

Разработка сложного программного обеспечения (ПО) в кратчайшие сроки и с наименьшими затратами требует правильной организации всего процесса создания ПО. Инженерно-техническую дисциплину, связанную с *созданием ПО и поддержанием его в рабочем состоянии* называют "программная инженерия". В нее входят анализ и постановка задачи, построение алгоритмов и разработка структур данных, проектирование программ и написание кода, отладка и тестирование ПО, документирование, сопровождение и многие другие организационные и технические стороны создания ПО. Все эти процессы исследуются и постоянно совершенствуются в рамках дисциплин *конструирование ПО и Технология программирования*.

Конструирование ПО – это детальное создание работающего программного обеспечения посредством комбинации кодирования, верификации, модульного тестирования, интеграционного тестирования и отладки.

Технология программирования – совокупность методов и средств, используемых в процессе разработки программного обеспечения. Как и любая другая, технология программирования представляет собой набор правил и инструкций, включающих:

- перечень и указание последовательности выполнения технологических операций;
- перечисление условий, при которых выполняется та или иная операция;
- описания самих операций, где для каждой операции определены исходные данные, результаты, а также инструкции, нормативы, стандарты, критерии и методы оценки и т. п.

Сегодня разработка ПО – серьезный и ресурсоемкий бизнес. Существенные доходы могут быть получены только при условии предоставления пользователям таких новых возможностей, за которые они готовы будут заплатить деньги. Для этого при составлении технического задания на разработку ПО необходимо учитывать многое: например, производительность процессоров и объем оперативной памяти в компьютерах потенциальных пользователей, скорость и объем доступного интернет трафика, удобство интерфейса, экономическую целесообразность применения ПО или высокое качество развлечений и т.п.

Большая часть трудностей при разработке ПО связана, в том числе, с организацией совместной работы многих людей, возможно удаленных друг от друга географически. Причем работа этих людей должна быть организована так, чтобы затраты на разработку были меньше доходов от продажи ПО. Рассчитывая стоимость разработки ПО необходимо учитывать, что в затраты входит не только зарплата разработчиков, затраты на оборудование и инструменты разработки, приобретение лицензий. Необходимо тратить деньги и на исследование потребностей клиентов и на проведение рекламы и другой маркетинговой деятельности.

Необходимо учитывать также, что практически значимые программные системы всегда содержат ошибки, т.е. нельзя утверждать, что она всегда правильно решает все поставленные перед ней задачи. Этот факт связан с практической невозможностью доказательства отсутствия ошибок или полной проверки работы ПО. Поэтому разработка сложного ПО связана с нахождением разумного компромисса между затратами на разработку и качеством ее результата. В затраты входят все виды используемых ресурсов, из которых наиболее важны затраченное время и бюджет проекта. Удовлетворение пользователей от работы с программой определяется термином *качество программы*, которое включает в себя набор предоставляемых возможностей, надежность, удобство использования, гибкость, удобство внесения изменений и исправления ошибок.

Создание качественного ПО в заданные сроки и с минимальной стоимостью возможно лишь при использовании современной *технологии программирования*.

ЛАБОРАТОРНАЯ РАБОТА № 1

Этапы разработки программного обеспечения. Договор на разработку. Техническое задание

Цель работы: ознакомиться с процессом заключения договора на разработку программного обеспечения (ПО) и правилами подготовки технического задания (ТЗ) на разработку ПО.

Продолжительность работы – 4 часа.

Содержание

Теория. Процесс заключения договора.....	2
Порядок разработки ТЗ	4
Порядок выполнения лабораторной работы.....	8
Вопросы.....	8
Приложения	9

1. Процесс заключения договора

Создание программного обеспечения – процесс дорогой и ресурсоемкий. Любой продукт (не только ПО!) создается либо по чьему-то заказу, либо за счет внутренних ресурсов компании с перспективой дальнейшей продажи. Чаще всего создание ПО выполняется компаниями-разработчиками по заказам других организаций и начинается этот процесс с заключения договора на создание продукта.

Необходимо понимать, что заключение договора – процесс юридический, подробно определен законодательством РФ (ст. 432 ГК) и предполагает две стадии:

1. предложение (оферта);
2. ее принятие (акцепт).

Соответственно можно выделить следующие стадии заключения договора:

- преддоговорные контакты сторон (переговоры);
- оферта;
- рассмотрение оферты;
- акцепт оферты.

При этом две стадии — оферта и акцепт оферты — являются обязательными для всех случаев заключения договора.

Стадия *преддоговорных контактов* сторон (переговоров) носит факультативный характер и используется по усмотрению сторон, вступающих в договорные отношения. На этапе преддоговорного периода стороны формулируют стоящие перед ними экономические задачи, которые предстоит реализовать.

Оферт (оформленной документально, направленной и полученной адресатом) присуще еще одно важное свойство — безотзывность. Принцип безотзывности оферты, т.е. невозможности для оферента отзывать свое предложение о заключении договора в период с момента получения его адресатом и до истечения установленного срока для ее акцепта, сформулирован в виде презумпции (ст. 436 ГК).

Для того чтобы договор был признан заключенным, необходим полный и безоговорочный *акцепт*, т.е. согласие лица, получившего оферту, на заключение договора на предложенных в оферте условиях (ст. 443 ГК).

Момент заключения договора определяет время его вступления в силу, т.е. обязательность для сторон условий заключенного договора (п. 1 и 2 ст. 425 ГК). Именно с этого момента исполнитель несет полную финансовую и правовую ответственность за исполнение принятых на себя обязательств.

Разработка ПО – это всегда новое и штучное. Даже выпуск новой версии уже существующего ПО – это всегда развитие и совершенствование. Разработка ПО – это всегда дорого и сложно. Поэтому при заключении договора на разработку необходимо учитывать не только желание заказчиков, но и технические (опыт и навыки коллектива разработчиков, материальное обеспечение) и организационные возможности компании.

