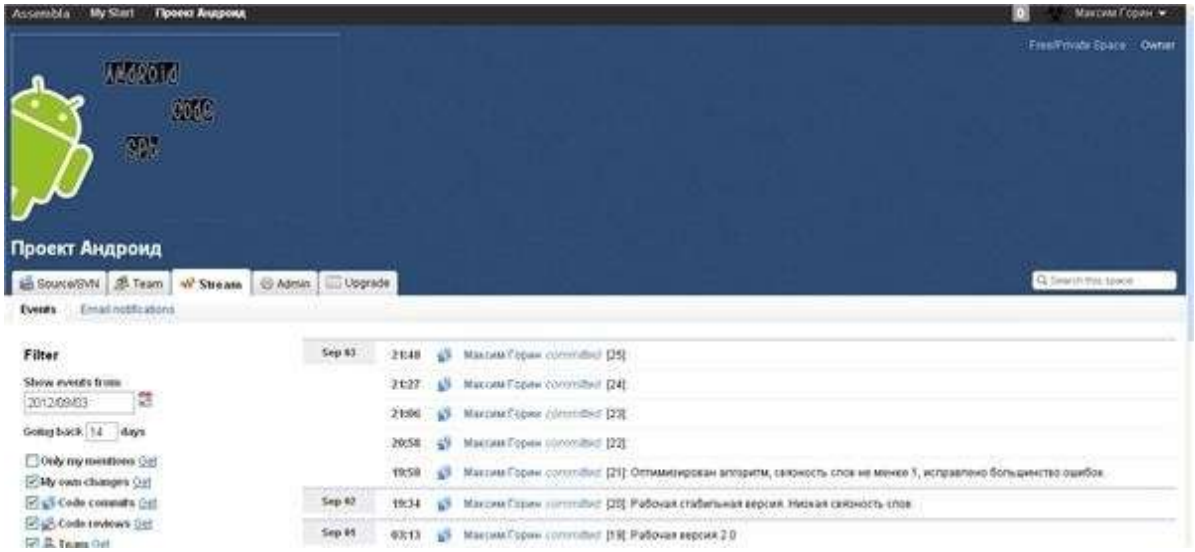
Работа с сетью.

С Subversion можно работать как посредством сети интернет, так и локально.

Воспользуемся сервисом Assembla (https://www.assembla.com/). Зарегистрировавшись там, вы получите 1 Gb места под репозиторий. Создав его и настроив, вы получите ссылку вида https://subversion.assembla.com/svn/название_репозитория, которую можно использовать в любом SVN клиенте. К примеру, чтобы в Visual SVN добавить свой проект в репозиторий, вам нужно нажать Add Solution to Subversion, после чего указать локальное хранилище вашего проекта, нажать Далее и ввести вашу ссылку. Все, теперь сверху появится панель с основными SVN-функциями (Show Log, Update, Commit, Switch Branch, Branch и Merge) и можно приступать к полноценной работе.



В общем и целом, сайт позволяет так же настроить репозиторий и получить на неделю или две пробный премиум (платные доп. функции, источник жизни сайта). Основные функции же полностью бесплатны. Чтобы товарищи по команде могли работать с общим для команды репозиторием, на assembla.com, необходима их регистрация на сайте. После регистрации, пользователи могут быть добавлены владельце репозитория в список команды. На самом сайте можно посмотреть всю информацию о проекте и изменениях, и даже поставить свой баннер с ссылкой.



Так, например, выглядит страница Stream, где отображаются последние изменения.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Что такое системы контроля версий (СКВ) и для решения каких задач они предназначаются?

2. Объясните следующие понятия СКВ и их отношения: хранилище, commit, история, рабочая копия.

3. Что представляют собой и чем отличаются централизованные и децентрализованные СКВ? Приведите примеры СКВ каждого вида.

4. Опишите действия с СКВ при единоличной работе с хранилищем.

5. Опишите порядок работы с общим хранилищем в централизованной СКВ.

6. Что такое и зачем может быть нужна разность (diff)?

7. Что такое и зачем может быть нужно слияние (merge)?
8. Что такое конфликты (conflict) и каков процесс их разрешения (resolve)?
9. Поясните процесс синхронизации с общим хранилищем («обновления») в децентрализованной СКВ.
10. Что такое и зачем могут быть нужны ветви (branches)?
11. Объясните смысл действия rebase в СКВ Git.
12. Как и зачем можно игнорировать некоторые файлы при commit?

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Современные технологии и инструменты интеграции»

Лабораторная работа № 2. Разработка и интеграция модулей проекта (командная работа)

Цель: закрепление практических навыки работы с системой Visual Studio 2019, MS SQL Server, проведение интеграции программных модулей.

Выполнив работу, Вы будете:

уметь:

- У1 использовать выбранную систему контроля версий.
- У2 использовать методы для получения кода с заданной функциональностью и степенью качества.
- У5. организовывать заданную интеграцию модулей в программные средства на базе имеющейся архитектуры и автоматизации бизнес-процессов.
- У6. определять источники и приемники данных.
- У7. использовать приемы работы в системах контроля версий.
- У17 использовать инструментальные средства отладки программных продуктов.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019, MS SQl Server.

Задание:

1. Разработать базу данных, включающую в себя таблицу Пользователи, используя среду MS SQl Server.
2. Создать приложение с окном авторизации пользователя.
3. Создать модель данных на основе разработанной ранее базы данных.
4. Сохранить данные пользователя в созданной базе данных.
5. Создать отчет о проделанной работе.

Краткие теоретические сведения:

Для создания базы данных в СУБД, запустите MS SQL Server Management Studio. Установите соединение с сервером.

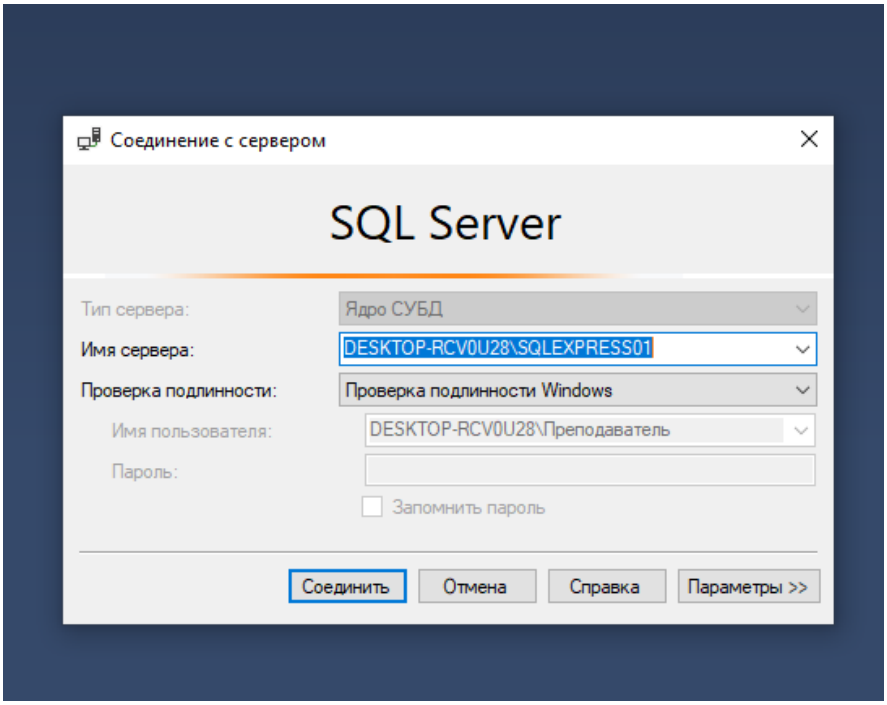


Рис. 1. Соединение с сервером

Для реализации базы данных, хранящей сведения о пользователях необходимо создать две сущности (таблицы): Пользователь (хранит основные сведения,

характеризующие пользователя) и Секретный вопрос (хранит список секретных вопросов, необходимых для авторизации) с указанными атрибутами.

На ключевых полях КодПользователя и КодСекретногоВопроса настройте спецификацию идентификатора с начальным значением – 1 и шагом приращения – 1 для автоматического заполнения указанных полей.

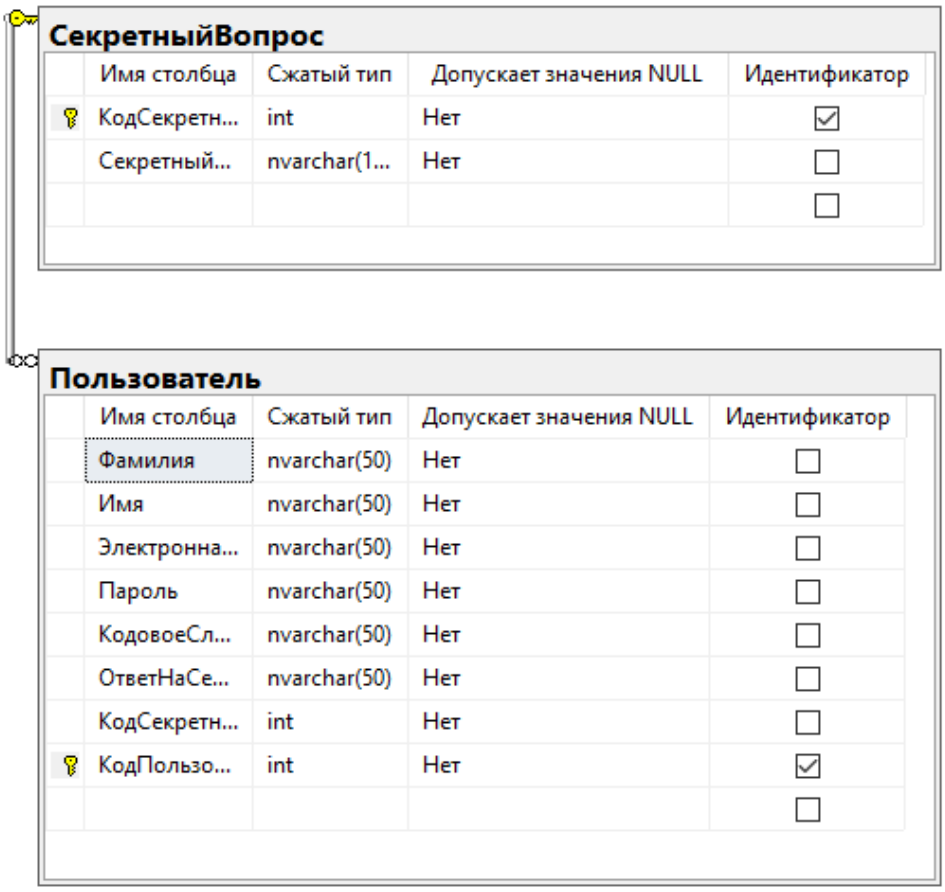


Рис. 2. Диаграмма базы данных

Окно авторизации следующим образом (Рис. 2).

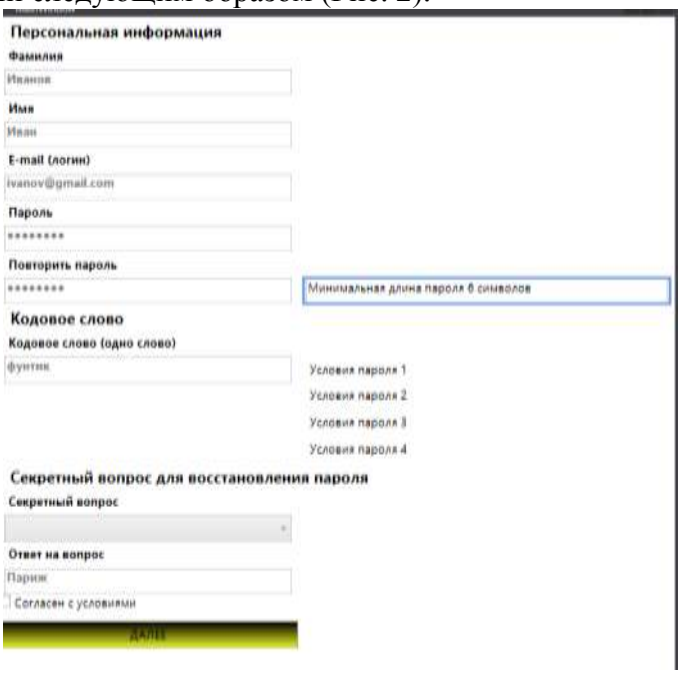
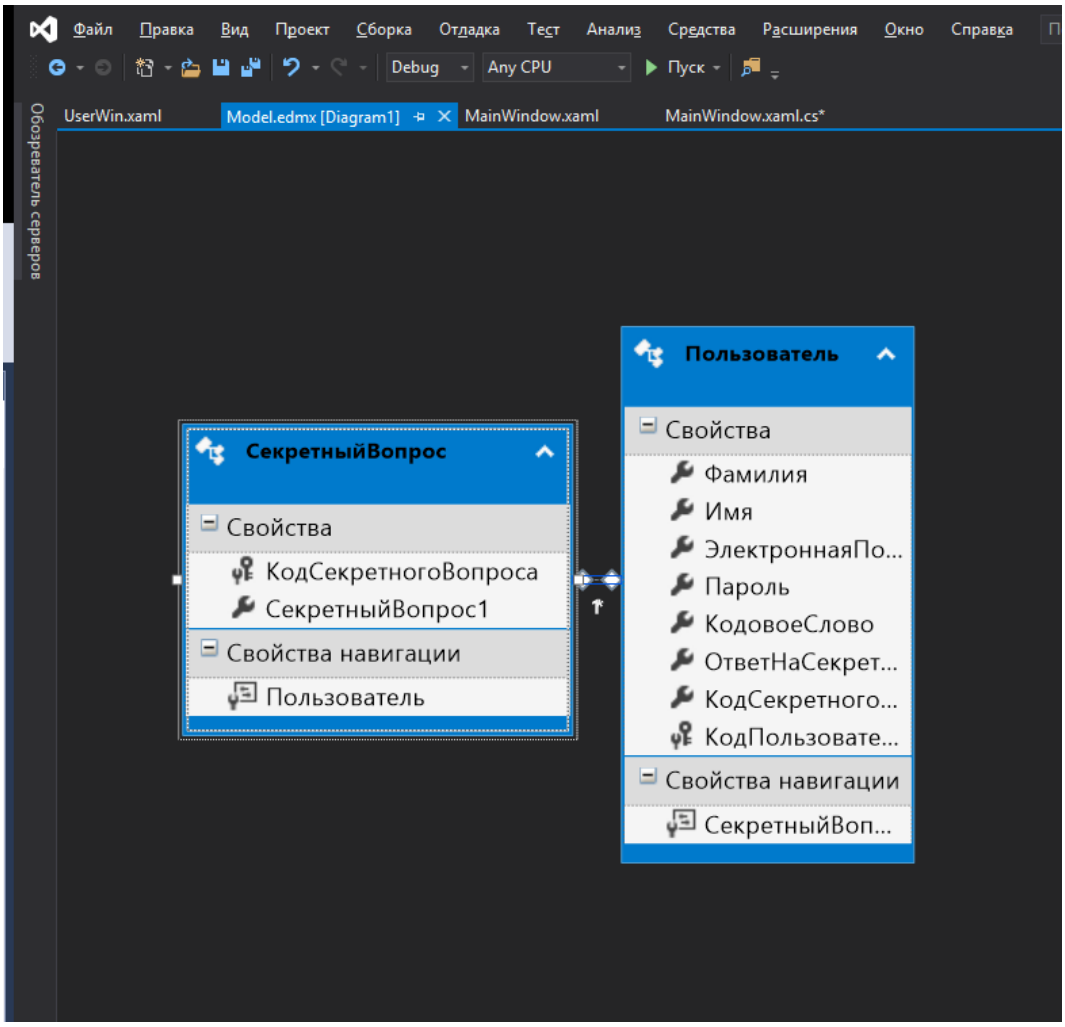


Рис 3. Окно авторизации

Код страницы авторизации показан в приложении 1.
Для передачи данных необходимо создать модель данных.



