ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ИРКУТСКОЙ ОБЛАСТИ «ЧЕРЕМХОВСКИЙ ГОРНОТЕХНИЧЕСКИЙ КОЛЛЕДЖ ИМ. М.И. ЩАДОВА»

РАССМОТРЕНО
на заседании ЦК
«Информатики и ВТ»
Протокол №10
«06» июнь 2023 г.
Председатель: Чипиштанова Д.В.

УТВЕРЖДАЮ
Зам. директора по УР
О.В. Папанова
«07» июнь 2023 г.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

для выполнения
практических (лабораторных) работ студентов
по профессиональному модулю

ПМ.03 Ревьюирование программных продуктов
программы подготовки специалистов среднего звена

09.02.07 Информационные системы и программирование

Разработал преподаватель:
Коровина Н.С.

ДОКУМЕНТ ПОДПИСАН
ЭЛЕКТРОННОЙ ПОДПИСЬЮ

Сертификат: 581AC1D7F71D0307B55183ED46A8CD46
Владелец: Сычев Сергей Николаевич
Действителен: с 05.06.2023 до 28.08.2024

2023 г.

СОДЕРЖАНИЕ

1.ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Методические указания по выполнению практических (лабораторных) работ по профессиональному модулю **ПМ.03 Ревьюирование программных продуктов** предназначены для студентов специальности **09.02.07 «Информационные системы и программирование»,** составлены в соответствии с рабочей программой профессионального модуля **ПМ.03 Ревьюирование программных продуктов** и направлены на достижение следующих целей:

– овладение обучающимися видом профессиональной деятельности

1. Осуществлять ревьюирование программного кода в соответствии с технической документацией.
2. Выполнять измерение характеристик компонент программного продукта для определения соответствия заданным критериям.
3. Производить исследование созданного программного кода с использованием специализированных программных средств с целью выявления ошибок и отклонения от алгоритма.
4. Проводить сравнительный анализ программных продуктов и средств разработки, с целью выявления наилучшего решения согласно критериям, определенным техническим заданием.

– формирование у обучающегося умений осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития;

Методические указания являются частью учебно-методического комплекса по профессиональному модулю **ПМ.03 Ревьюирование программных продуктов** и содержат задания, указания для выполнения практических работ, теоретический минимум и т.п. Перед выполнением практической работы каждый студент обязан показать свою готовность к выполнению работы:

- ответить на теоретические вопросы преподавателя.

По окончанию работы студент оформляет отчет в тетради и защищает свою работу.

В результате выполнения полного объема практических работ студент должен у**меть:**

- Работать с проектной документацией, разработанной с использованием графических языков спецификаций.
- Применять стандартные метрики по прогнозированию затрат, сроков и качества. Определять метрики программного кода специализированными.
- Выполнять оптимизацию программного кода с использованием специализированных программных средств. Использовать методы и технологии тестирования и ревьюирование кода и проектной документации.
- Проводить сравнительный анализ программных продуктов. Проводить сравнительный анализ средств разработки программных продуктов. Разграничивать подходы к менеджменту программных проектов.

При проведении практических работ применяются следующие технологии и методы обучения:

1. проблемно-поисковых технологий
2. тестовые технологии

Правила выполнения практических работ:

1. Запомните порядок проведения практических (лабораторной) работ, правила их оформления.
2. Изучите теоретические аспекты практической (лабораторной) работы
3. Выполните задания практической (лабораторной) работы.
4. Оформите отчет в тетради.

Требования к рабочему месту:

- посадочные места по количеству студентов,
- рабочее место преподавателя,
- дидактическое обеспечение дисциплины:
- сборник заданий для практической (лабораторной) работы студентов

Технические средства обучения:

- Интерактивная доска, компьютер, диапроектор.
- Аппаратное обеспечение компьютеров:

Материнская плата GIGABYTE B450M DS3H

Системная плата совместима с процессорами от AMD. Она поддерживает сокет AM4, этот параметр необходимо учитывать при выборе подходящего чипа. Для доступа в Интернет применяется адаптер RealtekGbE с максимальной скоростью соединения 1000 Мбит/с. Обработкой звука занимается адаптер Realtek ALC887, он поддерживает схему 7.1 для объемного и качественного звучания.

2) Процессор AMD Ryzen 5 1600

Процессоры серии Ryzen – одни из наиболее мощных в линейке от AMD.

Модель имеет архитектуру Zen, ядро Summit Ridge и техпроцесс в 14 нм. Работает устройство с использованием 6 ядер. Диапазон частот 3200–3600 МГц сочетается со множителем 32. Двухканальная память модели принадлежит типу DDR4.

3) Видеокарта AMD Radeon Pro WX 2100

Видеокарта AMD RadeonPro WX 2100 относится к профессиональному классу. Частота работы видеочипа равна 1219 МГц. Установлена скоростная память GDDR5 с эффективной частотой 6000 МГц и пропускной способностью 96 ГБ/с. Максимальное энергопотребление адаптера – лишь 50 Вт.

4) 2 ТБ Жесткий диск Seagate 5900 SkyHawk

В качестве интерфейса подключения изготовители решили применить высокопродуктивный SATA III, благодаря чему скорость обмена данными с другими компонентами ПК может достигать 6 Гбит/с – огромная пропускная способность.

Передача данных осуществляется на скорости, максимум которая может

равняться 180 Мбайт/с.

Оперативнаяпамять AMD Radeon R7 Performance Series 8 ГБ

В 8-гигабайтный комплект входят два 4-гигабайтных модуля. Тип памяти – DDR4. Использует тактовую частоту 2666 МГц. Пропускная способность памяти равна 21300 МБ/с. Помимо тактовой, устройство может использовать другие частоты. Минимально допустимая частота – 1600 МГц. Модули характеризуются таймингами 16-18-18-35. Напряжение питания памяти, равное 1.2 В, соответствует стандартному показателю для DDR4.

- компьютер с лицензионным программным обеспечением и мультимедиа проектор (преподавательский);
- компьютеры с лицензионным программным обеспечением;

Критерии оценки:

Оценки «5» (отлично) заслуживает студент, обнаруживший при выполнении заданий всестороннее, систематическое и глубокое знание учебно - программного материала, учения свободно выполнять профессиональные задачи с всесторонним творческим подходом, обнаруживший познания с использованием основной и дополнительной литературы, рекомендованной программой, усвоивший взаимосвязь изучаемых и изученных дисциплин в их значении для приобретаемой специальности, проявивший творческие способности в понимании, изложении и использовании учебно- программного материала, проявивший высокий профессионализм, индивидуальность в решении поставленной перед собой задачи, проявивший неординарность при выполнении практического задания.

Оценки «4» (хорошо) заслуживает студент, обнаруживший при выполнении заданий полное знание учебно- программного материала, успешно выполняющий профессиональную задачу или проблемную ситуацию, усвоивший основную литературу, рекомендованную в программе, показавший систематический характер знаний, умений и навыков при выполнении теоретических и практических заданий по профессиональному модулю ПМ.06 Сопровождение информационных систем.

Оценки «3» (удовлетворительно) заслуживает студент, обнаруживший при выполнении практических и теоретических заданий знания основного учебно-программного материала в объеме, необходимом для дальнейшей учебной и профессиональной деятельности, справляющийся с выполнением заданий, предусмотренных программой, допустивший погрешности в ответе при защите и выполнении теоретических и практических заданий, но обладающий необходимыми знаниями для их устранения под руководством преподавателя, проявивший какую-то долю творчества и индивидуальность в решении поставленных задач.

Оценки «2» (неудовлетворительно) заслуживает студент, обнаруживший при выполнении практических и теоретических заданий проблемы в знаниях основного учебного материала, допустивший основные принципиальные ошибки в выполнении задания или ситуативной задачи, которую он желал бы решить или предложить варианты решения, который не проявил творческого подхода, индивидуальности.

В соответствии с учебным планом программы подготовки специалистов среднего звена по специальности **09.02.07 «Информационные системы и программирование»** и рабочей программой на практические работы по профессиональному модулю **ПМ.03 Ревьюирование программных продуктов** отводится 34 часов.

2. ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ РАБОТ

№ п/п	Название практической работы (указать раздел программы, если это необходимо)	Количество часов
МДК 03.01 Моделирование и анализ программного обеспечения		
1	Практическая работа №1. «Создание и изучение возможностей репозитория проекта.	2
2	Практическая работа №2. «SharePoint Services. Создание учетных записей портала. Назначение прав доступа к библиотекам. Добавление содержимого. Разработка технической документации. Размещение документации и её обсуждение на портале SharePoint. Организация дискуссии технического задания. Создание кризиса – митинга (дискуссия, посвященная решению кризисной проблемы)	2
3	Практическая работа №3. «Сравнительный анализ средств просмотра видео. Сравнительный анализ браузеров».	2
4	Практическая работа №4. «Сравнительный анализ офисных пакетов. Обратное проектирование алгоритма»	2
5	Практическая работа №5. «Планирование code-review.».	2
6	Практическая работа №6. «Проверки на стороне клиента»	2
7	Практическая работа №7. «Проверки на стороне сервера».	2
8	Практическая работа №8. «Настройки доступа к репозиторию».	2
МДК 03.02. Управление проектами		
9	Практическая работа №1. «Использование метрик программного продукта»	2
10	Практическая работа №2. «Проверка целостности программного кода. Офускация кода»	2
11	Практическая работа №3. «Анализ потоков данных. Использование метрик стилистики»	2
12	Практическая работа №4. «Выполнение измерений характеристик кода в среде Visual Studio».	2
13	Практическая работа №5. «Выполнение измерений характеристик кода в среде (например, Eclipse C/C++ и др.)»	2
14	Практическая работа №6. «Определение проекта и его границ. Организационные структуры. Методологии ведения проектов».	2
15	Практическая работа №7. «Построение команды проекта. Начало и завершение проекта»	2
16	Практическая работа №8. «Коммуникации в проекте. Построение иерархической структуры работ проекта».	2
17	Практическая работа №9. «Смета затрат на разработку и реализацию проекта»	4
Итого		**36**

3. СОДЕРЖАНИЕ ПРАКТИЧЕСКИХ РАБОТ

МДК 03.01 Моделирование и анализ программного обеспечения

Практическая работа №1.

Создание и изучение возможностей репозитория проекта.

Цель: получить опыт практической работы с системой контроля версий на примере Tortoise SVN; получить навыки создания репозитория.

Задание 1

1. Создание хранилища (репозитория) SVN

1.1. На одном из локальных дисков создайте папку, в которой в дальнейшем будет находится репозиторий SVN для проекта. Лучше, чтобы эта папка находилась на достаточно надежном носителе. Пусть это будет папка E:\MyRepository\MyProject.

1.2. Нажмите правой кнопкой мыши на созданной папке и выберите команду TortoiseSVN\Create repository here.

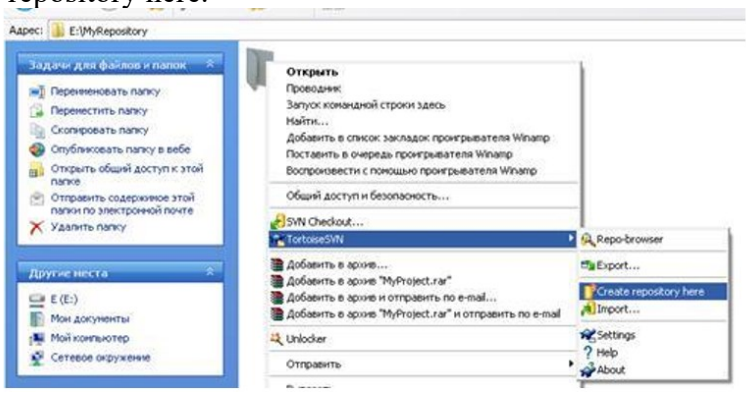


Рис.1.2 Создание репозитория

1.3. Появится окно с сообщением, что репозиторий успешно создан. После чего нажмите кнопку ОК.

1.4.

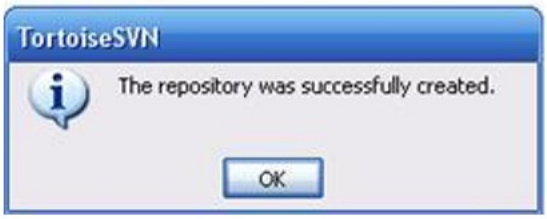


Рис.1.3 Окно сообщения об успешном создании хранилище

1.5. Далее появятся файлы как показано ниже на рисунке/

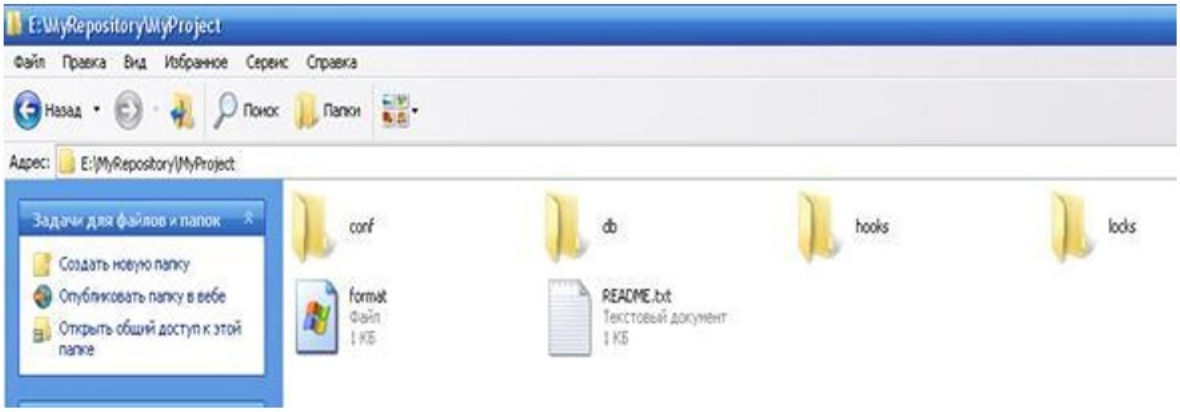


Рис.1.4 Структура репозитория

Примечание: Хранилище будет создано внутри новой папки. Не редактируйте эти файлы самостоятельно!!! В случае возникновения ошибок убедитесь, что папка пуста и доступна для записи.

2. Импорт данных в хранилище Предполагая, что хранилище уже у вас есть, и вы желаете добавить в него новую папку со всей еѐ структурой, просто выполните следующие шаги:
2.1. При помощи обозревателя хранилища (TortoiseSVN → Repo-browser) создайте новую папку проекта под именем 123 непосредственно в хранилище.

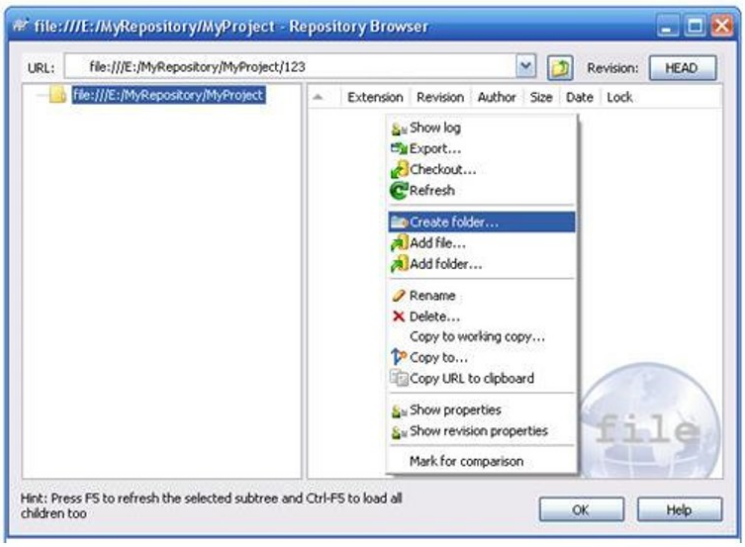


Рис.1.5 Обозреватель хранилища

2.2. Откройте общий доступ хранилище вкладе Доступ пометив флажком в Открыть общий доступ к этой папке и Разрешить изменение файлов в сети.

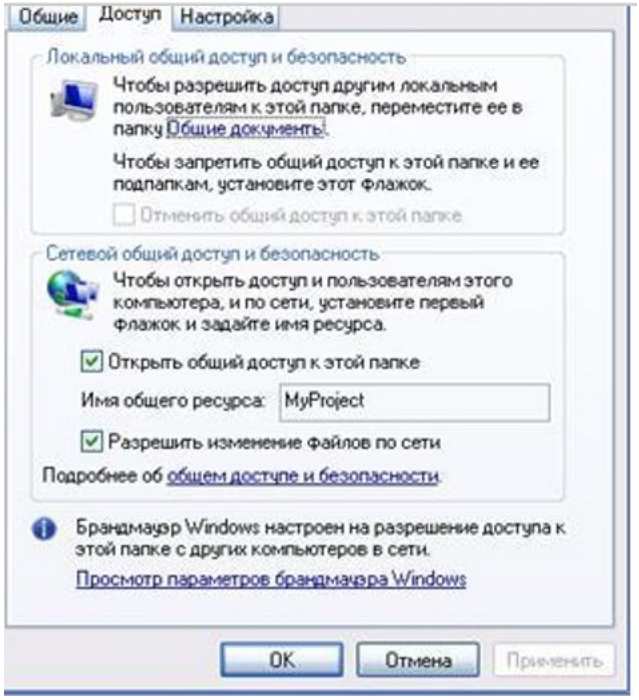


Рис.1.6 Открытие общего доступа к репозиторий.

3. Извлечение рабочей копии

3.1. Для получения рабочей копии вам надо произвести извлечение из хранилища. Выберите в Проводнике Windows папку, в которой хотите поместить вашу рабочую копию. Сделайте правый щелчок для вызова контекстного меню и выберите команду TortoiseSVN -> Извлечь ...,(TortoiseSVN → Checkout...,)

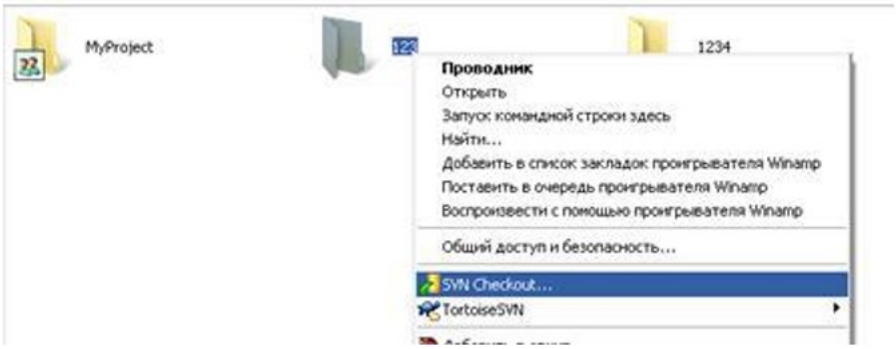


Рис.1.7 Меню для извлечения рабочей копии

3.2. Появившемся диалоговом окне Checkout в поле URL of Repository вставьте путь к репозиторий и нажимаем ОК, далее еще раз нажимаете ОК. Для того что вставить выделите репозиторий – папку и с помощью правой кнопки мыши откройте обозревателя хранилища (TortoiseSVN → Repo-browser) в поле URL скопируйте путь. В поле Checkout directory автоматически появляется путь к рабочей копии, т. е. к папке.

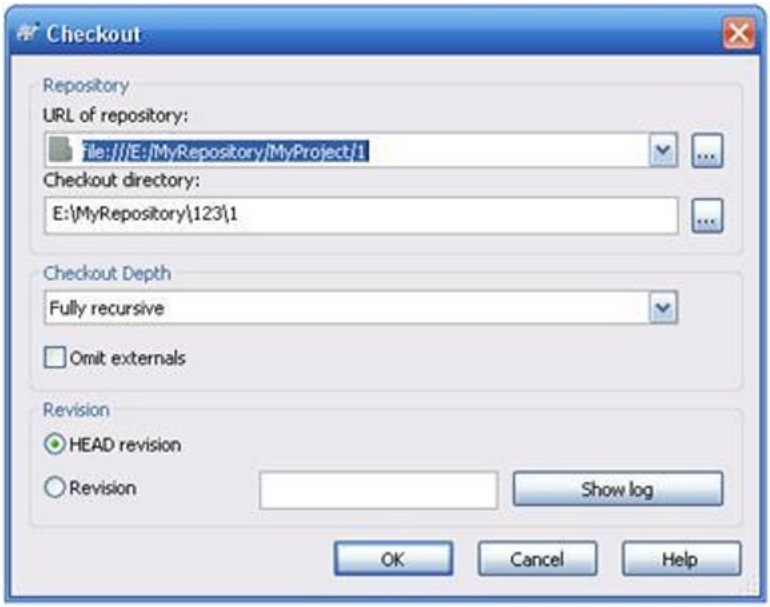


Рис.1.8 Путь к репозиторий

3.3. Выделите рабочую копию, т. е. папку под именем 123 и нажмите F5, либо правой кнопкой мыши нажмите обновить.

3.4. Повторите действия 3.1–3.3 для рабочей копии 1234.

Задание 2:

1. Аналогично проделать те же действие, но проделав эти же действие на двух разных компьютерах.
2. В одном компьютере например, комп1 создаете репозиторию и одну рабочую копию, т. е. папку под именем 123.
3. Во втором компьютере например, комп2 создаете только рабочую копию, т. е. папку под именем 1234.
4. Во втором компьютере соедините рабочую копию с репозиторием указывая путь file:///ip-компьютера/название_репозитория.
5. В комп1 создайте любой файл и отправьте его на комп2. Таким образом проделав 5 ревизий.
6. Чтобы узнать IP- компьютера нажмите пуск →выполнить→ cmd, либо C:\Windows\System32\cmd.exe /k %windir%\system32\ipconfig.exe
7. Откроется окно командой строки. Введите ipconfig, где вы увидите примерное IP-компьютера: 192.168.8.xxx

Итог работы: файлы, отчет

Практическая работа № 2

«SharePoint Services. Создание учетных записей портала. Назначение прав доступа к библиотекам. Добавление содержимого. Разработка технической документации. Размещение документации и её обсуждение на портале SharePoint. Организация дискуссии технического задания. Создание кризиса – митинга (дискуссия, посвященная решению кризисной проблемы)

Цель Изучить:

- способы создания учетных записей портала.

- назначение прав доступа к библиотекам.
- добавление содержимого.
- разработку технической документации.
- размещение документации и её обсуждение на портале SharePoint.
- организацию дискуссии технического задания.
- создание кризиса – митинга (дискуссия, посвященная решению кризисной проблемы)

Задание:

1. Создать портал SharePoint.
2. Добавить новых пользователей портала SharePoint
3. На портале создать: библиотеки документов и рисунков, список, доску обсуждений.
4. Определить права доступа к созданным библиотекам и списку.
5. Добавить материалы в библиотеки.
6. Создать дочерний сайт.
7. Настроить параметры времени портала, изменить тему. Прочие настройки – по усмотрению.

Итог работы: отчет

Практическая работа №3.

Сравнительный анализ средств просмотра видео.
Сравнительный анализ браузеров

Цель: изучение и сравнение офисные пакеты; овладеть навыками прямого и обратного проектирования; получение навыков выполнения анализа.

Задание 1. Сравнить средств для просмотра видео.

Задание 2. Сравнить браузеров. Вывод оформить в таблицу

Обязательные критерии для сравнения (по каждому критерию оформить таблицу)
1. Безопасность и приватность.
2. Удобство.
3.Переносимость

Итог работы: отчет.

Практическая работа №4.

Сравнительный анализ офисных пакетов.
Обратное проектирование алгоритма

Цель: изучение и провести анализ офисных пакетов; овладеть навыками прямого и обратного проектирования; получение навыков выполнения анализа.

Задание 1. Сравнить и проанализировать офисные пакеты MS Windows и OpenOffice.

Задание 2. Реализовать прямое проектирование в архитектуре «файл-сервер». Изменить структуру БД и осуществить обратное проектирование. Реализовать прямое проектирование в архитектуре «клиент-сервер», сгенерировать SQL – код создания базы данных на основе физической модели данных.

Этап прямого проектирования в архитектуре «файл-сервер».

Рассмотрим исходные логические и физические модели данных (Рис.1, Рис.2).