Неотъемлемой частью договора являются следующие документы (оформляемые как приложения к договору):

- техническое задание;
- план-график выполнения работ;
- классификация дефектов и сроки их устранения.

По общепринятым на сегодня как международным так и Российским стандартам процесс разработки включает следующие действия:

- *анализ требований к системе* – определение ее функциональных возможностей, пользовательских требований, требований к надежности и безопасности, требований к внешним интерфейсам и т. д.;
- *подготовительную работу* – выбор модели жизненного цикла (см. далее), стандартов, методов и средств разработки, а также составление плана работ;
- *проектирование структуры системы* – определение состава необходимого оборудования, программного обеспечения и операций, выполняемых обслуживающим персоналом;
- *анализ требований к программному обеспечению* – определение функциональных возможностей, включая характеристики производительности, среды функционирования компонентов, внешних интерфейсов, спецификаций надежности и безопасности, эргономических требований, требований к используемым данным, установке, приемке, пользовательской документации, эксплуатации и сопровождению;
- *детальное проектирование программного обеспечения* – подробное описание компонентов программного обеспечения и интерфейсов между ними, обновление пользовательской документации, разработка и документирование требований к тестам и плана тестирования компонентов программного обеспечения, обновление плана интеграции компонентов;
- *кодирование и тестирование программного обеспечения* – разработку и документирование каждого компонента, а также совокупности тестовых процедур и данных для их тестирования, тестирование компонентов, обновление пользовательской документации, обновление плана интеграции программного обеспечения;
- *интеграцию программного обеспечения* – сборку программных компонентов в соответствии с планом интеграции и тестирование программного обеспечения на соответствие квалификационным требованиям, представляющим собой набор критериев или условий, которые необходимо выполнить, чтобы квалифицировать программный продукт, как соответствующий своим спецификациям и готовый к использованию в заданных условиях эксплуатации;
- *квалификационное тестирование программного обеспечения* – тестирование программного обеспечения в присутствии заказчика для демонстрации его соответствия требованиям и готовности к эксплуатации; при этом проверяется также готовность и полнота технической и пользовательской документации;

- *интеграцию системы* – сборку всех компонентов системы, включая программное обеспечение и оборудование;
- *квалификационное тестирование системы* – тестирование системы на соответствие требованиям к ней и проверка оформления и полноты документации;
- *установку программного обеспечения* – установку программного обеспечения на оборудовании заказчика и проверку его работоспособности;
- *приемку программного обеспечения* – оценку результатов квалификационного тестирования программного обеспечения и системы в целом и документирование результатов оценки совместно с заказчиком, окончательную передачу программного обеспечения заказчику.

Стадия *определения требований* к ПО – одна из важнейших и определяет в значительной степени успех всего проекта. Она включает следующие этапы:

- Планирование работ, предваряющее работы над проектом. Определение целей разработки, предварительная экономическая оценка проекта, построение плана-графика выполнения работ, создание и обучение рабочей группы. Иногда вводят дополнительно начальную стадию – анализ осуществимости системы.
- Проведение обследования объекта разработки, в рамках которого осуществляются предварительное выявление требований к будущей системе, определение структуры, определение перечня целевых функций, анализ распределения функций по подразделениям и сотрудникам, выявление функциональных взаимодействий между подразделениями, информационных потоков внутри подразделений и между ними, внешних по отношению к организации объектов и внешних информационных воздействий, анализ существующих средств автоматизации деятельности.

Техническое задание представляет собой документ, в котором сформулированы основные цели разработки, требования к программному продукту, определены сроки и этапы разработки и регламентирован процесс приемо-сдаточных испытаний. В разработке технического задания участвуют как представители заказчика, так и представители исполнителя. В основе этого документа лежат исходные требования заказчика, анализ передовых достижений техники, результаты выполнения научно-исследовательских работ, предпроектных исследований, научного прогнозирования и т.п. В зависимости от сложности разрабатываемого ПО, *разработка технического задания* может быть как этапом разработки, так и четкими условиями заключенного договора.

2. Порядок разработки технического задания

Разработка технического задания выполняется в следующей последовательности.

Прежде всего, проводят анализ требований к ПО. *Требование* — это любое условие, которому должна соответствовать разрабатываемая система или программное средство. *Требованием* может быть возможность, которой система должна обладать и ограничение, которому система должна удовлетворять.

В соответствии с Глоссарием терминов программной инженерии IEEE ITILv3 (IT Infrastructure Library) [43], являющимся общепринятым международным стандартным глоссарием, *требование* это:

1. Условия или возможности, необходимые пользователю для решения проблем или достижения целей;
2. Условия или возможности, которыми должна обладать система или системные компоненты, чтобы выполнить контракт или удовлетворять стандартам, спецификациям или другим формальным документам;

3. Документированное представление условий или возможностей для пунктов 1 и 2.

В соответствии со стандартом разработки *требований* ISO/IEC 29148, *требование* – это утверждение, которое идентифицирует эксплуатационные, функциональные параметры, характеристики или ограничения проектирования продукта или процесса, которое однозначно, проверямо и измеримо. *Требования* необходимы для приемки продукта или процесса (потребителем или внутренним руководящим принципом обеспечения качества).

Глоссарий ITILv3 определяет такое понятие, как *набор требований* – документ, содержащий все *требования* к продукту, а также к новой или измененной ИТ-услуге.

Требование должно обладать следующими характеристиками:

1. *Единичность* — требование описывает одну и только одну вещь.
2. *Завершенность* — требование полностью определено в одном месте и вся необходимая информация присутствует.
3. *Последовательность* — требование не противоречит другим требованиям и полностью соответствует документации.
4. *Атомарность* — требование нельзя разделить на более мелкие.
5. *Отслеживаемость* — требование полностью или частично соответствует деловым нуждам как заявлено заинтересованными лицами и задокументировано.
6. *Актуальность* — требование не стало устаревшим с течением времени.
7. *Выполнимость* — требование может быть реализовано в рамках проекта.
8. *Недвусмысленность* — требование определено без обращения к техническому жаргону, акронимам и другим скрытым формулировкам. Оно выражает объекты и факты, а не субъективные мнения. Возможна одна и только одна его интерпретация. Определение не содержит нечетких фраз, использование отрицательных и составных утверждений запрещено.
9. *Обязательность* — требование представляет собой определенную заинтересованым лицом характеристику, отсутствие которой ведет к неполноценности решения, которая не может быть проигнорирована. Необязательное требование — противоречие самому понятию требования.
10. *Проверяемость* — реализованность требования может быть проверена.