После создания модели создаем новый экземпляр класса модели данных и используя технологию Entity Framework передаем данные в базу.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    qwest.ItemsSource = context.СекретныйВопрос.Select(i => i.СекретныйВопрос1).ToList();

}
private void SignIn_Click(object sender, RoutedEventArgs e)
{
    context.Пользователь.Add(new Пользователь
    {
        Фамилия = lName.Text,
        Имя = name.Text,
        ЭлектроннаяПочта = eMail.Text,
        Пароль = passw.Password,
        КодовоеСлово = word.Text,
        ОтветНаСекретныйВопрос = otvet.Text,
        КодСекретногоВопроса = context.СекретныйВопрос.Where(i => i.СекретныйВопрос1 == qwest.Text).S

    });
    context.SaveChanges();
    MessageBox.Show($"Пользователь {lName.Text} добавлен") ;

}
```

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Что такое интеграция?
2. Как реализовывается создание базы данных?
3. Что такое модель данных?
4. Как передаются данные в MSSQL Server

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Современные технологии и инструменты интеграции»

Лабораторная работа № 3. Отладка отдельных модулей программного проекта

Цель: усвоить знание основ модульного программирования; освоить способы создания и применения модулей.

Выполнив работу, Вы будете:

уметь:

- У2 использовать методы для получения кода с заданной функциональностью и степенью качества.
- У6. определять источники и приемники данных.
- У8. выполнять отладку, используя методы и инструменты условной компиляции (классы Debug и Trace)).
- У11. выявлять ошибки в системных компонентах на основе спецификаций.
- У12 использовать различные транспортные протоколы и стандарты форматирования сообщений.
- У17 использовать инструментальные средства отладки программных продуктов.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание:

1. Провести отладка отдельных модулей программного проекта

Краткие теоретические сведения:

Модульное программирование основано на понятии модуля – программы или функционально завершенного фрагмента программы.

Модуль характеризуют:

1. один вход и один выход. На входе программный модуль получает определенный набор исходных данных, выполняет их обработку и возвращает один набор выходных данных;
2. функциональная завершенность. Модуль выполняет набор определенных операций для реализации каждой отдельной функции, достаточных для завершения начатой обработки данных;
3. логическая независимость. Результат работы данного фрагмента программы не зависит от работы других модулей;
4. слабые информационные связи с другими программными модулями. Обмен информацией между отдельными модулями должен быть минимален;
5. размер и сложность программного элемента в разумных рамках.

С помощью модулей решаются различные профессиональные задачи обработки данных разного типа.

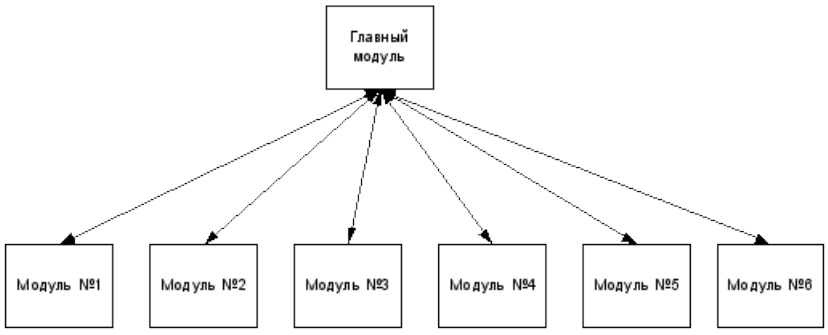


Рис. 1. Схема подключения модулей

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.

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.

Выполнение работы:

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.

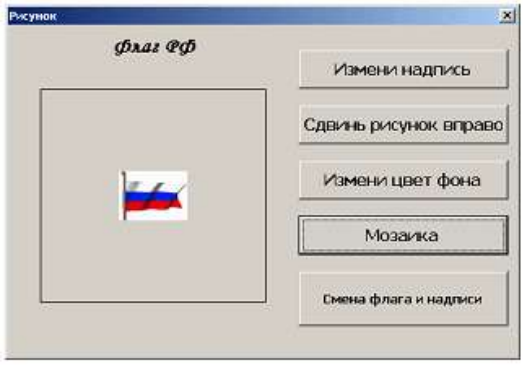


Рис. 2. Форма для выполнения задания

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.

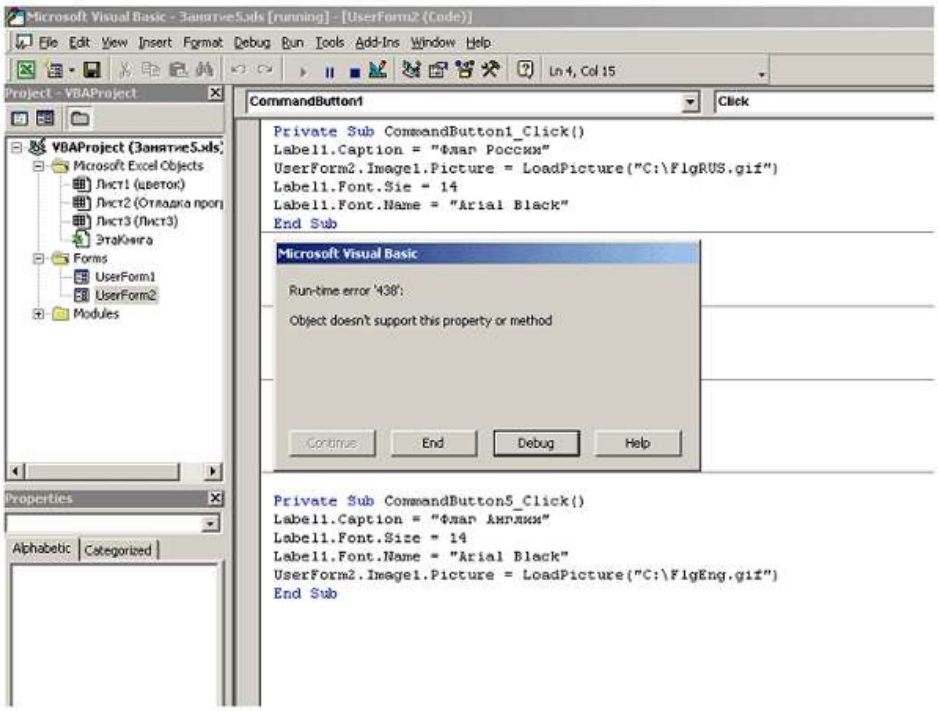


Рис. 3. Окно редактирования кода с окном сообщения об ошибке

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

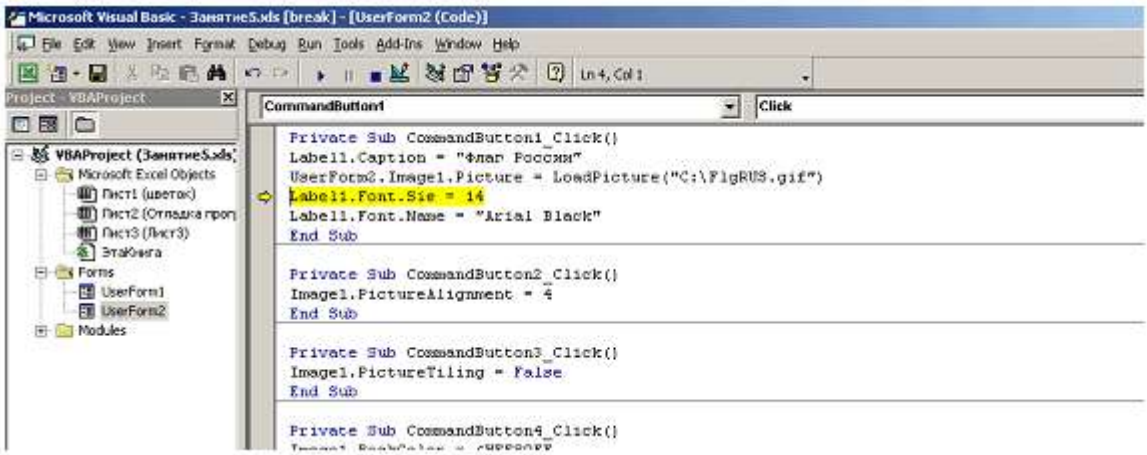


Рис. 4. Окно редактирования кода с указанной ошибкой

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.

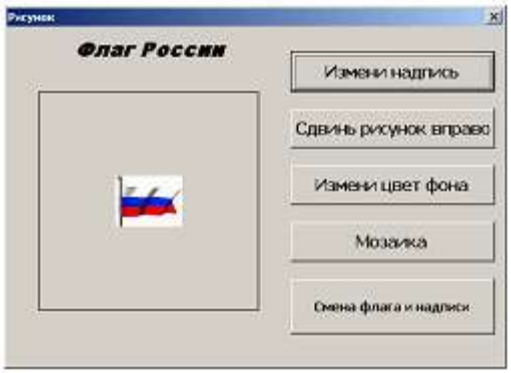


Рис. 5. Работа кнопки «Измени надпись»

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

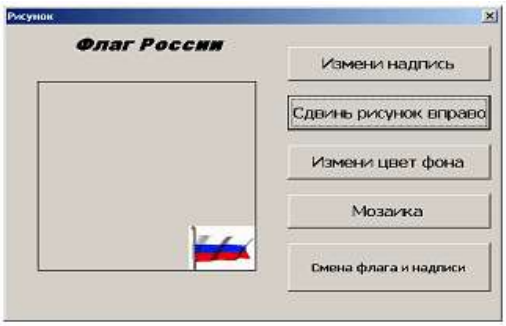


Рис. 6. Работа кнопки «Сдвинь рисунок вправо»

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

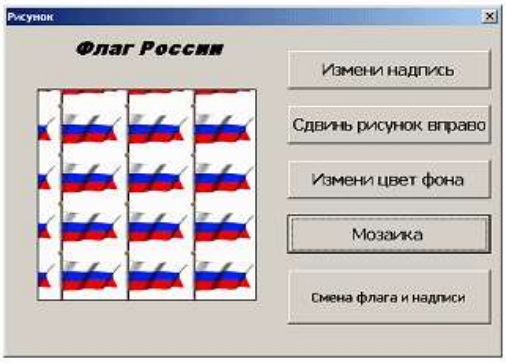


Рис. 7. Работа кнопки «Мозаика»

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

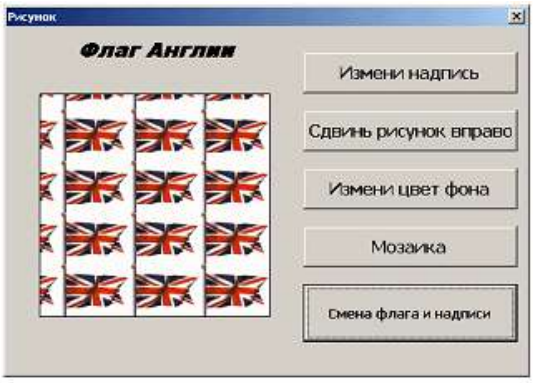


Рис. 8. Работа кнопки «Смена флага и надписи»

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.

Задание 3. Сохраните ваш проект в папке на рабочем столе с вашим ФИО и номером группы. А также составьте отчет.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

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

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Современные технологии и инструменты интеграции»

Лабораторная работа № 4. Организация обработки исключений

Цель: изучить операторы, используемые при обработке исключительных ситуаций, возникающих во время выполнения вычислительных процессов, получить практические навыки в составлении программ.

Выполнив работу, Вы будете:

уметь:

- У2 использовать методы для получения кода с заданной функциональностью и степенью качества.
- У3. анализировать проектную и техническую документацию.
- У6. определять источники и приемники данных.
- У8. выполнять отладку, используя методы и инструменты условной компиляции (классы Debug и Trace)).
- У15 создавать классы- исключения на основе базовых классов.
- модуля.
- У17 использовать инструментальные средства отладки программных продуктов.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание: оптимизировать программу, производящую простые арифметические операции над числами (сложение, вычитание, умножение и деление), используя обработку исключительных ситуаций (обработка ввода чисел и операции деления).

Пример:

```
using System;
namespace ConsoleApplication {
 class OurClass {
   static void Main(string[] args) {
 float num1 = 1, num2 = 2, summarize, multiply, sub, divide = 0;
 Console.Write("Введите первое число:");
 try { num1 = float.Parse(Console.ReadLine()); }
 catch {
  Console.WriteLine("Неправильный формат числа!\n"+
      "В качестве значения первого числа будет 1");
}
Console.Write("Введите второе число:");
 try { num2 = float.Parse(Console.ReadLine()); }
     catch {
  Console.WriteLine("Неправильный формат числа!\n"+
       "В качестве значения второго числа будет 2");
 }
summarize = num1 + num2; multiply = num1 * num2; sub = num1 - num2;
 try { divide = num1 / num2; }
 catch(DivideByZeroException) {
  Console.WriteLine("Нельзя делить на нуль!");
 }
     Console.WriteLine(
     "\n" + num1 + " + " + num2 + " = " + summarize +
"\n" + num1 + " * " + num2 + " = " + multiply +
"\n" + num1 + " - " + num2 + " = " + sub +
"\n" + num1 + " / " + num2 + " = " + divide);
Console.Write("\nДля выхода из программы нажмите [Enter]:");
     string anykey = Console.ReadLine();
   }
 }
}
```

Краткие теоретические сведения:

Схема обработки исключений в C#

Язык C# наследовал схему исключений языка C++, внеся в нее свои коррективы. Рассмотрим схему подробнее и начнем с синтаксиса конструкции try-catch-finally:

```
try {...}
catch (T1 e1) {...}
...
catch(Tk ek) {...}
finally {...}
```

Всюду в тексте модуля, где синтаксически допускается использование блока, этот блок можно сделать охраняемым, добавив ключевое слово try. Вслед за try-блоком могут следовать catch-блоки, называемые блоками-обработчиками исключительных ситуаций, их может быть несколько, они могут и отсутствовать. Завершает эту последовательность finally-блок - блок финализации, который также может отсутствовать. Вся эта конструкция может быть вложенной - в состав try-блока может входить конструкция try-catch-finally.

Выбрасывание исключений. Создание объектов Exception

В теле try-блока может возникнуть исключительная ситуация, приводящая к выбрасыванию исключений. Формально выбрасывание исключения происходит при выполнении оператора throw. Этот оператор, чаще всего, выполняется в недрах операционной системы, когда система команд или функция API не может сделать свою работу. Но этот оператор может быть частью программного текста try-блока и выполняться, когда в результате проведенного анализа становится понятным, что дальнейшая нормальная работа невозможна.

Синтаксически оператор throw имеет вид:

```
throw[выражение]
```

Выражение throw задает объект класса, являющегося наследником класса Exception. Обычно это выражение new, создающее новый объект. Если оно отсутствует, то повторно выбрасывается текущее исключение. Если исключение выбрасывается операционной системой, то она сама классифицирует исключение, создает объект соответствующего класса и автоматически заполняет его поля.

В рассматриваемой нами модели исключения являются объектами, класс которых представляет собой наследника класса Exception. Этот класс и многочисленные его наследники является частью библиотеки FCL, хотя и разбросаны по разным пространствам имен. Каждый класс задает определенный тип исключения в соответствии с классификацией, принятой в Framework .Net. Вот лишь некоторые классы исключений из пространства имен System: ArgumentException, ArgumentOutOfRangeException, ArithmeticException, Bad ImageFormatException, DivideByZeroException, OverflowException. В пространстве имен System.IO собраны классы исключений, связанных с проблемами ввода-вывода: DirectoryNotFoundException, FileNotFoundException и многие другие. Имена всех классов исключений заканчиваются словом Exception. Разрешается создавать собственные классы исключений, наследуя их от класса Exception.

При выполнении оператора throw создается объект te, класс TE которого характеризует текущее исключение, а поля содержат информацию о возникшей исключительной ситуации. Выполнение оператора throw приводит к тому, что нормальный процесс вычислений на этом прекращается. Если это происходит в охраняемом try-блоке, то начинается этап "захвата" исключения одним из обработчиков исключений.

Захват исключения

Блок catch - обработчик исключения имеет следующий синтаксис:

```
catch (T e) {...}
```

Класс T, указанный в заголовке catch -блока, должен принадлежать классам исключений. Блок catch с формальным аргументом e класса T потенциально способен

захватить текущее исключение te класса TE, если и только если объект te совместим по присваиванию с объектом e. Другими словами, потенциальная способность захвата означает допустимость присваивания e = te, что возможно, когда класс TE является потомком класса T. Обработчик, класс T которого является классом Exception, является универсальным обработчиком, потенциально он способен захватить любое исключение, поскольку все они являются его потомками.

Потенциальных захватчиков может быть много, исключение захватывает лишь один - тот из них, кто стоит первым в списке проверки. Каков порядок проверки? Он довольно естественный. Вначале проверяются обработчики в порядке следования их за try -блоком, и первый потенциальный захватчик становится активным, захватывая исключение и выполняя его обработку. Отсюда становится ясно, что порядок следования в списке catch -блоков крайне важен. Первыми идут наиболее специализированные обработчики, далее по мере возрастания универсальности. Так, вначале должен идти обработчик исключения DivideByZeroException, а уже за ним - ArithmeticException. Универсальный обработчик, если он есть, должен стоять последним. За этим наблюдает статический контроль типов. Если потенциальных захватчиков в списке catch -блоков нет (сам список может отсутствовать), то происходит переход к списку обработчиков охватывающего блока. Напомню, что try -блок может быть вложен в другой try -блок. Когда же будет исчерпаны списки вложенных блоков, а потенциальный захватчик не будет найден, то произойдет подъем по стеку вызовов. На рис. 1 показана цепочка вызовов, начинающаяся с точки "большого взрыва" - процедуры Main.

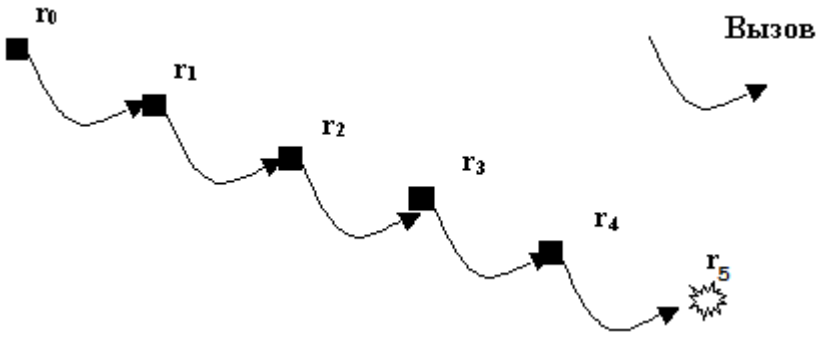


Рис. 1. Цепочка вызовов, хранящаяся в стеке вызовов

Исключение возникло в последнем вызванном методе цепочки - на рисунке метод r5. Если у этого метода не нашлось обработчиков события, способных обработать исключение, то это пытается сделать метод r4, вызвавший r5. Если вызов r5 находится в охраняемом блоке метода r4, то начнет проверяться список обработчиков в охраняемом блоке метода r4. Этот процесс подъема по списку вызовов будет продолжаться, пока не будет найден обработчик, способный захватить исключение, или не будет достигнута начальная точка - процедура Main. Если и в ней нет потенциального захватчика исключения, то сработает стандартный обработчик, прерывающий выполнение программы с выдачей соответствующего сообщения.

Параллельная работа обработчиков исключений

Обработчику исключения - catch-блоку, захватившему исключение, - передается текущее исключение. Анализируя свойства этого объекта, обработчик может понять причину, приведшую к возникновению исключительной ситуации, попытаться ее исправить и в случае успеха продолжить вычисления. Заметьте, в принятой C# схеме без возобновления обработчик исключения не возвращает управление try-блоку, а сам пытается решить проблемы. После завершения catch-блока выполняются операторы текущего метода, следующие за конструкцией try-catch-finally.

Зачастую обработчик исключения не может исправить ситуацию или может выполнить это лишь частично, предоставив решение оставшейся части проблем вызвавшему методу - предшественнику в цепочке вызовов. Механизм, реализующий такую возможность - это тот же механизм исключений. Как правило, в конце своей работы обработчик исключения выбрасывает исключение, выполняя оператор throw. При этом у него есть две возможности: повторно выбросить текущее исключение или выбросить новое исключение, содержащее дополнительную информацию.

Некоторые детали будут пояснены позже при рассмотрении примеров.

Таким образом, обработку возникшей исключительной ситуации могут выполнять несколько обработчиков, принадлежащие разным уровням цепочки вызовов.

Блок finally

До сих пор ничего не было сказано о важном участнике схемы обработки исключений - блоке finally. Напомню, рассматриваемая схема является схемой без возобновления. Это означает, что управление вычислением неожиданно покидает try-блок. Просто так этого делать нельзя - нужно выполнить определенную чистку. Прежде всего удаляются все локальные объекты, созданные в процессе работы блока. В языке C++ эта работа требовала вызова деструкторов объектов. В C#, благодаря автоматической сборке мусора, освобождением памяти можно не заниматься, достаточно освободить стек. Но в блоке try могли быть заняты другие ресурсы - открыты файлы, захвачены некоторые устройства. Освобождение ресурсов, занятых try-блоком, выполняет finally-блок. Если он присутствует, он выполняется всегда, сразу же после завершения работы try-блока, как бы последний ни завершился. Блок try может завершиться вполне нормально без всяких происшествий и управление достигнет конца блока, выполнение может быть прервано оператором throw, управление может быть передано другому блоку из-за выполнения таких операторов как goto, return - во всех этих случаях, прежде чем управление будет передано по предписанному назначению (в том числе, прежде чем произойдет захват исключения), предварительно будет выполнен finally-блок, который освобождает ресурсы, занятые try-блоком, а параллельно будет происходить освобождение стека от локальных переменных.

Схема Бертрана обработки исключительных ситуаций

Схема обработки исключительных ситуаций, предложенная в языке C#, обладает одним существенным изъяном - ее можно применить некорректно. Она позволяет, в случае возникновения исключительной ситуации, уведомить о ее возникновении и спокойно продолжить работу, что в конечном счете приведет к неверным результатам. Из двух зол - прервать вычисление с уведомлением о невозможности продолжения работы или закончить вычисления с ошибочным результатом вычисления - следует выбирать первое. Некорректно примененная схема C# приведет к ошибочным результатам. Приведу несколько примеров. Представьте, оформляется заказ на отдых где-нибудь на Канарах. В ходе оформления возникает исключительная ситуация - нет свободных мест в гостинице - обработчик исключения посылает уведомление с принесением извинений, но оформление заказа продолжается. Вероятно, предпочтительнее отказаться от отдыха на Канарах и выбрать другое место, чем оказаться без крыши над головой, ночуя на берегу океана. Эта ситуация не является критически важной. А что, если в процессе подготовки операции выясняется, что проведение ее в данном случае опасно? Никакие извинения не могут избавить от вреда, нанесенного операцией. Операция должна быть отменена.

Бертран Мейер в книге [1], в которой все механизмы, используемые в объектной технологии, тщательно обосновываются, предложил следующую схему обработки исключительных ситуаций. В основе ее лежит подход к проектированию программной системы на принципах Проектирования по Контракту. Модули программной системы, вызывающие друг друга, заключают между собой контракты. Вызывающий модуль обязан обеспечить истинность предусловия, необходимого

для корректной работы вызванного модуля. Вызванный модуль обязан гарантировать истинность постусловия по завершении своей работы. Если в вызванном модуле возникает исключительная ситуация, то это означает, что он не может выполнить свою часть контракта. Что должен делать обработчик исключительной ситуации? У него только две возможности - Retry и Rescue. Первая (Retry) - попытаться внести некоторые коррективы и вернуть управление охраняемому модулю, который может предпринять очередную попытку выполнить свой контракт. Модуль может, например, в следующей попытке запустить другой алгоритм, использовать другой файл, другие данные. Если все закончится успешно и работа модуля будет соответствовать его постусловию, то появление исключительной ситуации можно рассматривать как временные трудности, успешно преодоленные. Если же ситуация возникает вновь и вновь, тогда обработчик события применяет вторую стратегию (Rescue), выбрасывая исключение и передавая управление вызывающему модулю, который и должен теперь попытаться исправить ситуацию. Важная тонкость в схеме, предложенной Бертраном, состоит в том, что исключение, выбрасываемое обработчиком, следует рассматривать не как панику, не как бегство, а как отход на заранее подготовленные позиции. Обработчик исключения должен позаботиться о восстановлении состояния, предшествующего вызову модуля, который привел к исключительной ситуации, и это гарантирует нахождение всей системы в корректном состоянии.

Схема Бертрана является схемой с возобновлением, и она наиболее точно описывает разумную стратегию обработки исключительных ситуаций. Не следует думать, что эта схема не может быть реализована на C#, просто она требует понимания сути и определенной структурной организации модуля. Приведу возможную реализацию такой схемы на C#:

```
public void Pattern()
{
        do
        {
                try
                {
                        bool Danger = false;
                        Success = true;
                        MakeJob();
                        Danger = CheckDanger();
                        if (Danger)
                                throw (new MyException());
                        MakeLastJob();
                }
                catch (MyException me)
                {
                        if(count > maxcount)
                                throw(new MyException("Три попытки были
                                        безуспешны"));
                                        Success = false; count++;
                        //корректировка ситуации
                        Console.WriteLine("Попытка исправить ситуацию!");
                        level +=1;
                }
        }while (!Success);
}
```

Рассмотрим комментарии к этой процедуре, которую можно рассматривать как некоторый образец организации обработки исключительной ситуации:

Конструкция try-catch блоков помещается в цикл do-while(!Success), завершаемый в случае успешной работы охраняемого блока, за чем следит булева переменная Success.

В данном образце предполагается, что в теле охраняемого блока анализируется возможность возникновения исключительной ситуации и, в случае обнаружения опасности, выбрасывается собственное исключение, класс которого задан программно. В

соответствии с этим тело try -блока содержит вызов метода MakeJob, выполняющего некоторую часть работы, после чего вызывается метод CheckDanger, выясняющий, не возникла ли опасность нарушения спецификации и может ли работа быть продолжена. Если все нормально, то выполняется метод MakeLastJob, выполняющий заключительную часть работы. Управление вычислением достигает конца try -блока, он успешно завершается и, поскольку остается истинной переменная Success, значение true которой установлено в начале try -блока, то цикл while, окаймляющий охраняемый блок и его обработчиков исключений, также успешно завершается.

Если в методе CheckDanger выясняется, что нормальное продолжение вычислений невозможно, то выбрасывается исключение класса MyException. Оно перехватывается обработчиком исключения, стоящим за try -блоком, поскольку класс MyException указан как класс формального аргумента.

Для простоты приведен только один catch -блок. В общем случае их может быть несколько, но все они строятся по единому образцу. Предполагается, что обработчик исключения может сделать несколько попыток исправить ситуацию, после чего повторно выполняется охраняемый блок. Если же число попыток, за которым следит переменная count, превосходит максимально допустимое, то обработчик выбрасывает новое исключение, задавая дополнительную информацию и передавая тем самым обработку ошибки на следующий уровень - вызываемой программе.

Когда число попыток еще не исчерпано, обработчик исключения переменной Success дает значение false, гарантирующее повтор выполнения try -блока, увеличивает счетчик числа попыток и пытается исправить ситуацию.

Как видите, эта схема реализует два корректных исхода обработки исключительной ситуации - Retry и Rescue - повтор с надеждой выполнить обязательства и передачи управления вызывающей программе, чтобы она предприняла попытки исправления ситуации, когда вызванная программа не смогла с этим справиться.

Доведем этот образец до реально работающего кода, где угроза исключения зависит от значения генерируемого случайного числа, а обработчик исключения может изменять границы интервала, повышая вероятность успеха.

Определим первым делом собственный класс исключений:

```
public class MyException :Exception
{
        public MyException()
        {}
        public MyException (string message) : base(message)
        {}
        public MyException (string message, Exception e) :
                base(message, e)
        {}
}
```

Минимум того, что нужно сделать, определяя свои исключения, - это задать три конструктора класса, вызывающие соответствующие конструкторы базового класса Exception.

В классе Excepts, методом которого является наш образец Pattern, определим следующие поля класса:

```
Random rnd = new Random();
               int level = -10;
               bool Success; //true - нормальное завершение
               int count =1; // число попыток выполнения
               const int maxcount =3;
```

Определим теперь методы, вызываемые в теле *охраняемого блока*:

```
void MakeJob()
{
```

```
        Console.WriteLine("Подготовительные работы завершены");
}
bool CheckDanger()
{
        //проверка качества и возможности продолжения работ
        int low = rnd.Next(level,10);
        if ( low > 6) return(false);
        return(true);
}
void MakeLastJob()
{
        Console.WriteLine("Все работы завершены успешно");
}
```

В классе `Testing` зададим метод, вызывающий метод `Pattern`:

```
public void TestPattern()
{
        Excepts ex1 = new Excepts();
        try
        {
                ex1.Pattern();
        }
        catch (Exception e)
        {
                Console.WriteLine("исключительная ситуация при
                        вызове Pattern");
                Console.WriteLine(e.ToString());
        }
}
```

Обратите внимание, что вызов метода Pattern находится внутри охраняемого блока. Поэтому, когда Pattern не справится с обработкой исключительной ситуации, ее обработку возьмет на себя универсальный обработчик, стоящий за try -блоком.

На рис. 2 показаны три варианта запуска метода TestPattern. В одном из них исключительной ситуации при вызове метода Pattern вообще не возникало, в другом - ситуация возникала, но коррекция обработчика исключения помогла и при повторе выполнения охраняемого блока в Pattern все прошло нормально. В третьем варианте метод Pattern не смог справиться с исключительной ситуацией, и она обрабатывалась в catch -блоке метода TestPattern.

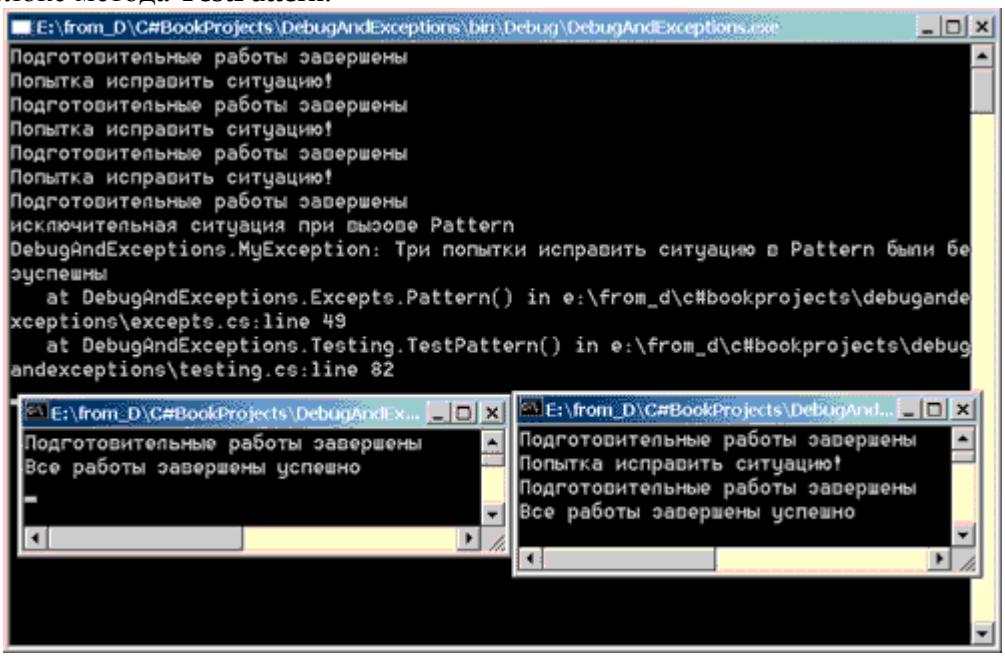


Рис. 2. Обработка исключительных ситуаций. Три случая

Класс Exception

Рассмотрим устройство базового класса Exception, чтобы понять, какую информацию может получить обработчик исключения, когда ему передается объект, задающий текущее исключение.

Основными свойствами класса являются:

- Message - строка, задающая причину возникновения исключения. Значение этого свойства устанавливается при вызове конструктора класса, когда создается объект, задающий исключение;
- HelpLink - ссылка (URL) на файл, содержащий подробную справку о возможной причине возникновения исключительной ситуации и способах ее устранения;
- InnerException - ссылка на внутреннее исключение. Когда обработчик выбрасывает новое исключение для передачи обработки на следующий уровень, то текущее исключение становится внутренним для вновь создаваемого исключения;
- Source - имя приложения, ставшего причиной исключения;
- StackTrace - цепочка вызовов - методы, хранящиеся в стеке вызовов в момент возникновения исключения;
- TargetSite - метод, выбросивший исключение.

Из методов класса отметим метод GetBaseException. При подъеме по цепочке вызовов он позволяет получить исходное исключение -- первопричину возникновения последовательности выбрасываемых исключений.

Класс имеет четыре конструктора, из которых три уже упоминались. Один из них - конструктор без аргументов, второй - принимает строку, становящуюся свойством Message, третий - имеет еще один аргумент: исключение, передаваемое свойству InnerException.

В предыдущий пример я внес некоторые изменения. В частности, добавил еще один аргумент при вызове конструктора исключения в catch -блоке метода Pattern:

```
throw(new MyException("Все попытки Pattern безуспешны", me));
```

В этом случае у создаваемого исключения заполняется свойство InnerException. Для слежения за свойствами исключений я добавил метод печати всех свойств, вызываемый во всех обработчиках исключений:

```
static public void PrintProperties(Exception e)
{
        Console.WriteLine("Свойства исключения:");
        Console.WriteLine("TargetSite = {0}", e.TargetSite);
        Console.WriteLine("Source = {0}", e.Source);
        Console.WriteLine("Message = {0}",e.Message);
        if (e.InnerException == null)
                Console.WriteLine("InnerException = null");
        else Console.WriteLine("InnerException = {0}",
                e.InnerException.Message);
        Console.WriteLine("StackTrace = {0}", e.StackTrace);
        Console.WriteLine("GetBaseException = {0}",
                e.GetBaseException());
}
```

Корректное применение механизма исключений должно поддерживаться целенаправленными усилиями программиста. Следует помнить о двух важных правилах:

- обработка исключений должна быть направлена не столько на уведомление о возникновении ошибки, сколько на корректировку возникшей ситуации;

– если исправить ситуацию не удается, то программа должна быть прервана так, чтобы не были получены некорректные результаты, не удовлетворяющие спецификациям программы.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Как создается защищенный блок кода?
2. Как описывается процедура обработки конкретного исключения?
3. Как генерируется исключение?
4. Как можно ограничить список исключений, которые могут генерироваться в функции?

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Инструментарий тестирования и анализа качества программных средств»

Лабораторная работа № 5. Применение отладочных классов в проекте

Цель: изучить отладочные классы в проекте.

Выполнив работу, Вы будете:

уметь:

- У2 использовать методы для получения кода с заданной функциональностью и степенью качества.
- У6. определять источники и приемники данных.
- У8. выполнять отладку, используя методы и инструменты условной компиляции (классы Debug и Trace)).
- У11. выявлять ошибки в системных компонентах на основе спецификаций.
- У12 использовать различные транспортные протоколы и стандарты форматирования сообщений.
- У15 создавать классы-исключения на основе базовых классов.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание:

Для выбранного по индивидуальному заданию программного продукта разработать техническое задание в соответствии с ГОСТ 19.201-78, предполагая, что сначала разрабатывается ТЗ, а затем будет написана программа для ТЗ. Отчет по лабораторной работе должен содержать разделы технического задания.

Краткие теоретические сведения:

Классы Debug и Trace

Атрибут условной компиляции Conditional характеризует метод, но не отдельный оператор. Иногда хотелось бы иметь условный оператор печати, не создавая специального метода, как это было сделано в предыдущем примере. Такую возможность и многие другие полезные свойства предоставляют классы Debug и Trace.

Классы Debug и Trace - это классы-двойники. Оба они находятся в пространстве имен Diagnostics, имеют идентичный набор статических свойств и методов с идентичной семантикой. В чем же разница? Методы класса Debug имеют атрибут условной компиляции с константой DEBUG, действуют только в Debug-конфигурации проекта и игнорируются в Release-конфигурации. Методы класса Trace включают два атрибута Conditional с константами DEBUG и TRACE и действуют в обеих конфигурациях.

Одна из основных групп методов этих классов - методы печати данных: Write, WriteIf, WriteLine, WriteLineIf. Методы перегружены, в простейшем случае позволяют выводить некоторое сообщение. Методы со словом If могут сделать печать условной, задавая условие печати в качестве первого аргумента метода, что иногда крайне полезно. Методы со словом Line дают возможность дополнять сообщение символом перехода на новую строку.

По умолчанию методы обоих классов направляют вывод в окно Output. Однако это не всегда целесообразно, особенно для Release-конфигурации. Замечательным свойством методов классов Debug и Trace является то, что они могут иметь много "слушателей", направляя вывод каждому из них. Свойство Listeners этих классов возвращает разделяемую обоими классами коллекцию слушателей - TraceListenerCollection. Как и всякая коллекция, она имеет ряд методов для добавления новых слушателей: Add, AddRange, Insert - и возможность удаления слушателей: Clear, Remove, RemoveAt и другие методы. Объекты этой коллекции в

качестве предка имеют абстрактный класс TraceListener. Библиотека FCL включает три неабстрактных потомка этого класса:

- DefaultTraceListener - слушатель этого класса, добавляется в коллекцию по умолчанию, направляет вывод, поступающий при вызове методов классов Debug и Trace, в окно Output;
- EventLogTraceListener - посылает сообщения в журнал событий Windows;
- TextWriterTraceListener - направляет сообщения объектам класса TextWriter или Stream ; обычно один из объектов этого класса направляет вывод на консоль, другой - в файл.

Можно и самому создать потомка абстрактного класса, предложив, например, XML-слушателя, направляющего вывод в соответствующий XML-документ. Как видите, система управления выводом очень гибкая, позволяющая получать и сохранять информацию о ходе вычислений в самых разных местах.

Помимо свойства Listeners и методов печати, классы Debug и Trace имеют и другие важные методы и свойства:

- Assert и Fail, проверяющие корректность хода вычислений - о них мы поговорим особо;
- Flush - метод, отправляющий содержание буфера слушателю (в файл, на консоль и так далее). Следует помнить, что данные буферизуются, поэтому применение метода Flush зачастую необходимо, иначе метод может завершиться, а данные останутся в буфере;
- AutoFlush - булево свойство, указывающее, следует ли после каждой операции записи данные из буфера направлять в соответствующий канал. По умолчанию свойство выключено, и происходит только буферизация данных;
- Close - метод, опустошающий буфера и закрывающий всех слушателей, после чего им нельзя направлять сообщения.

У классов есть и другие свойства и методы, позволяющие, например, заниматься структурированием текста сообщений.

Рассмотрим пример работы, в котором отладочная информация направляется в разные каналы - окно вывода, консоль, файл:

```
public void Optima()
{
        double x, y=1;
        x= y - 2*Math.Sin(y);
        FileStream f = new FileStream("Debuginfo.txt",
                FileMode.Create, FileAccess.Write);
        TextWriterTraceListener writer1 =
                new TextWriterTraceListener(f);
        TextWriterTraceListener writer2 =
                new TextWriterTraceListener(System.Console.Out);
        Trace.Listeners.Add( writer1);
        Debug.Listeners.Add( writer2);
        Debug.WriteLine("Число слушателей:" +
                Debug.Listeners.Count);
        Debug.WriteLine("автоматический вывод из буфера:"+
                Trace.AutoFlush);
        Trace.WriteLineIf(x<0, "Trace: " + "x= " + x.ToString()
                 + " y = " + y);
        Debug.WriteLine("Debug: " + "x= " + x.ToString() +
                " y = " + y);
        Trace.Flush();
        f.Close();
}
```

В коллекцию слушателей вывода к слушателю по умолчанию добавляются еще два слушателя класса TextWriterTraceListener. Заметьте, что хотя они добавляются методами разных классов Debug и Trace, попадают они в одну коллекцию. Как и обещано, один из

этих слушателей направляет вывод в файл, другой на консоль. На рис. 1 на фоне окна кода показаны три канала вывода - окно Output, консоль, файл - содержащие одну и ту же информацию.

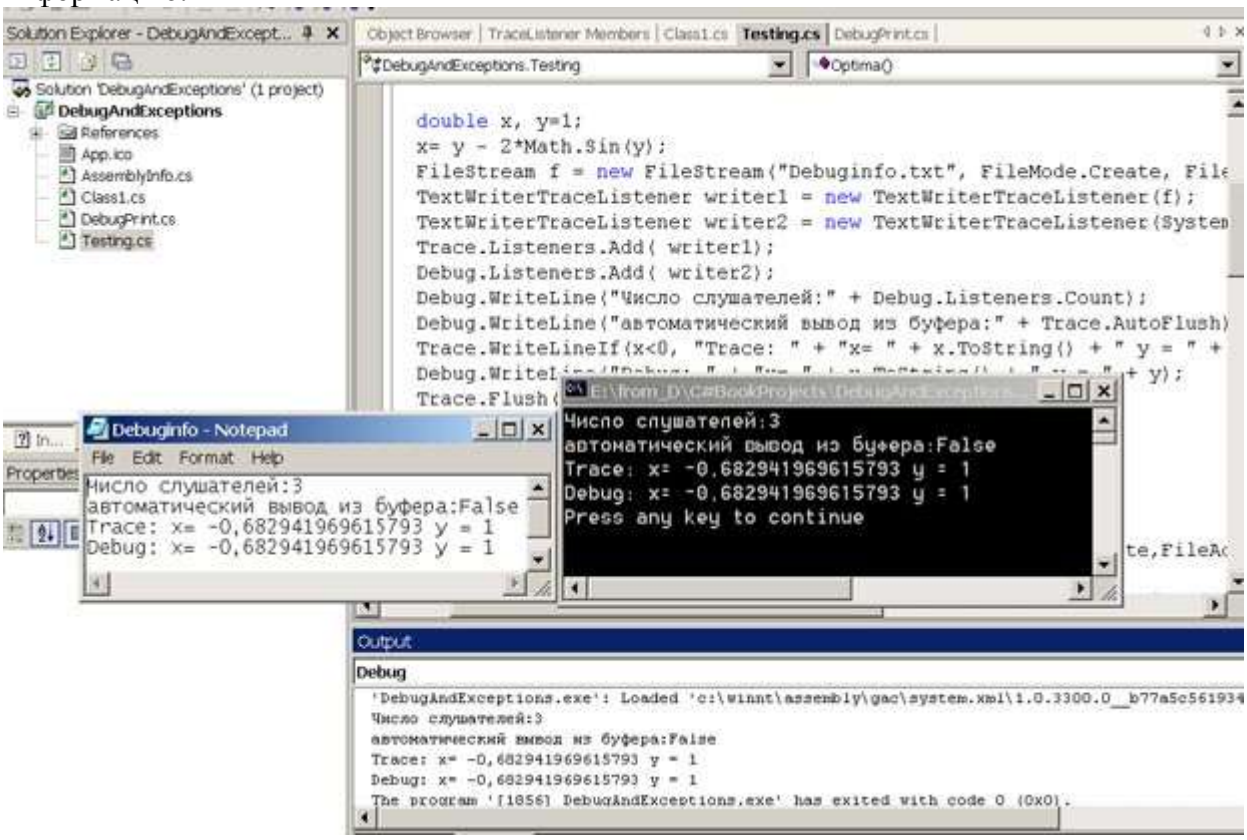


Рис. 1. Три канала вывода

Метод Флойда и утверждения Assert

Раннее большие надежды возлагались на формальные методы доказательства правильности программ, позволяющие доказывать корректность программ аналогично доказательству теорем. Реальные успехи формальных доказательств невелики. Построение такого доказательства не проще написания корректной программы, а ошибки столь же возможны и часты, как и ошибки программирования. Тем не менее, эти методы оказали серьезное влияние на культуру проектирования корректных программ, появление в практике программирования понятий предусловия и постусловия, инвариантов и других важных понятий.

Одним из методов доказательства правильности программ был метод Флойда, при котором программа разбивалась на участки, окаймленные утверждениями - булевскими выражениями (предикатами). Истинность начального предиката должна была следовать из входных данных программы. Затем для каждого участка доказывалось, что из истинности предиката, стоящего в начале участка, после завершения выполнения соответствующего участка программы гарантируется истинность следующего утверждения - предиката в конце участка. Конечный предикат описывал постусловие программы.

Схема Флойда используется на практике, по крайней мере, программистами, имеющими вкус к строгим методам доказательства. Утверждения становятся частью программного текста. Само доказательство может и не проводиться: чаще всего у программиста есть уверенность в справедливости расставленных утверждений и убежденность, что при желании он мог бы провести и строгое доказательство. В C# эта схема поддерживается тем, что классы Debug и Trace имеют метод Assert, аргументом которого является утверждение. Что происходит, когда вычисление достигает

соответствующей точки и вызывается метод Assert? Если истинно булево выражение в Assert, то вычисления продолжаются, не оказывая никакого влияния на нормальный ход вычислений. Если оно ложно, то корректность вычислений под сомнением, их выполнение приостанавливается и появляется окно с уведомлением о произошедшем событии, что показано на рис. 2.

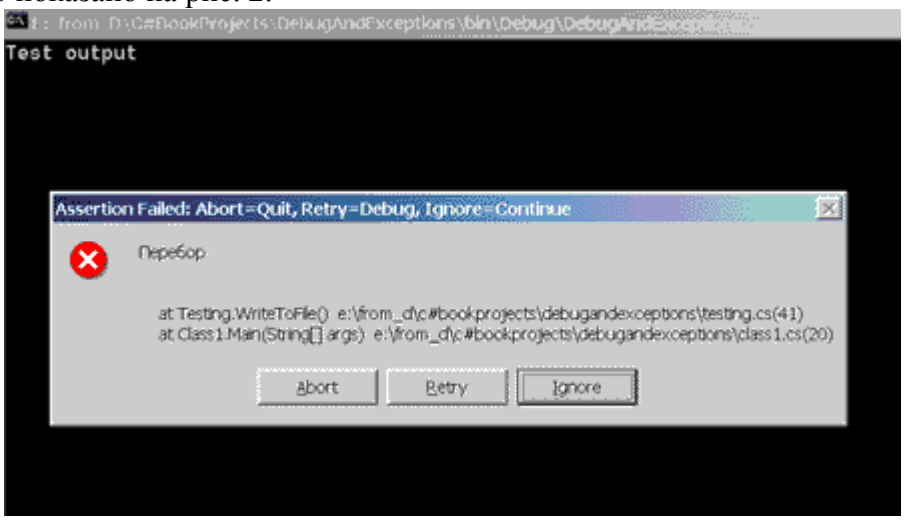


Рис. 2. Нарушение утверждения Assert

В этой ситуации у программиста есть несколько возможностей:

- прервать выполнение, нажав кнопку Abort;
- перейти в режим отладки (Retry);
- продолжить вычисления, проигнорировав уведомление.

В последнем случае сообщение о возникшей ошибке будет послано всем слушателям коллекции TraceListenerCollection.

Рассмотрим простой пример, демонстрирующий нарушение утверждения:

```
public void WriteToFile()
{
        Stream myFile = new
         FileStream("TestFile.txt",FileMode.Create,FileAccess.Write);
        TextWriterTraceListener myTextListener =
                new TextWriterTraceListener(myFile);
        int y = Debug.Listeners.Add(myTextListener);
        TextWriterTraceListener myWriter =
                new TextWriterTraceListener(System.Console.Out);
        Trace.Listeners.Add(myWriter);
        Trace.AutoFlush = true;
        Trace.WriteLine("автоматический вывод из буфера:"
                + Trace.AutoFlush);
        int x = 22;
        Trace.Assert(x<=21, "Перебор");
        myWriter.WriteLine("Вывод только на консоль");
        //Trace.Flush();
        //Вывод только в файл
        byte[] buf = {(byte)'B',(byte)'y'};
        myFile.Write(buf,0, 2);
        myFile.Close();
}
```

Как и в предыдущем примере, здесь создаются два слушателя, направляющие вывод отладочных сообщений на консоль и в файл. Когда произошло нарушение утверждения Assert, оно было проигнорировано, но сообщение о нем автоматически было направлено всем слушателям. Метод также демонстрирует возможность параллельной работы с консолью и файлом. На рис. 3 показаны результаты записи в файл:

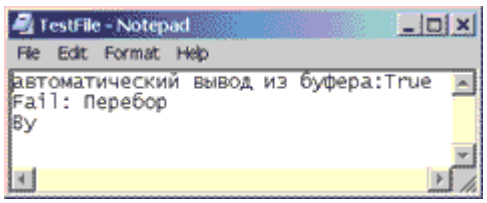


Рис. 3. Файл с записью сообщения о нарушении утверждения Assert

Вариацией метода Assert является метод Fail, всегда приводящий к появлению окна с сообщением о нарушении утверждения, проверка которого осуществляется обычным программным путем.

Классы StackTrace и BooleanSwitch

В библиотеке FCL имеются и другие классы, полезные при отладке. Класс StackTrace позволяет получить программный доступ к стеку вызовов. Класс BooleanSwitch предоставляет механизм, аналогичный константам условной компиляции. Он разрешает определять константы, используемые позже в методе условной печати WriteIf классов Debug и Trace. Мощь этого механизма в том, что константы можно менять в файле конфигурации проекта, не изменяя код проекта и не требуя его перекомпиляции.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Назначение технического задания?
2. Кто составляет и утверждает ТЗ?
3. На каком этапе разработки программного изделия составляется ТЗ?
4. Какими документами регламентируется написание ТЗ?

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Инструментарий тестирования и анализа качества программных средств»

Лабораторная работа № 6. Отладка проекта

Цель: закрепление практических навыки работы с системой Visual Studio 2019; научиться использовать инструментальные средства, помогающие провести отладку приложений.

Выполнив работу, Вы будете:

уметь:

- У6. определять источники и приемники данных.
- У8. выполнять отладку, используя методы и инструменты условной компиляции (классы Debug и Trace)).
- У17 использовать инструментальные средства отладки программных продуктов.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание:

1. Создание программы по индивидуальному варианту задания.
2. Проведите отладку программы всеми доступными вам средствами среды разработки.

Краткие теоретические сведения:

В C#, как и в других появившихся до .NET языках, главная методика по отладке состоит в добавлении точек останова и изучении того, что происходит в коде в конкретные моменты во время его выполнения.

Точки останова

Точку останова (breakpoint) в Visual Studio можно помещать на любую строку кода, которая в действительности выполняется. Самый простой способ — щелчок на необходимой строке в окне редактора кода внутри затененной области вдоль левого края окна документа (или выделение нужной строки и нажатие клавиши <F9>). Это приводит к размещению в данной строке точки останова, которая вызывает прерывание процесса выполнения и передачу управления отладчику. Как и в предыдущих версиях Visual Studio, точка останова обозначается большим кружком слева от соответствующей строки в окне редактора кода. Кроме того, Visual Studio выделяет саму строку, отображая ее текст и фон разными цветами. Щелчок на кружке приводит к удалению точки останова.

Если останов на определенной строке каждый раз не подходит для решения имеющейся проблемы, можно создать так называемую условную точку останова. Для этого выберите в меню Debug (Отладка) пункт Windows --- Breakpoints (Окно --- Точки останова). Откроется диалоговое окно, позволяющее указать желаемые детали для точки останова. В этом окне можно выполнять следующие действия:

- Указать, что выполнение должно прерываться лишь после прохождения точки останова определенное количество раз.
- Указать, что точка останова должна вступать в действие при каждом n-ном достижении строки, например, при каждом 20-м ее выполнении (это удобно при отладке больших циклов).
- Задать точки останова относительно переменных, а не команд. В таком случае наблюдение будет вестись за значением указанной переменной, и точки останова будут активизироваться при каждом изменении значения этой переменной. Однако этот вариант может сильно замедлить выполнение кода, поскольку на проверку, не изменилось ли значение отслеживаемой переменной после выполнения каждой очередной инструкции, будет тратиться дополнительное время процессора.

Слежения

После срабатывания точки останова обычно необходимо просмотреть значения переменных. Проще всего это сделать, наведя курсор мыши на имя интересующей переменной прямо в окне редактора кода. Это приводит к появлению небольшого всплывающего окошка, в котором показано значение данной переменной; это окошко можно развернуть, чтобы просмотреть дополнительные детали.

Для просмотра значений переменных можно также использовать *окно Autos (Автоматические)*. Окно Autos представляет собой окно с вкладками, которое появляется лишь тогда, когда программа выполняется в режиме отладки. Если вы его не видите, попробуйте выбрать в меню Debug (Отладка) пункт Windows --- Autos (Окна --- Автоматические).

В этом окне рядом с переменными, которые являются классами или структурами, отображается пиктограмма с изображением знака "плюс", щелчок на которой позволяет развернуть представление переменной и просмотреть значения ее полей.

Три предлагаемых в этом окне вкладки предназначены для наблюдения за переменными трех разных категорий:

- Вкладка *Autos (Автоматические)* позволяет просматривать значения нескольких последних переменных, к которым осуществлялся доступ в процессе выполнения программы.
- Вкладка *Locals (Локальные)* позволяет просматривать значения переменных, к которым получается доступ в методе, выполняемом в текущий момент
- Вкладка *Watch (Слежение)* позволяет просматривать значения любых интересующих переменных за счет явного указания их имен непосредственно в окне Watch.

Исключения

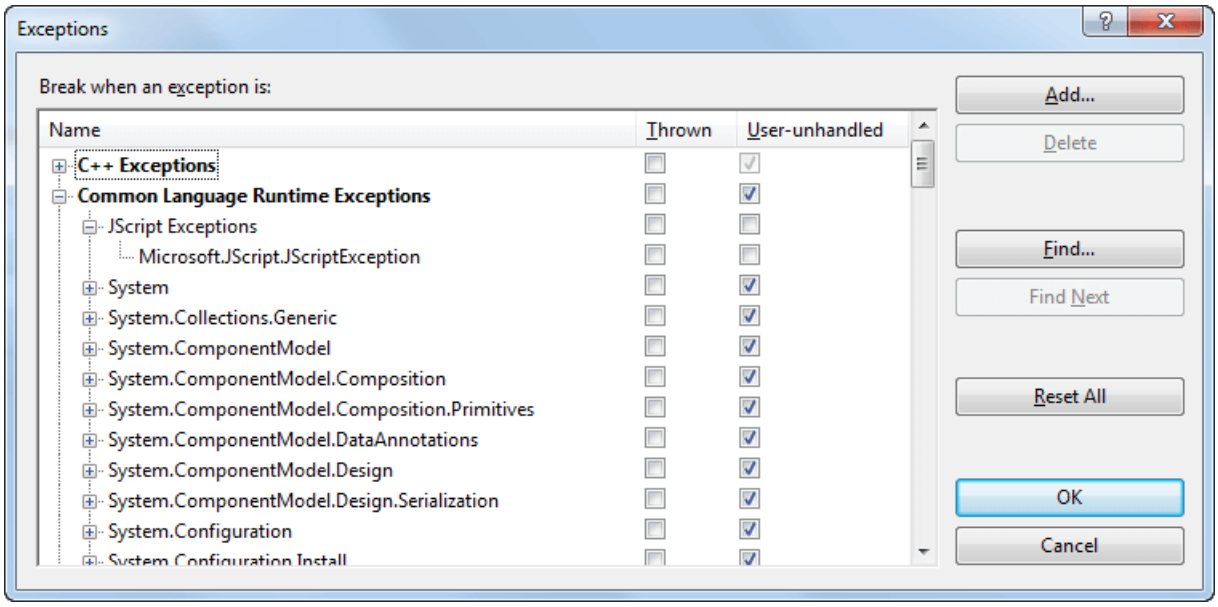
Исключения являются замечательным средством для обеспечения надлежащей обработки ошибок в поставляемом приложении. В случае правильного применения они позволяют обрести уверенность в том, что приложению удастся справиться с трудностями, а перед пользователем никогда не появится диалоговое окно с техническим описанием неполадки. К сожалению, во время отладки исключения не столь замечательны. На то имеются *две причины*:

- если исключение возникает во время отладки, часто не нужно, чтобы оно обрабатывалось автоматически, особенно если это подразумевает аккуратное завершение программы. Напротив, отладчик должен помочь выяснить, по какой причине возникло это исключение. Конечно же, трудность состоит в том, что в случае написания качественного надежного и отказоустойчивого кода, программа будет автоматически обрабатывать практически все, в том числе и неполадки, которые требуется обнаружить.
- если возникло исключение, для которого не было предусмотрено обработчика, исполняющая среда .NET все равно будет пытаться его найти. К сожалению, к моменту, когда она обнаружит, что обработчик не существует, выполнение программы будет завершаться. Никакого стека вызовов, следовательно, не останется, и просматривать значения каких-либо переменных будет невозможно, потому что все они будут находиться за пределами области видимости.

Конечно, можно устанавливать точки останова в блоках *catch*, но это часто особо не помогает, поскольку при достижении блока catch поток управления по определению покинет соответствующий блок *try*. Это означает, что переменные, значения которых, скорее всего, следовало изучить для выяснения того, что пошло не так, покинут область видимости. Не будет даже возможности просматривать трассировочные данные стека для выяснения, какой метод выполнялся во время срабатывания оператора *throw*, поскольку управление уже покинет этот метод. Разумеется, помещение точки останова в оператор throw позволит решить эту проблему, но надо учитывать, что при написании кода

защищенным образом операторов throw будет в коде очень много. Как тогда угадать, какой из них срабатывает и приводит к генерации исключения?

На самом деле в Visual Studio предлагается очень действенное решение. Если заглянуть в **меню Debug (Отладка)**, то можно будет обнаружить там пункт *Exceptions (Исключения)*. В случае выбора этого пункта открывается диалоговое окно Exceptions (Исключения). Это окно позволяет указывать, что должно происходить при выдаче исключения. Здесь можно указать, что выполнение должно продолжаться или же останавливаться с переходом в режим отладки, в случае чего произойдет останов, а отладчик окажется прямо на самом операторе throw:



Visual Studio известно обо всех классах исключений, которые доступны в базовых классах .NET, и о многих таких исключениях, которых могут выдаваться за пределами среды .NET. Распознавать автоматически специальные классы исключений, создаваемые разработчиками, Visual Studio не умеет, но позволяет вручную добавлять такие классы исключений в список и, следовательно, указывать, какие из таких исключений должны приводить к немедленному прекращению выполнения приложения. Для этого необходимо щелкнуть на кнопке *Add (Добавить)*, которая активизируется при выборе самого верхнего узла в дереве, и ввести имя специального класса исключения.

Дополнительные команды отладки исходного кода

Компиляция практически всего коммерческого программного обеспечения на стадии отладки и на стадии подготовки окончательной версии продукта должна проводиться немного по-разному. Среда Visual Studio способна понимать это, поскольку сохраняет информацию обо всех параметрах, которые ей надлежит передавать компилятору. Для поддержки разных вариантов компоновки проекта Visual Studio потребуется сохранять подобную информацию в более чем одном экземпляре. Разные экземпляры такой информации называются конфигурациями. При создании проекта Visual Studio автоматически предлагает на выбор две таких конфигурации, которые называются, соответственно, **Debug (Отладка)** и **Release (Выпуск)**:

- *Конфигурация Debug* обычно указывает, что никакие операции по оптимизации выполняться не должны, в исполняемом коде должна присутствовать дополнительная отладочная информация, а компилятор должен предполагать, что в коде определен препроцессорный символ отладки Debug, если только он не был явно отменен с помощью директивы #undefined.
- *Конфигурация Release* указывает, что компилятор должен проводить в отношении компилируемого кода оптимизацию, в исполняемом коде не должно

присутствовать никакой дополнительной информации, а компилятор не должен предполагать наличие препроцессорного символа Debug.

Можно также определять собственные конфигурации. Это необходимо, например, для компоновки приложения с несколькими отличающимися версиями. Раньше из-за проблем, связанных с поддержкой кодировки Unicode в Windows NT, но не в Windows 95, для многих проектов на C++ было принято создавать две конфигурации — одну для Unicode, а вторую для *MBCS* (multibyte character set — набор многобайтных символов).

Выполнение работы:

1. Разработайте проект в соответствии с вариантом.

2. перейдите в режим отладки. При остановке выполнения программы добавьте в окно просмотра наименования нескольких интересующих Вас переменных.

3. Вызовите окно просмотра. Далее выполняйте программу по шагам. В окне просмотра можно наблюдать интересующие Вас переменные.

4. Установите в программе несколько контрольных точек и запустите программу на выполнение.

5. Просмотрите значения интересующих Вас переменных с помощью окна просмотра, так же как это было предложено в предыдущем пункте.

6. При остановке выполнения программы откройте окно и просмотрите значения интересующих Вас переменных. Измените значение какой-нибудь переменной, записав новое в окно.

7. Выполните такое же задания, используя точки останова (Breakpoint).

8. Продолжите отладку разрабатываемой программы, используя навыки работы с отладчиком.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Что такое отладка?
2. Какие инструменты отладки вам известны?
3. Методы отладки.

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Инструментарий тестирования и анализа качества программных средств»

Лабораторная работа № 7. Инспекция кода модулей проекта

Цель: закрепление практических навыков работы с системой Visual Studio 2019; научиться проводить инспекцию кода модулей проекта.

Выполнив работу, Вы будете:

уметь:

- У2 использовать методы для получения кода с заданной функциональностью и степенью качества.
- У5. организовывать заданную интеграцию модулей в программные средства на базе имеющейся архитектуры и автоматизации бизнес-процессов.
- У6. определять источники и приемники данных.
- У8. выполнять отладку, используя методы и инструменты условной компиляции (классы Debug и Trace)).
- У17 использовать инструментальные средства отладки программных продуктов.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание:

1. Изучить теоретическую часть.
2. Выполнить задание, следуя указаниям.
3. Ответить на контрольные вопросы (в устной форме).
4. Предъявить преподавателю результаты работы программы и исходные коды.
5. Оформить отчет в соответствии с ходом работы (тема, цель, условие задачи, программный код, результаты тестирования программы, выводы).

Краткие теоретические сведения:

Не во всех случаях возможна разработка автоматических или хотя бы четко формализованных ручных тестов для проверки функциональности программной системы. В некоторых случаях выполнение программного кода, подвергаемого тестированию, невозможно в условиях, создаваемых тестовым окружением (например, во встроенных системах, если программный код предназначен для обработки исключительных ситуаций, создаваемых только при установке системы на реальное оборудование). В других случаях верифицируется не программный код, а проектная документация на систему, которую нельзя "выполнить" или создать для нее отдельные тестовые примеры. И в тех и в других случаях обычно прибегают к методу экспертных исследований программного кода или документации на *корректность* или *непротиворечивость*.

Такие экспертные исследования обычно называют инспекциями или просмотрами. Существует два типа инспекций – неформальные и формальные.

Формальная инспекция является четко управляемым процессом, структура которого обычно четко определяется соответствующим стандартом проекта. Таким образом, все формальные инспекции имеют одинаковую структуру и одинаковые выходные документы, которые затем используются при разработке.

Факт начала формальной инспекции четко фиксируется в общей базе данных проекта. Также фиксируются документы, подвергаемые инспекции, списки замечаний, отслеживаются внесенные по замечаниям изменения. Этим формальная инспекция похожа на автоматизированное тестирование – списки замечаний имеют много общего с отчетами о выполнении тестовых примеров.

В ходе формальной инспекции группой специалистов осуществляется независимая проверка соответствия инспектируемых документов исходным документам. Независимость проверки обеспечивается тем, что она осуществляется инспекторами, не участвовавшими в разработке инспектируемого документа. Входами процесса формальной инспекции являются инспектируемые документы и исходные документы, а

выходами – материалы инспекции, включающие *список* обнаруженных несоответствий и решение об изменении статуса инспектируемых документов. рис. 1 иллюстрирует *место* формальной инспекции в процессе разработки программных систем.

Рис.1. Место формальной инспекции в процессе разработки программных систем

Выполнение работы:

Задание. Следуя указаниям, создайте программу, которая запрашивает у нового сотрудника имя, фамилию и дату рождения. Вы будете хранить эту информацию в свойствах нового класса с именем Person, и создадите метод класса, который будет вычислять текущий возраст нового сотрудника. Этот проект научит вас создавать собственные классы, экземпляры классов (объекты) а также как использовать эти классы в процедурах событий вашей программы.

Добавление в ваш проект нового класса

Класс, определенный пользователем, позволяет определить в программе ваши собственные объекты, которые имеют свойства, методы и события, точно так же, как объекты, создаваемые на формах Windows с помощью элементов управления из Области элементов. Чтобы добавить в ваш проект новый класс, щелкните в меню Проект (Project) на команде Добавить класс (AddClass), а затем определите этот класс с помощью кода программы и нескольких новых ключевых слов VisualBasic.

Создание проекта PersonClass

1. Запустите *MicrosoftVisualStudio2019,* затем создайте в своей папке новый проект с именем **MyPersonClass.**
2. Используйте элемент управления *Label* и добавьте в верхней части формы Form1 длинную метку.
3. Используйте элемент управления *TextBox* и нарисуйте под меткой два широких текстового поля.
4. Используйте элемент управления *DateTimePicker* и нарисуйте под текстовыми полями объект выбора даты и времени.
5. Используйте элемент управления *Button* и нарисуйте под объектом выбора даты и времени кнопку.
6. Установите для объектов формы следующие свойства:

Объект	Свойство	Установка
Label1	Text	Введите имя, фамилию и дату рождения сотрудника.
TextBox1	Text	Имя
TextBox2	Text	Фамилия
Button1	Text	Отобразить запись
Form1	Text	Класс Person

7. Ваша форма должна выглядеть примерно так.

Это базовый интерфейс пользователя для формы, которая определяет запись нового сотрудника фирмы. (Эта форма не подключена к базе данных, так что храниться может только одна запись.) Теперь вы должны добавить в проект класс для хранения информации из этой записи.

8. Щелкните на команде *Добавить класс(AddClass)* в меню *Проект (Project)*. VisualStudio откроет диалоговое окно *Добавление нового элемента (AddNewItem)*, показанное ниже.

Диалоговое окно Добавление нового элемента дает возможность задать имя вашего класса. Когда вы присвоите имя, обратите внимание, что вы можете сохранить в новом модуле класса несколько классов и указать имя, которое будет для них общим.

9. Введите в текстовом поле *Имя(Name)* имя *Person.vb*, а затем щелкните *Добавить*. *VisualStudio* откроет в Редакторе кода пустой модуль класса и добавит имя файла *Person.vb* в ваш проект в *Обозревателе решений*, как показано на рисунке.

Объявление переменных класса

Под оператором программы PublicClassPerson введите следующие объявления переменных:

Здесь вы объявляете две переменные, которые будут использованы исключительно в модуле класса для хранения значений двух строковых свойств. Переменные объявлены с помощью ключевого слова Private, так как по соглашению VisualBasic программисты должны держать внутренние переменные класса закрытыми - другими словами, недоступными для просмотра извне самого модуля класса.

Создание свойств

1. Под объявлением переменных введите следующий оператор программы и нажмите клавишу (Enter):

Этот оператор создает свойство вашего класса с именем FirstName, которое имеет тип String. Когда вы нажмете (Enter), VisualStudio немедленно создаст структуру кода для остальных элементов объявления свойства. Требуемыми элементами являются: блок Get, который определяет, что программисты увидят, когда будут проверять свойство FirstName, блок Set, который определяет, что произойдет, когда свойство FirstName будет установлено или изменено, и оператор EndProperty, который отмечает конец процедуры свойства.

2. Заполните структуру процедуры свойства так, чтобы она выглядела, как показано ниже.

Ключевое слово Return указывает, что при обращении к свойству FirstName будет возвращена строковая переменная Name1. При установке значения свойства блок Set присваивает переменной Name1 строковое значение. Обратите особое внимание на переменную Value, используемую в процедурах свойств для обозначения значения,

которое присваивается свойству класса при его установке. Хотя этот синтаксис может выглядеть странно, просто поверьте мне - именно так создаются свойства в элементах управления, хотя более сложные свойства будут иметь здесь дополнительную программную логику, которая будет проверять значения и производить вычисления.

3. Под оператором EndProperty введите для свойства LastName вашего класса вторую процедуру свойства. Она должна выглядеть так, как показано ниже.

Эта процедура свойства аналогична первой, за исключением того, что она использует вторую строковую переменную (Name2), которую вы объявили в верхней части кода класса. Вы закончили определять два свойства вашего класса. Теперь перейдем к методу с именем Age, который будет определять текущий возраст нового сотрудника на основе даты рождения.

Создание метода

Под процедурой свойства LastName введите следующее определение функции:

Чтобы создать метод класса, который выполняет некое действие, добавьте в ваш класс процедуру Sub. Хотя многие методы не требуют для выполнения своей работы аргументов, метод Age, определенный мной, требует для своих вычислений аргумент Birthday типа Date. Это метод использует для вычитания даты рождения нового сотрудника из текущей системной даты метод Subtract, и возвращает значение, выраженное в днях, деленных на 365.25 - примерную длину одного года в днях. Функция Int преобразует это значение в целое, и это число с помощью оператора Return возвращается в вызывающую процедуру - как и в случае с обычной функцией.

Определение класса закончено! Вернитесь к форме *Form1* и используйте новый класс в процедуре события.

Совет. Хотя в данном примере это и не делалось, в реальном проекте полезно добавить в модуль класса логику для проверки типов данных. Это делается для того, чтобы неправильное использование свойств или методов, не приводило к возникновению ошибок времени исполнения, из-за которых выполнение программы может прерваться.

Создание объекта с помощью нового класса

1. Щелкните в Обозревателе решений на значке Form1.vb, а затем на кнопкеОткрыть в конструкторе. Появится интерфейс пользователя Form1.

2. Чтобы открыть в Редакторе кода процедуру события Button1_Click, сделайте двойной щелчок мышью на кнопке **Отобразить запись.**

3. Введите следующие операторы программы:

Эта процедура сохраняет в объекте с именем Employee, который имеет тип Person, значения, введенные пользователем. Ключевое слово New указывает, что вы хотите немедленно создать новый экземпляр объекта Employee. Теперь нужно объявить переменную с помощью класса, созданного вами самими! Затем процедура объявляет переменную с именем DOB типа Date. Она будет хранить дату, введенную пользователем, и устанавливает свойства FirstName и LastName объекта Employeeравными имени и фамилии, введенным в два объекта текстовых полей формы. Значение, возвращаемое объектом выбора даты и времени, сохраняется в переменной DOB, а последний оператор программы отображает окно сообщения, содержащее свойства FirstName и LastName, а также возраст нового сотрудника, определенный методом Age, который при передаче в него переменной DOB возвращает целое значение. Как только вы определили класс в модуле класса, его легко можно использовать в процедуре события.

4. Чтобы запустить программу, щелкните на кнопке Начать отладку (F5).В среде разработки появится интерфейс пользователя, готовый к приему ваших данных.

5. Введите в текстовое поле FirstName ваше имя, а в текстовое поле LastName - фамилию.

6. Щелкните на раскрывающемся списке объекта выбора даты и времени, и прокрутите его до вашей даты рождения.

Совет. Вы можете быстро прокрутить список, щелкнув в открытом диалоговом окне объекта выбора даты на поле года. Появятся небольшие стрелки прокрутки, и вы сможете переходить сразу на год вперед или назад. Также можно быстро перейти на нужный вам месяц, щелкнув на поле месяца, а затем на месяце в появившемся меню.

Ваша форма будет выглядеть примерно так.

7. Щелкните на кнопке **Отобразить запись**. Ваша программа сохраняет значения имени и фамилии в свойствах и использует метод Age для вычисления текущего возраста нового сотрудника. Появится диалоговое окно с результатом.

8. Чтобы закрыть это окно сообщения, щелкните на **OK**, а затем поэкспериментируйте с несколькими различными значениями дат, щелкая на **Отобразить запись** каждый раз, когда вы меняете значение поля даты рождения.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Что такое инспекция кода?
2. Какие виды инспекции кода вы знаете?
3. Определите понятия класс, экземпляр класса, объект.
4. На какие этапы можно разбить процесс создания экземпляров класса.
5. Как создать базовый файл класса, какой код он содержит.

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Инструментарий тестирования и анализа качества программных средств»

Лабораторная работа № 8. Тестирование интерфейса пользователя средствами инструментальной среды разработки

Цель: формирование навыков разработки графических пользовательских интерфейсов с использованием средств MS Visual Studio 2010.

Выполнив работу, Вы будете:

уметь:

- У2 использовать методы для получения кода с заданной функциональностью и степенью качества.
- У6. определять источники и приемники данных.
- У9. оценивать размер минимального набора тестов.
- У10. разрабатывать тестовые пакеты и тестовые сценарии.
- У11. выявлять ошибки в системных компонентах на основе спецификаций.
- У13 выполнять тестирование интеграции.
- У16 выполнять ручное и автоматизированное тестирование программного модуля.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание:

1. Определить предметную область и сферу применения программного продукта.
2. Определить целевую аудиторию.
3. Построить описательную модель пользователя (профиль). При необходимости — выделить группы пользователей.
4. Сформировать множество сценариев поведения пользователей на основании составленной модели.
5. Выделить функциональные блоки приложения и схему навигации между ними (структуру диалога).
6. Провести тестирование интерфейса пользователя.

Краткие теоретические сведения:

Тео Мандел в своей работе выделяет четыре этапа разработки пользовательского интерфейса, а именно:

- сбор и анализ информации от пользователей;
- разработка пользовательского интерфейса;
- построение пользовательского интерфейса;
- подтверждение качества пользовательского интерфейса.

Первый шаг – определение профиля пользователя. Профиль пользователя отвечает на вопрос: «Что представляет наш пользователь?». Он позволяет нам составить представление о возрасте, образовании, предпочтениях пользователей.

Второй шаг – анализ стоящих перед пользователями задач.

Анализ стоящих перед пользователями задач – это определение того, чего хотят пользователи и каким образом они собираются решать свои задачи.

Концептуальное проектирование есть определение общей структуры и взаимодействия продукта. По определению Алана Купера, концептуальные принципы проектирования «помогают определить сущность продукта и его место в более широком контексте использования, который требуется пользователям».

Концептуальное проектирование включает:

- определение типа интерфейса будущего приложения (монопольный, временный, фоновый);

– организацию инфраструктуры взаимодействия;

Согласно определению Алана Купера, тип интерфейса определяет поведенческую сущность продукта, то есть то, как он предъявляет себя пользователю. Тип интерфейса – это способ описать то, как много внимания пользователь будет уделять взаимодействию с продуктом, и каким образом продукт будет реагировать на это внимание.

Следует отметить зависимость типа интерфейса от используемой технической платформы: персонального компьютера, Интернет, информационный киоск, мобильное устройство, бытовая техника.

Применительно к программам, которые разрабатываются для современных персональных компьютеров, в литературе также используется термин «настольное приложение».

Интерфейс настольных приложений можно отнести к одному из трёх типов: монопольный, временный и фоновый.

К приложениям *монопольного типа* относятся программы, которые полностью завладевают вниманием пользователя на длительные периоды времени. Для продуктов с монопольным интерфейсом характерна длительная работа в течение длительных отрезков времени. В процессе работа пользователя монопольный продукт является его основным инструментом и преобладает над остальными.

Приложение *временного типа* приходит и уходит, предлагая одну функцию и ограниченный набор связанных с этой функцией элементов управления. Приложение этого типа вызывается при необходимости, делает свою работу и быстро исчезает, позволяя пользователю продолжить прерванную (как правило, в окне монопольного приложения) деятельность. Типичный пример сценария работы с временным приложением – вызов Проводника Windows для поиска и открытия другого файла в то время, когда пользователь уже редактирует один файл в MS Word.

Фоновыми называют приложения, которые в нормальном «рабочем» состоянии не взаимодействуют с пользователем. Такие программы выполняют задачи, которые в целом важны, но не требуют вмешательства пользователя. Примеры: драйвер принтера, подключение к сети.

Инфраструктура взаимодействия включает варианты поведения приложения. Создание инфраструктуры взаимодействия предполагает выполнение шести шагов [2, с. 164]:

Шаг 1. Определение форм-фактора, типа приложения и способов управления.

Шаг 2. Определение функциональных и информационных элементов.

Шаг 3. Определение функциональных групп и иерархических связей между ними.

Шаг 4. Макетирование общей инфраструктуры взаимодействия.

Шаг 5. Создание ключевых сценариев.

Шаг 6. Выполнение проверочных сценариев для верификации решений.

Форм-фактор – это зависимость вида пользовательского интерфейса от используемой технической платформы.

Функциональные и информационные элементы – это зримые представления функций и данных, доступные пользователю посредством интерфейса. Это конкретные проявления функциональных и информационных потребностей, выявленных на стадии выработки требований.

Информационные элементы – это, как правило, фундаментальные объекты интерактивных продуктов.

Функциональные элементы – это операции, которые могут выполняться над информационными объектами и представляющими эти объекты элементами интерфейса. В большинстве случаев функциональные элементы представляют собой инструменты, работающие с информационными элементами, а также контейнеры, содержащие информационные элементы.

Макетирование общей инфраструктуры взаимодействия Аланом Купером [2, с. 169] охарактеризовано как «фаза прямоугольников», поскольку эскизы будущего интерфейса начинаются с разделения каждого представления на прямоугольные области, соответствующие панелям, элементам управления и другим высокоуровневым контейнерам. При этом каждому прямоугольнику даётся своё название и показывается, каким образом одна группа элементов может влиять на другие. Содержательно этот шаг предназначен для исследования различных вариантов представления информации и функциональности в интерфейсе, при этом затраты на внесение изменений должны быть минимальны.

Известны два вида макетов: с жёсткой компоновкой и без компоновки.

При этом макет с жёсткой компоновкой:

- содержит взаимное расположение элементов и визуальную информацию о приоритетах;
- ограничивает работу графического дизайнера.
    - для макета без компоновки характерно то, что он:
- не содержит графического представления элементов;
- содержит текстовое описание элементов и их приоритетов;
- не ограничивает работу графического дизайнера.

Сценарий определяется Аланом Купером как средство описания идеального для пользователя взаимодействия. Истоки этого понятия восходят к публикациям сообщества HCI (Human-Computer Interaction – взаимодействие человека и компьютера), где оно увязывалось с указанием на метод решения задач проектирования через конкретизацию, которая понималась как использование специально составленного рассказа, чтобы одновременно конструировать и иллюстрировать проектные решения [2, с. 148]. Применение сценарного подхода к проектированию, как показано в книге Кэрролла «Making Use» (Carroll, 2000), сосредоточено на описании того, как пользователи решают задачи. Такое описание включает характеристику обстановки рабочей среды, а также агентов, или действующих лиц, которые являются абстрактными представителями пользователей.

Сценарии, основанные на персонажах, есть краткие описания одного или более персонажей, применяющих программный продукт для достижения конкретных целей. Сценарии позволяют начинать проектирование с рассказа, описывающего идеальный с точки зрения персонажа опыт, при этом фокусируя внимание на людях, их образе мысли и поведении.

Процесс выработки требований с использованием персонажей и сценариев состоит из следующих пяти шагов:

Шаг 1. Постановка задач и определение образа продукта.

Шаг 2. Мозговой штурм.

Шаг 3. Выявление ожиданий персонажей.

Шаг 4. Разработка контекстных сценариев.

Шаг 5. Выявление требований.

Ключевой сценарий описывает взаимодействие персонажа с системой в терминах лексикона инфраструктуры взаимодействия. Он отражает магистральные пути внутри интерфейса, используемые персонажем чаще всего (например, ежедневно). Ключевые сценарии сосредоточены на задачах. Например, в случае приложения для работы с электронной почтой ключевые действия – это просмотр и создание новых сообщений, а не настройка нового почтового сервера.

Ключевые сценарии, как правило, являются результатом развития контекстных сценариев, но целенаправленно описывают взаимодействие персонажа с различными функциональными и информационными элементами, составляющими общую инфраструктуру взаимодействия.

Контекстные сценарии сосредоточены на целях, же ключевые сценарии больше сосредоточены на задачах, намеки на которые или описания которых содержатся в контекстных сценариях.

Выполнение работы:

В качестве основы для выполнения данной лабораторной работы предлагается использовать одно из ранее разработанных ими приложений.

Предметная область и сфера применения. Правильное определение этих аспектов является основой для разработки UI в частности и всего приложения в целом. **Определение целевой аудитории**, направлен на выделение из общей массы группы (или групп) потенциальных пользователей разрабатываемой программы. Естественно, что цели, задачи, способности и возможности групп пользователей будут существенно различаться.

Модель пользователя, или *профиль*, формируется в результате анализа целевых групп. Она отражает наиболее общие черты, характерные для представителей группы и может представлять следующую информацию о пользователе:

- социальные и демографические характеристики (возраст, пол, основной язык, род занятий, потребности, привычки и т.п.).
- уровень компьютерной грамотности.
- цель и задачи, решаемые пользователем.
- окружение (рабочее место, конфигурация оборудования, используемая операционная система и т.п.)
- требования, специфичные для конкретной целевой группы.

После выделения одного или нескольких основных профилей пользователей и определения задач, стоящих перед ними, переходят к следующему этапу проектирования. Он связан с составлением **пользовательских сценариев**. Сценарий — это описание действий, выполняемых пользователем в рамках решения конкретной задачи на пути достижения его цели. Очевидно, что достигнуть некоторой цели можно, решая ряд задач. Каждую их них пользователь может решать несколькими способами, следовательно, должно быть сформировано несколько сценариев. Чем больше их будет, тем ниже вероятность того, что некоторые ключевые объекты и операции будут упущены.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Что такое интерфейс?
2. Какие типы пользовательских интерфейсов существуют?
3. Перечислите этапы разработки пользовательских интерфейсов?
4. К какому типу интерфейсов будет относиться интерфейс, разработанный в данной лабораторной работе?
5. Какие модели интерфейсов существуют?
6. Какая модель интерфейса будет использована в данной работе?
7. Что такое диалог?
8. Какие типы диалогов существуют?
9. Какие формы диалога Вы знаете?
10. Какой тип диалога и какая форма диалога будет использована в данной работе?

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Инструментарий тестирования и анализа качества программных средств»

Лабораторная работа № 9. Разработка тестовых модулей проекта для тестирования отдельных модулей

Цель: Ознакомление с видами оптимизации программы, оптимизация индивидуального модуля по выбранному параметру (время выполнения, объем памяти).

Выполнив работу, Вы будете:

уметь:

- У9. оценивать размер минимального набора тестов.
- У10. разрабатывать тестовые пакеты и тестовые сценарии.
- У11. выявлять ошибки в системных компонентах на основе спецификаций.
- У12 использовать различные транспортные протоколы и стандарты форматирования сообщений.
- У13 выполнять тестирование интеграции.
- У16 выполнять ручное и автоматизированное тестирование программного модуля.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание:

Разработать тестовые модулей проекта для тестирования отдельных модулей, провести оптимизацию программы по выбранному параметру.

Краткие теоретические сведения:

Оптимизация – преобразование программы, сохраняющее ее семантику (конструкции языка программирования), но уменьшающие ее размер и время выполнения.

Виды оптимизация программы:

- глобальная (всей программы);
- локальная (нескольких соседних операторов, образующих линейный участок);
- квазилокальная (фрагментов программы фиксированной структуры, например, циклов).

Способы оптимизации:

1. Разгрузка участков повторяемости: вынесение вычислений из многократно проходимых исполняемых участков программы на участки программы, редко проходимые. Таким образом, это преобразование тела цикла или рекурсивных процедур.

2. Упрощение действий: улучшение программы за счет замены групп вычислений на группу вычислений, дающих тот же результат с точки зрения всей программы, но имеющих меньшую сложность.

а) упрощение действий происходит при замене сложных операций в выражениях более простыми: x / 0.4 -> x*0.25;

б) преобразование по объединению или расчленению циклов, по перестановке заголовков циклов, по удалению избыточных выражений (замене их на переменную).

3. Реализация действия: действия над константами заменяются на константы; ликвидация константных распознавателей -замена условного оператора на одну из его ветвей, если его выбирающее условие-выражение имеет постоянное значение; удаление из программы ненужных пересылок вида:

Y=F(W), X=Y на X=F(W)

4. Чистка программы (удаление ненужных конструкций): недостижимых операторов, существенных операторов, неиспользуемых переменных, видов, операций.

5. Сокращение размера программы: вынесение одинаковых конструкций в начальную или конечную точку программы; поиск в программе похожих объектов и формирование их в виде процедуры.

6. Экономия памяти -уменьшение объема памяти, отводимые под информационные объекты программы (например, параметры процедуры).

Выполнение работы:

1. Для индивидуального модуля выбрать параметр оптимизации и определить его количественные характеристики.
2. Провести оптимизацию программы по выбранному параметру.
3. Сравнить характеристики исходного модуля и модуля, полученного в результате оптимизации.
4. Оформить отчет, содержащий описание, обоснование и результаты оптимизации программы.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Почему необходимо проводить оптимизацию, а не минимизацию программы?