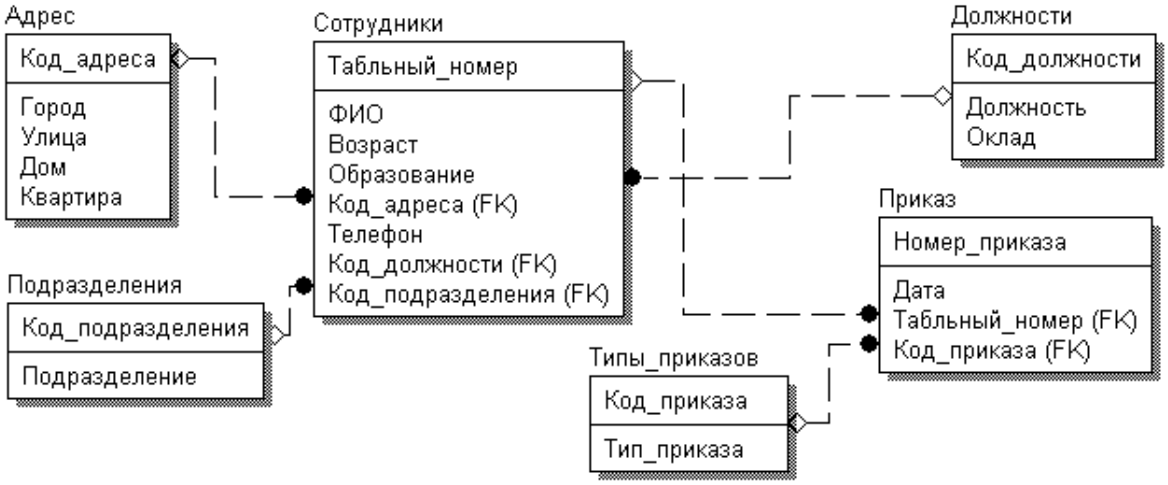


Рис.1. Логическая модель проектируемой ИС

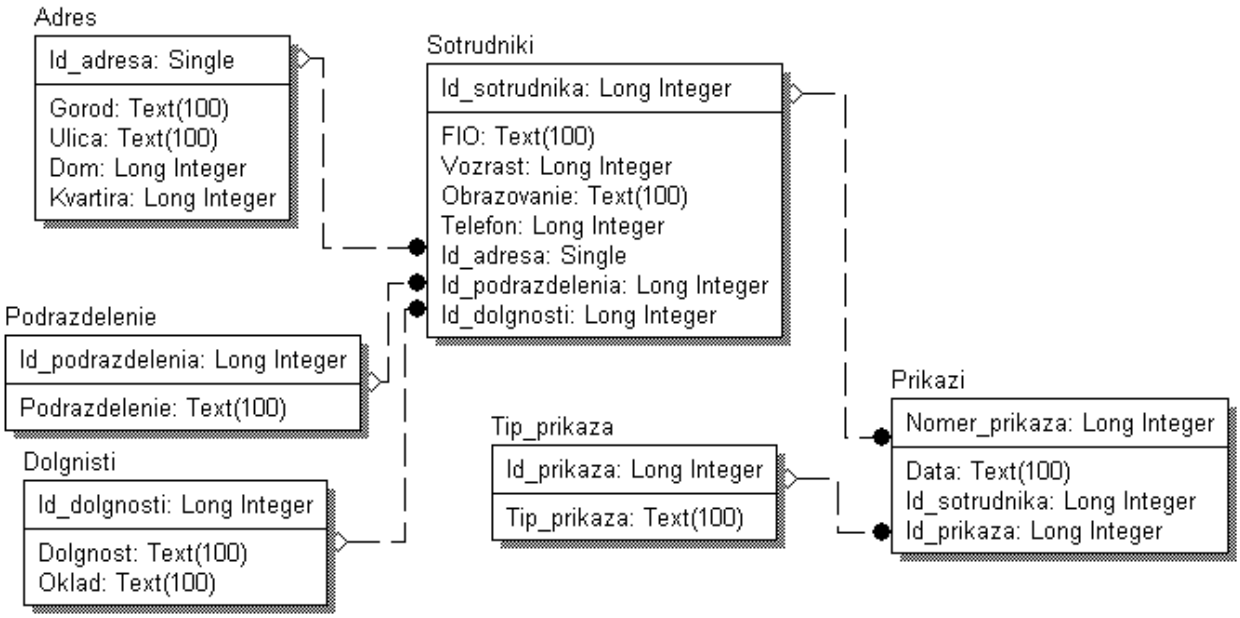


Рис.2. Физическая модель проектируемой ИС

Открываем физическую модель ИС и выбираем Access в качестве нужного типа СУБД, после чего типы данных в физической модели изменятся, так как по умолчанию она может быть настроена на другую СУБД.

Создаем пустую базу данных в Access и подключаемся к ней (Рис.3, Рис.4).

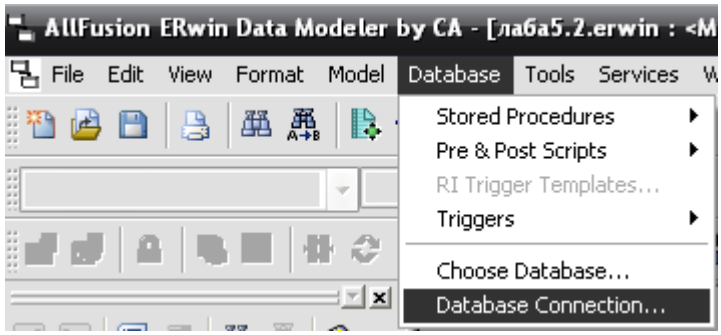


Рис.3. Подключение к СУБД Access

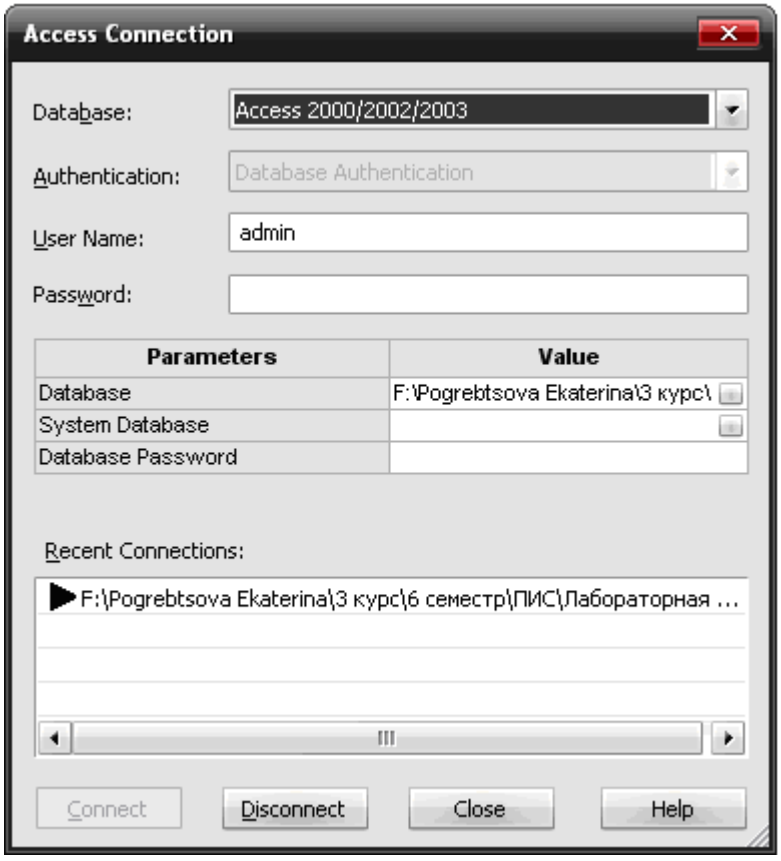


Рис.4. Выбор БД Access

Далее в меню выбираем Tools/ Forward Engineer/Shema Generation.

В открывшемся окне на вкладке Options в пункте Index поставили галочки напротив пунктов Primary Key и Foreign Key, отвечающих за генерацию первичных и внешних ключей (Рис.5).

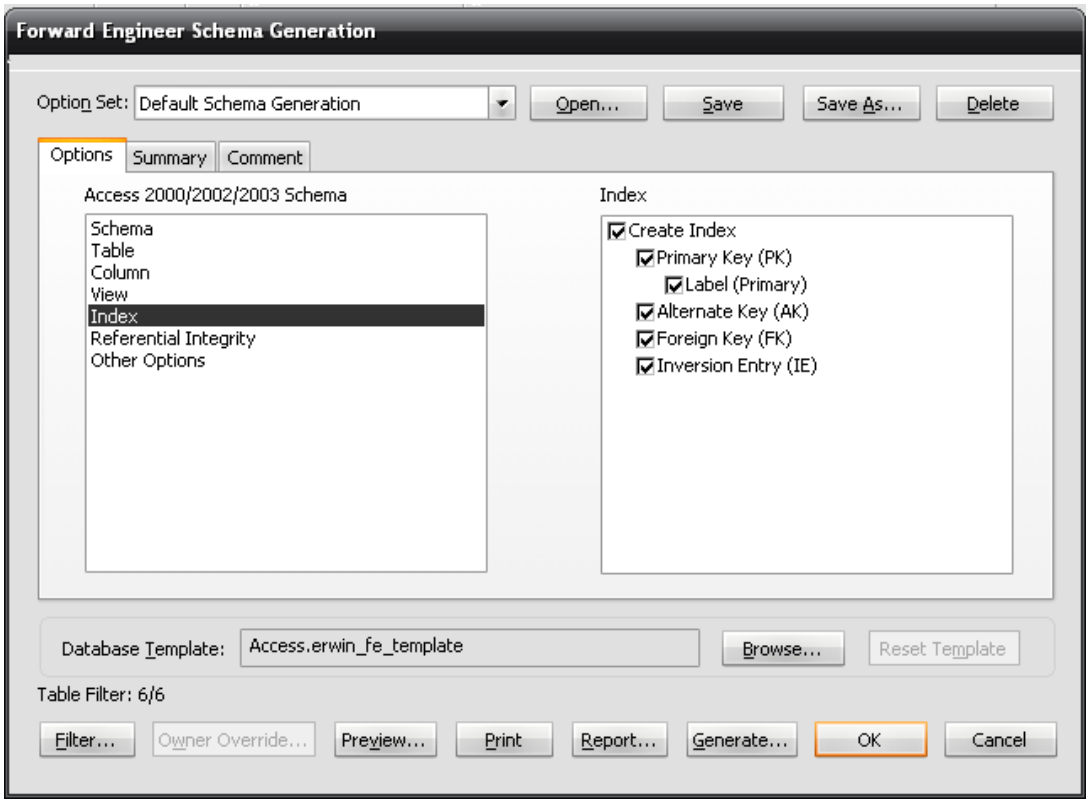


Рис.5. Установки по генерации схемы для базы данных Access

После завершения операции по переносу физической модели в Access заходим в полученную базу данных и проверяем результат (Рис.6).

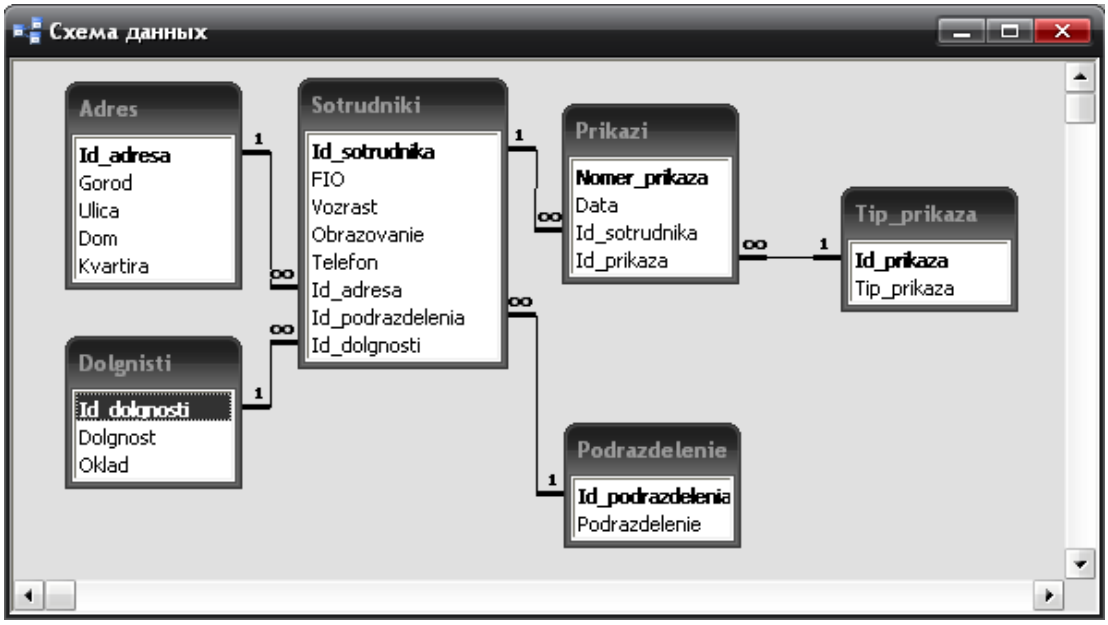


Рис.6. Схема данных в Access

Этап обратного проектирования.

В базе данных Access в таблице Адрес добавили поле e-mail и сохранили изменения. Далее зашли в Erwin и в меню выбрали Tools/ Reverse Engineer. В открывшемся окне выбрали тип новой модели - физическая, и СУБД из которой будем импортироваться физическая модель – Access (Рис.7).

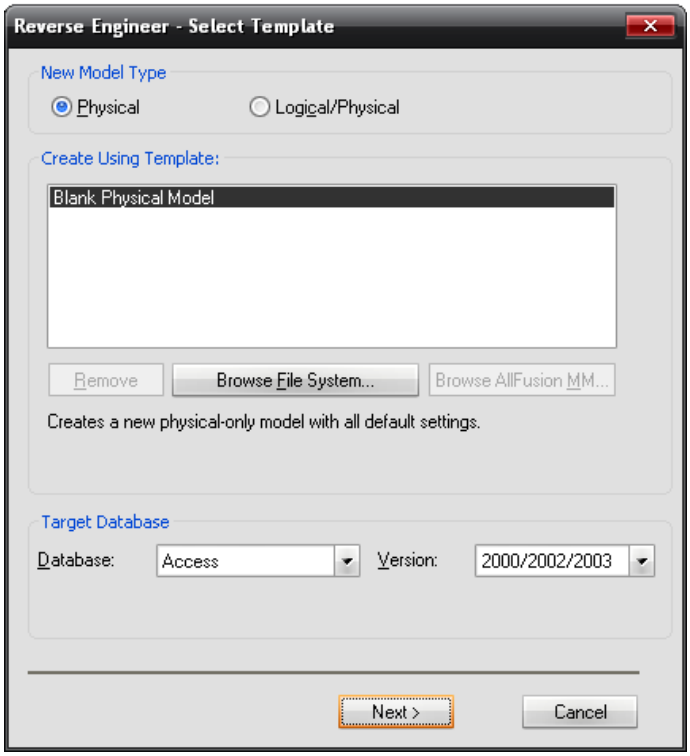


Рис.7. Установки обратного проектирования

Далее настраиваем параметры проектирования (Рис.8).

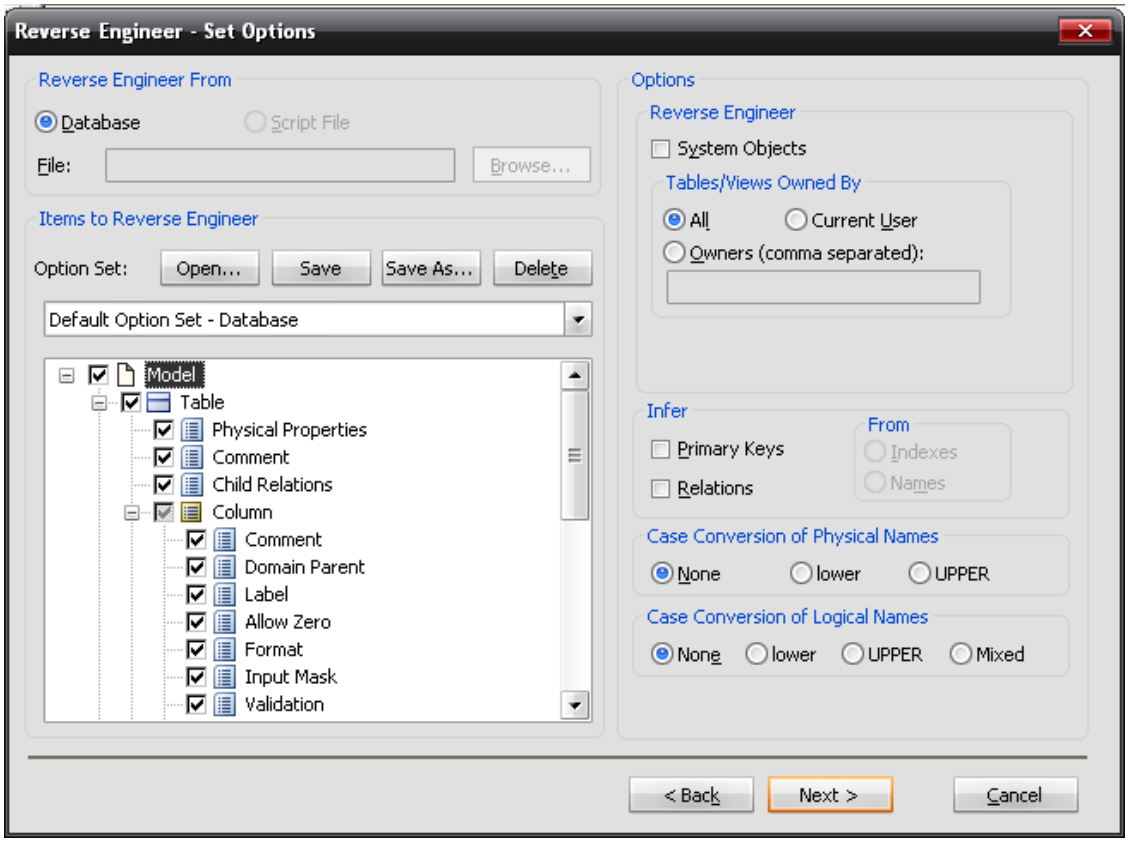


Рис.8. Установки по генерированию схемы для Erwin.

Подключение к Access аналогично режиму прямого проектирования.

Получаем физическую модель (Рис.9).

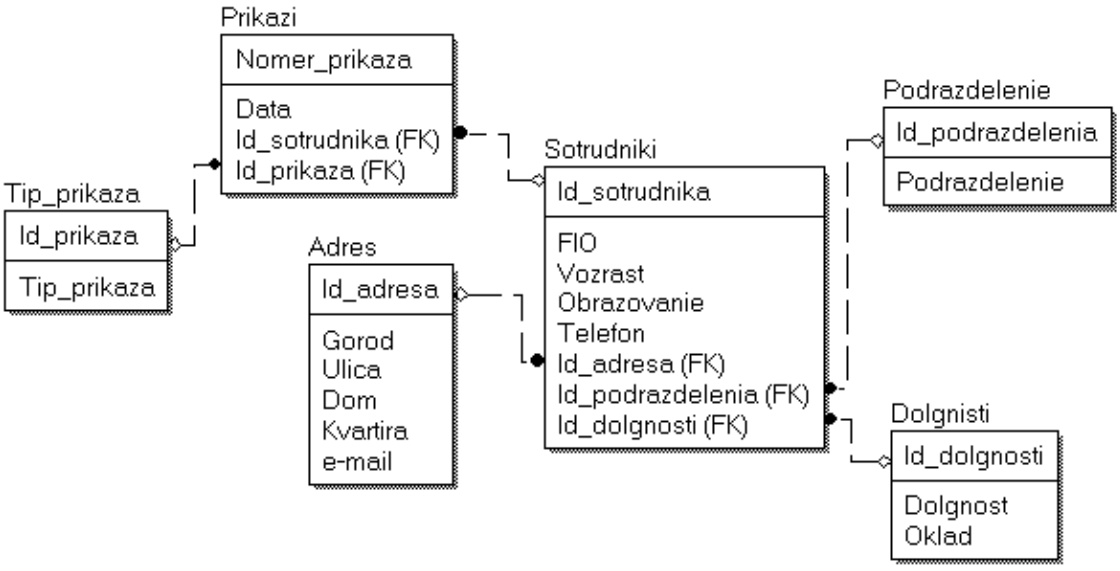


Рис.9. Физическая модель, полученная из БД Access

Этап проектирования БД для архитектуры “клиент-сервер”.

В среде Erwin открыли физическую модель ИС, изменили тип СУБД на Microsoft SQL Server, в меню выбрали *Tools/ Forward Engineer/Shema Generation*.

В открывшемся окне на вкладке *Options* в пункте *Index* поставили галочки напротив пунктов *Primary Key* и *Foreign Key*, отвечающих за генерацию первичных и внешних ключей. Нажали кнопку *Preview* (Рис.10).

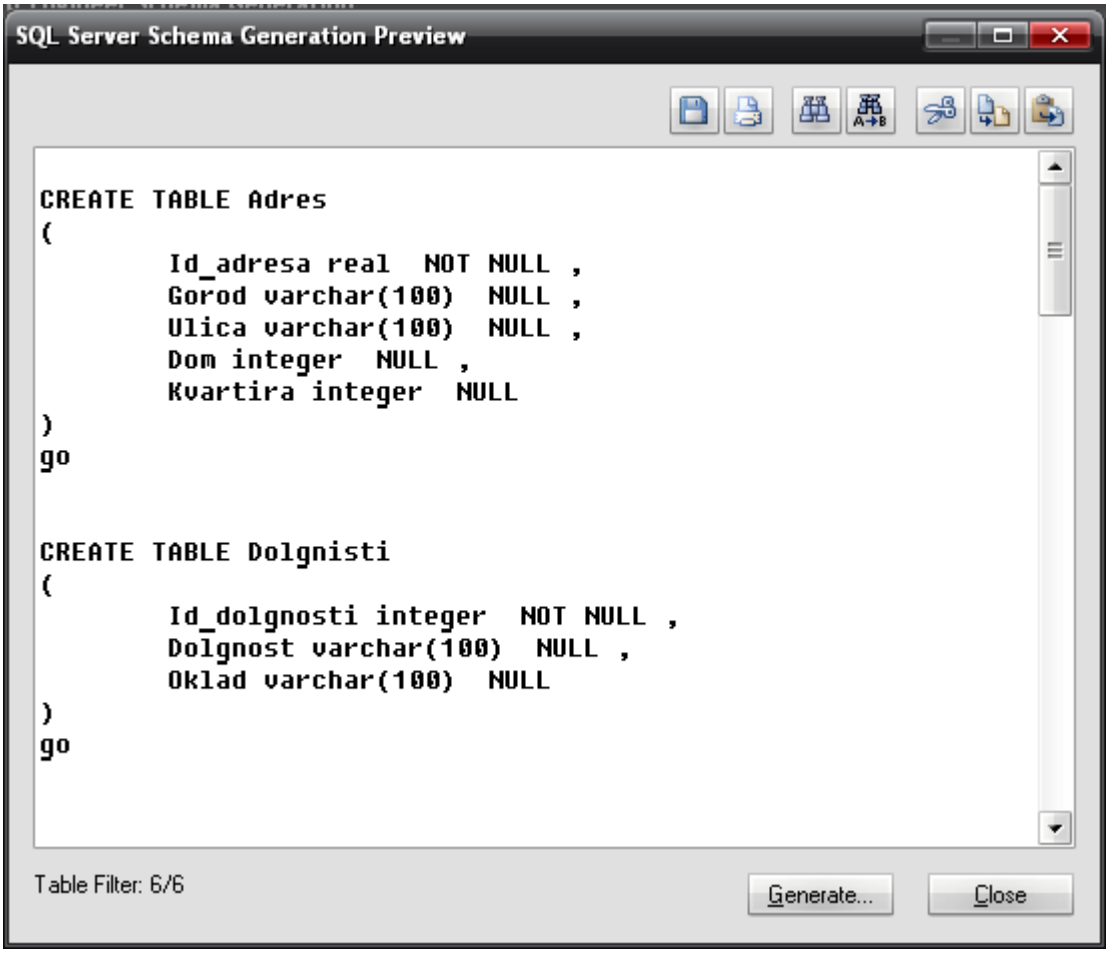


Рис.10. Генерация SQL-кода для MS SQL

Итог работы: отчет, файлы

Практическая работа №5.

«Планирование code-review».

Цель: освоение процесс проверки и анализа кода задачи разработчиком перед ее релизом.