В соответствии с ITILv3 все требования в проекте можно разделить на следующие группы:

1. *Функциональные* — реализуют саму бизнес-функцию.
2. *Управленческие* — требования к доступным и безопасным сервисам; относятся к размещению системы, администрированию и безопасности.
3. *Эргономические* — к удобству работы конечных пользователей.
4. *Архитектурные* — требования к архитектуре системы.
5. *Взаимодействия* — к взаимосвязям между существующими приложениями и программными средствами и новым приложением.
6. *Сервисного уровня* — описывают поведение сервиса, качество его выходных данных и другие качественные аспекты, измеряемые заказчиком.

Важно:

Спецификация требований не содержит деталей дизайна или реализации (кроме известных ограничений), данных о планировании проекта или сведений о тестировании. Это относится к требованиям к проекту, а не к продукту и формируются на следующих стадиях разработки ПО.

Далее устанавливают набор выполняемых функций, а также перечень и характеристики исходных данных. Затем определяют перечень результатов, их характеристики и способы представления.

Далее уточняют среду функционирования программного обеспечения: конкретную комплектацию и параметры технических средств, версию используемой операционной системы и, возможно, версии и параметры другого установленного программного обеспечения, с которым предстоит взаимодействовать будущему программному продукту.

В случаях, когда разрабатываемое программное обеспечение собирает и хранит некоторую информацию или включается в управление каким-либо техническим процессом, необходимо также четко регламентировать действия программы в случае сбоев оборудования и энергоснабжения.

Общие положения по оформлению ТЗ

Техническое задание в рамках лабораторной работы оформляют в соответствии с ГОСТ 19.106-78 на листах формата А4.

Титульный лист оформляют в соответствии с ГОСТ 19.104-78. Для внесения изменений и дополнений в техническое задание на последующих стадиях разработки программы или программного изделия выпускают дополнение к нему. Согласование и утверждение дополнения к техническому заданию проводят в том же порядке, который установлен для технического задания.

Техническое задание должно содержать следующие разделы:

- введение;
- наименование и область применения;
- основание для разработки;
- назначение разработки;
- технические требования к программе или программному изделию;
- технико-экономические показатели;
- стадии и этапы разработки;
- порядок контроля и приёмки;
- приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них. При необходимости допускается в техническое задание включать приложения.

Содержание разделов

Введение должно включать краткую характеристику области применения программы или программного продукта, а также объекта (например, системы) в котором предполагается их использовать. Основное назначение введения – продемонстрировать актуальность данной разработки и показать, какое место эта разработка занимает в ряду подобных.

В разделе "Наименование и область применения" указывают наименование, краткую характеристику области применения программы или программного изделия и объекта, в котором используют программу или программное изделие.

В разделе "Основание для разработки" должны быть указаны:

- документ (документы), на основании которых ведется разработка. Таким документом может служить план, приказ, договор и т. п.;
- организация, утвердившая этот документ, и дата его утверждения;

- наименование и (или) условное обозначение темы разработки.

В разделе "Назначение разработки" должно быть указано функциональное и эксплуатационное назначение программы или программного изделия.

Раздел "Технические требования к программе или программному изделию" должен содержать следующие подразделы:

- требования к функциональным характеристикам;
- требования к надёжности;
- условия эксплуатации;
- требования к составу и параметрам технических средств;
- требования к информационной и программной совместимости;
- требования к маркировке и упаковке;
- требования к транспортированию и хранению;
- специальные требования.

В подразделе "Требования к функциональным характеристикам" должны быть указаны требования к составу выполняемых функций, организации входных и выходных данных, временным характеристикам и т.п.

В подразделе "Требования к надёжности" должны быть указаны требования к обеспечению надёжного функционирования (обеспечение устойчивого функционирования, контроль входной и выходной информации, время восстановления после отказа и т.п.)

В подразделе "Условия эксплуатации" должны быть указаны условия эксплуатации (температура окружающего воздуха, относительная влажность и т.п. для выбранных типов носителей данных), при которых должны обеспечиваться заданные характеристики, а также вид обслуживания, необходимое количество и квалификация персонала.

В подразделе "Требования к составу и параметрам технических средств" указывают необходимый состав технических средств с указанием их технических характеристик.

В подразделе "Требования к информационной и программной совместимости" должны быть указаны требования к информационным структурам на входе и выходе и методам решения, исходным кодам, языкам программирования. При необходимости должна обеспечиваться защита информации и программ.

В подразделе "Требования к маркировке и упаковке" в общем случае указывают требования к маркировке программного изделия, варианты и способы упаковки.

В подразделе "требования к транспортированию и хранению" должны быть указаны для программного изделия условия транспортирования, места хранения, условия хранения, условия складирования, сроки хранения в различных условиях.

В разделе "Технико-экономические показатели" должны быть указаны: ориентировочная экономическая эффективность, предполагаемая годовая потребность, экономические преимущества разработки по сравнению с лучшими отечественными и зарубежными образцами или аналогами.

В разделе "Стадии и этапы разработки" устанавливают необходимые стадии разработки, этапы и содержание работ (перечень программных документов, которые должны быть разработаны, согласованы и утверждены), а так же, как правило, сроки разработки и определяют исполнителей.

В разделе "Порядок контроля и приёмки" должны быть указаны виды испытаний и общие требования к приёмке работы.

В приложениях к техническому заданию, при необходимости, приводят:

- перечень научно-исследовательских и других работ, обосновывающих разработку;

- схемы алгоритмов, таблицы, описания, обоснования, расчёты и другие документы, которые могут быть использованы при разработке;
- другие источники разработки.