2. От чего зависит выбор метода оптимизации?
3. Почему большое внимание уделяется циклическим участкам?
4. К каким нежелательным последствиям может привести оптимизация?

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Инструментарий тестирования и анализа качества программных средств»

Практическая работа № 4. Выполнение функционального тестирования

Цель: описать набор тестовых сценариев для верификации ПО. Оптимизировать тестовый набор. Научиться составлять спецификацию для разработки тестов.

Выполнив работу, Вы будете:

уметь:

- У9. оценивать размер минимального набора тестов.
- У10. разрабатывать тестовые пакеты и тестовые сценарии.
- У11. выявлять ошибки в системных компонентах на основе спецификаций.
- У12 использовать различные транспортные протоколы и стандарты форматирования сообщений.
- У13 выполнять тестирование интеграции.
- У14 организовывать постобработку данных.
- У16 выполнять ручное и автоматизированное тестирование программного модуля.
- У17 использовать инструментальные средства отладки программных продуктов.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание:

Составить спецификацию для тест дизайна сайта, согласно вашему варианту, используя полученные выше знания. Основываясь на спецификации требований, написать набор тест-кейсов для тестирования сайта.

Краткие теоретические сведения:

Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом.

Функциональные тесты основываются на функциях, выполняемых системой, и могут проводиться на всех уровнях тестирования (компонентном, интеграционном, системном, приемочном). Как правило, эти функции описываются в требованиях, функциональных спецификациях или в виде случаев использования системы (use cases).

Тестирование функциональности может проводиться в двух аспектах:

1. требования
2. бизнес-процессы

Тестирование в перспективе «требования» использует спецификацию функциональных требований к системе как основу для дизайна тестовых случаев (Test Cases). В этом случае необходимо сделать список того, что будет тестироваться, а что нет, приоритезировать требования на основе рисков (если это не сделано в документе с требованиями), а на основе этого приоритезировать тестовые сценарии (test cases). Это позволит сфокусироваться и не упустить при тестировании наиболее важный функционал.

Тестирование в перспективе «бизнес-процессы» использует знание этих самых бизнес-процессов, которые описывают сценарии ежедневного использования системы. В этой перспективе тестовые сценарии (test scripts), как правило, основываются на случаях использования системы (use cases).

Преимущества функционального тестирования:

- имитирует фактическое использование системы;

Недостатки функционального тестирования:

- возможность упущения логических ошибок в программном обеспечении;

– вероятность избыточного тестирования.

Достаточно распространенной является автоматизация функционального тестирования.

Тестовый случай (Test Case)

Тестовый случай (Test Case) - это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части.

Под тест кейсом понимается структура вида:

Action > Expected Result > Test Result

Пример:

Action	Expected Result	Test Result (passed/failed/blocked)
Open page "login"	Login page is opened	Passed

Виды Тестовых Случаев

Тест кейсы разделяются по ожидаемому результату на позитивные и негативные:

Позитивный тест кейс использует только корректные данные и проверяет, что приложение правильно выполнило вызываемую функцию.

Негативный тест кейс оперирует как корректными так и некорректными данными (минимум 1 некорректный параметр) и ставит целью проверку исключительных ситуаций (срабатывание валидаторов), а также проверяет, что вызываемая приложением функция не выполняется при срабатывании валидатора.

Структура Тестовых Случаев (Test Case Structure)

На просторах интернета вы сможете найти очень много информации о структуре тест кейсов, уровне их детализации и количестве проверок в них, я собираюсь рассказать о подходе используемом мной, и который я хочу предложить использовать вам.

Каждый тест кейс должен иметь 3 части:

PreConditions	Список действий, которые приводят систему к состоянию пригодному для проведения основной проверки. Либо список условий, выполнение которых говорит о том, что система находится в пригодном для проведения основного теста состояния.
Test Case Description	Список действий, переводящих систему из одного состояния в другое, для получения результата, на основании которого можно сделать вывод о удовлетворении реализации, поставленным требованиям
PostConditions	Список действий, переводящих систему в первоначальное состояние (состояние до проведения теста - initial state)

Примечание: Post Conditions не является обязательной частью. Это скорее всего - правило хорошего тона: "намусорил - убери за собой". Это особенно актуально при автоматизированном тестировании, когда за один прогон можно наполнить базу данных сотней или даже тысячей некорректных документов.

Пример тест кейса:

do A1, verify B1

do A2, verify B2

do A3, verify B3

В приведенном примере конечная проверка - B3. Это значит, что именно она является ключевой. Значит, A1 и A2 - это действия, приводящие систему в

тестопригодное состояние. А B1 и B2 - условия того, что система находится в состоянии пригодном для тестирования. Таким образом имеем:

Action	Expected Result	Test Result (passed/failed/blocked)
PreConditions		
do A1	verify B1	
do A2	verify B2	
Test Case Description		
do A3	verify B3	
PostConditions		

PostConditions в данном примере не были описаны, но по логике вещей надо выполнить шаги, которые бы вернули систему в первоначальное состояние. (например, удалили созданную запись, или отменили бы изменения, сделанные в документе).

Теперь ответим на вопрос: "Почему данное разбиение удобно использовать?"

Ответ: конечная проверка одна, т.е. в случае если тест провален (test failed) будет сразу ясно из-за чего. Т.к. если провальными окажутся проверки B1 и/или B2, то тест кейс будет заблокирован (test blocked), из-за того, что функцию невозможно привести в тестопригодное состояние (состояние пригодное для проведения тестирования), но это не значит, что тестируемая функция не работает.

Action	Expected Result	Test Result (passed/failed/blocked)
PreConditions		
do A1	verify B1	passed
do A2	verify B2	failed
Test Case Description:		
do A3	verify B3	blocked
PostConditions		

Детализация описания тест кейсов (Test Case Specification)

Бытует много разных мнений об уровне детализации при написании тест кейсов, а также количестве проверок в одном тест кейсе. Все они по-своему правильные. Мое мнение, что уровень детализации тест кейсов должен быть таков, чтобы обеспечивать разумное соотношение времени прохождения к тестовому покрытию. Т.е. до тех пор, пока покрытие тестами определенного функционала не меняется, можно уменьшать детализацию тест кейсов.

Пример тест кейса 1:

Проверка отображения страницы		
Действие	Ожидаемый результат	Результат теста

Открыть страницу "Вход в систему"	- Окно "Вход в систему" открыто - Название окна - Вход в систему - Логотип компании отображается в правом верхнем углу - На форме 2 поля - Имя и Пароль - Кнопка Вход доступна - Ссылка "забыл пароль" - доступна	...

Пример тест кейса 2:

Название: Проверка отображения страницы

Действие: Открыть страницу "Вход в систему"

Проверка: Проверьте, что отображаемая страница соответствует странице на картинке 1 (и прилагаем изображение страницы "Вход в систему")

В примере 1 и 2 покрытие будет одинаковым, но вот время, которое потребуется для прохождения, будет разным. Мне кажется, что второй пример будет даже нагляднее.

В дополнение хочется сказать, что решение о виде тест кейса и детализации его описания принимает человек, ответственный за его создание - Тест Дизайнер или Тест Аналитик, обладающий необходимым опытом, и который знает тест дизайн не по наслышке и имеет опыт практического применения техник тест дизайна. Во многих компаниях эта роль не выделяется отдельно, а доверяется обычным тестировщикам, что в случае недостаточной квалификации может привести к переписке тест кейсов.

Тест Дизайн (Test Design)

Тест дизайн – это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы), в соответствии с определёнными ранее критериями качества и целями тестирования.

План работы над тест дизайном

анализ имеющихся проектных артефактов: документация (спецификации, требования, планы), модели, исполняемый код и т.д.

написание спецификации по тест дизайну (Test Design Specification)

проектирование и создание тестовых случаев (Test Cases)

Роли, ответственные за тест дизайн

Тест аналитик - определяет "ЧТО тестировать?"

Тест дизайнер - определяет "КАК тестировать?"

Попросту говоря, задача тест аналитиков и дизайнеров сводится к тому, чтобы используя различные стратегии и техники тест дизайна, создать набор тестовых случаев, обеспечивающий оптимальное тестовое покрытие тестируемого приложения. Однако, на большинстве проектов эти роли не выделяется, а доверяется обычным тестировщикам, что не всегда положительно сказывается на качестве тестов, тестировании и, как из этого следует, на качестве программного обеспечения (конечного продукта).

Техники тест дизайна (Test Design Technics)

Многие люди тестируют и пишут тестовые случаи (test cases), но не многие пользуются специальными техниками тест дизайна. Постепенно, набираясь опыта они осознают, что постоянно делают одну и ту же работу, поддающуюся конкретным правилам. И тогда они находят, что все эти правила уже описаны.

Предлагаем вам ознакомиться с кратким описанием наиболее распространенных техник тест дизайна:

Эквивалентное Разделение (Equivalence Partitioning - EP). Как пример, у вас есть диапазон допустимых значений от 1 до 10, вы должны выбрать одно верное значение внутри интервала, скажем, 5, и одно неверное значение вне интервала - 0.

Анализ Граничных Значений (Boundary Value Analysis - BVA). Если взять пример выше, в качестве значений для позитивного тестирования выберем минимальную и максимальную границы (1 и 10), и значения больше и меньше границ (0 и 11). Анализ граничных значений может быть применен к полям, записям, файлам, или к любого рода сущностям, имеющим ограничения.

Причина / Следствие (Cause/Effect - CE). Это, как правило, ввод комбинаций условий (причин), для получения ответа от системы (Следствие). Например, вы проверяете возможность добавлять клиента, используя определенную экранную форму. Для этого вам необходимо будет ввести несколько полей, таких как "Имя", "Адрес", "Номер Телефона" а затем, нажать кнопку "Добавить" - эта "Причина". После нажатия кнопки "Добавить", система добавляет клиента в базу данных и показывает его номер на экране - это "Следствие".

Предугадывание ошибки (Error Guessing - EG). Это когда тест аналитик использует свои знания системы и способность к интерпретации спецификации на предмет того, чтобы "предугадать" при каких входных условиях система может выдать ошибку. Например, спецификация говорит: "пользователь должен ввести код". Тест аналитик, будет думать: "Что, если я не введу код?", "Что, если я введу неправильный код? ", и так далее. Это и есть предугадывание ошибки.

Исчерпывающее тестирование (Exhaustive Testing - ET) - это крайний случай. В пределах этой техники вы должны проверить все возможные комбинации входных значений, и в принципе, это должно найти все проблемы. На практике применение этого метода не представляется возможным, из-за огромного количества входных значений.

Практическое применение техник тест дизайна при разработке тест кейсов

Многие знают, что такое тест дизайн, но не все умеют его применять. Чтобы немного прояснить ситуацию, мы решили предложить Вашему вниманию последовательный подход к разработке тестовых случаев (тест кейсов), используя самые простейшие техники тест дизайна:

Эквивалентное Разделение (Equivalence Partitioning), далее в тексте - EP

Анализ Граничных Значений (Boundary Value Analysis), далее в тексте - BVA

Предугадывание ошибки (Error Guessing), далее в тексте - EG

Причина / Следствие (Cause/Effect), далее в тексте - CE

План разработки тест кейсов предлагается следующий:

1. Анализ требований.
2. Определение набора тестовых данных на основании EP, BVA, EG.
3. Разработка шаблона теста на основании CE.
4. Написание тест кейсов на основании первоначальных требований, тестовых данных и шагов теста.

Далее на примере, рассмотрим предложенный подход.

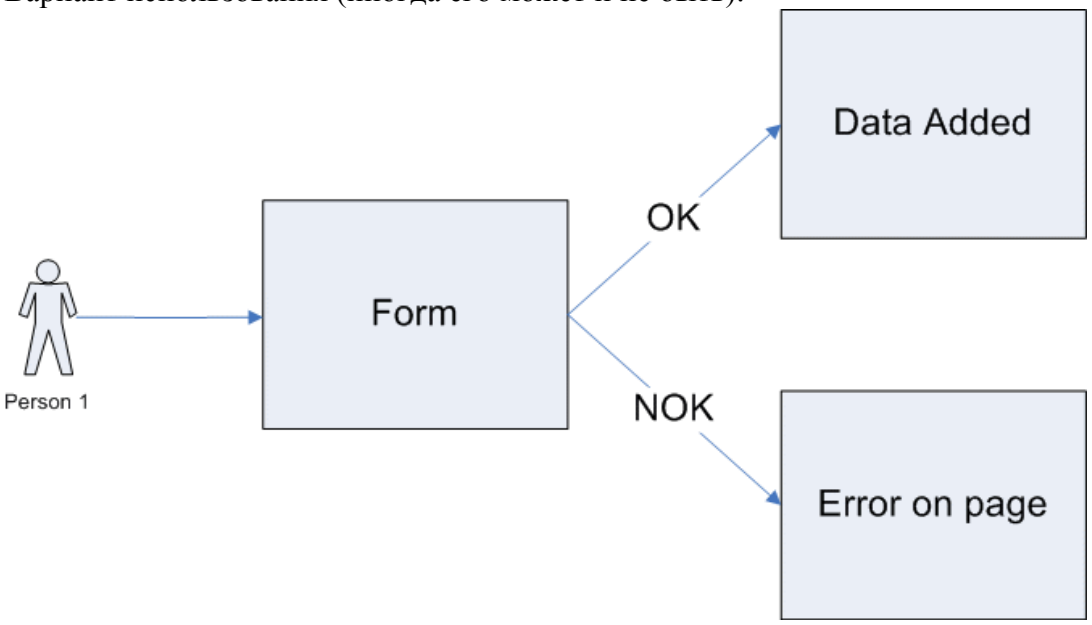
Пример:

Протестировать функциональность формы приема заявок, требования к которой предоставлены в следующей таблице:

Элемент	Тип элемента	Требования
Тип обращения	combobox	Набор данных: Консультация Проведение тестирования Размещение рекламы Ошибка на сайте

		* - на процесс выполнения операции приема заявок не влияет.
Контактное лицо	editbox	1. Обязательное для заполнения 2. Максимально 25 символов 3. Использование цифр и спец символов не допускается
Контактный телефон	editbox	Обязательное для заполнения Допустимые символы "+" и цифры "+" можно использовать только в начале номера Допустимые форматы: начинается с плюса - 11-15 цифр +31612361264 +375291438884 без плюса - 5-10 цифр, например: 0613261264 2925167
Сообщение	text area	1. Обязательное для заполнения 2. Максимальная длина 1024 символа
Отправить	button	Состояние: 1. По умолчанию - не активна (Disabled) 2. После заполнения обязательных полей становится активна (Enabled) Действия после нажатия 1. Если введенные данные корректны - отправка сообщения 2. Если введенные данные НЕ корректны - валидационное сообщение

Вариант использования (иногда его может и не быть):



1. Анализ требований

Читаем, анализируем требования и выделяем для себя следующие нюансы:

– какие из полей обязательные для заполнения?

- имеют ли поля ограничения по длине или по размерности (границы)?
- какие из полей имеют специальные форматы?

2. Определение набора тестовых данных

Отталкиваясь от требований к полям, используя техники тест дизайна начинаем определение набора тестовых данных:

- в зависимости от того обязательное поле или нет, определим какие поля необходимо проверить на пустое значение, так как оно может вызывать ошибку (В результирующей таблице оранжевый цвет)
- т.к. исчерпывающее тестирование не представляется возможным из-за огромного числа всевозможных комбинаций значений, в первую очередь необходимо определить минимальный набор данных. Это можно сделать используя такие техники, как EP и BVA. (В результирующей таблице голубой цвет)

На форме присутствует поле, имеющее составной тип (цифры используются совместно с символами), обладает специальным форматом данных и поэтому выделение тестовых данных для него - это достаточно трудоемкая задача. В пределах данной работы ограничимся только простой проверкой форматов и основных требований описанных в форме приема заявок.

По завершению генерации данных используя стандартные техники, можно добавить некоторое количество значений на основании личного опыта (техника EG) - это будет использование спец. символов, очень длинных строк, разных форматов данных, регистров в строках (Upper, Lowwer, Mixed cases), отрицательные и нулевые значения, кейворды Null - NaN - Infinity и т.д. Сюда можно включить все, что вы полагаете может вывести приложение из строя (в результирующей таблице фиолетовый цвет)

Примечание:

Отметим, что количество тестовых данных после окончательной генерации будет достаточно большим, даже при использовании специальных техник тест дизайна. Поэтому ограничимся лишь несколькими значениями для каждого поля, так как цель данной работы показать именно процесс создания тест кейсов, а не процесс получения конкретных тестовых данных.

2.1 Выбор тестовых данных для каждого отдельно взятого поля

Поле Тип обращения. Так как все данные входят в 1 класс эквивалентности, то есть не изменяют сам процесс выполнения приема заявки, берем любою (1-ю) позицию в листе с ожидаемым результатом ОК. Но т.к. реализовано поле как лист, имеет также смысл рассмотреть и граничные условия (техника BVA), т.е. берем первый и последний элементы. Итого: 1-я и последняя позиции в листе. Ожидаемый результат при использовании - ОК.

Поле Контактное лицо. Это обязательное поле размером от 1 до 25 символов (включая границы). Проверка на обязательность добавляет к тестовым данным пустое значение. Проведем анализ граничных условий (BVA), получим набор: 0, 1, 2, 24, 25 и 26 символов. Пустое значение (0 символов) уже было добавлено при анализе обязательности поля для ввода, поэтому при BVA мы не будем добавлять его еще раз. (если его добавить второй раз, произойдет дублирование тестовых данных, которое не приведет к нахождению новых дефектов, а значит повторное добавление в домен не имеет смысла). В связи с тем, что значения 2 и 24 символа являются, с нашей точки зрения, некритичными, их можно не добавлять. В итоге получаем, что минимальный набор данных для тестирования поля - это строки 1 и 25 - ОК, и 0 (пустое значение), 26 символов - NOK.

поле Контактный телефон состоит из нескольких частей: код страны, код оператора, номер телефон (который может быть составной и разделенный дефисами). Для определения правильного набора тестовых данных необходимо рассматривать каждую составную часть по отдельности. Применяя BVA и EP, получим:

для номеров с плюсом:

По BVA получим номера с 10, 11, 12 и 14, 15, 16 цифрами, где 10 и 16 - NOK, а 11, 12, 14, 15 - OK

Рассматривая полученные данные с позиции ЕР выделим, что 11, 12, 14, 15 входят в один класс эквивалентности. Поэтому при тестировании мы можем использовать любое из них, но так как 11 и 15 - это границы интервала, то на наш взгляд их пропускать нельзя. Следовательно, мы можем уменьшить набор значений до двух, исключив 12 и 14, а оставив 11 и 15 для проверки граничных условий.

Итого имеем:

11 и 15 цифр - OK, (+12345678901, +123456789012345)

10 и 16 цифр - NOK; (+1234567890, +1234567890123456)

для номеров без плюса:

По BVA получим номера с 4, 5, 6 и 9, 10, 11 цифрами.

Действуя аналогично примеру для номеров телефонов с плюсом, исключим значения 6 и 9, оставив 5 и 10.

Итого имеем:

5 и 10 цифр - OK, (12345, 1234567890)

4 и 11 цифр - NOK; (1234, 12345678901)

поле Сообщение. подбор данных проводим по аналогии с полем Контактное лицо. На выходе получаем значения: строки 1 и 1024 - OK, и 1025 символов - NOK.

Результирующая таблица данных, для использования при последующем составлении тест кейсов

Поле	OK /NOK	Значение	Комментарий
Тип обращения	OK	Консультация	первый в списке
		Ошибка на сайте	последний в списке
	NOK		
Контактное лицо	OK	йцукенгшщзйцукенгшщзйцуке	25 символов нижний регистр
		а	1 символ
		ЙЦУКЕНГШЩЗФЫВАПРОЛДЖЯЧСМИ	25 символов ВЕРХНИЙ регистр
		ЙЦУКЕНГШЩЗфывапролджЯЧСМИ	25 символов СМеШаННыЙ регистр
	NOK		пустое значение
		йцукенгшщзйцукенгшщзйцукей	длина больше максимальной(26 символов
		@#$%^&;.?,>|\/№"!()_{}[<~	спец. символы (ASCII)
		12345678901234567890123456	только цифры

		adsadasdasdas dasdasd asasdsads(...)sas	очень длинная строка (~1Mb)
Контактный телефон	OK	+12345678901	с плюсом - минимальная длина
		+123456789012345	с плюсом - максимальная длина
		12345	без плюса - минимальная длина
		1234567890	без плюса - максимальная длина
	NOK		пустое значение
		+1234567890	с плюсом - < минимальной длины
		+1234567890123456	с плюсом - > максимальной длины
		1234	без плюса - < минимальной длины
		12345678901	без плюса - > максимальной длины
		+YYYXXXyyyxxzz	с плюсом - буквы вместо цифр
		yyyxxxxzz	без плюса - буквы вместо цифр
		+###-$$$-%^-&^-&!	спец. символы (ASCII)
		12323123231232132312 32(...)99	очень длинная строка (~1Mb)
Сообщение	OK	йццуйцуйц(...)йцу	максимальная длина (1024 символа)
	NOK		пустое значение
		йццуйцуйц(...)йцуц	длина больше

			максимальной (1025 символов)
		adsadasdasdas dasdasd asasdsads(...)sas	очень длинная строка (~1Mb)
		@##$$$%^&^&	только спец. символы (ASCII)

3. Разрабатываем шаблон теста

На основании техники CE и, по возможности, имеющихся вариантов использования (Use case) создадим шаблон планируемого теста. Данный документ будет представлять собой шаги и ожидаемые результаты теста, но без конкретных данных, которые подставляются на следующем этапе разработки тест кейсов.

Пример шаблона тест кейса

Действие	Ожидаемый результат
1. Открываем форму отправки сообщения	Форма открыта Все поля по умолчанию пусты Обязательные поля помечены - * Кнопка "Отправить" не активна
2. Заполняем поля формы: Тип обращения Контактное лицо Контактный телефон Сообщение	Поля заполнены Кнопка "Отправить" - активна (Enabled)
3. Нажимаем кнопку "Отправить"	Если введенные данные корректны - Сообщение "Заявка отправлена"выведено на экран. Новая заявка появилась в списке на странице "Заявки". Если введенные данные НЕ корректны -; Валидационное сообщение со всеми ошибками выведено на экран. Заявка НЕ появилась в списке на странице "Заявки".

4. Написание тест кейсов на основании первоначальных требований, тестовых данных и шаблона теста

После того, как тестовые данные и шаги теста готовы приступаем непосредственно к разработке тест кейсов. Здесь нам помогут такие методы комбинирования как:

Последовательный перебор. Представляет собой перебор всех возможных комбинаций имеющихся значений. Таким образом получается, что количество тест кейсов будет равно произведению количества вариантов тестовых данных для каждого поля. Для нашего конкретного примера мы получим 1170 тест кейсов.

Попарный перебор (Pairwise Testing). Зачастую, сбои вызывают не сложное сочетание всех параметров, а сочетание лишь пары параметров. Техника попарного перебора, позволяет создать тестовые наборы, комбинирующие данные из двух полей. Благодаря этому, количество полученных на выходе тест кейсов в разы меньше, чем при комбинировании того же набора данных при последовательном переборе. Отметим также, что в данный момент существует несколько алгоритмов генерации комбинаций для попарного тестирования: Orthogonal Arrays Testing, All pairs, IPO (In-Parameter Order). Так например, при использовании техники All Pairs в нашем конкретном случае мы получим всего 118 тест кейсов.

По завершению подготовки комбинаций данных, подставляем их в шаблон тест кейса, и в результате имеем набор тестовых случаев, покрывающий тестируемые нами требования к форме приема заявок.

Примечание:

Напоминаем, что тест кейсы разделяются по ожидаемому результату на позитивные и негативные тест кейсы.

Пример позитивного тест кейса (все поля OK):

Действие	Ожидаемый результат
1. Открываем форму отправки сообщения	Форма открыта Все поля по умолчанию пусты Обязательные поля помечены - * Кнопка "Отправить" не активна
2. Заполняем поля формы: Тип обращения = Консультация Контактное лицо = йцукенгшщзйцукенгшщзйцуке Контактный телефон = +7-916-111-11-11 Сообщение	Поля заполнены Кнопка "Отправить" - активна (Enabled)
3. Нажимаем кнопку "Отправить"	Сообщение "Заявка отправлена"выведено на экран. Новая заявка появилась в списке на странице "Заявки".

Пример негативного тест кейса (поле Контактное лицо - NOK):

Действие	Ожидаемый результат
1. Открываем форму отправки сообщения	Форма открыта Все поля по умолчанию пусты Обязательные поля помечены - * Кнопка "Отправить" не активна
2. Заполняем поля формы: Тип обращения = Консультация Контактное лицо = @#$%^&;.?,>|\/№"!()_{}[<~ Контактный телефон = (916)333-33-33 Сообщение = йццуйцуйц(...)йцу - 1024 символа	Поля заполнены Кнопка "Отправить" - активна (Enabled)
3. Нажимаем кнопку "Отправить"	Валидационное сообщение со всеми ошибками выведено на экран: "В поле "Контактное лицо" запрещено использование цифр и спец. символов." Заявка НЕ появилась в списке на странице "Заявки".

Выполнение работы:

1) Разработать тестовые наборы для функционального тестирования.
2) Провести тестирование программы и представить результаты в виде таблицы.
3) Выработать рекомендации для корректировки тестируемой программы.
4) Представить отчет по лабораторной работе для защиты.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

.

Контрольные вопросы:

1. Что такое тестирование ПС?
2. Чем тестирование отличается от отладки ПС?
3. Для чего проводится функциональное тестирование?
4. Каковы правила тестирования программы «как черного ящика»?
5. Как проводится тестирования программы по принципу «белого ящика»?
6. Как осуществляется сборка программы при модульно тестировании?

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Инструментарий тестирования и анализа качества программных средств»

Практическая работа № 5. Тестирование интеграции

Цель: Изучение назначения и задач интеграционного тестирования.

Выполнив работу, Вы будете:

уметь:

- У5. организовывать заданную интеграцию модулей в программные средства на базе имеющейся архитектуры и автоматизации бизнес-процессов.
- У6. определять источники и приемники данных.
- У9. оценивать размер минимального набора тестов.
- У10. разрабатывать тестовые пакеты и тестовые сценарии.
- У11. выявлять ошибки в системных компонентах на основе спецификаций.
- У12 использовать различные транспортные протоколы и стандарты форматирования сообщений.
- У13 выполнять тестирование интеграции.
- У14 организовывать постобработку данных.
- У15 создавать классы- исключения на основе базовых классов.
- У16 выполнять ручное и автоматизированное тестирование программного модуля.
- У17 использовать инструментальные средства отладки программных продуктов.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание:

1. Напишите план интеграционного тестирования в соответствии с рекомендациями.
2. Исходя из количества описанных кластеров и заявленных для них тестовых примеров (последний пункт плана), создайте соответствующее количество юнит-тестов.
3. Отладьте и запустите все юнит-тесты. При этом не требуется исправление ошибок в исходном коде, если таковые были обнаружены.
4. Оцените результаты выполнения юнит-тестирования и сделайте соответствующие выводы.

Краткие теоретические сведения:

Интеграционное тестирование — это тестирование программного обеспечения на корректность взаимодействия нескольких модулей, объединенных в единое целое. Несмотря на то, что результатом тестирования и верификации отдельных модулей, составляющих программную систему, является заключение о том, что эти модули являются внутренне непротиворечивыми и соответствуют требованиям, это не гарантирует их корректную совместную работу. Целью интеграционного тестирования является проверка соответствия проектируемых единиц функциональным, приёмным и требованиям надежности. Тестирование этих проектируемых единиц — объединения, множества или группа модулей — выполняются через их интерфейс, используя тестирование «чёрного ящика».

Интеграционное тестирование называют еще тестированием архитектуры системы. Результаты выполнения интеграционных тестов – один из основных источников информации для процесса улучшения и уточнения архитектуры системы, межмодульных и межкомпонентных интерфейсов. Т.е. с интеграционные тесты проверяют корректность взаимодействия компонент системы.

Интеграционное тестирование, как правило, представляет собой итеративный процесс, при котором проверяется функциональность все более и более увеличивающейся в размерах совокупности модулей.

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Интеграционные тесты должны проверять совместную работу отдельных модулей или наборов модулей, между которыми есть зависимости. При этом тесты должны проверить, правильно ли обрабатываются входные данные отдельных методов при вызове их при разных ситуациях и наборах исходных данных (функциональная сторона интеграционного тестирования) и проверить стабильность работы модуля при разных исходных данных (проверка на надежность работы модуля интеграционным тестированием).

Существует несколько подходов к интеграционному тестированию:

- Снизу вверх. Сначала собираются и тестируются модули самих нижних уровней, а затем по возрастанию к вершине иерархии. Данный подход требует готовности всех собираемых модулей на всех уровнях системы.
- Сверху вниз. Данный подход предусматривает движение с высокоуровневых модулей, а затем направляется вниз. При этом используются заглушки для тех модулей, которые находятся ниже по уровню, но включение которых в тест еще не произошло.
- Большой взрыв. Все модули всех уровней собираются воедино, а затем тестируется. Данный метод экономит время, но требует тщательной проработки тест кейсов.

Преимущество.

Интеграционное тестирование позволяет имитировать действия пользователей и быстро получать подтверждение, что программный продукт успешно взаимодействует с другими системами. Такой подход гарантирует сразу несколько преимуществ:

1. Предотвращение появления критичных ошибок в опытно-промышленной эксплуатации
2. Снижение влияния человеческого фактора
3. Экономия затрат на исправление дефектов

Главной задачей интеграционного тестирования является поиск ошибок, связанных с взаимодействием модулей системы или нескольких систем. В результате все смежные системы и модули одной системы должны работать согласованно.

Способы проведения интеграционного тестирования подбираются в зависимости от интеграционных решений.

Выполнение работы:

Разработайте класс, содержащий минимум 3 метода, один из которых соответствует вашему варианту: по заданным параметрам a и b вычислите параметр c.

Вариант	c
1	
2	
3	
4	
5	
6	
7	

8	$8a - b^2$
9	$5a + \frac{b}{4}$
10	$4(a + b^2)$ $12a^2 - 8b$ $a^3 - 3b$ $(a^2 - b)/2$ $(a - b)(a + b)$ $\frac{a(b^2 + 16)}{5}$ $2(\frac{a^2}{4} - b)$ $4(b + a) - a^2$
11	3a2-b
12	ab-4a

Два других (остальных) метода должны передавать или получать значения из метода вашего варианта. Т.е. всеми тремя методами должно быть организовано взаимодействие.

Создайте отдельный тестирующий класс (как для модульного тестирования), в котором опишите минимум 3 теста с различными параметрами, охватывающими различные неэквивалентные значения.

Примечание: при интеграционном тестировании при одном вызове функции, тестируются все 3 метода.

1. Разработайте тест-плана – руководства к действию для тестировщиков;
2. Сформируйте тестовые данных и создайте тест-кейсов;
3. Реализуйте сценарии для запуска тест-кейсов;
4. Выполните тест-кейсы и исправьте ошибки;
5. Повторите цикл тестирования до успешной интеграции.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

.

Контрольные вопросы:

1. Дайте определение понятия интеграционное тестирование.
2. В чем сущность метода «белого ящика»?
3. В чем сущность метода «черного ящика».
4. Цели интеграционного тестирования.
5. Преимущества «раннего начала» тестирования.

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Тема «Инструментарий тестирования и анализа качества программных средств»

Практическая работа № 6. Документирование результатов тестирования

Цель: составить итоговый отчет о результатах тестирования приложения.

Выполнив работу, Вы будете:

уметь:

- У16 выполнять ручное и автоматизированное тестирование программного модуля.

Материальное обеспечение: персональный компьютер, среда программирования Visual Studio 2019.

Задание:

1. Провести документирование результатов тестирования программного средства.
2. Составить отчет по практической работе.

Краткие теоретические сведения:

Итоговый отчет можно разделить на части с соответствующей информацией:

- Приветствие.
- Общая информация (Common Information).
- Тестовое окружение (Test Platform).
- Рекомендации QA (QA Recommendations).
- Детализированная информация (Detailed Information).
- Окончание содержимого.

Приветствие

Свое письмо с отчетом необходимо начать с приветствия всех адресатов. Если по каким-либо причинам произошла задержка данных отчета, либо не весь запланированный функционал был проверен, то эту информацию необходимо предоставить в начале письма. Следует извиниться за задержку и указать адекватные причины произошедшего. Также в самом начале письма следует указывать, если были какие-то внешние факторы, препятствующие проверке какой-то части функционала.

Если во время тестирования не произошло никаких форс-мажорных обстоятельств, то достаточно обычного вежливого приветствия и далее уже переход к следующим пунктам.

Общая информация (Common Information)

В данной части отчета описывается, какие виды тестов проводились. Зачастую указываются модули, которые тестировались или функционал. Стоит удостовериться, не забыта ли какая-то часть функционала, особенно это актуально, когда нужно собрать итоговый отчет, соединив в себе данные о разных видах тестов и функционале.

Тестовое окружение (Test Platform)

Как правило, в этой части указываются:

- Название проекта.
- Номер сборки.
- Ссылка на проект (сборку). Необходимо убедиться, что зайдя по этой
- ссылке вы действительно попадаете на проект или можете установить приложение.

При указании данных в этой части отчета нужно быть очень внимательным, т.к. неправильная ссылка на сборку или неверный номер сборки не дают достоверной информации всем заинтересованным людям, а также затрудняют работу человеку, собирающему финальный отчет.

Рекомендации QA (QA Recommendations)

Данная часть отчета является наиболее важной, т.к. здесь отражается общее состояние сборки. Здесь показывается аналитическая работа тестировщика, его рекомендации по улучшению функционала, наиболее слабые места и наиболее критичные дефекты, динамика изменения качества проекта.

В этом разделе должна быть информация о следующем:

- Указан функционал (часть функционала), который заблокирован для проверки. Даны пояснения почему этот функционал не проверен (указаны наиболее критичные дефекты).
- Произведен анализ качества проверенного функционала. Следует указать, улучшилось оно или ухудшилось по сравнению с предыдущей версией, какое качество на сегодняшний момент, какие факторы повлияли на выставление именно такого качества сборки.
- Если качество сборки ухудшилось, то обязательно должны быть указаны регрессионные места.
- Наиболее нестабильные части функционала следует выделить и указать причину, по которой они таковыми являются.
- Даны рекомендации по тому функционалу и дефектам, скорейшее исправление которых является наиболее приоритетным.
- Список наиболее критичных для сборки дефектов, с указанием названия и их критичности.
- Для отчета уровня Smoke обязательно указать весь нестабильный функционал. Если сборка является релизной или предрелизной, то любое ухудшение качества является критичным и важно об этом сообщить менеджеру как можно раньше.

Помимо всего вышеуказанного для релизных и предрелизных сборок в отчете о качестве продукта важно указывать следующее:

- Дана информация о всех проблемах, характерных сборке. Проведен анализ, насколько оставшиеся проблемы являются критичными для конечного пользователя.
- Указаны дефекты, которые следует исправить, чтобы качество конечной сборки было выше.

Детализированная информация (Detailed Information)

В данной части отчета описывается более подробная информация о проверенных частях функционала, устанавливается качество каждой проверенной части функционала(модуля) в отдельности. В зависимости от типа проводимых тестов, эта часть отчета будет отличаться.

Smoke

При оценке качества функционала на уровне Smoke теста, оно может быть либо Приемлемым, либо Неприемлемым. Качество сборки зависит от нескольких факторов:

- Если это релизная или предрелизная сборка, то для выставления Приемлемого качества на уровне Smoke не должно быть найдено функциональных дефектов.
- Наличие нового функционала. Новый функционал, который впервые поставляется на тестирование, не должен содержать дефектов уровня Smoke для выставления Приемлемого качества всей сборки.
- Чтобы установить сборке Приемлемое качество, не должно быть дефектов уровня Smoke у того функционала, по которому планируется проводить полные тесты.
- Все наиболее важные части функционала отрабатывают корректно, тогда качество всего функционала на уровне Smoke может быть оценено, как Примлемое.

В части о детализированной информации качества сборки следует более подробно описать проблемы, которые были найдены во время теста.

DV

В этой части отчета указывается качество о проведении валидации дефектов.

Здесь должна быть следующая информация:

- Общее количество всех дефектов, поступивших на проверку.
- Количество неисправленных дефектов и их процент от общего количества.
- Список дефектов, которые не были проверены и причины, по которым этого не было сделано.
- Наглядная таблица с неисправленными дефектами.

По вышеуказанным результатам выставляется качество теста. Если процент неисправленных дефектов < 10%, то качество Приемлемое, если > 10%, то качество Неприемлемое.

NFT

При проведении полного теста нового функционала качество отдельно проверенного функционала может быть: Высокое, Среднее, Низкое.

В отчете следует отдельно указывать информацию о качестве каждой части нового функционала. В этой части отчета должна быть следующая информация:

- Дана общая оценка реализации нового функционала (сгруппированная по качеству).
- Подробная (детальная) информация о качестве каждой из частей новой функциональности.
- Проведен анализ каждой из новых функций в отдельности.
- Даны ясные пояснения о выставлении соответствующего качества.
- Даны рекомендации по улучшению качества (какие проблемы следует исправить).
- Показана таблица с новыми функциями (название), их качеством, статусом фуннкции из CQ.

AT, MAT, Regression

Если проводились тесты указанных уровней, то в первую очередь при написании отчета нужно анализировать динамику изменения качества проверенной функциональности в сравнении с более ранними версиями сборки. Также как и у предыдущего вида тестов, качество этих может быть: Высокое, Среднее, Низкое.

Для указанных видов тестов в данной части отчета должна быть описана информация следующего характера:

- Дана сравнительная характеристика каждой из частей функционала в сравнении с предыдущими версиями сборки.
- Подробная (детальная) информация о качестве каждой из частей проверенной функциональности.
- Даны ясные пояснения о выставлении соответствующего качества каждой функции в отдельности.
- Даны рекомендации по улучшению качества (какие проблемы следует исправить).

Окончание содержимого

В завершении содержимое отчета должно включать в себя информацию следующего характера:

- Ссылка на тест-план.
- Ссылка на документ feature matrix (если таковой имеется).
- Ссылка на документ со статистикой (если таковой имеется).
- Общее количество всех новых дефектов.
- Подпись высылающего отчет.

Данные ссылки должны быть корректными, необходимо проверить достоверную ли информацию получает пользователь, открывший ссылку. Следует обращать особое

внимание на подпись, удостоверьтесь, что указана именно ваша подпись либо какая-то универсальная для определенного проекта подпись.

Выполнение работы:

1. Запустить ранее созданное приложение.
2. Составить итоговый отчет по результатам тестирования приложения.
3. Оформить отчет и защитить практическую работу.

Форма представления результата:

1. Текст должен быть написан шрифтом Times New Roman, 12. Интервал между строками и абзацами – 1,5. Отступ слева 1,5. Ориентация текста – по ширине страницы. Скриншоты необходимо подписать. Название практической работы, цель работы, ход работы, вывод, ответы на контрольные вопросы должны быть выделены жирным шрифтом.
2. Заключение (выводы).
3. Список используемой литературы.

Контрольные вопросы:

1. Какая структура итогового отчета о результатах тестирования?
2. Что содержится в разделе Приветствие?
3. Что содержится в разделе Общая информация?
4. Что содержится в разделе Тестовое окружение?
5. Что содержится в разделе Рекомендации QA?
6. Что содержится в разделе Детализированная информация?
7. Что содержится в разделе Окончание содержимого?

Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент учел и рассмотрел особенности предметной области и грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент не полностью учел и рассмотрел особенности предметной области и представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, студент не учел особенности предметной области, отчет составлен неграмотно – оценка «неудовлетворительно».

Код страницы авторизации