Задание:

1. Выбрать инструмент для Code Review:
   - GitHub Code Review https://github.com/features/code-review/
   - Upsource https://www.jetbrains.com/ru-ru/upsource/
   - Crucible https://www.atlassian.com/ru/software/crucible
   - Collaborator https://smartbear.com/product/collaborator/overview/
   - Beanstalk https://beanstalkapp.com/

2. Выполнить Чеклист для ревьювера:
   - Ознакомиться и понять цель и суть задачи
   - Проверить общую архитектуру и подход к решению
   - Проверить реализацию
   - Проверить мелкие детали (имена функций и переменных и т.д.)
   - Проверить наличие тестов и документации по необходимости
3. Подвести итоги
   - Список ToDo: изменения, которые необходимо внести в код после ревью

- Вопросы: обозначить свои вопросы по частям кода, которые непонятны после ревью
- Предложения по улучшению: внести свои предложения и пожелания по коду задачи и/или связанных задач. Например, договориться о создании задачи по обновлению старого метода в других участках кода на новый и завести на это ToDo и задачу в трекере задач и поставить ее в тех. долг команды.

Итог работы: файл, защита

Практическая работа №6.

Проверки на стороне клиента.

Цель: рассмотреть Управлении этапом разработки кода программных компонентов». «Построение приложений в командном проекте

Задание:
В соответствии с подготовленным техническим заданием выполнить разработку спецификаций на программный продукт, которые должны включать:
- спецификации процессов;
- словарь терминов;
- диаграммы переходов состояний;
- диаграммы потоков с детализацией.

2.Оформить отчет.
Содержание отчета:
-тема практической работы;
- цель работы;
- задание на практическую работу;
- разработанные спецификации процессов;
- словарь терминов;
- диаграммы переходов состояний;
- диаграммы потоков с детализацией;
- выводы по проделанной работе.

Итог работы: отчет, файлы

Практическая работа №7.

«Проверки на стороне сервера».

Цель: рассмотреть тестовые случае в командном проекте, формирование отчетов.
Задание:

1. Проведите сравнительный анализ состава рабочих элементов в шаблонах MSFforCMMIProcessImprovement 6.0 и MSFforAgileSoftwareDevelopment 6.0.
2. Проанализируйте процесс управления рисками проектов в методологии MSFforCMMIProcessImprovement 6.0.
3. Проанализируйте процесс управления командами проектов в методологии MSFforCMMIProcessImprovement 6.0.
4. Написать отчет в тетради

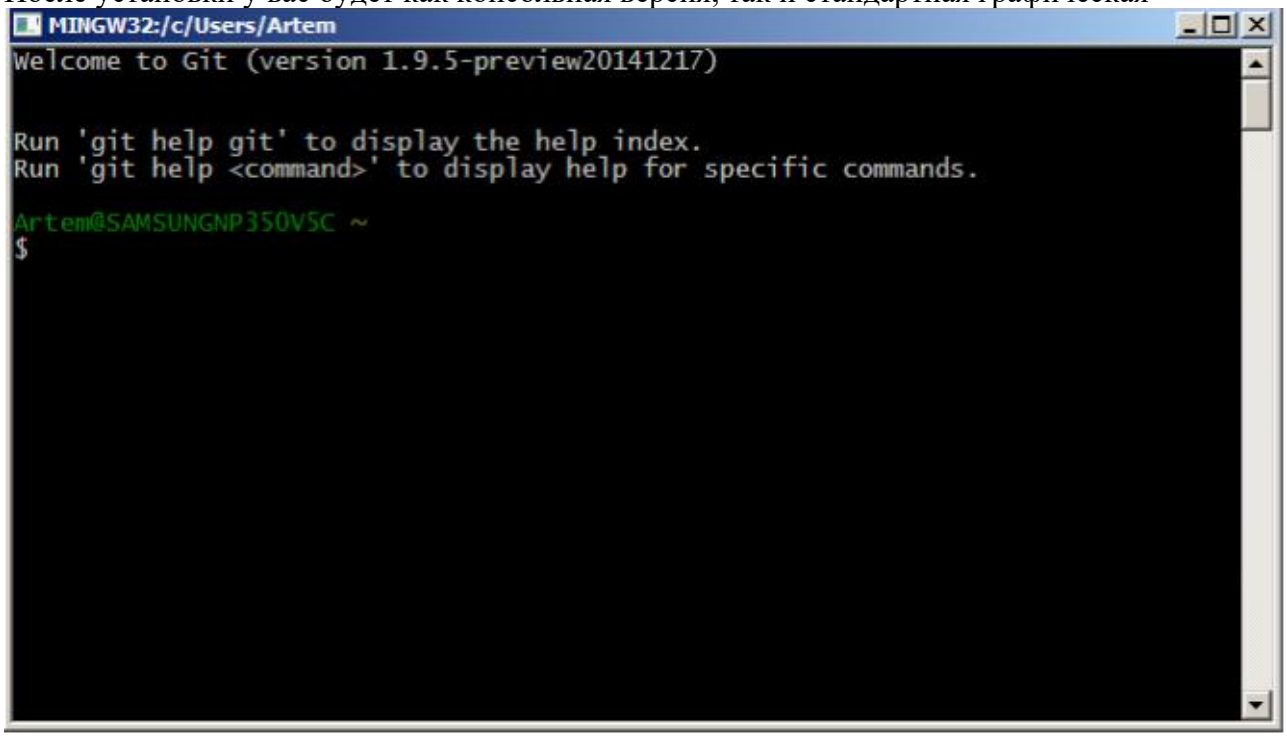
Итог работы: отчет

Практическая работа №8.

Настройки доступа к репозиторию.

Цель: ознакомиться с основами системы контроля версий Git, научиться работать с регулярными выражениями

Задание1. Установить Git в Windows
http://msysgit.github.com/
После установки у вас будет как консольная версия, так и стандартная графическая



Первоначальная настройка Git

Теперь, когда Git установлен в вашей системе, необходимо настроить среду для работы с Git'ом под себя. Это нужно сделать только один раз — при обновлении версии Git'а настройки сохранятся. Но вы можете поменять их в любой момент, выполнив те же команды снова.

В состав Git'а входит утилита git config, которая позволяет просматривать и устанавливать параметры, контролирующие все аспекты работы Git'а и его внешний вид.

Имя пользователя

Первое, что вам следует сделать после установки Git'а, — указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в Git'е содержит эту информацию, и она включена в коммиты, передаваемые вами, и не может быть далее изменена:

```
$ git config global user.name "John Doe"
$ git config global user.email johndoe@example.com
```

Проверка настроек

Если вы хотите проверить используемые настройки, можете использовать команду git config list, чтобы показать все, которые Git найдёт:

```
$ git config list user.name=Scott Chacon
user.email=schacon@gmail.com color.status=auto
```

Если вам нужна помощь при использовании Git'а, есть два способа открыть страницу руководства по любой команде Git'а:

```
$ git help <команда> $ git <команда> help
```

Задание 2. Создание Git-репозитория

Для создания Gitрепозитория существуют два основных подхода. Первый подход — импорт в Git уже существующего проекта или каталога. Второй — клонирование уже существующего репозитория с сервера.

Создание репозитория в существующем каталоге

Если вы собираетесь начать использовать Git для существующего проекта, то вам необходимо перейти в проектный каталог и в командной строке ввести

```
$ git init
```

Эта команда создаёт в текущем каталоге новый подкаталог с именем .git содержащий все необходимые файлы репозитория — основу Gitрепозитория. На этом этапе ваш проект ещё не находится под версионным контролем.

Если вы хотите добавить под версионный контроль существующие файлы (в отличие от пустого каталога), вам стоит проиндексировать эти файлы и осуществить первую фиксацию изменений. Осуществить это вы можете с помощью нескольких команд git add указывающих индексируемые файлы, а
затем commit:

```
$ git add *.c
$ git add README
$ git commit m 'initial project version'
```

Мы разберём, что делают эти команды чуть позже. На данном этапе, у вас есть Git-репозиторий с добавленными файлами и начальным коммитом.

Клонирование существующего репозитория

Если вы желаете получить копию существующего репозитория Git, например, проекта, в котором вы хотите поучаствовать, то вам нужна команда git clone. Если вы знакомы с другими системами контроля версий, такими как Subversion, то заметите, что команда называется clone, а не checkout. Это важное отличие — Git получает копию практически всех данных, что есть на сервере. Каждая версия каждого файла из истории проекта забирается (pulled) с сервера, когда вы выполняете git clone. Фактически, если серверный диск выйдет из строя, вы можете использовать любой из клонов на любом из клиентов, для того чтобы вернуть сервер в то состояние, в котором он находился в момент клонирования.

Клонирование репозитория осуществляется командой git clone [url]. Например, если вы хотите клонировать библиотеку Ruby Git, известную как Grit, вы можете сделать это следующим образом:

```
$ git clone git://github.com/schacon/grit.git
```

Эта команда создаёт каталог с именем grit, инициализирует в нём каталог .git, скачивает все данные для этого репозитория и создаёт (checks out) рабочую копию последней версии. Если вы зайдёте в новый каталог grit, вы увидите в нём проектные файлы, пригодные для работы и использования. Если вы хотите клонировать репозиторий в каталог, отличный от grit, можно это указать в следующем параметре командной строки:

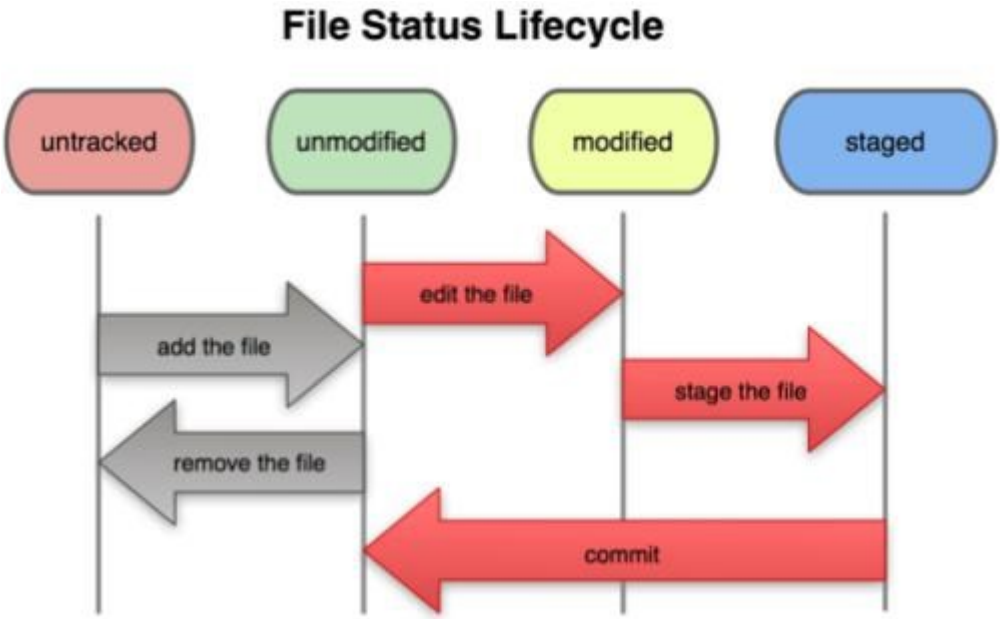
```
$ git clone git://github.com/schacon/grit.git mygrit
```

Эта команда делает всё то же самое, что и предыдущая, только результирующий каталог будет назван mygrit.

В Git'e реализовано несколько транспортных протоколов, которые вы можете использовать. В предыдущем примере использовался протокол git://, вы также можете встретить http(s):// или user@server:/path.git, использующий протокол передачи SSH.

Запись изменений в репозиторий

Итак, у вас имеется Gitрепозиторий и рабочая копия файлов для некоторого проекта. Вам нужно делать некоторые изменения и фиксировать “снимки” состояния (snapshots) этих изменений в вашем репозитории каждый раз, когда проект достигает состояния, которое вам хотелось бы сохранить.



Запомните, каждый файл в вашем рабочем каталоге может находиться в одном из двух состояний: под версионным контролем (отслеживаемые) и нет (неотслеживаемые). Отслеживаемые файлы — это те файлы, которые были в последнем слепке состояния проекта (snapshot); они могут быть неизменёнными, изменёнными или подготовленными к коммиту (staged). Неотслеживаемые файлы — это всё остальное, любые файлы в вашем рабочем каталоге, которые не входили в ваш последний слепок состояния и не подготовлены к коммиту. Когда вы впервые клонируете репозиторий, все файлы будут отслеживаемыми и неизменёнными, потому что вы только взяли их из хранилища (checked them out) и ничего пока не редактировали.

Как только вы отредактируете файлы, Git будет рассматривать их как изменённые, т.к. вы изменили их с момента последнего коммита. Вы индексируете (stage) эти изменения и затем фиксируете все индексированные изменения, а затем цикл повторяется. Этот жизненный цикл изображён на рисунке 2.1.

Определение состояния файлов

Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся

— это команда git status. Если вы выполните эту команду сразу после клонирования, вы увидите чтото вроде этого:

```
$ git status
# On branch master
nothing to commit, working directory clean
```

Это означает, что у вас чистый рабочий каталог, другими словами — в нём нет отслеживаемых изменённых файлов. Git также не обнаружил неотслеживаемых файлов, в противном случае они бы были перечислены здесь. И наконец, команда сообщает вам на какой ветке (branch) вы сейчас находитесь. Пока что это всегда ветка master — это ветка по умолчанию; в этой лабораторной работе это не важно.

Отслеживание новых файлов

Для того чтобы начать отслеживать (добавить под версионный контроль) новый файл, используется команда git add. Чтобы начать отслеживание файла README, вы можете выполнить следующее:

```
$ git add README
```

Если вы выполните команду status, то увидите, что файл README теперь отслеживаемый и индексированный:

```
$ git status
#On branch master
#Changes to be committed:
#(use "git reset HEAD <file>..." to unstage)
# new file: README
#
```

Вы можете видеть, что файл проиндексирован по тому, что он находится в секции “Changes to be committed”. Если вы выполните коммит в этот момент, то версия файла, существовавшая на момент выполнения вами команды git add, будет добавлена в историю снимков состояния. Как вы помните, когда вы ранее выполнили git init, вы затем выполнили git add (файлы) — это было сделано для того, чтобы добавить файлы в вашем каталоге под версионный контроль. Команда git add принимает параметром путь к файлу или каталогу, если это каталог, команда рекурсивно добавляет (индексирует) все файлы в данном каталоге.

Индексация изменённых файлов

Давайте модифицируем файл, уже находящийся под версионным контролем. Если вы измените отслеживаемый файл benchmarks.rb и после этого снова выполните команду status, то результат будет примерно следующим:

```
$ git status
#On branch master
#Changes to be committed:
#(use "git reset HEAD <file>..." to unstage)

#new file: README
#
#Changes not staged for commit:
#(use "git add <file>..." to update what will be committed)

#modified: benchmarks.rb
#
```

Файл benchmarks.rb находится в секции “Changes not staged for commit” — это означает, что отслеживаемый файл был изменён в рабочем каталоге, но пока не проиндексирован. Чтобы проиндексировать его, необходимо выполнить команду git add (это многофункциональная команда, она используется для добавления под версионный контроль новых файлов, для индексации изменений, а также для других целей, например для указания файлов с исправленным конфликтом слияния). Выполним git add, чтобы проиндексировать benchmarks.rb, а затем снова выполним git status:

```
$ git add benchmarks.rb $ git status
#On branch master
#Changes to be committed:
#(use "git reset HEAD <file>..." to unstage)

#new file: README
```

```
#modified: benchmarks.rb
```

Теперь оба файла проиндексированы и войдут в следующий коммит. В этот момент вы, предположим, вспомнили одно небольшое изменение, которое вы хотите сделать в benchmarks.rb до фиксации. Вы открываете файл, вносите и сохраняете необходимые изменения и вроде бы готовы к коммиту. Но давайте ещё раз выполним git status:

```
$ git status
#On branch master
#Changes to be committed:
#(use "git reset HEAD <file>..." to unstage)

#new file: README
#modified: benchmarks.rb
#
#Changes not staged for commit:
#(use "git add <file>..." to update what will be committed)

#modified: benchmarks.rb
#
```

Теперь benchmarks.rb отображается как проиндексированный и непроиндексированный одновременно. Как такое возможно? Такая ситуация наглядно демонстрирует, что Git индексирует файл в точности в том состоянии, в котором он находился, когда вы выполнили команду git add. Если вы выполните коммит сейчас, то файл benchmarks.rb попадёт в коммит в том состоянии, в котором он находился, когда вы последний раз выполняли команду git add, а не в том, в котором он находится в вашем рабочем каталоге в момент выполнения git commit. Если вы изменили файл после выполнения git add, вам придётся снова выполнить git add, чтобы проиндексировать последнюю версию файла:

```
$ git add benchmarks.rb $ git status
#On branch master
#Changes to be committed:
#(use "git reset HEAD <file>..." to unstage)

#new file: README
#modified: benchmarks.rb
#
```

Игнорирование файлов

Зачастую, у вас имеется группа файлов, которые вы не только не хотите автоматически добавлять в репозиторий, но и видеть в списках неотслеживаемых. К таким файлам обычно относятся автоматически генерируемые файлы (различные логи, результаты сборки программ и т.п.). В таком случае, вы можете создать файл .gitignore с перечислением шаблонов соответствующих таким файлам. Вот пример файла

.gitignore: *.[oa] *~

Первая строка предписывает Git'у игнорировать любые файлы заканчивающиеся на .o или .a — объектные и архивные файлы, которые могут появиться во время сборки кода. Вторая строка предписывает игнорировать все файлы заканчивающиеся на тильду (~), которая используется во многих текстовых редакторах, например Emacs, для обозначения временных файлов. Вы можете также включить каталоги log, tmp или pid; автоматически

создаваемую документацию; и т.д. и т.п. Хорошая практика заключается в настройке файла .gitignore до того, как начать серьёзно работать, это защитит вас от случайного добавления в репозиторий файлов, которых вы там видеть не хотите.

Фиксация изменений

Теперь, когда ваш индекс настроен так, как вам и хотелось, вы можете зафиксировать свои изменения. Запомните, всё, что до сих пор не проиндексировано — любые файлы, созданные или изменённые вами, и для которых вы не выполнили git add после момента редактирования — не войдут в этот коммит. Они останутся изменёнными файлами на вашем диске. В нашем случае, когда вы в последний раз выполняли git status, вы видели что всё проиндексировано, и вот, вы готовы к коммиту. Простейший способ зафиксировать изменения — это набрать git commit:

$ git commit

Эта команда откроет выбранный вами текстовый редактор. (Редактор устанавливается системной переменной $EDITOR — обычно это vim или emacs, хотя вы можете установить ваш любимый с помощью команды git config global core.editor).

В редакторе будет отображён следующий текст (это пример окна Vim'a):

```
#Please enter the commit message for your changes. Lines starting
#with '#' will be ignored, and an empty message aborts the commit.
#On branch master
#Changes to be committed:
#(use "git reset HEAD <file>..." to unstage)

#new file: README
#modified: benchmarks.rb
~
~
~
".git/COMMIT_EDITMSG" 10L, 283C
```

Вы можете видеть, что комментарий по умолчанию для коммита содержит закомментированный

результат работы ("выхлоп") команды git status и ещё одну пустую строку сверху. Вы можете удалить эти комментарии и набрать своё сообщение или же оставить их для напоминания о том, что вы фиксируете. (Для ещё более подробного напоминания, что же именно вы поменяли, можете передать аргумент v в команду git commit. Это приведёт к тому, что в комментарий будет также помещена дельта/diff изменений, таким образом вы сможете точно увидеть всё, что сделано.) Когда вы выходите из редактора, Git создаёт для вас коммит с этим сообщением (удаляя комментарии и вывод diff'a).

Есть и другой способ — вы можете набрать свой комментарий к коммиту в командной строке вместе с командой commit, указав его после параметра m, как в следующем примере:

$ git commit m "Story 182: Fix benchmarks for speed" [master]: created 463dc4f: "Fix benchmarks for speed" 2 files changed, 3 insertions(+), 0 deletions()
create mode 100644 README

Итак, вы создали коммит! Вы можете видеть, что коммит вывел вам немного информации о себе: на какую ветку вы выполнили коммит (master), какая контрольная сумма SHA1 у этого коммита (463dc4f), сколько файлов было изменено, а также статистику по добавленным/удалённым строкам в этом коммите.

Запомните, что коммит сохраняет снимок состояния вашего индекса. Всё, что вы не проиндексировали, так и остается в рабочем каталоге как изменённое; вы можете сделать

ещё один коммит, чтобы добавить эти изменения в репозиторий. Каждый раз, когда вы делаете коммит, вы сохраняете снимок состояния вашего проекта, который позже вы можете восстановить или с которым можно сравнить текущее состояние.

Игнорирование индексации

Несмотря на то, что индекс может быть удивительно полезным для создания коммитов именно такими, как вам и хотелось, он временами несколько сложнее, чем вам нужно в процессе работы. Если у вас есть желание пропустить этап индексирования, Git предоставляет простой способ. Добавление параметра a в команду git commit заставляет Git автоматически индексировать каждый уже отслеживаемый на момент коммита файл, позволяя вам обойтись без git add:

```
$ git status
#On branch master

#Changes not staged for commit:

#modified: benchmarks.rb

$ git commit a m 'added new benchmarks' [master 83e38c7] added new benchmarks
1 files changed, 5 insertions(+), 0 deletions()
```

Обратите внимание на то, что в данном случае перед коммитом вам не нужно выполнять git add
для файла benchmarks.rb.

Просмотр истории коммитов

После того как вы создадите несколько коммитов, или же вы склонируете репозиторий с уже существующей историей коммитов, вы, вероятно, захотите оглянуться назад и узнать, что же происходило с этим репозиторием. Наиболее простой и в то же время мощный инструмент для этого — команда git log. Данные примеры используют очень простой проект, названный simplegit. Чтобы получить этот
проект, выполните:

```
git clone git://github.com/schacon/simplegitprogit.git
```

В результате выполнения git log в данном проекте, вы должны получить чтото вроде этого: `$ git log`

```
commit ca82a6dff817ec66f44342007202690a93763949 Author: Scott Chacon
<schacon@geemail.com> Date: Mon Mar 17 21:52:11 2008 0700
changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 Author: Scott Chacon
<schacon@geemail.com> Date: Sat Mar 15 16:40:33 2008 0700
removed unnecessary test code

commit a11bef06a3f659402fe7563abf99ad00de2209e6 Author: Scott Chacon
<schacon@geemail.com> Date: Sat Mar 15 10:31:28 2008 0700
first commit
```

По умолчанию, без аргументов, git log выводит список коммитов созданных в данном репозитории в обратном хронологическом порядке. То есть самые последние коммиты показываются первыми. Как вы можете видеть, эта команда отображает каждый коммит

вместе с его контрольной суммой SHA1, именем и электронной почтой автора, датой создания и комментарием.

Существует превеликое множество параметров команды git log и их комбинаций, для того чтобы показать вам именно то, что вы ищете.

Отмена изменений

На любой стадии может возникнуть необходимость чтолибо отменить. Здесь мы рассмотрим несколько основных инструментов для отмены произведённых изменений. Будьте осторожны, ибо не всегда можно отменить сами отмены. Это одно из немногих мест в Git'e, где вы можете потерять свою работу если сделаете чтото неправильно.

Изменение последнего коммита

Одна из типичных отмен происходит тогда, когда вы делаете коммит слишком рано, забыв добавить какието файлы, или напутали с комментарием к коммиту. Если вам хотелось бы сделать этот коммит ещё раз, вы можете выполнить commit с опцией amend:
$ git commit amend

Эта команда берёт индекс и использует его для коммита. Если после последнего коммита не было никаких изменений (например, вы запустили приведённую команду сразу после предыдущего коммита), то состояние проекта будет абсолютно таким же и всё, что вы измените, это комментарий к коммиту.

Появится всё тот же редактор для комментариев к коммитам, но уже с введённым комментарием к последнему коммиту. Вы можете отредактировать это сообщение так же, как обычно, и оно перепишет предыдущее.

Для примера, если после совершения коммита вы осознали, что забыли проиндексировать изменения в файле, которые хотели добавить в этот коммит, вы можете сделать чтото подобное:

```
$ git commit m 'initial commit' $ git add forgotten_file
$ git commit amend
```

Все три команды вместе дают один коммит — второй коммит заменяет результат первого.

Работа с удалёнными репозиториями

Чтобы иметь возможность совместной работы над какимлибо Gitпроектом, необходимо знать, как управлять удалёнными репозиториями. Удалённые репозитории — это модификации проекта, которые хранятся в интернете или ещё гдето в сети. Их может быть несколько, каждый из которых, как правило, доступен для вас либо только на чтение, либо на чтение и запись. Совместная работа включает в себя управление удалёнными репозиториями и помещение (push) и получение (pull) данных в и из них тогда, когда нужно обменяться результатами работы. Управление удалёнными репозиториями включает умение добавлять удалённые репозитории, удалять те из них, которые больше не действуют, умение управлять различными удалёнными ветками и определять их как отслеживаемые (tracked) или нет и прочее. Данный раздел охватывает все перечисленные навыки по управлению удалёнными репозиториями.