В случаях, если какие-либо требования, предусмотренные техническим заданием, заказчик не предъявляет, следует в соответствующем месте указать «Требования не предъявляются».

3. Порядок выполнения работы.

1. Изучить теоретические сведения, приведенные в описании лабораторной работы.
2. Разработать техническое задание на программный продукт (см. варианты заданий в приложении 1).
3. Оформить работу в соответствии с ГОСТ 19.106-78. При оформлении использовать MS Office. Примеры оформления ТЗ приведены в приложении 2 и 3.
4. Сдать и защитить работу.

Отчет по лабораторной работе должен включать в себя:

1. Постановку задачи.
2. Техническое задание на программный продукт.

Защита отчета по лабораторной работе заключается в предъявлении преподавателю полученных результатов (на экране монитора), демонстрации полученных навыков и ответах на вопросы преподавателя.

Контрольные вопросы

1. Какие процедуры выполняются при заключении договора.
2. Этапы разработки программного обеспечения.
3. Постановка задачи и предпроектные исследования.
4. Функциональные и эксплуатационные требования к программному продукту.
5. Правила разработки технического задания.
6. Основные разделы технического задания.

ПРИЛОЖЕНИЕ 1.

ВАРИАНТЫ ЗАДАНИЙ НА ЛАБОРАТОРНУЮ РАБОТУ

Для отработки навыков коллективной разработки ПО лабораторной работы выполняются группами студентов от 2 до 5 человек. Лабораторные работы №№ 1-8 выполняются каждой группой для одного и того же варианта.

Примеры, приведенные в перечне ниже, не являются обязательными. Фантазия студентов в выборе темы приветствуется и ничем, кроме возможности реализации, не ограничивается.

1. Разработать программный модуль «Система контроля ж/д переезда». Программный модуль предназначен для управления движением автомобилей через железнодорожный переезд.
2. Разработать программный модуль «Касса в магазине». Программный модуль предназначен для контроля штрих-кодов товаров, учета расходования товара и коррекции склада.
3. Разработать программный модуль «Система контроля взлета-посадки». Модуль должен отслеживать текущую обстановку на территории аэродрома и учитывать воздушную обстановку.
4. Разработать программный модуль «Электронный секретарь». Программа предназначена для записи мероприятий, предупреждения в реальном времени о плановых мероприятиях, отслеживания юбилеев коллег и партнеров.
5. Разработать приложение для поликлиник «Электронная запись к врачу». Приложение предназначено для больных, желающих попасть на прием как в день обращения, так и на месяц вперед.
6. Разработать программный модуль «Система заказа фотографий», содержащий возможности заказать фотографии любого размера на разных видах фотобумаги с различных носителей информации.
7. Разработать программный модуль «Умный дом». Модуль предназначен для удаленного управления загородным домом.
8. Разработать программный модуль «Бронирование авиабилетов в турфирмах». Необходимо обеспечить возможность бронирования билетов как прямых рейсов, так и перелетов с учетом стыковки рейсов разных авиакомпаний.
9. Разработать программный модуль «Автоматизированная система поздравления друзей и коллег». Должен учитываться пол, возраст и увлечения. Поздравления не должны быть одинаковыми и не могут повторяться для людей знакомых друг с другом.

Примечание: При разработке программы не ограничивайтесь функциями, приведенными в варианте. Подойдите к задаче творчески – постарайтесь предусмотреть расширение возможностей проектируемой программы. Обязательно использование структурного и модульного подхода к программированию. Желательно использование объектного подхода.

ПРИЛОЖЕНИЕ 2.

**ПРИМЕР РАЗРАБОТКИ ТЕХНИЧЕСКОГО ЗАДАНИЯ
НА ПРОГРАММНЫЙ ПРОДУКТ**

Министерство образования Российской Федерации

Московский государственный институт электронной техники

(технический университет)

Кафедра Информатики и программного обеспечения вычислительных
систем

УТВЕРЖДАЮ

Зав. Кафедрой ИПОВС,

д.т.н., проф. _____ Гагарина Л.Г.

«__» _____ 2015 г.

ПРОГРАММА СОРТИРОВКИ ОДНОМЕРНОГО МАССИВА

Техническое задание на лабораторную работу

Листов 3

Руководитель, к.т.н., доцент _____ Петров А.А.

Исполнитель, студент гр. МП 33 _____ Власов С.Е.

МОСКВА, 2015

*Рисунок 2.1. Пример оформления титульного листа технического задания на учебный
программный продукт*

1. Введение

Настоящее техническое задание распространяется на разработку программы сортировки одномерного массива методами пузырька, прямого выбора, Шелла и быстрой сортировки, предназначеннной для использования школьниками старших классов при изучении курса школьной информатики.

2. Основание для разработки

- 2.1. Программа разрабатывается на основе учебного плана кафедры «Информатики и программного обеспечения вычислительных систем»
- 2.2. Наименование работы
«Программа сортировки одномерного массива»
- 2.3. Исполнитель: компания BestSoft.
- 2.4. Соисполнители: нет.

3. Назначение

Программа предназначена для использования школьниками при изучении темы «Обработка одномерных массивов» в курсе «Информатика».

4. Требования к программе или программному изделию

- 4.1. Требования к функциональным характеристикам
 - 4.1.1. Программа должна обеспечивать возможность выполнения следующих функций:
 - ввод размера массива и самого массива;
 - хранение массива в памяти;
 - выбор метода сортировки;
 - вывод текстового описания метода сортировки;
 - вывод результата сортировки.
 - 4.1.2. Исходные данные:
 - размер массива, заданный целым числом;
 - массив;
 - 4.1.3. Организация входных и выходных данных
Входные данные поступают с клавиатуры.
Выходные данные отображаются на экране и при необходимости выводятся на печать.
- 4.2. Требования к надежности
Предусмотреть контроль вводимой информации.
Предусмотреть блокировку некорректных действий пользователя при работе с системой.
- 4.3. Требования к составу и параметрам технических средств
Система должна работать на IBM совместимых персональных компьютерах.
Минимальная конфигурация:
 - тип процессора – Pentium и выше;
 - объем оперативного запоминающего устройства – 32 Мб и более;
 - объем свободного места на жестком диске – 40 Мб.
Рекомендуемая конфигурация:
 - тип процессора – Intel Core 5;
 - объем оперативного запоминающего устройства – 512 Мб;
 - объем свободного места на жестком диске – 600 Мб.