```
<Window x:Class="Olesya.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Olesya"
        mc:Ignorable="d"
        Title="MainWindow" Height="800" Width="800" Loaded="Window_Loaded"
WindowStartupLocation="CenterScreen">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />

        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="0.8*"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Label Content="Персональная информация" Grid.Column="0" Grid.Row="0"
FontSize="18" FontWeight="Bold" Margin="10,0"/>
        <Label Content="Фамилия" Grid.Column="0" Grid.Row="1" FontWeight="Bold"
FontSize="14" Margin="8,0"/>
        <TextBox x:Name="lName" Text="Иванов"  Grid.Column="0" Grid.Row="2"
FontWeight="Bold" FontSize="14" Margin="8,0" Foreground="Gray"
GotFocus="LName_GotFocus"/>
        <Label Content="Имя" Grid.Column="0" Grid.Row="3" FontWeight="Bold" FontSize="14"
Margin="8,0"/>
        <TextBox x:Name="name" Text="Иван"  Grid.Column="0" Grid.Row="4"
FontWeight="Bold" FontSize="14" Margin="8,0" Foreground="Gray"/>
        <Label Content="E-mail (логин)" Grid.Column="0" Grid.Row="5" FontWeight="Bold"
FontSize="14" Margin="8,0"/>
        <TextBox x:Name="eMail" Text="ivanov@gmail.com"  Grid.Column="0" Grid.Row="6"
FontWeight="Bold" FontSize="14" Margin="8,0" Foreground="Gray"/>
        <Label Content="Пароль" Grid.Column="0" Grid.Row="7" FontWeight="Bold"
FontSize="14" Margin="8,0"/>
        <PasswordBox x:Name="passw" Password="********"  Grid.Column="0" Grid.Row="8"
FontWeight="Bold" FontSize="14" Margin="8,0" Foreground="Gray"/>
```