Отображение удалённых репозиториев

Чтобы просмотреть, какие удалённые серверы у вас уже настроены, следует выполнить команду git remote. Она перечисляет список имёнсокращений для всех уже указанных удалённых дескрипторов. Если вы склонировали ваш репозиторий, у вас должен отобразиться, по крайней мере, origin — это имя по умолчанию, которое Git присваивает серверу, с которого вы склонировали:

```
$ git clone git://github.com/schacon/ticgit.git
Initialized empty Git repository in /private/tmp/ticgit/.git/ remote:
Counting objects: 595, done.
remote: Compressing objects: 100% (269/269), done. remote: Total 595
(delta 255), reused 589 (delta 253)
Receiving objects: 100% (595/595), 73.31 KiB | 1 KiB/s, done.
Resolving deltas: 100% (255/255), done.
$ cd ticgit $ git remote origin
```

Чтобы посмотреть, какому URL соответствует сокращённое имя в Git, можно указать команде опцию v:

```
$ git remote v
origin git://github.com/schacon/ticgit.git (fetch) origin
git://github.com/schacon/ticgit.git (push)
```

Если у вас больше одного удалённого репозитория, команда покажет их все. Например, мой репозиторий Grit выглядит следующим образом.

```
$ cd grit
$ git remote v
bakkdoor git://github.com/bakkdoor/grit.git cho45
git://github.com/cho45/grit.git defunkt
git://github.com/defunkt/grit.git koke git://github.com/koke/grit.git
origin git@github.com:mojombo/grit.git
```

Это означает, что мы легко можем получить изменения от любого из этих пользователей.

Добавление удалённых репозиториев

В предыдущих разделах мы упомянули и немного продемонстрировали добавление удалённых репозиториев, сейчас мы рассмотрим это более детально. Чтобы добавить новый удалённый Gitрепозиторий под именемсокращением, к которому будет проще обращаться, выполните git remote add [сокращение] [url]:

```
$ git remote origin
$ git remote add pb git://github.com/paulboone/ticgit.git $ git
remote v
origin git://github.com/schacon/ticgit.git pb
git://github.com/paulboone/ticgit.git
```

Теперь вы можете использовать в командной строке имя pb вместо полного URL. Например, если вы хотите извлечь (fetch) всю информацию, которая есть в репозитории Павла, но нет в вашем, вы можете выполнить git fetch pb:

```
$ git fetch pb
remote: Counting objects: 58, done.
remote: Compressing objects: 100% (41/41), done. remote: Total 44
(delta 24), reused 1 (delta 0) Unpacking objects: 100% (44/44), done.
From git://github.com/paulboone/ticgit
 * [new branch]      master     > pb/master
 * [new branch]      ticgit     > pb/ticgit
```

Ветка master Павла теперь доступна локально как pb/master. Вы можете слить (merge) её в одну из своих веток или перейти на эту ветку, если хотите её проверить.

Fetch и Pull

Как вы только что узнали, для получения данных из удалённых проектов, следует выполнить: `$ git fetch [имя удал. сервера]`

Данная команда связывается с указанным удалённым проектом и забирает все те данные проекта, которых у вас ещё нет. После того как вы выполнили команду, у вас должны появиться ссылки на все ветки из этого удалённого проекта. Теперь эти ветки в любой момент могут быть просмотрены или слиты.

Когда вы клонируете репозиторий, команда clone автоматически добавляет этот удалённый репозиторий под именем origin. Таким образом, git fetch origin извлекает все наработки, отправленные (push) на этот сервер после того, как вы склонировали его (или получили изменения с помощью fetch). Важно отметить, что команда fetch забирает данные в ваш локальный репозиторий, но не сливает их с какимилибо вашими наработками и не модифицирует то, над чем вы работаете в данный момент. Вам необходимо вручную слить эти данные с вашими, когда вы будете готовы.

Если у вас есть ветка, настроенная на отслеживание удалённой ветки, то вы можете использовать команду git pull. Она автоматически извлекает и затем сливает данные из удалённой ветки в вашу текущую ветку. Этот способ может для вас оказаться более простым или более удобным. К тому же по умолчанию команда git clone автоматически настраивает вашу локальную ветку master на отслеживание удалённой ветки master на сервере, с которого вы клонировали (подразумевается, что на удалённом сервере есть ветка master). Выполнение git pull, как правило, извлекает (fetch) данные с сервера, с которого вы изначально склонировали, и автоматически пытается слить (merge) их с кодом, над которым вы в данный момент работаете.

Push

Когда вы хотите поделиться своими наработками, вам необходимо отправить (push) их в главный репозиторий. Команда для этого действия простая: git push [удал. сервер] [ветка]. Чтобы отправить вашу ветку master на сервер origin (повторимся, что клонирование, как правило, настраивает оба этих имени автоматически), вы можете выполнить следующую команду для отправки наработок на сервер:

```
$ git push origin master
```

Эта команда срабатывает только в случае, если вы клонировали с сервера, на котором у вас есть права на запись, и если никто другой с тех пор не выполнял команду push. Если вы и ктото ещё одновременно клонируете, затем он выполняет команду push, а затем команду push выполняете вы, то ваш push точно будет отклонён. Вам придётся сначала вытянуть (pull) их изменения и объединить с вашими. Только после этого вам будет позволено выполнить push.

Инспекция удалённого репозитория

Если хотите получить побольше информации об одном из удалённых репозиториев, вы можете использовать команду git remote show [удал. сервер]. Если вы выполните эту команду с некоторым именем, например, origin, вы получите чтото подобное:

```
$ git remote show origin * remote origin
URL: git://github.com/schacon/ticgit.git
Remote branch merged with 'git pull' while on branch master master
Tracked remote branches master
  ticgit
```

Она выдаёт URL удалённого репозитория, а также информацию об отслеживаемых ветках. Эта команда любезно сообщает вам, что если вы, находясь на ветке master, выполните git pull, ветка master с удалённого сервера будет автоматически влита в вашу сразу после получения всех необходимых данных. Она также выдаёт список всех полученных ею ссылок.

Это был пример для простой ситуации, и наверняка вы встретились с чемто подобным. Однако, если вы используете Git более интенсивно, вы можете увидеть гораздо большее количество информации от git remote show:

```
$ git remote show origin
*remote origin
URL: git@github.com:defunkt/github.git
Remote branch merged with 'git pull' while on branch issues issues
Remote branch merged with 'git pull' while on branch master master
New remote branches (next fetch will store in remotes/origin) caching
Stale tracking branches (use 'git remote prune') libwalker
  walker2
 Tracked remote branches acl
  apiv2
  dashboard2 issues master postgres
Local branch pushed with 'git push' master:master
```

Данная команда показывает какая именно локальная ветка будет отправлена на удалённый сервер по умолчанию при выполнении git push. Она также показывает, каких веток с удалённого сервера у вас ещё нет, какие ветки всё ещё есть у вас, но уже удалены на сервере. И для нескольких веток показано, какие удалённые ветки будут в них влиты при выполнении git pull.

Удаление и переименование удалённых репозиториев

Для переименования ссылок в новых версиях Git'a можно выполнить git remote rename, это изменит сокращённое имя, используемое для удалённого репозитория. Например, если вы хотите переименовать pb в paul, вы можете сделать это следующим образом:

```
$ git remote rename pb paul $ git remote
origin
paul
```

Стоит упомянуть, что это также меняет для вас имена удалённых веток. То, к чему вы обращались как pb/master, стало paul/master.

Если по какойто причине вы хотите удалить ссылку (вы сменили сервер или больше не используете определённое зеркало, или, возможно, контрибьютор перестал быть активным), вы можете использовать git remote rm:

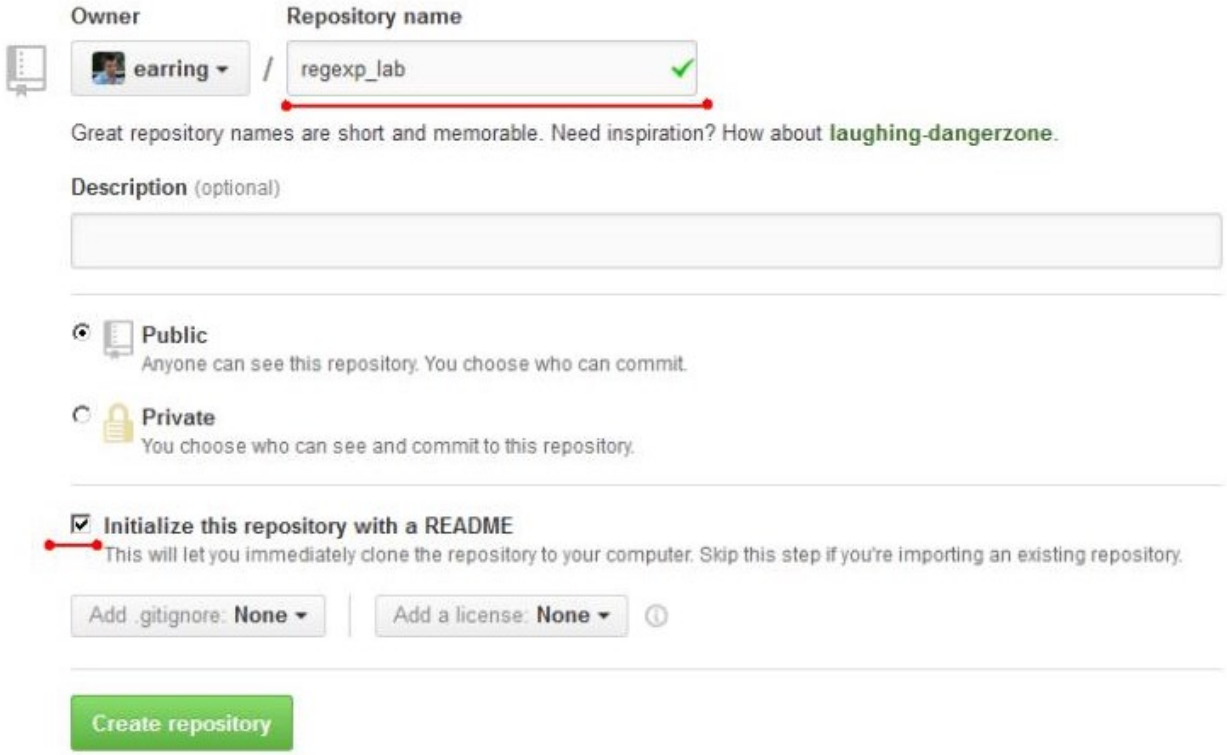
```
$ git remote rm paul $ git remote
origin
```

Работа над проектом с использованием Git, Github и IntelliJ IDEA

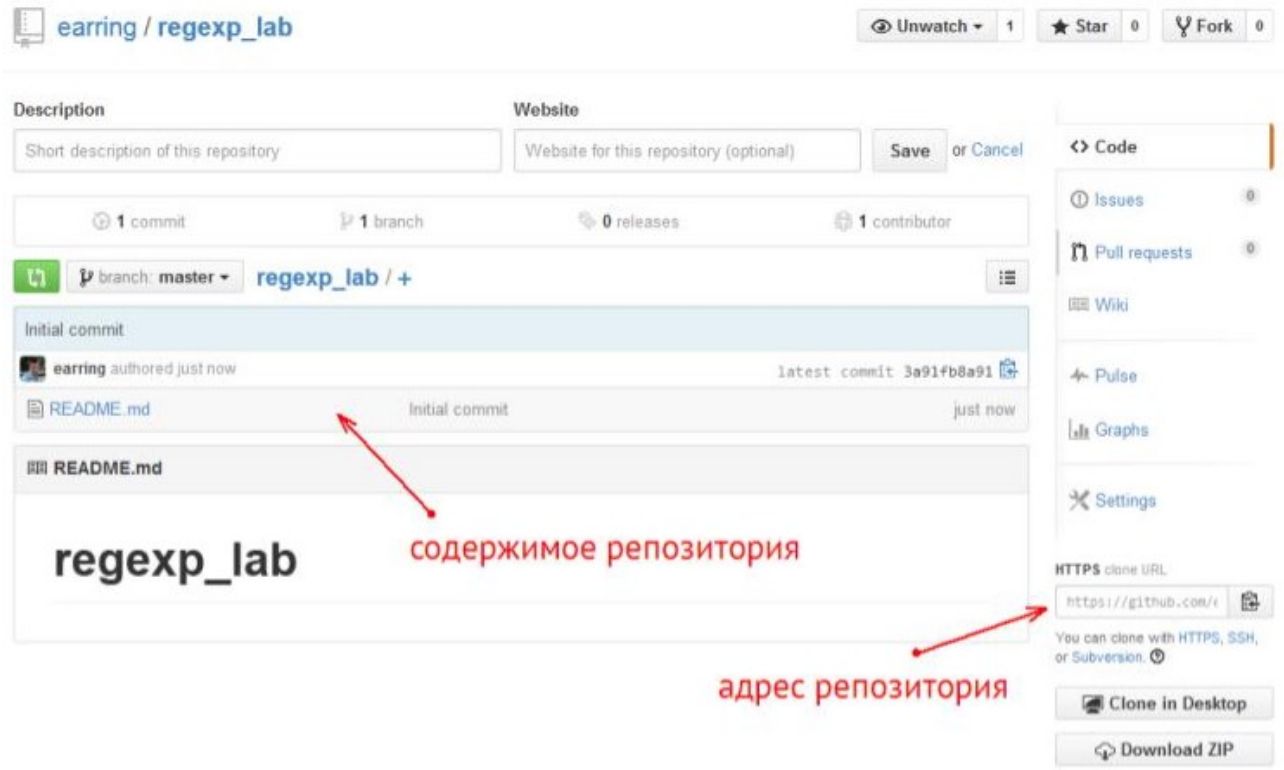
Создание репозитория на GitHub. Для возможности работать с удаленными репозиториями на GitHub необходима регистрация. После регистрации необходимо создать новый репозиторий.

Subscribe to your news feed

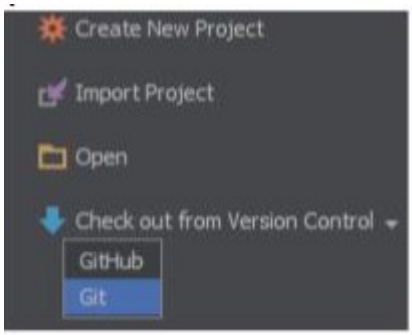
Отметьте необходимые параметры для начала работы: адрес репозитория, приватность и начальную инициализацию



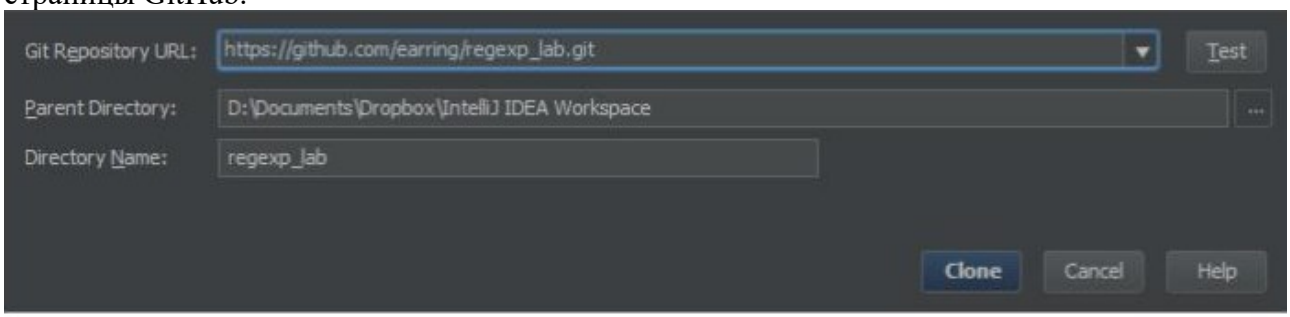
В итоге вам откроется страница с содержимым репозитория.



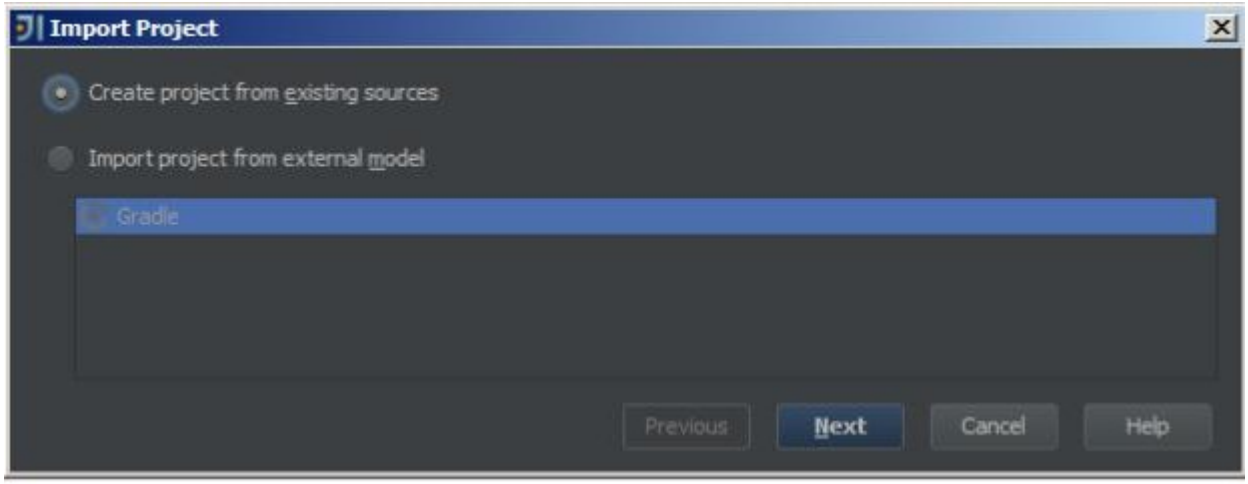
После запуска IntelliJ IDEA там необходимо закрыть открытый проект (последний запускавшийся проект), и в начальном меню выбрать пункт “Check out from Version Control” -> Git.



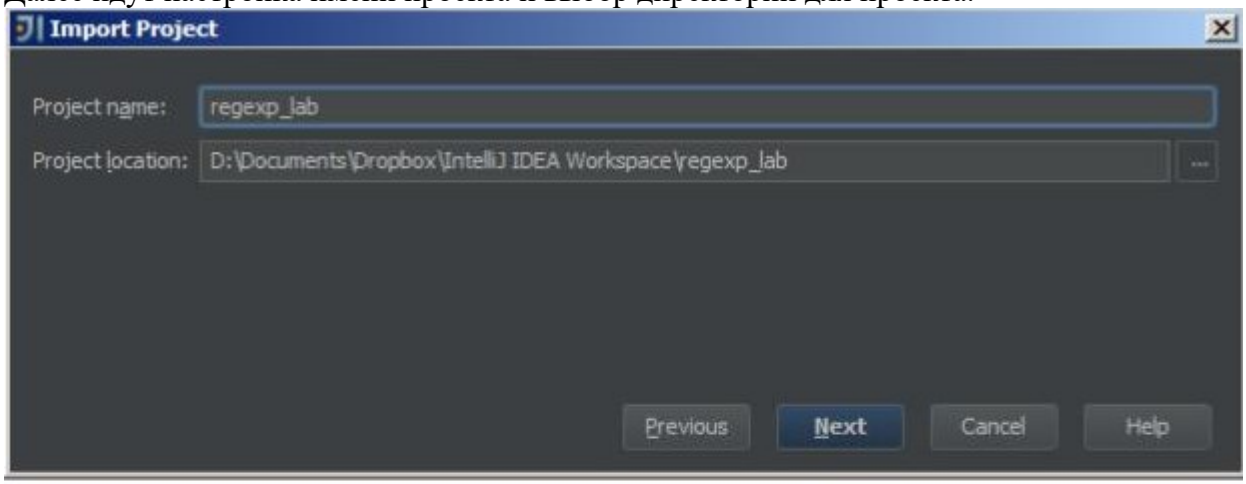
После этого необходимо настроить параметры доступа к только что созданному репозиторию. В строку Git Repository URL необходимо скопировать адрес репозитория со страницы GitHub.

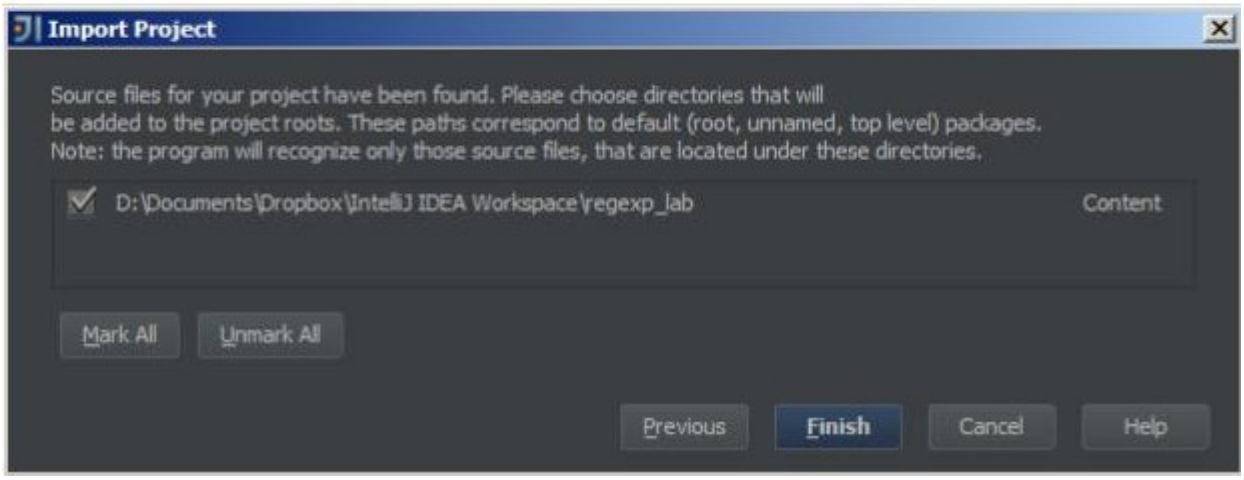


“Clone”. Далее надо согласиться с созданием проекта в IntelliJ IDEA, а также задать источник создания проекта, в данном случае - это “existing sources”.



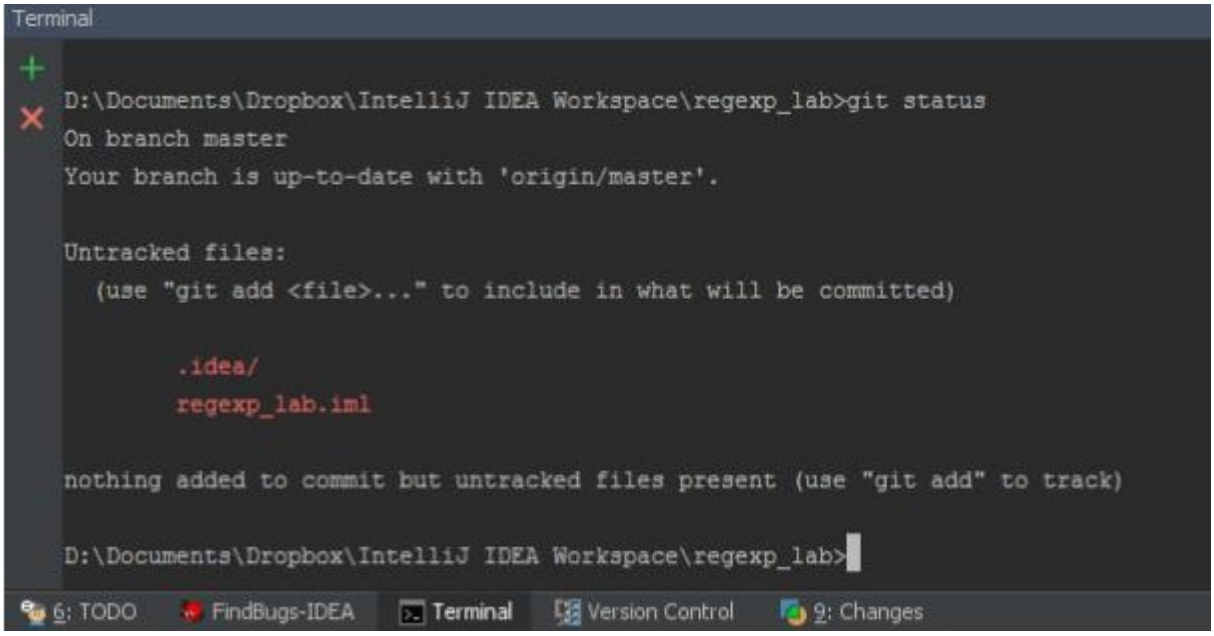
Далее идут настройка имени проекта и выбор директорий для проекта.