4.4. Требования к программной совместимости

Программа должна работать под управлением семейства операционных систем Win 32 (Windows 95/98/2000/ME/XP и т. п.).

5. Требования к программной документации

- 5.1. Разрабатываемые программные модули должны быть самодокументированы, т. е. тексты программ должны содержать все необходимые комментарии.
- 5.2. Разрабатываемая программа должна включать справочную информацию о работе программы, описания методов сортировки и подсказки учащимся.
- 5.3. В состав сопровождающей документации должны входить:
 - 5.3.1. Пояснительная записка на 5 листах, содержащая описание разработки.
 - 5.3.2. Руководство пользователя.

ПРИЛОЖЕНИЕ 3.

ПРИМЕР ТЕХНИЧЕСКОГО ЗАДАНИЯ НА РАЗРАБОТКУ

«Утверждаю»

Профессор кафедры ВС

_____ (Иванов И.И.)

«___» _____ 2015 г.

Техническое задание
на разработку «Модуля автоматизированной
системы оперативно-диспетчерского управления
теплоснабжением корпусов МИЭТ»

Москва, 2015

1. Введение

Работа выполняется в рамках проекта «Автоматизированная система оперативно-диспетчерского управления электро-, теплоснабжением корпусов МИЭТ»

2. Основание для разработки

- 2.1. Основанием для данной работы служит договор № 1234 от 10 марта 2014 г.
- 2.2. Наименование работы «Модуль автоматизированной системы оперативно-диспетчерского управления теплоснабжением корпусов МИЭТ».
- 2.3. Исполнители: ОАО “Лаборатория создания программного обеспечения”
- 2.4. Соисполнители: нет.

3. Назначение разработки

Создание модуля для контроля и оперативной корректировки состояния основных параметров теплообеспечения корпусов МИЭТ.

4. Технические требования

4.1. Требования к функциональным характеристикам

4.1.1. Состав выполняемых функций

Разрабатываемое ПО должно обеспечивать:

- сбор и анализ информации о расходовании тепла, горячей и холодной воды по данным теплосчетчиков SA-94 на всех тепловых выходах;
- сбор и анализ информации с устройств управления системами воздушного отопления и кондиционирования типа РТ1 и РТ2 (разработки кафедры СММЭ и ТЦ);
- предварительный анализ информации на предмет нахождения параметров в допустимых пределах и сигнализирование при выходе параметров за пределы допуска;
- выдачу рекомендаций по дальнейшей работе;
- отображение текущего состояния по набору параметров -циклически постоянно (режим работы круглосуточный), при сохранении периодичности контроля прочих параметров;
- визуализацию информации по расходу теплоносителя:
- текущую, аналогично показаниям счетчиков;
- с накоплением за прошедшие сутки, неделю, месяц – в виде почасового графика для информации за сутки и неделю;
- суточный расход – для информации за месяц.

Для устройств управления приточной вентиляцией текущая информация должна содержать номер приточной системы и все параметры, выдаваемые на собственный индикатор.

По отдельному запросу осуществляются внутренние настройки;

В конце отчетного периода система должна архивировать данные.

4.1.2. Организация входных и выходных данных

Исходные данные в систему поступают в виде значений с датчиков, установленных в помещениях института. Эти значения отображаются на компьютере диспетчера. После анализа поступивший информации оператор диспетчерского пункта устанавливает необходимые параметры для устройств, регулирующих отопление и

вентиляцию в помещениях. Возможно также автоматическая установка некоторых параметров для устройств регулирования.

Основной режим использования системы – ежедневная работа.

4.2. Требования к надежности

Для обеспечения надежности необходимо проверять корректность получаемых данных с датчиков.

4.3. Условия эксплуатации и требования к составу и параметрам технических средств

Для работы системы должен быть выделен ответственный оператор.

Требования к составу и параметрам технических средств уточняются на этапе эскизного проектирования системы.

4.4. Требования к информационной и программной совместимости

Программа должна работать на платформах Windows 98/NT/2000

4.5. Требования к транспортировке и хранению

Программа поставляется на лазерном носителе информации. Программная документация поставляется в электронном и печатном виде.

4.6. Специальные требования

- программное обеспечение должно иметь дружественный интерфейс, рассчитанный на пользователя (в плане компьютерной грамотности) квалификации;
- ввиду объемности проекта, задачи предполагается решать поэтапно, при этом модули ПО, созданные в разное время должны предполагать возможность наращивания системы и быть совместимы друг с другом, поэтому документация на принятое эксплуатационное ПО должна содержать полную информацию, необходимую для работы программистов с ним;
- язык программирования – по выбору исполнителя, должен обеспечивать возможность интеграции программного обеспечения с некоторыми видами периферийного оборудования (например счетчик SA-94 и т.п.)

5. Требования к программной документации

Основными документами, регламентирующими разработку будущих программ, должны быть документы Единой Системы Программной Документации (ЕСПД): Руководство пользователя, руководство администратора, описание применения.

6. Технико-экономические показатели

Эффективность системы определяется удобством использования системы для контроля и управления основными параметрами теплообеспечения помещений МИЭТ, а также экономической выгодой, полученной от внедрения аппаратно-программного комплекса.

7. Порядок контроля и приемки

После передачи Исполнителем отдельного функционального модуля программы Заказчику, последний имеет право тестировать модуль в течении 7 дней. После

тестирования Заказчик должен принять работу по данному этапу или в письменном виде изложить причину отказа принятия. В случае обоснованного отказа Исполнитель обязуется доработать модуль.