```
        <Label Content="Повторить пароль" Grid.Column="0" Grid.Row="9" FontWeight="Bold"
FontSize="14" Margin="8,0"/>
        <PasswordBox x:Name="repPass" Password="********"  Grid.Column="0" Grid.Row="10"
FontWeight="Bold" FontSize="14" Margin="8,0" Foreground="Gray"/>
        <Label Content="Минимальная длина пароля 6 символов"  Grid.Column="1"
Grid.Row="10" FontSize="14" Margin="8,0" />

        <Label Content="Кодовое слово" Grid.Column="0" Grid.Row="11" FontSize="18"
FontWeight="Bold" Margin="10,0"/>
        <Label Content="Кодовое слово (одно слово)" Grid.Column="0" Grid.Row="12"
FontWeight="Bold" FontSize="14" Margin="8,0"/>
        <TextBox x:Name="word" Text="фунтик"  Grid.Column="0" Grid.Row="13"
FontWeight="Bold" FontSize="14" Margin="8,0" Foreground="Gray"/>
        <Label Content="Условия пароля 1" Grid.Column="1" Grid.Row="13"  FontSize="14"
Margin="8,0"/>
        <Label Content="Условия пароля 2" Grid.Column="1" Grid.Row="14"  FontSize="14"
Margin="8,0"/>
        <Label Content="Условия пароля 3" Grid.Column="1" Grid.Row="15"  FontSize="14"
Margin="8,0"/>
        <Label Content="Условия пароля 4" Grid.Column="1" Grid.Row="16"  FontSize="14"
Margin="8,0"/>

        <Label Content="Секретный вопрос для восстановления пароля" Grid.Column="0"
Grid.Row="17" Grid.ColumnSpan="2" FontSize="18" FontWeight="Bold" Margin="10,0"/>
        <Label Content="Секретный вопрос" Grid.Column="0" Grid.Row="18" FontWeight="Bold"
FontSize="14" Margin="8,0"/>

        <ComboBox x:Name="qwest" Grid.Column="0" Grid.Row="19" FontWeight="Bold"
FontSize="14" Margin="8,0" Foreground="Gray"/>

        <Label Content="Ответ на вопрос" Grid.Column="0" Grid.Row="20" FontWeight="Bold"
FontSize="14" Margin="8,0"/>
        <TextBox x:Name="otvet" Text="Париж"  Grid.Column="0" Grid.Row="21"
FontWeight="Bold" FontSize="14" Margin="8,0" Foreground="Gray"/>

        <CheckBox Content="Согласен с условиями" Grid.Column="0" Grid.Row="22"
FontSize="14" />
        <Button x:Name="SignIn" Content="ДАЛЕЕ" Grid.Column="0" Grid.Row="23"
FontSize="14" Click="SignIn_Click" >
            <Button.Background>
                <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                    <GradientStop Color="Black" Offset="0"/>
                    <GradientStop Color="#FFE2EC1B" Offset="1"/>
                </LinearGradientBrush>
            </Button.Background>
        </Button>
        <Button x:Name="go" Content="Перейти" Grid.Row="23" Grid.Column="1"
Margin="100,0" Click="Go_Click"/>
    </Grid>
</Window>
```