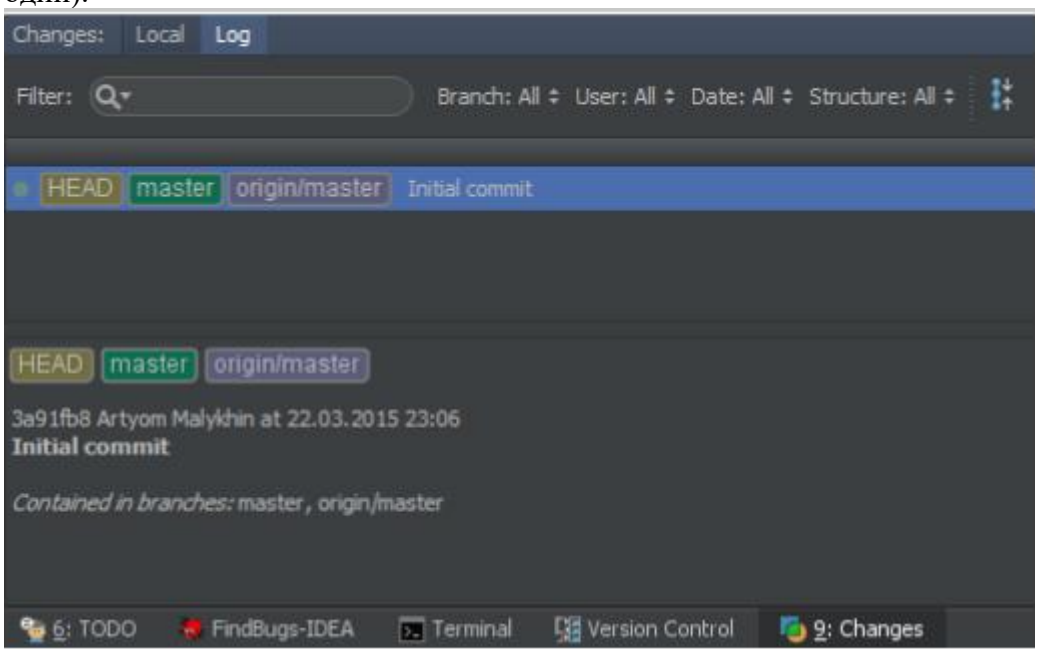


После этого проект (пока пустой) будет открыт в среде разработки.

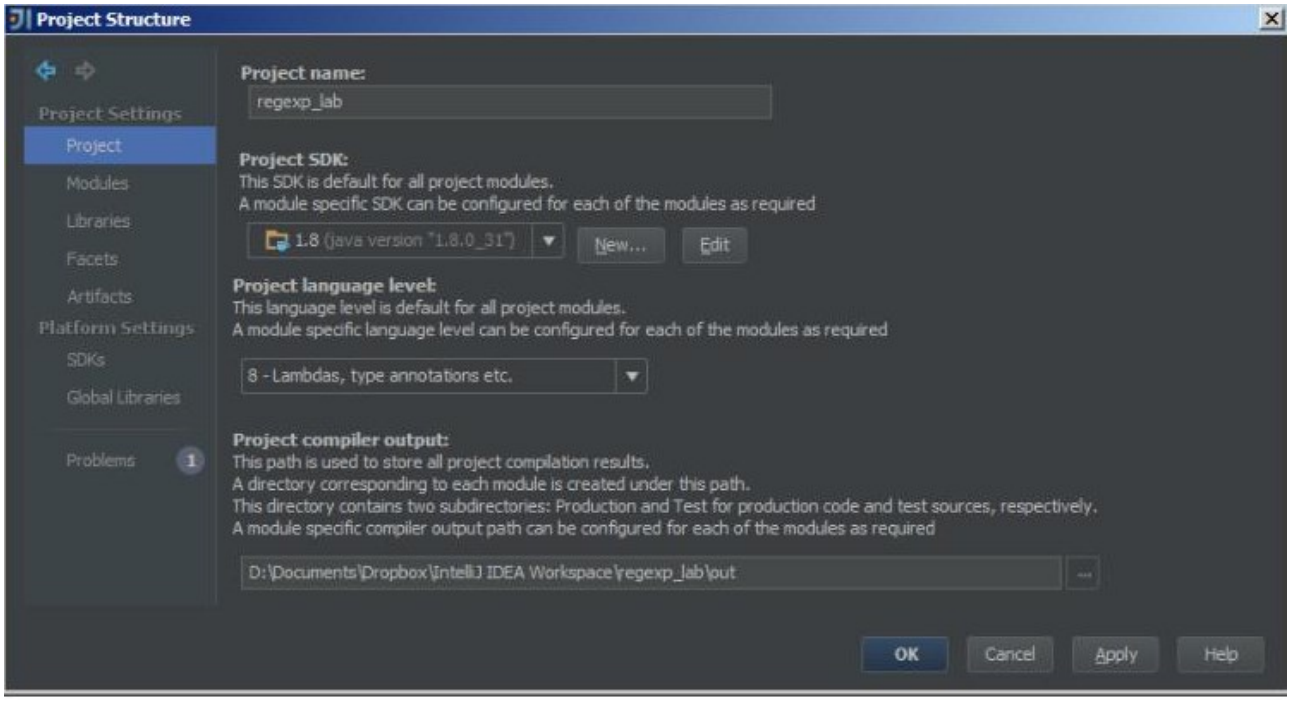
Во встроенном терминале (в составе нижних вкладок) можно выполнять непосредственно консольные программы. Если команда git не выполняется, то проверьте наличие в системной переменной Path пути к папке, где лежат исполняемые файлы Git. При установке по умолчанию это C:\Program Files (x86)\Git\bin.



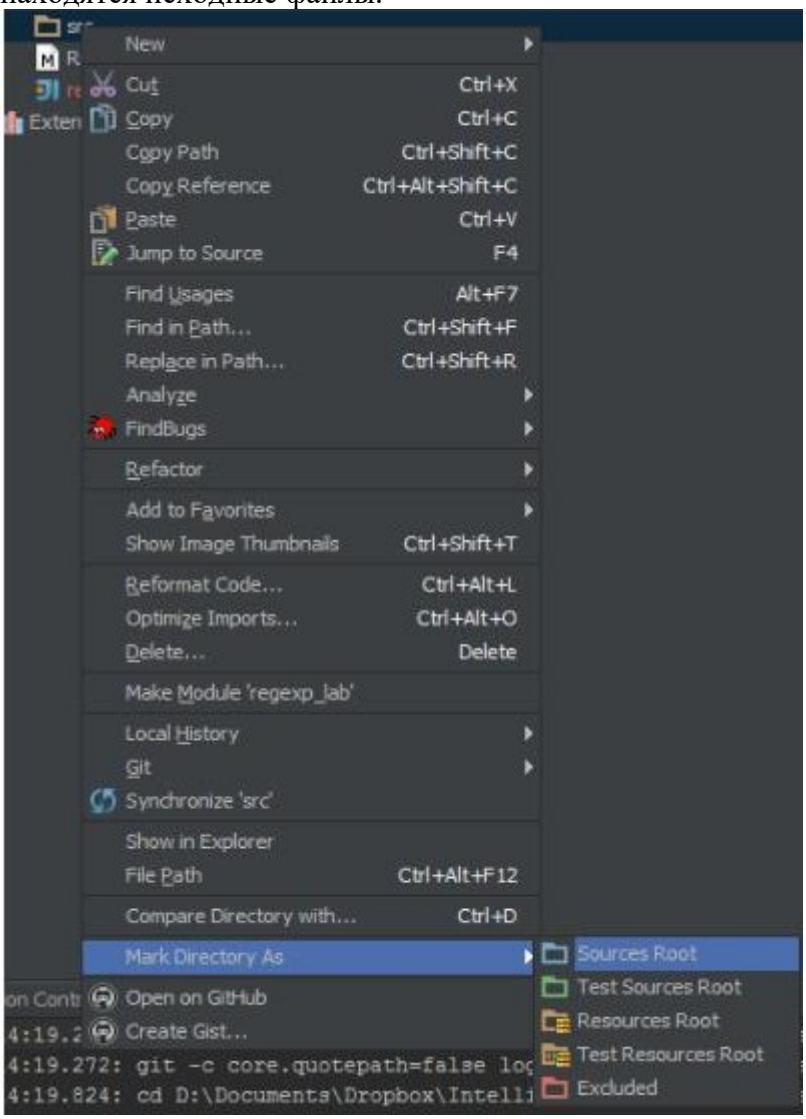
Во вкладке “Changes” можно просмотреть историю изменений в локальном репозитории, например, полученные коммиты. На рисунке изображен начальный коммит (пока он только один).



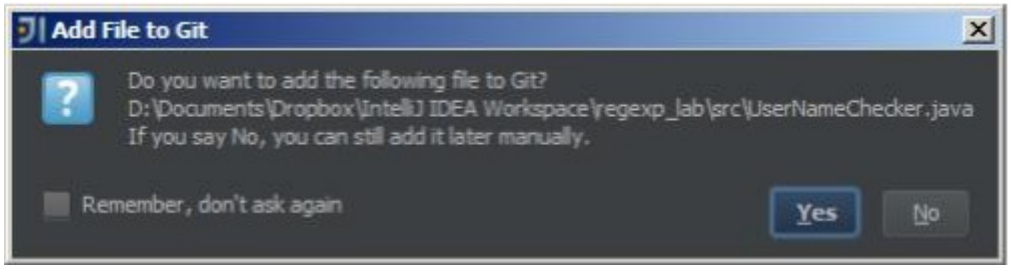
Для начала работы над проектом нужно настроить работу проекта с Java. В контекстном меню проекта - “Open Module Settings”. Настройки должны быть как на рисунке ниже.



Для хранения исходных кодов создайте папку src. Далее необходимо разметить папки, какую роль каждая из них выполняет. Например, для папки src нужно отметить, что в ней находятся исходные файлы.



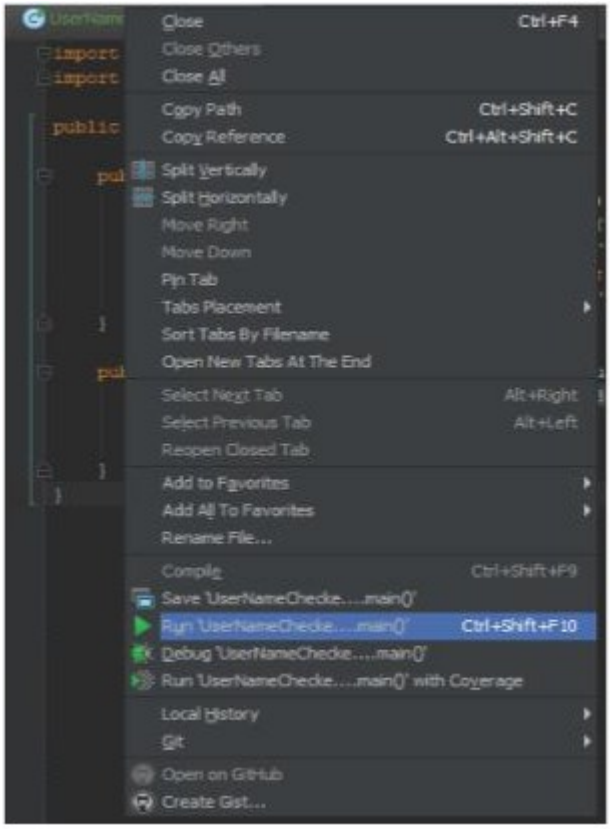
После этого станет возможно создать исходные файлы Java из контекстного меню. После создания Java файла, среда разработки предложит добавить этот файл в Git.



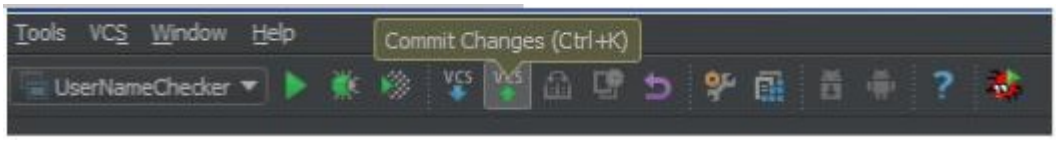
В качестве примера приведен исходный файл простой программы, демонстрирующий проверку логина на соответствие условиям “длина от 3 до 15 символов, может включать в себя латинские символы в нижнем регистре, цифры, символ подчеркивания и тире”.

```
import java.util.regex.Matcher; import java.util.regex.Pattern;
public class UserNameChecker {

public static void main(String[] args){ System.out.println("Results of
    checking:"); System.out.println(checkWithRegExp("_@BEST"));
    System.out.println(checkWithRegExp("vovan"));
    System.out.println(checkWithRegExp("vo"));
    System.out.println(checkWithRegExp("Z@OZA"));
    }

public static boolean checkWithRegExp(String userNameString){ Pattern p =
    Pattern.compile("^[az09_]{3,15}$"); Matcher m =
    p.matcher(userNameString);
    return m.matches();
    }
    }
```

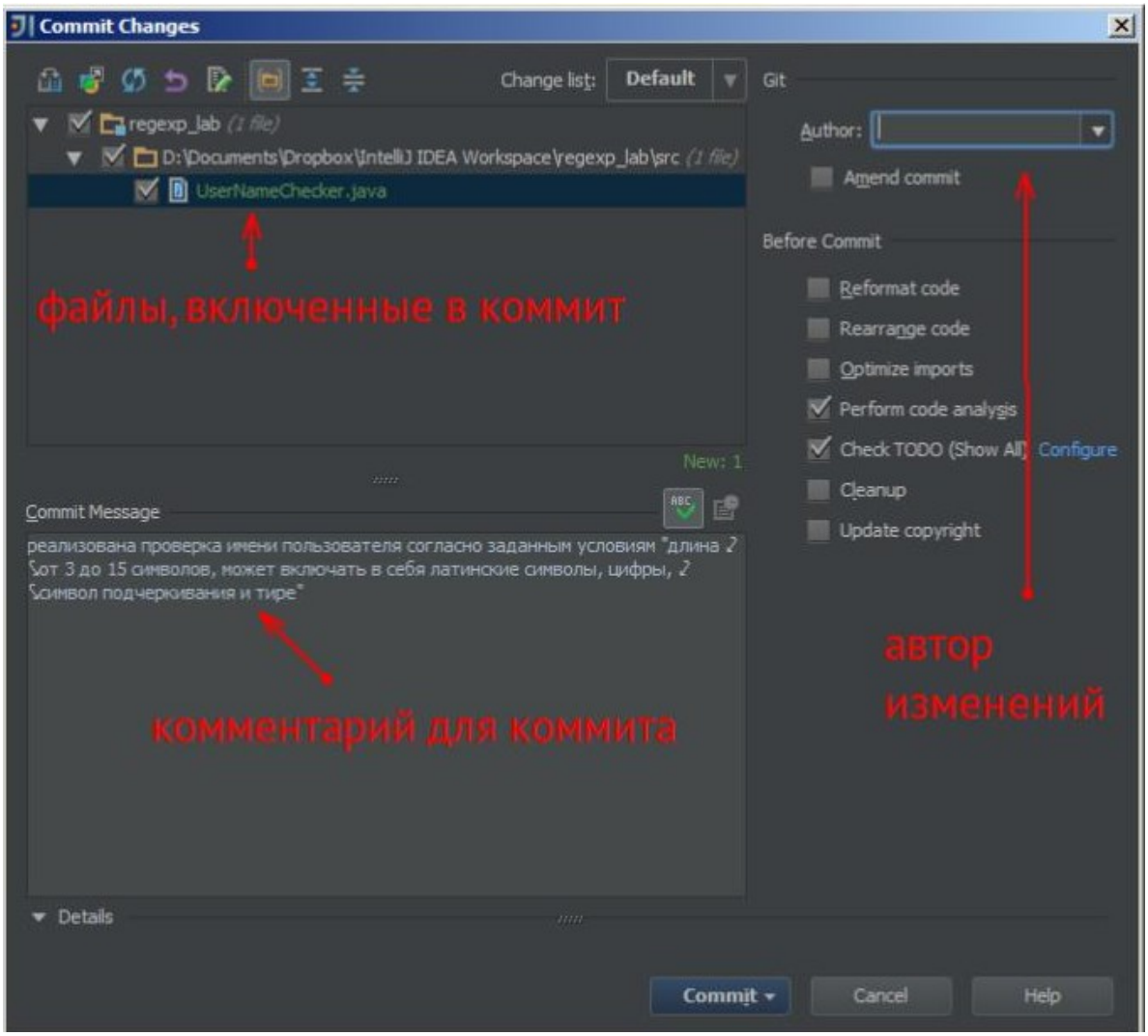
Запустить на исполнение программу можно посредством контекстного меню на вкладке Java файла, в котором находится mainметод.



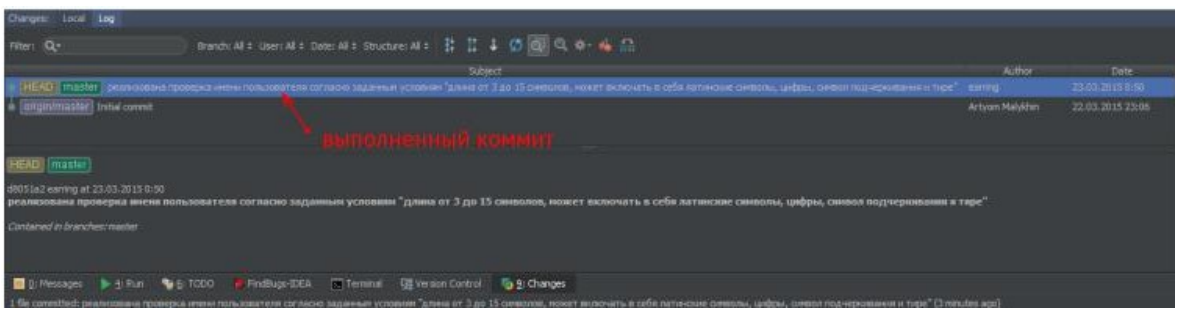
После успешной проверки первой реализации нужно отправить коммит, т.е зафиксировать свою реализацию. Выберите пункт меню “Commit changes”



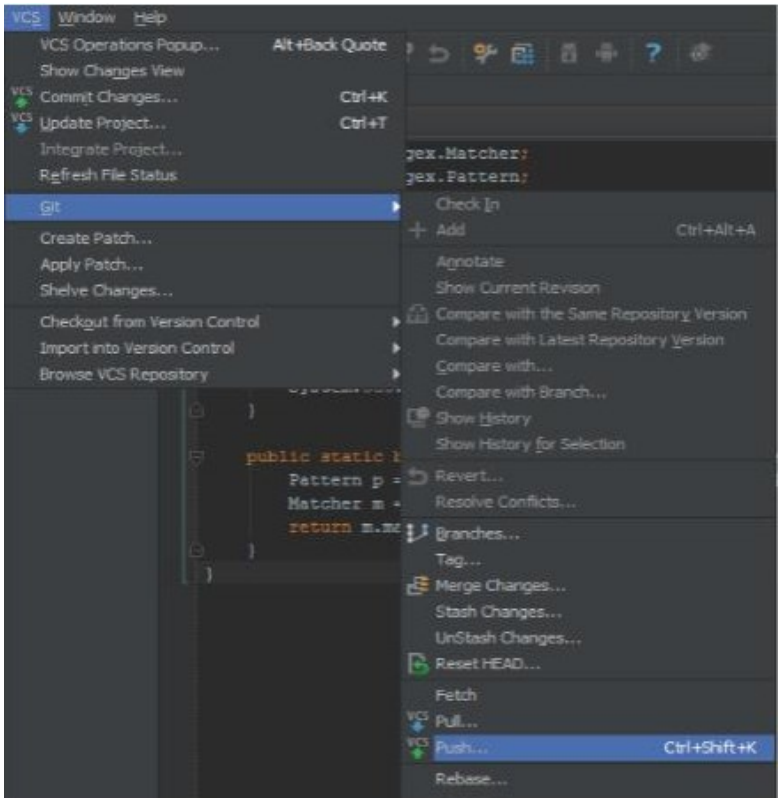
После этого будет показано окно, где можно “оформить” коммит.

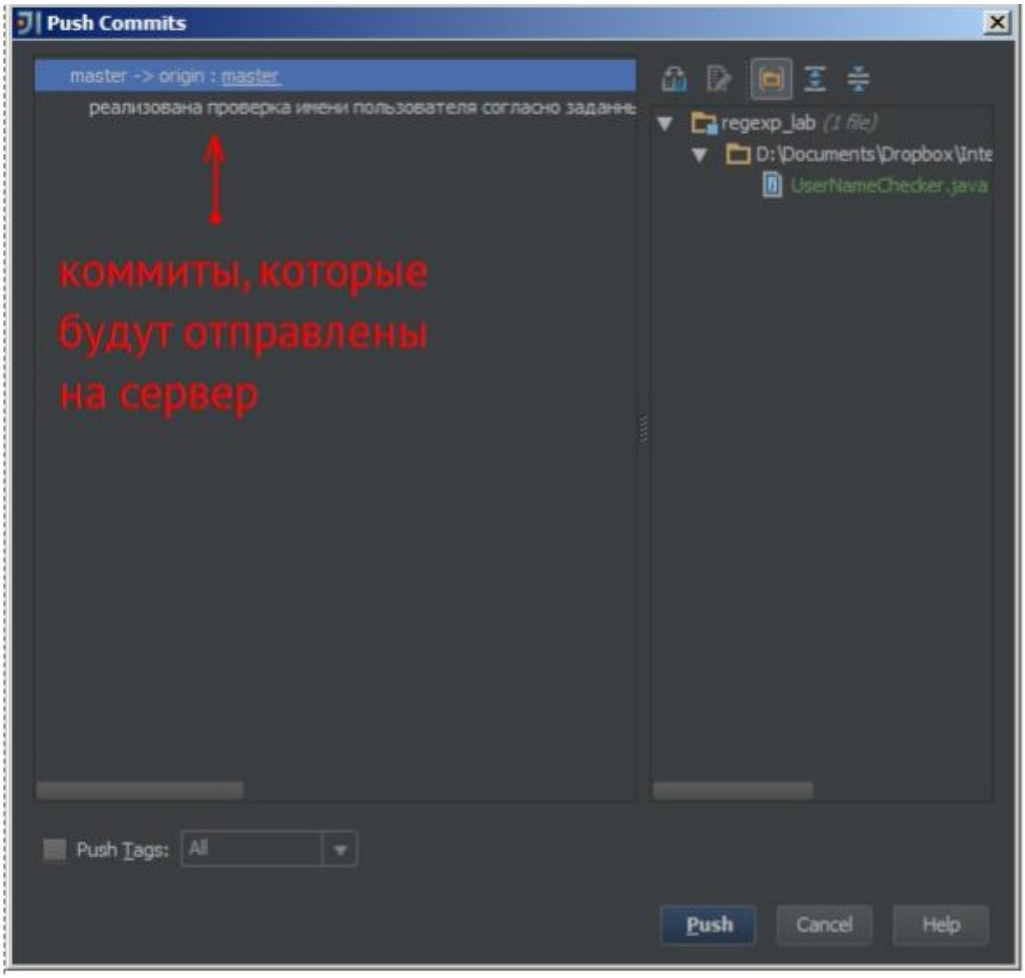


После нажатия “Commit” изменения будут отмечены в локальном репозитории, что можно увидеть во подвкладке Log вкладки Changes.

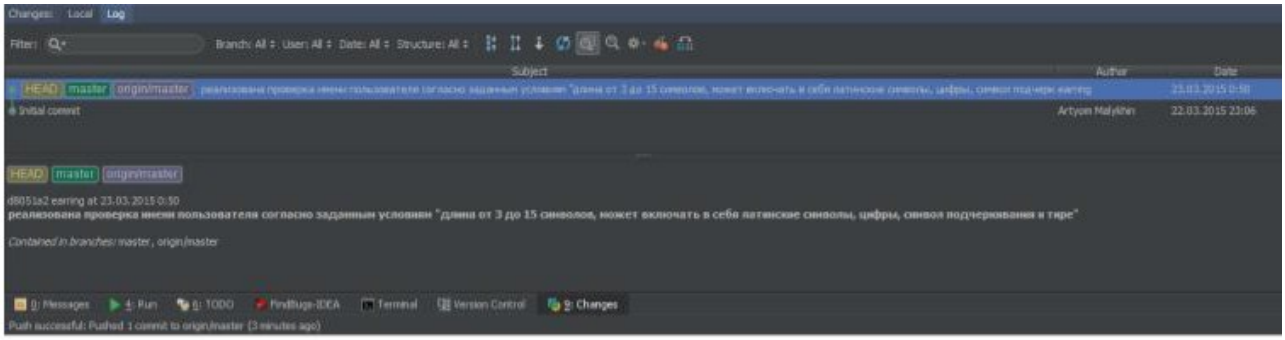


Теперь эти изменения можно отправить на GitHub. Это делается следующим образом.