8. Календарный план работ

№ этапа	Название этапа	Сроки этапа	Чем заканчивается этап
1	Изучение предметной области. Проектирование системы. Разработка предложений по реализации системы.	01.02.200_ - 28.02.200_	Предложения по работе системы. Акт-сдачи приемки.
2	Разработка программного модуля по сбору и анализу информации со счетчиков и устройств управления. Внедрение системы для одного из корпусов МИЭТ.	01.03.200_ - 31.08.200_	Программный комплекс решающий поставленные задачи для пилотного корпуса МИЭТ. Акт сдачи-приемки
3	Тестирование и отладка модуля. Внедрение системы во всех корпусах МИЭТ	01.09.200_ - 30.12.200_	Готовая система контроля теплообеспечения МИЭТ, установленная в диспетчерском пункте. Программная документация. Акт сдачи-приемки работ

Руководитель работ

Сидоров С.В.

ПРИЛОЖЕНИЕ 4.

ПРИМЕР ТИПОВОГО ДОГОВОРА НА РАЗРАБОТКУ ПО

ДОГОВОР № _____
на разработку программного обеспечения

г. Москва

" ____ " 20 ____ г.

Заказчик, в лице _____ (полное и сокращённое наименование заказчика), именуемый в дальнейшем действующего на основании _____ (наименование и реквизиты документа, на основании которого действует представитель) _____, с одной стороны, и _____ (полное и сокращённое наименование контрагента) _____, именуем _____ в дальнейшем Исполнитель, в лице _____ (указать должность, фамилию, имя, отчество представителя) _____, действующего на основании _____ (указать наименование и реквизиты документа, на основании которого действует представитель) _____, с другой стороны, совместно именуемые далее Стороны, а каждая в отдельности Сторона, заключили настоящий договор (далее – Договор) о нижеследующем.

1. ПРЕДМЕТ ДОГОВОРА

1.1. Исполнитель обязуется выполнить работы по разработке _____[1] программного обеспечения _____[2] (далее ПО), в том числе его компонент (_____)[3], а также оказать услуги по гарантийной технической поддержке разработанного по настоящему Договору ПО. С момента передачи Исполнителем Заказчику результатов работ по разработке ПО в порядке, указанном в п.3.1. Договора, исключительное право на ПО в полном объеме без ограничений принадлежит Заказчику. Требования к ПО, а также представляемые Исполнителем результаты работ по разработке ПО, определяются в Техническом задании (Приложение) (далее – ТЗ), которое подписывается Сторонами и является неотъемлемой частью Договора. Результаты работ Исполнитель передает на материальном носителе (_____[4]).

1.2. Перечень и сроки выполнения работ определяются в Плане-графике выполнения работ (Приложение), который подписывается Сторонами и является неотъемлемой частью Договора.

1.3. Заказчик обязуется принять и оплатить работы, выполненные Исполнителем по Договору.

2. СТОИМОСТЬ И ПОРЯДОК РАСЧЕТОВ

2.1. Стоимость работ по разработке ПО составляет _____ (____) ___, кроме того НДС (18%) – _____ (____) ___, итого с учетом НДС – _____ (____) ___. В стоимость работ по Договору включена стоимость материальных носителей на которых передаются результаты работ.

2.2. Заказчик производит оплату работ по разработке ПО в соответствии с Графиком платежей (Приложение к Договору) на основании счетов, представленных Исполнителем.

2.3. Оплата работ производится в российских рублях путем перечисления Заказчиком денежных средств на расчетный счет Исполнителя, указанный в Статье 15 Договора. Датой исполнения обязательств Заказчика по платежам считается дата списания денежных средств со счета Заказчика.

2.4. В платежно-расчетных документах НДС выделяется отдельной строкой.

2.5. Исполнитель предоставляет Заказчику счета-фактуры в порядке и сроки, установленные законодательством Российской Федерации.

3. ПОРЯДОК СДАЧИ-ПРИЕМКИ РАБОТ

3.1. По завершении выполнения работ Исполнитель уведомляет об этом Заказчика и передает Заказчику по Акту сдачи-приемки работ (в _____ (____)[5] экземплярах) (форма Акта сдачи-приемки работ приведена в Приложении № 4 к Договору) полный комплект программ с технической документацией, дистрибутивов, других материалов и документов, предусмотренных ТЗ, на материальных носителях и счет на оплату. Заказчик обязан провести проверку, принять выполненные работы и подписать все экземпляры Акта сдачи-приемки работ, _____ (____)[6] из которых направить Исполнителю в течение _____ (____)[7] рабочих дней с даты получения, либо в этот же срок направить Исполнителю мотивированный отказ от подписания Акта.

3.2. При наличии мотивированного отказа Заказчика от подписания Акта Стороны в течение _____ (____)[8] рабочих дней с даты получения Исполнителем мотивированного отказа согласовывают Протокол несоответствий по форме, предусмотренной Приложением № 6 к Договору, в котором

указываются также способы и сроки устранения замечаний. Выявленные несоответствия по согласованному Протоколу несоответствий устраняются Исполнителем в срок, предусмотренный в Протоколе несоответствий, без увеличения объема работ и без увеличения общей стоимости работ, указанной в п.2.1. Договора. После принятия Исполнителем мер для устранения замечаний процедура подписания Акта сдачи-приемки работ повторяется.

3.3. Исполнитель вправе по согласованию с Заказчиком досрочно выполнить работы.

4. ВНЕСЕНИЕ ИЗМЕНЕНИЙ

4.1. Любая из Сторон может потребовать внесения изменений в Договор, в том числе по условиям, срокам, составу и стоимости работ.

4.2. Требование Исполнителя о внесении изменений не является обязательным для Заказчика, если Исполнитель не докажет, что данное требование связано с технической невозможностью выполнить работы без изменения ТЗ.

4.3. Требование Заказчика о внесении изменений является обязательным для Исполнителя. Исполнитель в течение _____ (____)[9] рабочих дней со дня получения соответствующего требования направляет Заказчику смету по внесению соответствующих изменений с указанием стоимости работ и сроков их выполнения. Исполнитель может отказаться от исполнения требования Заказчика о внесении изменений только в случае технической невозможности их реализации.

4.4. Внесение изменений оформляется подписанием Сторонами Дополнительного соглашения (форма Дополнительного соглашения – Приложение к Договору).

5. ОТВЕТСТВЕННОСТЬ СТОРОН

5.1. Исполнитель обязуется выполнять работы, указанные в п.1.1. Договора, самостоятельно. При необходимости, по согласованию с Заказчиком, Исполнитель вправе привлекать к исполнению Договора третьи лица, без дополнительной оплаты Заказчиком, неся за них полную ответственность, в т. ч. по конфиденциальности предоставляемой информации, за качество выполнения работ, а также за убытки в порядке, предусмотренном в п. 5.13. Договора. Допуск представителей третьих лиц осуществляется в порядке, предусмотренном в п.5.12 Договора.

5.2. За неисполнение или ненадлежащее исполнение обязательств по Договору Стороны несут ответственность в соответствии с законодательством Российской Федерации.

5.3. В каждом случае нарушения любого из: сроков выполнения работ, установленных в Приложении № 2 к Договору, сроков устранения выявленных несоответствий, установленных в Протоколе несоответствий, сроков внесения изменений в соответствии с п.4.3 Договора, Исполнитель выплачивает Заказчику неустойку в размере _____ (____)[10] %, включая НДС, от общей стоимости работ, указанной в п.2.1. Договора, за каждый календарный день просрочки, но не более _____ (____)[11] % от этой суммы за каждый случай.

5.4. В случае нарушения сроков оплаты, установленных в Приложении № 3 к Договору, Заказчик уплачивает Исполнителю неустойку в размере _____ (____)[12]%, включая НДС, от суммы просроченного платежа за каждый календарный день просрочки, но не более _____ (____)[13] %, включая НДС, от этой суммы.

5.5. Обязательство Стороны по выплате неустойки возникает у нарушившей Стороны после получения ею письменного требования об уплате неустойки от другой Стороны.

5.6. Исполнитель не несёт ответственности за нарушение сроков выполнения работ, предусмотренных Договором, произошедших по вине Заказчика в следующих случаях:

5.6.1. несвоевременное выполнение согласованных Сторонами работ, возложенных на Заказчика и/или привлеченных им третьих лиц;

5.6.2. несвоевременное предоставление запрашиваемой Исполнителем в ходе выполнения работ информации, предоставление которой предусмотрено Договором;

5.6.3 несвоевременное оказание содействия Исполнителю, предусмотренного Договором (включая, предоставление технических ресурсов, рабочих мест, удовлетворяющих требованиям Исполнителя).

5.7. Исполнитель вправе не приступать к работе, а начатую работу приостановить в случаях нарушения Заказчиком своих обязательств по Договору, которые повлекли для Исполнителя невозможность выполнения работы, либо в случаях, предусмотренных п. п. 5.6 Договора. О приостановлении работ Исполнитель обязан незамедлительно письменно уведомить Заказчика, указав причины, по которым работы приостановлены и необходимые действия Заказчика по их устранению. Об устранении причин приостановления работ Заказчик обязуется незамедлительно письменно уведомить Исполнителя.

5.8. В каждом случае нарушения сроков исправления дефектов, классифицируемых как критические (в соответствии с Приложением № 5 к Договору), установленных в Приложении № 5 к

Договору, Исполнитель выплачивает Заказчику неустойку в размере _____ (_____) [14] %, включая НДС, от общей стоимости работ по Договору, за каждый _____ [15] просрочки, но не более _____ (_____) [16] % от этой суммы за каждый случай.

5.9. В каждом случае нарушения сроков исправления дефектов, классифицируемых как важные (в соответствии с Приложением № 5 к Договору), установленных в Приложении № 5 к Договору, Исполнитель выплачивает Заказчику неустойку в размере _____ (_____) [17] %, включая НДС, от общей стоимости работ по Договору, за каждый _____ [18] просрочки, но не более _____ (_____) [19] % от этой суммы за каждый случай.

5.10. В каждом случае нарушения сроков исправления дефектов, классифицируемых как средние (в соответствии с Приложением № 5 к Договору), установленных в Приложении № 5 к Договору, Исполнитель выплачивает Заказчику неустойку в размере _____ (_____) [20] %, включая НДС, от общей стоимости работ по Договору, за каждый _____ [21] просрочки, но не более _____ (_____) [22] % от этой суммы за каждый случай.

5.11. В каждом случае нарушения сроков исправления дефектов, классифицируемых как незначительных (в соответствии с Приложением № 5 к Договору), установленных в Приложении № 5 к Договору, Исполнитель выплачивает Заказчику неустойку в размере _____ (_____) [23] %, включая НДС, от общей стоимости работ по Договору, за каждый _____ [24] просрочки, но не более _____ (_____) [25] % от этой суммы за каждый случай.

5.12. Исполнитель несет ответственность за действия своих работников, выполняющих работы в помещениях Заказчика. Допуск работников Исполнителя к выполнению работ в рамках Договора в помещения Заказчика производится после их ознакомления уполномоченными лицами Заказчика с нормативными документами Заказчика по обеспечению информационной безопасности. После ознакомления с вышеуказанными документами, работники Исполнителя подписывают обязательство о выполнении требований этих документов.

5.13. В случае причинения Заказчику убытков в результате нарушения работниками Исполнителя требований нормативных документов Заказчика по обеспечению информационной безопасности в ходе выполнения работ в помещениях Заказчика, Исполнитель обязан полностью возместить Заказчику причиненные ему убытки.

6. ГАРАНТИЙНЫЕ ОБЯЗАТЕЛЬСТВА

6.1. Исполнителем производится гарантийное обслуживание ПО сроком _____ (_____) [26] со дня подписания Сторонами Акта сдачи-приемки работ.