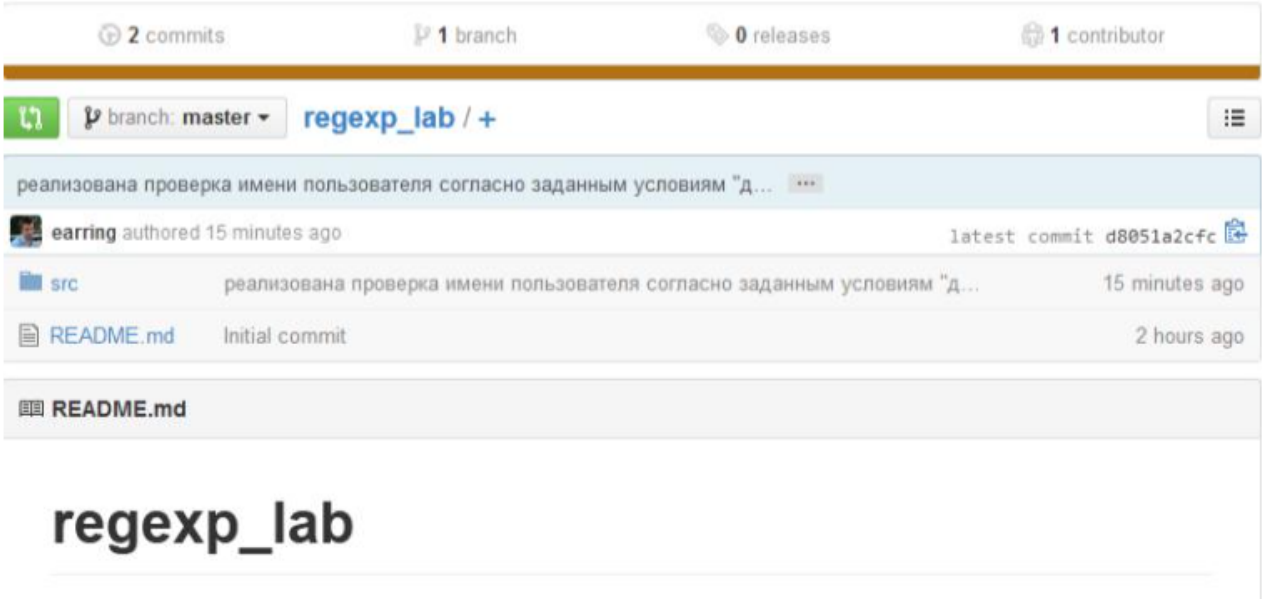


После выбора “Push” коммиты будут отправлены на сервер. Обзор изменений теперь выглядит следующим образом.



Можно увидеть, что метка “origin/master” теперь находится на последнем коммите, а не на самом первом, это значит, что теперь в удаленном репозитории (они по умолчанию называются origin) последним коммитом является тот же коммит, что является последним в локальном репозитории. Иными словами, набор коммитов на сервере и локальном компьютере стал одинаковым.

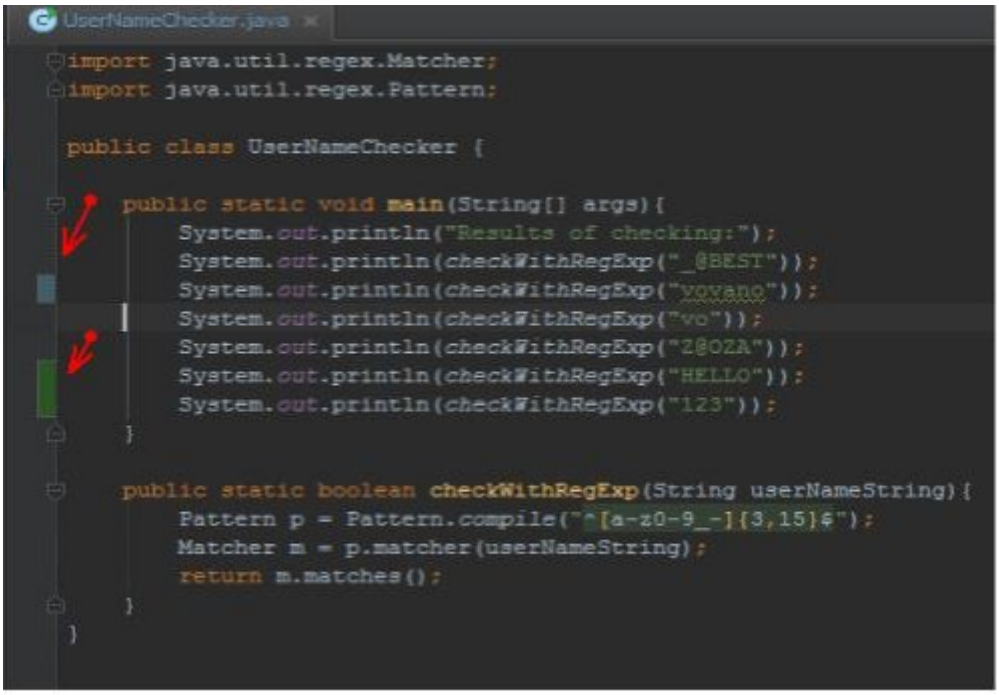
Можно также увидеть, что в репозитории в GitHub файлы также изменились.



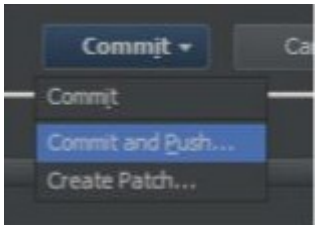
Внесем некоторые изменения в код программы. `import java.util.regex.Matcher;`

```
import java.util.regex.Pattern;

public class UserNameChecker {

public static void main(String[] args){ System.out.println("Results of
checking:"); System.out.println(checkWithRegExp("_@BEST"));
System.out.println(checkWithRegExp("vovano"));
System.out.println(checkWithRegExp("vo"));
System.out.println(checkWithRegExp("Z@OZA"));
System.out.println(checkWithRegExp("HELLO"));
System.out.println(checkWithRegExp("123"));
}
Public static boolean checkWithRegExp(String userNameString){ Pattern
p = Pattern.compile("^[az09_]{3,15}$"); Matcher m =
p.matcher(userNameString);
return m.matches();
}
}
```

Обратите внимание на цветные полосы возле строк кода. Синий цвет означает изменившуюся строку, зеленый добавившуюся, а на месте удаленных строк появится треугольник.



Для удобства процесс отправки коммита в локальный репозиторий и последующий процесс отправки всех коммитов в удаленный репозиторий можно выполнить одной командой. Необходимо перейти в меню “Commit Changes”, только вместо команды “Commit” выполнить команду “Commit and Push”.



Далее нужно провести процедуру push как сказано выше.

В процессе изменений программы процессы commit и push повторяются. Однако это самый простой вариант работы, когда над программой работает только один человек. При работе нескольких человек приходится обеспечивать их взаимодействие, в том числе, используя ветвление. Это будет рассмотрено подробно на следующей лабораторной работе.

Пример, использовавшийся в практической части, доступен по адресу https://github.com/earring/regexp_lab.

Задания для самостоятельной работы

Программа выполняется **индивидуально**. Необходимо реализовать программу своего варианта, используя Git и размещая её репозиторий на GitHub. Программа должна реализовываться поэтапно, т. е. каждое значительное изменение кода оформляется в виде коммита с ясным комментарием.
Рекомендуется продемонстрировать использование регулярных выражений.

Вариант №1, 7, 13, 19, 25

В программу передается текст из файла. В этом тексте должно быть подсчитано количество прилагательных, наречий и глаголов.

Вариант №2, 8, 14, 20, 26

В программу подается текст из файла, состоящего из строчек формата
Имя Фамилия|Возраст|ТелефонныйНомер|ЭлектроннаяПочта

Необходимо проверить данные на корректность, и по возможности, исправленную версию поместить в другой файл. Если данные ошибочные, то часть строки оставить пустой. К примеру, из строки
ИванИванов|27|+7999000 1 1 11|example@@yandex..ru
может после исправления получиться строка
Иван Иванов|27|+7 (999) 0001111|example@yandex.ru

Вариант №3, 9, 15, 21, 27

В программу подается текст из файла, в котором находится секретное письмо. Его надо очистить от конфиденциальных данных. Конфиденциальными данными считаются имена, фамилии, номера телефонов и данные, помогающие определить географическое положение адресата. Вместо данных, которые программа находит конфиденциальными, следует написать [censored]. Рекомендуется проверять работоспособность программы на художественных текстах.

Вариант №4, 10, 16, 22, 28

В программу подается текст из файла, в котором находится некий текст, либо набор текстов. Необходимо определить “настроение” текста. Это значит, что если в тексте будет употреблено много негативных слов, то текст будет иметь отрицательное значение “настроения”. Должно быть вычислено точно число “настроения”, для возможности сравнения различных текстов.

Вариант №5, 11, 17, 23, 29

В программу подается текст из файла, в котором находится текст, в котором нужно заменить числа, написанные прописью, на числа, написанные цифрами. Например, текст “сто одиннадцать тысяч фиолетовых оленей” будет записан как “111 000 фиолетовых оленей”. Считать, что числа прописью написаны без ошибок в грамматике, падеже и т. д.

Вариант №6, 12, 18, 24, 30

В программу передается текст из файла, в котором находится текст, в котором нужно во всех найденных телефонных номерах изменить код страны, а также привести номера к единому формату. К примеру, текст “Звоните по номеру +79000000000” должен быть заменен на текст “Звоните по номеру +1 (900) 0000000”. Возможные ошибки в написании телефонного номера учитывать.

МДК 03.02. Управление проектами

Практическая работа №1.

«Использование метрик программного продукта»

Цель: Научиться использовать Microsoft Visual Studio для разработки программ на языке C++. Получить практические навыки работы со средой визуальной разработки программ.

Задание 1. Создание приложения

Запускаем Microsoft Visual C++ .

После запуска системы мы увидим начальный пользовательский интерфейс, показанный на рис. 1.1.

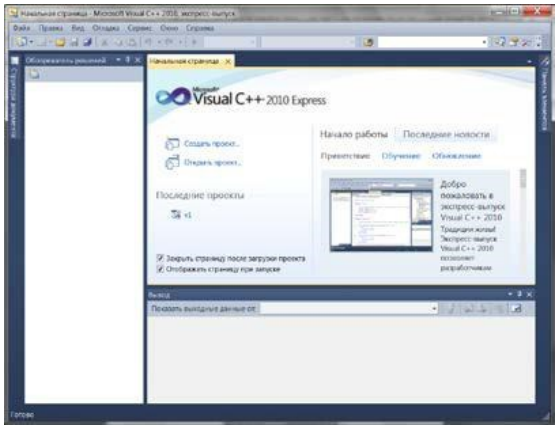


Рис. 1.1. Фрагмент стартовой страницы системы Visual Studio

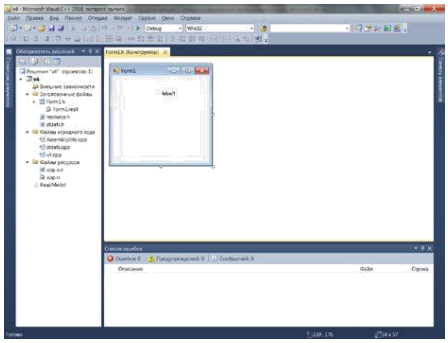
Для создания приложения, необходимо в пункте меню File выполнить команду New Project (Новый проект). В появившемся окне New Project в левой колонке находится список установленных шаблонов (Installed Templates). Среди них — шаблоны языков программирования, встроенных в Visual Studio, в том числе Visual Basic, Visual C#, Visual C++, Visual F# и др. Нам нужен язык Visual C++. В узле Visual C++ области типов проектов выберем среду CLR, а затем в области шаблонов (в средней колонке) выберем шаблон (Templates) Windows Forms Application Visual C++.

Теперь введем имя проекта (Name) v4 и щелкнем на кнопке OK, в результате увидим окно, представленное на рис. 1.2.

Рис. 1.2. Окно для проектирования пользовательского интерфейса

В этом окне изображена экранная форма — Form1. Первая программа будет отображать такую экранную форму, в которой будет что-либо написано, например «Microsoft Visual C++ 2010», также в форме будет расположена командная кнопка с надписью «Нажми меня». При нажатии кнопки будет появляться диалоговое окно с сообщением «Всем привет!» В программе четыре объекта: форму Form, надпись на форме Label, кнопка Button и диалоговое окно MessageBox с текстом «Всем привет!» (окно с приветом).

Добавить в форму названные элементы управления. Для этого понадобится панель элементов управления Toolbox (Панель управления), ее можно добавить, например, с помощью комбинации клавиш Ctrl+Alt+x или ViewToolbox. Итак, добавьте метку Label и кнопку Button в форму, дважды щелкая на этих элементах на панели Toolbox. А затем следует рас положить их примерно так, как показано на рис. 1.3.



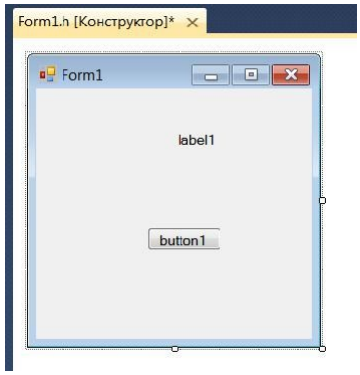


Рис. 1.3. Форма первого проекта

Каждый объект имеет свойства (properties- Свойства) . Свойств много, их можно увидеть, если щелкнуть правой кнопкой мыши в пределах формы и выбрать в контекстном меню команду Properties-Свойства, при этом появится панель свойств.

Для объекта label1 выбрать свойство Text и написать напротив этого поля «Microsoft Visual C++ 2010» (вместо текста label1). Для объекта button1 также в свойстве Text написать «Нажми меня».

Объекты не только имеют свойства, но и обрабатываются событиями. В задаче событием, которым управляем, является щелчок на командной кнопке. Для получения пустого обработчика этого события следует в свойствах кнопки button1 щелкнуть на значке молнии Events (события) и в списке всех возможных событий кнопки button1 выбрать двойным щелчком событие Click. После этого попадаем на вкладку программного кода Form1.h (см. рис. 1.4).

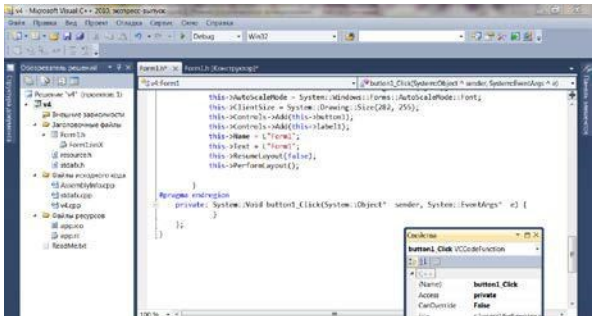


Рис 1.4 Вкладка программного кода

На вкладке Form1.h видно, что управляющая среда Visual C++ сгенерировала довольно таки много строк программного кода. В этом тексте уже можно найти те присваивания, которые сделали в панели свойств Properties. Например, для свойства Text кнопки Button управляющая среда назначила строку ≪Нажми меня≫: **this->button1->Text = L**"Нажми меня"**;**

Пустой обработчик события button1_Click:

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) { }

Здесь в фигурных скобках пишутся команды, подлежащие выполнению после щелчка на кнопке. В фигурных скобках обработчика события напишите:

MessageBox::Show("Всем привет!");

Теперь нажмите клавишу F5 и проверьте работоспособность программы (рис. 1.5).

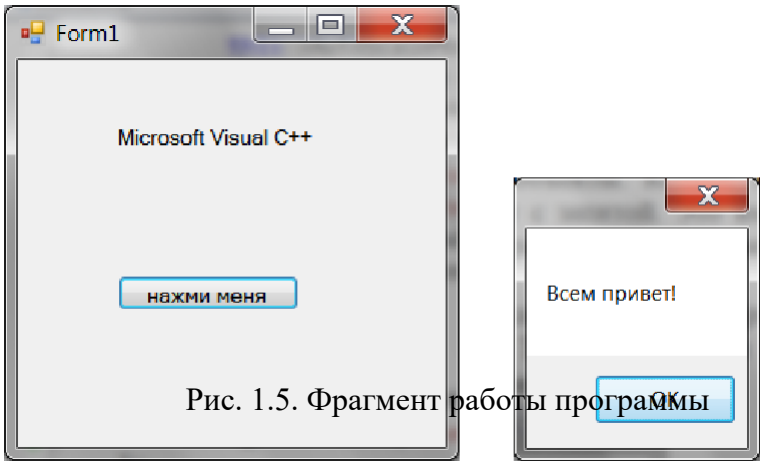


Рис. 1.5. Фрагмент работы программы

Задание 2. Обработка события MouseHover мыши

Событие MouseHover наступает тогда, когда пользователь указателем мыши «зависает» над каким-либо объектом, событие MouseHover происходит, когда указатель мыши наведен на элемент.

Таким образом, программа в данном примере должна содержать на экранной форме текстовую метку Label и кнопку Button. Метка должна отображать текст «Microsoft Visual C++ 2010»; при щелчке на командной кнопке, на которой попрежнему будет написано «Нажми меня», появится диалоговое окно с сообщением «Всем привет!» Кроме того, когда указатель мыши наведен на текстовую метку (то самое событие MouseHover), должно появиться диалоговое окно с текстом «Событие Hover».

Для решения этой задачи запустим Visual Studio 2010, щелкнем на пункте меню New Project. В появившемся окне New Project в левой колонке в узле Visual C++ выберем

среду CLR, а затем в области шаблоны (в средней колонке) выберем шаблон (Templates) Windows Forms Application Visual C++. В качестве имени проекта введем имя Hover и щелкнем на кнопке OK.

В дизайнере формы из панели Toolbox перетащим на форму метку Label и кнопку Button, а затем немного уменьшим размеры формы на свое усмотрение. Теперь добавим три обработчика событий в программный код. Для этого в панели Properties следует щелкнуть на значке молнии (Events) и двойным щелчком последовательно выбрать событие загрузки формы Form_Load, событие «щелчок на кнопке button1_Click» и событие label1_MouseHover.

При этом осуществится переход на вкладку программного кода Form1.h, и среда Visual Studio 2010 сгенерирует три пустых обработчика события (рис.1.6). Например, обработчик последнего события будет иметь вид:

```
private:System::Void lab el1_MouseHover(System::Object^ sender, System::EventArgs^
e){}
```

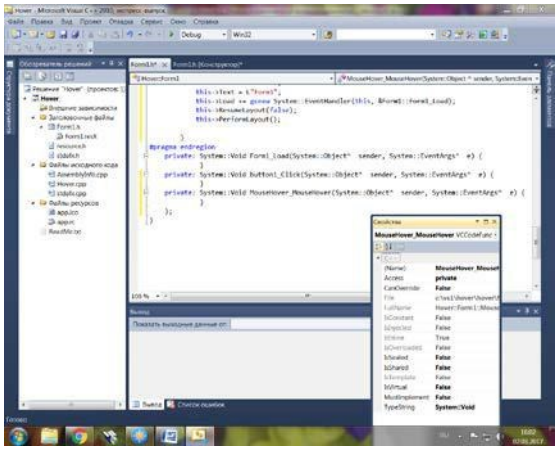


Рис. 1.6 Вкладка программного кода

Между фигурными скобками вставим вызов диалогового окна:

```
MessageBox::Show("Событие Hover!");
```

Теперь проверим возможности программы: нажимаем клавишу F5, «зависаем» указателем мыши над label1, щелкаем на кнопке button1. Все работает! (рис.1.7)

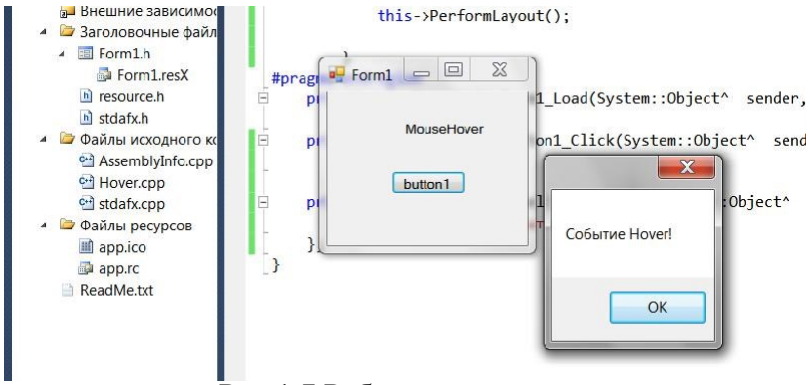


Рис.1.7 Работа приложения

Листинг. Фрагмент файла Form1.h, содержащего программный код с тремя обработчиками событий

```
//     .....
//     Программный код, расположенный выше, создан средой Visual
Studio
//     автоматически, поэтому автором не приводится
this->ResumeLayout(false);
```

```
this->PerformLayout();
}
#pragma endregion
//        Данная программа управляется тремя событиями. Событие загрузки формы
//      Form1_Load инициализирует надписи заголовка формы, текстовой метки
//       и кнопки. Событие щелчок на кнопке button1_Click вызывает появление
//       диалогового окна с текстом "Всем привет!". Событие, когда указатель
//      мыши наведен на метку, вызывает появление диалогового окна с текстом
//     "Событие Hover".
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^
e)
{ // Обработка события загрузки формы: this->Text =
"Приветствие";
// или Form1::Text = "Приветствие"; label1->Text =
"Microsoft Visual C++ 2010"; button1->Text = "Нажми меня";
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ // Обработка события щелчок на кнопке:
MessageBox::Show("Всем привет!");
}
private: System::Void label1_MouseHover(System::Object^ sender, System::EventArgs^ e)
{ // Обработка события, когда указатель мыши наведен на метку:
MessageBox::Show("Событие Hover!");
}
};
}
```

Контрольные вопросы

1. Из каких двух этапов состоит процесс проектирования программы Visual C++.
2. Что такое программа, основанная на диалоге.
3. Что такое Windows Forms Application.

Итог работы: файлы, защита

Практическая работа №2.
«Проверка целостности программного кода. Офускация кода»

Цель: Научиться разрабатывать и реализовывать простейшие программы на языке VC++. Получить практические навыки работы по использованию интерфейса CLR. Научиться связывать переменные и методы с элементами диалогового окна.

Задание. Ввод данных через текстовое поле TextBox с проверкой типа методом TryParse
При работе с формой очень часто ввод данных организуют через элемент управления текстовое поле TextBox . Напишем типичную программу, которая вводит через текстовое поле число, при нажатии командной кнопки извлекает из него квадратный корень и выводит результат на метку Label. В случае ввода не числа сообщает пользователю об этом.

Решая сформулированную задачу, запускаем Visual Studio, выбираем пункт меню File-New -Project. В окне New Project в узле Visual C++ выберем среду CLR, а затем в области шаблоны выберем шаблон (Templates) Windows Forms Application Visual C++. В качестве имени проекта введем имя Корень и щелкнем на кнопке OK.

Далее из панели элементов управления Toolbox (если в данный момент вы не видите панель элементов управления, то ее можно добавить, например, с помощью комбинации клавиш Ctrl+Alt+x или меню View - oolbox) в форму с помощью указателя мыши перетаскиваем текстовое поле TextBox, метку Label и командную кнопку Button. Получить названные элементы на проектируемой экранной форме можно , также дважды

щелкая указателем мыши на каждом элементе в панели Tools. Таким образом, в форме будут находиться три элемента управления. Расположим их на экранной форме.

Теперь следует изменить некоторые свойства элементов управления. Чтобы получить пустой обработчик загрузки формы, дважды щелкнем по проектируемой экранной форме. Сразу после этого мы попадаем на вкладку программного кода Form1.h. Здесь задаем свойствам формы (к форме обращаемся посредством ссылки this), кнопкам button1 и текстового поля textBox1, метке label1 следующие значения:

this->Text = "Извлечение квадратного корня"; button1->Text
= "Извлечь корень"; textBox1->Clear(); // Очистка текстового
поля label1->Text = nullptr; // или = String::Empty;

Нажмем клавишу F5 для выявления возможных опечаток, то есть синтаксических ошибок и предварительного просмотра дизайна будущей программы (рис.2.1).

Далее программируем событие button1 _Click — «щелчок мышью на кнопке Извлечь корень». Создать пустой обработчик этого события удобно, дважды щелкнув мышью на этой кнопке. Между двумя появившимися строчками программируем диагностику правильности вводимых данных, конвертирование строковой переменной в переменную типа Single и непосредственное извлечение корня (листинг).

Листинг. Фрагмент программы извлечения корня с проверкой типа методом TryParse

```
//    .............
//     Программный код, расположенный выше, создан средой Visual
Studio
//    автоматически, поэтому автором не приводится
this->ResumeLayout(false);
this->PerformLayout();
}
#pragma endregion
//     Программа вводит через текстовое поле число, при щелчке на командной
//       кнопке извлекает из него квадратный корень и выводит результат
//       на метку label1. В случае ввода не числа сообщает пользователю об
//     этом, выводя красным цветом предупреждение также на метку label1.
private: System::
Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
button1->Text = "Извлечь корень"; label1-
>Text = String::Empty;
//    или label1->Text = nullptr;
this->Text = "Извлечение квадратного корня"; textBox1->Clear(); // Очистка
текстового поля textBox1->TabIndex = 0; // Установка фокуса в текстовом поле
}
private: System:: // Обработка щелчка на кнопке "Извлечь корень":
Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
Single X; // - из этого числа будем извлекать корень
//    Преобразование из строковой переменной в Single: bool Число_ли =
Single::TryParse(textBox1->Text, System::Globalization::NumberStyles::Number,
System::Globalization::NumberFormatInfo::CurrentInfo, X);
//    Второй параметр - это разрешенный стиль числа (Integer,
//       шестнадцатеричное число, экспоненциальный вид числа и прочее).
//    Третий параметр форматирует значения на основе текущего языка
//    и региональных параметров из Панели управления - Язык и
//    региональные стандарты - число допустимого формата; метод

//    возвращает значение в переменную X if
(Число_ли == false)
```

```
{ // Если пользователь ввел не число: label1->Text =
"Следует вводить числа";
label1->ForeColor = Color::Red; // - цвет текста на метке return; // - выход из
процедуры
}
Single Y = (Single)Math::Sqrt(X); // - извлечение корня label1->ForeColor =
Color::Black; // - черный цвет текста на метке
label1->Text = String::Format("Корень из {0} равен {1:F5}", X, Y);
}
};
}
```

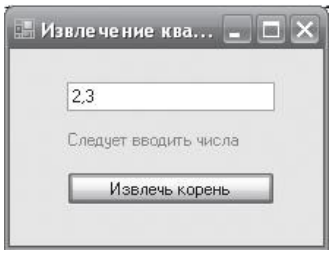


Рис.2.1 Фрагмент работы программы

Если пользователь ввел все-таки число, то будет выполняться следующий оператор извлечения квадратного корня Math::Sqrt(X). Математические функции Visual Studio являются методами класса Math. Их можно увидеть, набрав Math и введя так называемый *оператор разрешения области действия* (::). В раскрывающемся списке вы увидите множество математических функций: Abs, Sin, Cos, Min и т. д. и два свойства — две константы E = 2,718...
(основание натуральных логарифмов) и PI = 3,14... (число диаметров, уложенных вдоль окружности). Функция Math::Sqrt(X)возвращает значение типа double (двойной точности с плавающей запятой), которое мы *приводим* с помощью неявного преобразования (Single) к переменной одинарной точности.
Последней строчкой обработки события button1_Click является формирование строки label1->Text с использованием метода String::Format. Использованный формат ≪Корень из {0} равен {1:F5}≫ означает: взять нулевой выводимый элемент, то есть переменную X, и записать эту переменную вместо фигурных скобок; после чего взять первый выводимый элемент, то есть Y, и записать его вместо вторых фигурных скобок в формате с фиксированной точкой и пятью десятичными знаками после запятой.
Нажав клавишу F5, проверяем, как работает программа.
Результат работающей программы представлен на рис. 2.2.

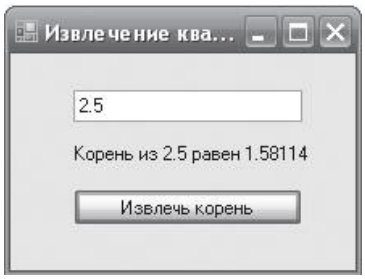


Рис.2.2. Извлечение квадратного корня

Контрольные вопросы

1. Опишите шаги, которые вы сделали, чтобы открыть диалоговую панель программы для ее визуальной настройки.
2. Что такое Class Wizard.
3. Как добавить элемент для ввода данных.

Итог работы: файлы, защита

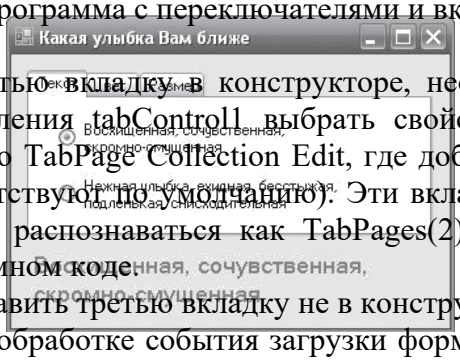
Практическая работа №3.

Цель: Научиться разрабатывать и реализовывать программу, использующую вкладки и переключатели, изменять шрифт вкладок.

Задание. Разработать программу, позволяющую выбрать текст из двух вариантов, задать цвет и размер шрифта этого текста на трех вкладках TabControl с использованием переключателей RadioButton.

Программируя поставленную задачу, запустим Visual Studio и выберем приложение в среде CLR шаблона Windows Forms Application Visual C++. Назовем этот проект Вкладки. Используя панель элементов Toolbox, в форму перетащим с помощью мыши элемент управления TabControl. Как видно, по умолчанию имеем две вкладки, а по условию задачи, как показано на рисунке, три вкладки. Добавить третью вкладку можно в конструкторе формы, а можно программно (рис.3.1).

Рис.3.1 Программа с переключателями и вкладками



Чтобы добавить третью вкладку в конструкторе, необходимо в свойствах (окно Properties) элемента управления tabControl1 выбрать свойство TabPages, в результате попадаем в диалоговое окно TabPage Collection Edit, где добавляем (кнопка Add) третью вкладку (первые две присутствуют по умолчанию). Эти вкладки нумеруются от нуля, то есть третья вкладка будет распознаваться как TabPages(2). Название каждой вкладки будем указывать в программном коде.

Рассмотрим, как добавить третью вкладку не в конструкторе, а в программном коде при обработке события загрузки формы (листинг). Однако прежде чем перейти на вкладку программного кода, для каждой вкладки выбираем из панели Toolbox по два переключателя RadioButton, а в форму перетаскиваем метку Label. Теперь с помощью щелчка правой кнопкой мыши в пределах формы переключаемся на редактирование программного кода.

Листинг Фрагмент программы, управляющей вкладками и переключателями

```
//     .............
//      Программный код, расположенный выше, создан средой Visual
Studio
//     автоматически, поэтому автором не приводится
this->ResumeLayout(false);
this->PerformLayout();
}
#pragma endregion
//      Программа, позволяющая выбрать текст из двух вариантов, задать цвет
//     и размер шрифта для этого текста на трех вкладках TabControl
//     с использованием переключателей RadioButton
private: System::
Void Form1_Load(System::Object^ sender, System::EventArgs^ e) {
//     Создание третьей вкладки "программно":
```

```
auto tabPage3 = gcnew System::Windows::Forms::TabPage(); tabPage3-
>UseVisualStyleBackColor = true;
//    Добавление третьей вкладки в существующий набор
//    вкладок tabControl1:
this->tabControl1->Controls->Add(tabPage3);
//    Добавление переключателей 5 и 6 на третью вкладку: tabPage3-
>Controls->Add(this->radioButton5); tabPage3->Controls->Add(this-
>radioButton6);
//    Расположение переключателей 5 и 6:
this->radioButton5->Location = System::Drawing::Point(20, 15); this->radioButton6-
>Location = System::Drawing::Point(20, 58); this->Text = "Какая улыбка вам ближе";
//    Задаем названия вкладок:
tabControl1->TabPages[0]->Text = "Текст"; tabControl1-
>TabPages[1]->Text = "Цвет"; tabControl1->TabPages[2]-
>Text = "Размер";
//    Эта пара переключателей изменяет текст:
radioButton1->Text =
"Восхищенная, сочувственная,\nскромно-смущенная"; radioButton2-
>Text = "Нежная улыбка, ехидная, бес" + "стыжая,\nподленькая,
снисходительная";
//    или
//    radioButton2->Text = "Нежная улыбка, бесстыжая," +
//    Environment::NewLine + "подленькая, снисходительная";
//    Эта пара переключателей изменяет цвет текста: radioButton3-
>Text = "Красный"; radioButton4->Text = "Синий";
//    Эта пара переключателей изменяет размет шрифта: radioButton5-
>Text = "11 пунктов"; radioButton6->Text = "13 пунктов"; label1->Text
= radioButton1->Text;
}
private: System::Void radioButton1_CheckedChanged(System::Object^ sender,
System::EventArgs^ e)
{ label1->Text = radioButton1->Text; }
private: System::Void radioButton2_CheckedChanged(System::Object^ sender,
System::EventArgs^ e)
{ label1->Text = radioButton2->Text; }
private: System::Void radioButton3_CheckedChanged(System::Object^ sender,
System::EventArgs^ e)
{ label1->ForeColor = Color::Red; }
private: System::Void radioButton4_CheckedChanged(System::Object^ sender,
System::EventArgs^ e)
{ label1->ForeColor = Color::Blue; }
private: System::Void radioButton5_CheckedChanged(System::Object^ sender,
System::EventArgs^ e)
{ label1->Font = gcnew System::Drawing:: Font(label1-
>Font->Name, 11); }
private: System::Void radioButton6_CheckedChanged(System::Object^ sender,
System::EventArgs^ e)
{ label1->Font = gcnew System::Drawing:: Font(label1-
>Font->Name, 13); }
};
}
```

Как видно из текста программы, при обработке события загрузки формы Form1_Load (этот участок программного кода можно было бы задать сразу после вызова

процедуры InitializeComponent) создаем «программно» третью вкладку. Заметьте, что мы ее объявили как auto, то есть тип переменной tabPage3 выводится из выражения инициализации в Visual C++. Далее добавляем новую вкладку tabPage3 в набор вкладок tabControl1, созданный в конструкторе. Затем «привязываем» пятый и шестой переключатели к третьей вкладке.

Каждая пара переключателей, расположенных на каком-либо элементе управления (в данном случае на различных вкладках), «отрицают» друг друга, то есть если пользователь выбрал один, то другой переходит в противоположное состояние. Отслеживать изменения состояния переключателей удобно с помощью обработки событий переключателей CheckChanged (см. листинг). Чтобы получить пустой обработчик этого события в конструкторе формы, следует дважды щелкнуть на соответствующем переключателе и таким образом запрограммировать изменения состояния переключателей.

Контрольные вопросы

1. Опишите, как добавить в программу вкладки.
2. По какому принципу работают переключатели RadioButton.
3. Опишите, как организовать работу группы переключателей.

Итог работы: файлы, ответы на контрольные вопросы, отчет

Практическая работа №4.

«Выполнение измерений характеристик кода в среде Visual Studio».

Цель: Научиться программировать консольное приложение, для вычисления математических функций, вводимых значений и вывода результата на экран. Получить практические навыки по использованию различных элементов графического интерфейса и операторов языка VC++.

Теоретические сведения

Иногда, например для научных расчетов, требуется организовать какой-нибудь самый простой ввод данных, выполнить весьма сложную математическую обработку введенных данных и оперативно вывести на экран результат вычислений.
Можно по-разному организовать такую программу, в том числе программируя так называемое *консольное приложение* (от англ. *console* — пульт управления). Под консолью обычно подразумевают экран компьютера и клавиатуру.

Задание: Напишем консольное приложение, которое приглашает пользователя ввести два числа , складывает их и выводит результат вычислений на консоль. Для этого запускаем Visual C++ 2010, далее создаем новый проект (New Project), в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, задаем имя решения (Name) — Сумма. После щелчка на кнопке OK попадаем сразу на вкладку программного кода (рис. 4.1).

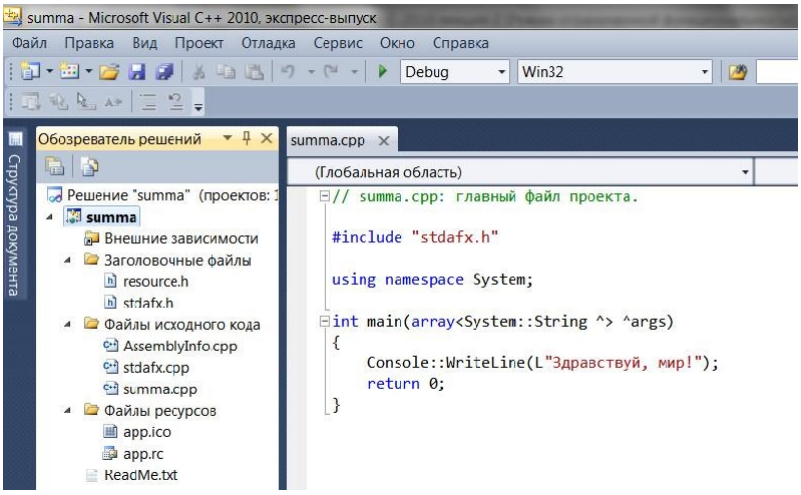


Рис. 4.1. Вкладка программного кода

Как видите, здесь управляющая среда Visual Studio приготовила несколько строк программного кода. Это вполне работоспособная программа. При запуске консольного или Windows-приложения C++ метод Main() является первым вызываемым методом. В фигурные скобки после Main() мы вставим собственный программный код (листинг). Фрагмент работы программы на рисунке 4.2.

Листинг Ввод и вывод данных в консольном приложении

```
//     Сумма.cpp: главный файл проекта.
//      Программа организует ввод двух чисел, их сложение и вывод суммы на консоль

#include "stdafx.h"
using namespace System;
int main(array<System::String ^> ^args)
{
//     Задаем строку заголовка консоли: Console::Title = "Складываю два числа:";
Console::BackgroundColor = ConsoleColor::Cyan; // - цвет фона Console::ForegroundColor =
ConsoleColor::Black; // - цвет текста Console::Clear();

//     Ввод первого слагаемого:
Console::WriteLine("Введите первое слагаемое:"); String^ Строка =
Console::ReadLine(); Single X, Y, Z;
//     Преобразование строковой переменной в число: X =
Single::Parse(Строка);
//     Ввод второго слагаемого: Console::WriteLine("Введите второе
слагаемое:");
Строка = Console::ReadLine(); Y =
Single::Parse(Строка);
Z=X+Y;
Console::WriteLine("Сумма = {0} + {1} = {2}", X, Y, Z);
//     Звуковой сигнал частотой 1000 Гц и длительностью 0.5 секунды:
Console::Beep(1000, 500);
//      Приостановить выполнение программы до нажатия какой-нибудь клавиши:
Console::ReadKey(); return
0;
}
```

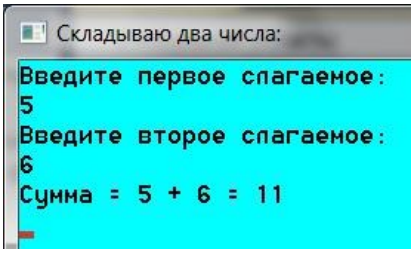


Рис. 4.2. Фрагмент работы консольного приложения

Итак, в данной программе Main() — это стартовая точка, с которой начинается ее выполнение. Обычно консольное приложение выполняется в окне на черном фоне. Чтобы как-то украсить традиционно черное окно консольного приложения, установим цвет фона окна BackgroundColor сине-зеленым (Cyan), а цвет символов, выводимых на консоль, черным (Black). Выводим строки в окно консоли методом WriteLine, а для считывания строки символов, вводимых пользователем, используем метод ReadLine. Далее объявляем три переменных типа Single для соответственно первого числа, второго и значения суммы. Тип данных Single применяется тогда, когда число, записанное в переменную, может иметь целую и дробную части.

Переменная типа Single занимает 4 байта. Для преобразования строки символов, введенных пользователем в числовое значение, используем метод Parse.

После вычисления суммы необходимо вывести результат вычислений из оперативной памяти на экран. Для этого воспользуемся форматированным выводом в фигурных скобках метода WriteLine объекта Console:

Console::WriteLine("Сумма = {0} + {1} = {2}", X, Y, Z)

Затем выдаем звуковой сигнал Beep, символизирующий об окончании процедуры и выводе на экран результатов вычислений. Последняя строка в программе Console::ReadKey(); предназначена для приостановки выполнения программы до нажатия какой-нибудь клавиши. Если не добавить эту строку, окно с командной строкой сразу исчезнет, и пользователь не сможет увидеть вывод результатов выполнения. Программа написана. Нажмите клавишу F5, чтобы увидеть результат.

Контрольные вопросы

1. Что такое форматированный ввод?
2. Какая команда помогает задерживать результат работы программы на экране?
3. Опишите вывод результата.

Итог работы: файлы, защита

Практическая работа №5.

«Выполнение измерений характеристик кода в среде (например, Eclipse C/C++ и др.)»

Цель: Научиться создавать элементы управления в форме «программным» способом, обрабатывать несколько событий одной процедурой. Получить практические навыки по использованию различных элементов графического интерфейса и операторов языка VC++.

Задание:

1. Создать новый проект с формой. При этом, как обычно, запускаем Visual Studio 2010, в окне New Project выбираем в среде

CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Чтобы к программному коду добавить пустой обработчик события загрузки формы, дважды щелкнем на

проектируемой экранной форме. Далее вводим программный код, представленный в листинге :

```
// ..............
// Программный код, расположенный выше, создан средой Visual
Studio
// автоматически, поэтому автором не приводится
this->Load += gcnew System::EventHandler(this,
&Form1::Form1_Load);
this->ResumeLayout(false);
}
#pragma endregion
// Программа создает командную кнопку в форме «программным»
способом,
// т.е. с помощью написания непосредственно программного кода,
не
// используя при этом панель элементов управления Toolbox.
Программа
// задает свойства кнопки: ее видимость, размеры, положение,
надпись
// на кнопке и подключает событие "щелчок на кнопке"
private: System::Void Form1_Load(System::Object^ sender,
System::EventArgs^ e)
{
// Создание кнопки без панели элементов управления:
Button^ button1 = gcnew Button();
// Задаем свойства кнопки:
button1->Visible = true;
// Ширина и высота кнопки:
button1->Size = Drawing::Size(100, 30);
// Расположение кнопки в системе координат формы:
button1->Location = Drawing::Point(100, 80);
button1->Text = "Новая кнопка";
// Добавление кнопки в коллекцию элементов управления
this->Controls->Add(button1);
// Подписку на событие Click для кнопки можно делать "вручную".
// Связываем событие Click с процедурой обработки этого события:
button1->Click += gcnew EventHandler(this,
&Form1::ЩелчокНаКнопке);
}
private: System::
Void ЩелчокНаКнопке(System::Object^ sender, System::EventArgs^
e)
{
MessageBox::Show("Нажата новая кнопка");
}
};}
```

Мы видим, что при обработке события загрузки формы создаем новый объект button1 стандартного класса кнопок. Задаем свойства кнопки: ее видимость (Visible), размеры (Size), положение (Location) относительно левого нижнего угла формы, надпись на кнопке — ≪Новая кнопка≫.

Далее необходимо организовать корректную работу с событием «щелчок на созданной нами командной кнопке». В предыдущих примерах мы для этой цели в

конструкторе формы дважды щелкали на проектируемой кнопке, и исполняемая среда автоматически генерировала пустой обработчик этого события в программном коде. Или опять же в конструкторе формы в панели свойств проектируемой кнопки щелкали мышью на значке молнии (Events) и в появившемся списке всех событий выбирали необходимое событие. Однако согласно условию задачи мы должны организовать обработку события «программным» способом без использования конструктора формы. Для этого в программном коде сразу после добавления командной кнопки в коллекцию элементов управления поставим оператор стрелки (->) после имени кнопки button1 и в раскрывающемся списке выберем необходимое событие Click. Затем, как приведено в листинге 3.2, осуществляем так называемую «подписку» на данное событие, то есть с помощью ключевого слова EventHandler связываем событие Click с процедурой обработки события. Мы его назвали ≪Щелчок-НаКнопке≫. Теперь создадим обработчик события «щелчок на кнопке», как показано в листинге. В этой процедуре предусматриваем вывод сообщения ≪ Нажата новая кнопка≫. На рисунке 5.1 приведен фрагмент работы программы.

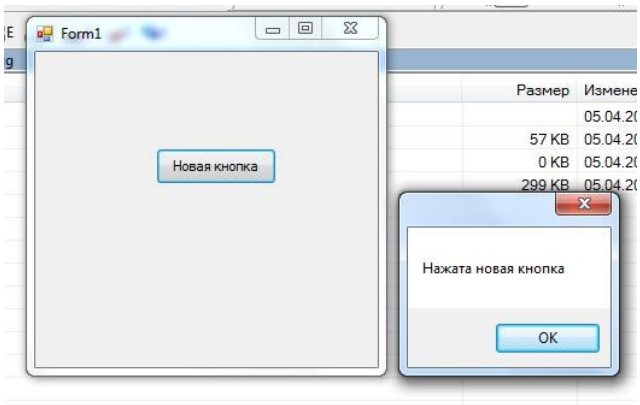


Рис.5.1 Фрагмент работы программы

В заключение отметим, что в случае создания пустого обработчика события в конструкторе формы строка подписки на событие формируется автоматически в методе InitializeComponent в файле Form1.h проекта.

Запустить Visual Studio 2010, в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем из панели элементов перенесем в форму две командные кнопки и текстовую метку. Далее через двойной щелчок мышью в пределах проектируемой формы создать пустой обработчик загрузки формы и перейдем к вкладке программного кода (листинг).

```
// .............
// Программный код, расположенный выше, создан средой Visual
Studio
// автоматически, поэтому автором не
приводится this->ResumeLayout(false); this-
>PerformLayout();
}
#pragma endregion
```

```
// В форме имеем две командные кнопки, и при нажатии указателем
мыши
//  любой из них получаем номер нажатой кнопки. При этом в
программе
// предусмотрена только одна процедура обработки событий
private: System::Void Form1_Load(System::Object^ sender,
System::EventArgs^ e)
{
Form1::Text = "Щелкните на кнопке"; label1->Text = nullptr;
// Связываем события Click от обеих кнопок с одной
процедурой КЛИК:
button1->Click += gcnew EventHandler(this, &Form1::КЛИК);
button2->Click += gcnew EventHandler(this, &Form1::КЛИК);
// Подпиской на событие называют связывание названия события
// с названием процедуры обработки события посредством
EventHandler
}
private: System::Void КЛИК(System::Object^ sender,
System::EventArgs^ e)
{
// String S = Convert.ToString(sender);
// получить текст, отображаемый на кнопке, можно таким образом:
Button^ Кнопка = (Button^)sender;
// или String^ НадписьНаКнопке = ((Button^)sender)->Text;
label1->Text = "Нажата кнопка " + Кнопка->Text; // или
Кнопка->Name
}
};}
```

Как видно из текста программы, при обработке события загрузки формы мы осуществляем так называемую подписку на событие, то есть связываем название события с названием процедуры обработки события КЛИК посредством метода (делегата) EventHandler. Этот метод делегирует (передает полномочия) обработку события button1->Click процедуре КЛИК. Заметим, что события Click от обеих кнопок мы связали с одной и той же процедурой КЛИК.

Далее создаем процедуру обработки события КЛИК, ее параметр sender содержит *ссылку на объект-источник события*, то есть кнопку, нажатую пользователем.
С помощью неявного преобразования можно конвертировать параметр sender в экземпляр класса Button и, таким образом, выяснить все свойства кнопки, которая инициировала событие. На рисунке 5.2 приведен пример работы написанной программы.

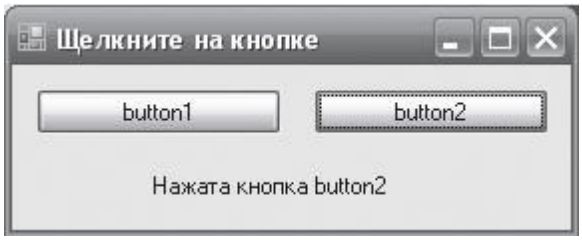


Рис.5.2 Фрагмент работы программы, определяющей нажатую кнопку

Контрольные вопросы

1. Опишите создание кнопок Button.
2. Опишите свойства Size, Point.
3. Что такое обработчик событий?
4. Опишите процедуру gcnew EventHandler.

5. Для чего нужен параметр sender?
6. Опишите связывание событий.

Итог работы: файлы, ответы на вопросы

Практическая работа №6.

«Определение проекта и его границ».
«Организационные структуры». «Методологии ведения проектов».

Цель: Научиться создавать приложение для обработки событий клавиатуры. Получить практические навыки по использованию различных элементов графического интерфейса и операторов языка VC++.

Задание: Выполните последовательность действий:

Напишите программу, информирующую пользователя о тех клавишах и комбинациях клавиш, которые тот нажал. Запустим Visual Studio 2010, в окне New Project выберем в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Затем из панели Toolbox перетащим в форму две текстовых метки Label.

Далее, поскольку нам потребуются клавишные события формы: KeyPress, KeyDown, KeyUp, уже привычным способом получим пустые обработчики этих событий. То есть в панели Properties щелкнем на пиктограмме молнии (Events), а затем в списке всех возможных событий выберем каждое из названных событий клавиатуры.

Программный код приведен **в листинге** :

```
// ..............
//  Программный код, расположенный выше, создан средой Visual
Studio автоматически
this->ResumeLayout(false);
this->PerformLayout();
}
#pragma endregion
// Программа, информирующая пользователя о тех клавишах
// и комбинациях клавиш, которые тот нажал
private: System::
Void Form1_Load(System::Object^ sender, System::EventArgs^
e) {
// Устанавливаем шрифт с фиксированной шириной (моноширинный):
Form1::Font = gcnew Drawing::
Font(FontFamily::GenericMonospace, 14.0F);
// Поскольку мы задали этот шрифт увеличенным (от 8 по умолчанию
// до 14), форма окажется пропорционально увеличенной
Form1::Text = "Какие клавиши нажаты сейчас:"; label1->Text
= String::Empty; label2->Text = String::Empty;
}
private: System::Void Form1_KeyPress(System::Object^
sender, System::Windows::Forms::KeyPressEventArgs^ e)
{
// Здесь событие нажатия клавиши: при удержании
// клавиши генерируется непрерывно
label1->Text = "Нажатая клавиша: " + e->KeyChar;
}
private: System::Void Form1_KeyDown(System::Object^ sender,
System::Windows::Forms::KeyEventArgs^ e)
{
// Здесь обрабатываем мгновенное событие первоначального
```

```
// нажатия клавиши
label2->Text = String::Empty;
// Если нажата клавиша Alt
if (e->Alt == true) label2->Text += "Alt: Yes\n";
else label2->Text += "Alt: No\n";
// Если нажата клавиша Shift
if (e->Shift == true) label2->Text += "Shift: Yes\n";
else label2->Text += "Shift: No\n";
// Если нажата клавиша Ctrl
if (e->Control == true) label2->Text += "Ctrl: Yes\n";
```

26

```
else label2->Text += "Ctrl: No\n";
label2->Text += String::Format(
"Код клавиши: {0} \nKeyData: {1} \nKeyValue:
{2}", e->KeyCode, e->KeyData, e->KeyValue);
}
private: System::Void Form1_KeyUp(System::Object^
sender, System::Windows::Forms::KeyEventArgs^ e) {

// Очистка меток при освобождении клавиши
label1->Text = String::Empty; label2->Text = String::Empty;
}
};
}
```

В первую метку label1 записываем сведения о нажатой обычной (то есть не модифицирующей и не функциональной) клавише при обработке события KeyPress.

Во вторую метку из аргумента события e (e->Alt, e->Shift и e->Control) получаем сведения, была ли нажата какая-либо модифицирующая клавиша (либо их комбинация). Обработчик события KeyUp очищает обе метки при освобождении клавиш.
На рисунке 6.1 приведен фрагмент работы программы.

25

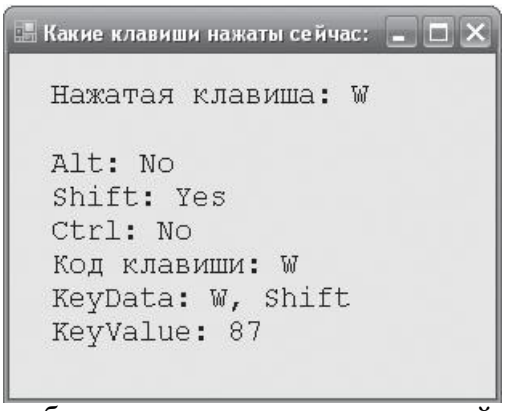


Рис. 6.1 Фрагмент работы программы, определяющей нажатую клавишу

Контрольные вопросы

1. Что означает модифицирующая клавиша?
2. Какое событие необходимо обработать, чтобы узнать нажата ли модифицирующая клавиша?
3. В какой момент нажатия клавиши генерируется событие KeyDown, KeyUp?

Итог работы: файлы, ответы на вопросы

Практическая работа №7.

«Построение команды проекта». «Начало и завершение проекта»

Цель: Изучить приемы работы с файлами и способы создания файлов. Получить практические навыки по использованию файлов для хранения информации.

Задание:

Напишите программу, содержащую на экранной форме текстовое поле и две командные кнопки. При щелчке мышью на первой кнопке происходит чтение текстового файла в текстовое поле в кодировке Unicode. При щелчке на второй кнопке отредактированный пользователем текст в текстовом поле сохраняется в файл на диске.

Запустим систему Visual Studio 2010 и в окне New Project выберем
в среде CLR узла Visual C++ приложение шаблона Windows Forms Application Visual C++. Далее в форму из панели Toolbox перенесем текстовое поле и две командные копки. Для текстового поля в окне Properties сразу укажем для свойства Multiline значение True, чтобы текстовое поле имело не одну строку, а столько, сколько поместится в растянутом указателем мыши поле. Одна кнопка предназначена для открытия файла, а другая — для сохранения файла на машинном носителе. В листинге приведен текст данной программы «Чтение/запись текстового файла в кодировке Unicode».

Листинг:

```
// ..............
// Программный код, расположенный выше, создан средой Visual
Studio
// автоматически, поэтому автором не приводится
this->ResumeLayout(false);
this->PerformLayout();
}
#pragma endregion
// Программа для чтения/записи текстового файла в кодировке
Unicode
String ^ filename;
// Объявляем filename здесь, чтобы эта переменная была "видна"
// в процедурах обработки обоих событий.
private: System::Void Form1_Load(System::Object^
sender, System::EventArgs^ e)
{
// Установка начальных значений:
textBox1->Multiline = true; textBox1->Clear();
textBox1->Size = Drawing::Size(268, 112);
button1->Text = "Открыть"; button1->TabIndex = 0;
button2->Text = "Сохранить";

Form1::Text = "Здесь кодировка
Unicode"; filename = "C:\\Text1.txt";
}
private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^ e)
{
// Щелчок на кнопке Открыть.
// Русские буквы будут корректно читаться,
// если открыть файл в кодировке UNICODE:
try
{
```

```
// Создание объекта StreamReader для чтения из файла:
auto Читатель = gcnew IO::StreamReader(filename);
// Непосредственное чтение всего файла в текстовое
поле: textBox1->Text = Читатель->ReadToEnd();
Читатель->Close(); // закрытие файла
// Читать текстовый файл в кодировке UNICODE в массив строк
// можно также таким образом (без Open и Close):
// array <String^>^ МассивСтрок =
// IO::File::ReadAllLines("C:\\Text1.txt");
}
catch (IO::FileNotFoundException^ Ситуация)
{ // Обработка исключительной ситуации:
MessageBox::Show(Ситуация->Message + «\nНет такого файла»,
"Ошибка", MessageBoxButtons::OK,
MessageBoxIcon::Exclamation);
}
catch (Exception^ Ситуация)
{
// Отчет о других ошибках:
MessageBox::Show(Ситуация->Message, "Ошибка",
MessageBoxButtons::OK,
MessageBoxIcon::Exclamation); }
}
private: System::Void button2_Click(System::Object^ sender,
System::EventArgs^ e)
{
// Щелчок на кнопке Сохранить:
try
{
// Создание объекта StreamWriter для записи в
файл: auto Писатель = gcnew
IO::StreamWriter(filename, false); Писатель-
>Write(textBox1->Text); Писатель->Close();

// Сохранить текстовый файл можно также таким образом
// (без Close), причем, если файл уже существует,
// то он будет заменен:
// IO::File::WriteAllText("C:\\tmp.tmp", textBox1->Text);
}
catch (Exception^ Ситуация)
{
// Отчет обо всех возможных ошибках:

MessageBox::Show(Ситуация->Message, "Ошибка",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation);
}
}
};
}
```

Блоки try, которые, как мы видим, используются в данном программном коде. Логика использования try следующая: *попытаться* (try) выполнить некоторую задачу, например прочитать файл. Если задача решена некорректно (например, файл не найден), то *«перехватить »* (catch) управление и *обработать* возникшую (*исключительную*, Exception) ситуацию.

При обработке события «щелчок на кнопке Открыть» организован ввод файла C:\Text1.txt. Обычно в этой ситуации пользуются элементом управления OpenFileDialog для выбора файла. Мы не стали использовать этот элемент управления для того, чтобы *не «заговорить»* проблему, а также свести к минимуму программный код.

Далее создаем объект (поток) Читатель для чтения из файла. Для большей выразительности операций с данным объектом мы назвали его русскими буквами.

При обработке события «щелчок на кнопке Сохранить» организована запись файла на диск аналогично через объект Писатель. При создании объекта Писатель первым аргументом является filename, а второй аргумент false указывает, что данные следует *не добавить* (append) к содержимому файла (если он уже существует), а *перезаписать* (overwrite). Запись на диск производится с помощью метода Write() из свойства Text элемента управления textBox1. На рисунке 7.1 приведен фрагмент работы программы.

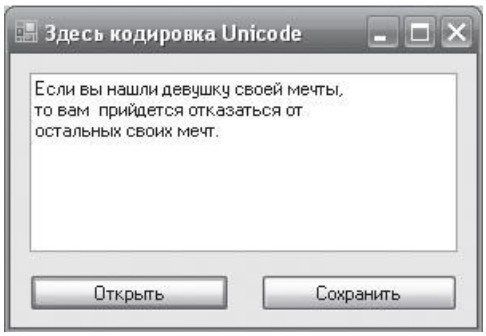


Рис.7.1 Чтение/запись текстового файла в кодировке Unicode

Запись текстового файла с помощью данной программы будет происходить *в формате (кодировке) Unicode*, как и чтение из файла. То есть вы сможете читать эти файлы Блокнотом, редактировать их, но каждый раз при сохранении файлов следить, чтобы кодировка была (оставалась) Unicode.

Контрольные вопросы

К чему свелась обработка исключительной ситуации ?
К каким методом происходит чтение файла filename?
К каким методом происходит закрытие файла?

Итог работы: файл, ответы на контрольные вопросы

Практическая работа №8.

«Коммуникации в проекте». «Построение иерархической структуры работ проекта».

Цель: Научиться создавать меню, передавать значения между диалоговыми окнами и главным окном приложения. Получить практические навыки в разработке программ.

Задание:

1. Создайте приложение с помощью Visual Studio 2010, в окне New Project выберте в среде CLR узла Visual C++ приложение шаблона
Windows Forms Application
Для создания меню на панели инструментов выберите *MenuStrip* . Дважды кликните на появившемся в нижней области окна объекте,
а затем перейдите на форму и в области *(Вводить здесь)* введите меню верхнего уровня с текстом *Цвет*.
Переместитесь на нижнюю область и введите текст *Черный.* Заполните элемент MenuStrip следующим образом:

Цвет
Черный
Красный
Синий
Зеленый

Запустите программу и поэкспериментируйте: выбирайте разные элементы созданного объекта.
Запрограммируйте событие *Click* для каждого пункта; например, для элемента *Черный* необходимо написать следующий код (дважды щелкнув на пункте, чтобы открыть код):

```
private: System::Void черныйToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
  this->txt->BackColor=System::Drawing::Color::Black;
}
```

Запрограммируйте аналогично остальные пункты *Цвет*. Запустите и отладьте приложение. Сохраните проект.

Контрольные вопросы

1. Каково основное назначение объекта MenuStrip?
2. Как запрограммировать необходимый пункт меню формы в Visual c++?
3. Какое свойство служит для изменения фона объекта?
4. С помощью какого свойства menu можно сделать недоступным какой-либо пункт?

Итог работы: файл, защита, ответы на вопросы.

Практическая работа №9.
«Смета затрат на разработку и реализацию проекта»

Цель: составить смету затрат на разработку и реализацию проекта. Изучить возможности программного продукта Microsoft Project, предназначенного для управления проектами

Задание1. Изучить теоретические сведения:

Разработка смет — процесс структуризации и систематизации стоимостных оценок, полученных на этапе оценки стоимости. Структуризация и систематизация данных о стоимости работ производится в соответствии со статьями затрат, принятыми в системе учета родительской организации проекта.

Смета — документ, содержащий список затрат проекта, полученных на основе объемов работ проекта, требуемых ресурсов и цен, структурированный по статьям.
Если в проекте (родительской организации) проектные сметы принято структурировать по работам, то процесс разработки смет значительно упрощается. Оценки, структурированные по работам, переносятся в смету и сводятся в единый документ.

Если же требованием компании является структуризация расходов в смете по статьям затрат, процесс несколько усложняется. Обычно выделяют:

- прямые затраты (расходы);
- накладные (косвенные) затраты;
- общие и административные накладные расходы.

Прямые затраты — расходы, непосредственно связанные с производством продукции, работ проекта; производственные расходы, включаемые в себестоимость продукции, в прямые издержки производства.

Прямые расходы напрямую связаны с пакетом работ. Они включают:

- затраты на оплату труда;
- затраты на материалы и оборудование;
- иные расходы, связанные с выполнением работ.

Именно на прямые расходы могут непосредственно влиять менеджер проекта и его команда. Влияние команды проекта на другие расходы ограниченно.

Накладные расходы (косвенные затраты) — расходы, сопровождающие, сопутствующие основному производству, но не связанные с ним напрямую, не входящие в стоимость труда и материалов. Накладные расходы не могут быть привязаны к какой-то конкретной работе, конкретному результату. Они относятся ко всему проекту в целом. Это затраты на:

- содержание и эксплуатацию основных средств;
- управление, организацию, обслуживание производства;
- командировки;
- обучение работников.

Общие и административные накладные расходы (постоянные расходы) — затраты, не связанные с каким-то конкретным проектом. Они относятся к расходам компании, но при этом имеют отношение и к проекту. К общим и административным расходам обычно относятся расходы на содержание аппарата управления, поддерживающих подразделений (бухгалтерия, секретариат, охрана и др.).

Задание: Оформите смету затрат на разработку и реализацию проекта.

Таблица 1 Расчет материальных затрат.

Наименование	
Единица измерения	
Количество	
Стоимость, руб.	
за единицу, руб.	
общая, руб.	
ИТОГО:	
Транспортные расходы (15%):	
ВСЕГО:	

Таблица 2 Смета выполненных работ.

Наименование вида выполненных работ	
Единица измерения	
Количество	
Стоимость, руб. выполненные работы в том числе З/П	
за единицу, руб.	
общая, руб.	
ИТОГО:	
Наименование	
Единица измерения	
Количество	
Стоимость, руб.	
за единицу, руб.	
общая, руб.	
Транспортные расходы (15%):	
Всего с транспортными расходами:	
Плановые накопления (4-8% от итого материалы и выполненные работы)	
Всего с плановыми накоплениями	

Сводка итогов:	
А) Материальные затраты	
Б) Выполненные работы и материалы	
Всего:	
Заработная плата:	
Начисления на заработную плату:	
Всего с начислениями:	
ВСЕГО ПО СМЕТЕ	

Задания 2:

1. Познакомиться с режимами и основными функциями программы Microsoft Project.
2. Изучить возможности создания и оптимизации проектов с помощью программы Microsoft Project.
3. Выполнить индивидуальное задание.
4. Подготовить отчет о проделанной работе.

Порядок выполнения работы

1. Основные сведения о программе

Для поддержки процесса управления проектом на различных этапах существует большое количество программных комплексов, целью применения которых является повышение эффективности реализации проекта, т. е. выполнение как всего проекта в целом, так и его отдельных этапов в заданные сроки и в рамках утвержденных денежных ресурсов. Основными задачами менеджера проекта является составление сетевого графика (расписания) проекта, распределение ресурсов между задачами проекта и слежение за ходом реализации проекта.

Внедрение компьютерных технологий в практику реализации проекта может оказать существенную помощь в эффективной реализации проектов.

Анализ существующих программных комплексов поддержки деятельности по управлению проектами показал, что для малых проектов, в которых удельный вес этапа реализации значителен, предпочтительным является пакет программ Microsoft Project (MS Project). Он позволяет эффективно выполнить структуризацию проекта путем разделения его на этапы и подзадачи, выявить критические задачи (задачи, длительность которых существенно влияет на длительность реализации всего проекта), получить сетевой график проекта, осуществить назначение ресурсов задачам проекта, контролировать загрузку ресурсов.

Microsoft Project позволяет эффективно управлять проектом на различных этапах его реализации. Он дает возможность выполнить структуризацию проекта путем разделения его на этапы, задачи и подзадачи, выявить критические задачи (задачи, длительность которых существенно влияет на длительность реализации всего проекта), получить сетевой график и календарный план проекта, осуществить назначение ресурсов задачам проекта, эффективно контролировать загрузку ресурсов. Пакет поддерживает все необходимые типы связей между задачами: FS (Finish-Start), SS (Start-Start), FF (Finish-Finish).

Поддерживая современные информационные технологии, пакет MS Project позволяет импортировать данные из файлов, созданных в среде других приложений, например MS Excel и MS Access. Неоспоримым достоинством пакета является наличие встроенного языка программирования Visual Basic For Application, что обеспечивает возможность разработки программных компонент, обеспечивающих решение специфических задач.

2. Построение модели

Методика использования пакета Microsoft Project для управления инновационным проектом на этапе подготовки к реализации, целью которой является получение сетевого графика и календарного плана проекта, может быть представлена в виде последовательности следующих шагов:

- создание календаря проекта (т. е. учет нерабочих и праздничных дней);
- составление списка задач, которые надо выполнить для успешной реализации проекта;
- определение связей между задачами;
- выявление задач, длительность реализации которых существенно влияет на длительность реализации всего проекта, и, возможно, изменение порядка выполнения задач проекта;
- формирование списка доступных для реализации проекта ресурсов;
- распределение ресурсов (назначение ресурсов конкретным задачам проекта).

Начало работы над проектом

Календарь

Одной из задач, решаемых на этапе подготовки к реализации проекта, является получение сетевого графика проекта. При построении сетевого графика MS Project использует календарь — таблицу, в которой отражены рабочие и нерабочие (выходные и праздничные) дни. Различают стандартный календарь и календари пользователя. Стандартный календарь предполагает, что рабочими днями являются все дни недели, за исключением субботы и воскресенья, и рабочий день длится 8 часов. Календарь пользователя учитывает реальные рабочие дни. Например, очевидно, что в России реальный календарь в январе и мае существенно отличается от стандартного.

При работе с проектом MS Project позволяет, помимо базового календаря, использовать несколько календарей пользователя. Можно, например, создать календарь для всего проекта в целом (основной календарь) и календарь для отдельного ресурса, который будет учитывать особенности его использования (например, недоступность ресурса в определенные дни или месяцы).

Задачи проекта

Задача – это некоторая деятельность, которую надо выполнить, чтобы завершить часть проекта. Работа над проектом начинается с составления списка задач проекта.

Большие, сложные задачи, как правило, могут быть естественным образом представлены в виде набора боле простых, более конкретных задач. Поэтому при составлении списка сначала записывают общую (обобщенную) задачу, затем – задачи, из которых эта общая задача состоит (подчиненные задачи).

При составлении списка задач проекта обычно используют метод, который часто называют «сверху — вниз». Суть метода заключается в том, что сначала составляют список главных (обобщенных) задач, затем этот список уточняется посредством добавления уточняющих задач.

Каждая задача проекта характеризуется длительностью, которая измеряется в минутах, часах, днях и неделях. Длительность обобщенной задачи определяется длительностью ее подчиненных задач. Длительность обобщенной задачи вычисляет MS Project.

Длительность подчиненной задачи нижнего уровня, т. е. задачи? у которой нет подчиненных задач, определяется временем необходимым для ее выполнения одной единицей ресурса. Например, один рабочий копает траншею в 5 метров 8 часов. Если для реализации проекта необходимо выкопать траншею длинной в 10 метров, то длительность задачи «Траншея» равна 16 часам. Длительность подчиненной задачи задает менеджер проекта на основе нормативной документации.

Контрольные точки проекта

В каждом проекте могут быть выделены этапы, после выполнения которых процесс реализации проекта переходит на новый качественный уровень. Например, после этапа согласования требований с заказчиком начинается этап подготовки технической документации. Можно считать, что последней задачей этапа согласования требований с заказчиком является задача утверждения требований. Подобные задачи, как правило, должны выполняться в конкретный, заранее установленный день, поэтому их называют контрольными точками. Длительность задач – контрольных точек – принимают равной нулю.

Ответы на контрольные вопросы.

1. Что такое смета проекта?

2. Что такое прямые затраты (расходы), накладные (косвенные) затраты, общие и административные накладные расходы?

Итог работы: отчет

4. ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ПРАКТИЧЕСКИХ РАБОТ

4.1 Печатные изделия:

Основные:

О-1. Поколодина, Е.В. Ревьюирование программных модулей: учебник для студ. учреждений сред. проф. образования / Е.В. Поколодина, Н.А. Долгова, Д.В. Ананьев. – Издательский центр «Академия», 2020. – 208 с.

Дополнительные:

Д-1. Кокорева О.И., Реестр Windovs XP: / О.И. Кокорева - М.: БХВ-Петербург, 2008. -560 с.

Д-2. Омельченко Л.Н., Федоров А.Ф., Реестр Windovs XP: самоучитель/ Л.Н. Омельченко, А.Ф. Федоров - М.: БХВ-Петербург, 2007. – 560 с.

Д-3. Голицына О.Л., Партыка Т.Л., Попов И.И. Программное обеспечение: учебное пособие/ О.Л. Голицына, Т.Л. Партыка, И.И. Попов - М.: ИД "ФОРУМ"-ИНФРА-М, 2006. – 432 с.

Д-4. Голицына О.Л., Партыка Т.Л., Попов И.И. Программное обеспечение: учебное пособие/ О.Л. Голицына, Т.Л. Партыка, И.И. Попов - М.: ИД "ФОРУМ"-ИНФРА-М, 2008. – 432 с.

Д-5. Ломов А.Ю. HTML, CSS, скрипты: практика создания сайтов / Ю.И. Волков. - М.: Питер, 2007. – 416 с.

Д-6. Титтел Э., Бурмейстер М. HTML для чайников/ Э.Титтел , М. Бурмейстер - М.: Вильямс, 2008. – 368 с.

Д-7. Полонская Е.Л., язык HTML: самоучитель/ Е.Л. Полонская - М.: Вильямс, 2004. – 320 с.

Д-8.Технология разработки программных продуктов: Практикум: учебник для студ. сред. проф. образования/ А. В. Рудаков, Федорова Г.Н. - 12-е изд., стер. – М.: Издательский центр «Академия», 2017. – 208 с.

Д-9. Богданов В. В., Управление проектами в Microsoft Project 2007. Учебный курс, Уч. Пособие. – Издат. Питер, 2015. – 592 с.

Д-10. Рудаков А. Технология разработки программных продуктов: учебник. / Рудаков А. – Москва: Академия, 2018. –208 с.

4.2 Электронные издания (электронные ресурсы)

1. Единое окно доступа к информационным ресурсам [Электронный ресурс]. –Режим доступа: http://window.edu.ru/
2. *Федоров Г.Н., Учебник: Разработка модулей программного обеспечения для компьютерных систем, ИЦ Академия, 2017. - 350с., 25 подключений*
3. *Федоров Г.Н., Учебник: Осуществление интеграции программных модулей ИЦ Академия, 2017. - 282с., 25 подключений;*
4. *Федоров Г.Н., Учебник: Разработка, администрирование и защита баз данных ИЦ Академия, 2017. - 282с., 25 подключений;*

5. ЛИСТ ИЗМЕНЕНИЙ И ДОПОЛНЕНИЙ, ВНЕСЕННЫХ В МЕТОДИЧЕСКИЕ УКАЗАНИЯ

№ изменения, дата внесения, № страницы с изменением	
Было	**Стало**
Основание: **Подпись лица, внесшего изменения**	