6.2. В случае обнаружения дефектов в разработанном ПО в течение гарантийного срока, в соответствии с Классификацией дефектов (Приложение к Договору) (не включая дефекты в базовом программном обеспечении и аппаратных компонентах, принадлежащих Заказчику), Исполнитель обязуется исправлять их без дополнительной оплаты со стороны Заказчика, в сроки, указанные в Приложении № 5 к Договору. Срок исправления любого из дефектов отсчитывается от даты получения Исполнителем письменного уведомления от Заказчика об обнаруженном дефекте. Гарантия предоставляется на программное обеспечение и полнофункциональные скомпилированные исполняемые модули программ, которые были переданы Исполнителем Заказчику в рамках Договора и оформлены Актом сдачи-приемки работ. Если изменения в исходные тексты (коды) программ или модификации исполняемых модулей были внесены вне рамок Договора, либо эксплуатация ПО производится Заказчиком на программно-аппаратной платформе, не соответствующей ТЗ, либо неработоспособность ПО вызвана внесением Заказчиком изменений в базовое программное обеспечение, влияющее на работу ПО, то гарантия на такую версию ПО не распространяется.

6.3. Если в период гарантийного срока обнаружатся ошибки и дефекты ПО, которые не позволяют продолжить нормальную эксплуатацию ПО до их устранения, то гарантийный срок продлевается на период устранения таких ошибок и дефектов. Устранение ошибок и дефектов осуществляется Исполнителем своими силами и без дополнительной оплаты со стороны Заказчика.

6.4. Исполнитель гарантирует отсутствие в разработанном ПО скрытых (недокументированных) функциональных возможностей, ведущих к финансовому ущербу для Заказчика.

6.5. Техническая поддержка ПО по истечении гарантийного срока, указанного в п. 6.1. Договора, осуществляется Исполнителем на основании отдельно заключаемых Сторонами договоров, в которых определяются стоимость и порядок ее осуществления. В любом случае стоимость ежегодной технической поддержки не может превышать _____ (_____) [27] %, включая НДС, от стоимости разработанного ПО.

6.6. Если Заказчик сочтет необходимым, то Исполнитель соглашается с тем, что на полученное Заказчиком от Исполнителя ПО распространяются требования нормативного документа Заказчика, определяющего порядок проведения контрольной компиляции с исходных текстов ПО. После

ознакомления уполномоченного представителя Исполнителя с данным нормативным документом, он подписывает обязательство от лица Исполнителя о выполнении требований этого документа.

7. КОНФИДЕНЦИАЛЬНОСТЬ

7.1. По взаимному согласию Сторон в рамках Договора конфиденциальной признается любая информация, касающаяся предмета Договора, хода его выполнения и полученных результатов. Каждая из Сторон обеспечивает защиту конфиденциальной информации, ставшей доступной ей в рамках Договора, от несанкционированного использования, распространения или публикации. Такая информация не будет передаваться третьим сторонам без письменного разрешения другой Стороны и использоваться в иных целях, кроме выполнения обязательств по Договору.

7.2. Любой ущерб, вызванный нарушением условий конфиденциальности, определяется и возмещается в соответствии с законодательством Российской Федерации.

7.3. Обязательства Сторон по защите конфиденциальной информации распространяются на все время действия Договора, а также в течение ____ (____) [28] после прекращения действия Договора.

7.4. Не является нарушением режима конфиденциальности предоставление Сторонами информации по запросу уполномоченных государственных органов в соответствии с законодательством Российской Федерации.

8. ДЕЙСТВИЕ ДОГОВОРА

8.1. Договор вступает в силу с момента подписания его Сторонами и действует до полного и надлежащего исполнения обязательств по нему.

8.2. Договор может быть расторгнут по соглашению Сторон. В любом случае досрочного расторжения Договора (за исключением случаев, указанных в п. п. 8.3. и 8.4. Договора) Стороны должны произвести между собой **взаиморасчеты** не позднее ____ (____) [29] рабочих дней со дня расторжения, на основании двухстороннего акта, с учетом того, что Заказчику должна быть возвращена вся сумма полученного Исполнителем **аванса**, включая НДС, а Исполнителю оплачен результат фактически выполненной части работ, но только в том случае, если выполненная часть работ может быть использована Заказчиком в дальнейшем и Заказчик согласен принять выполненную часть работ, при этом решение о приемке / не приемке части работ Заказчик принимает исключительно по своему усмотрению, без согласования с Исполнителем.

8.3. Заказчик вправе в любой момент без объяснения причин расторгнуть Договор в одностороннем внесудебном порядке, письменно уведомив об этом Исполнителя не позднее чем за ____ (____) [30] календарных дней до даты расторжения, указанной в уведомлении.

В этом случае Стороны должны произвести между собой взаиморасчеты не позднее ____ (____) [31] рабочих дней со дня расторжения, на основании двухстороннего акта, с учетом того, что Заказчику должна быть возвращена вся сумма полученного Исполнителем аванса, включая НДС, а Исполнителю оплачен результат фактически выполненной части работ.

8.4. В случае расторжения Договора по инициативе Исполнителя (если такая инициатива не была вызвана ненадлежащим исполнением Заказчиком своих обязательств по Договору (в случаях, указанных в п. п. 5.4., 5.6. и 5.7. Договора) или обстоятельствами, предусмотренными Разделом 9 Соглашения) Заказчику в течение ____ (____) рабочих дней со дня расторжения Договора должна быть возвращена вся сумма полученного Исполнителем аванса, включая НДС, и выплачена неустойка, включая НДС, в размере двойной ставки рефинансирования, установленной Банком России и действующей на день выплаты аванса, от суммы аванса за весь период, начиная с даты перечисления денежных средств Заказчиком по дату возврата аванса. Датой возврата аванса является дата зачисления денежных средств на счет Заказчика.

9. ФОРС-МАЖОР

9.1 Любая из Сторон может быть освобождена от ответственности в определенных случаях, которые возникли независимо от ее воли.

9.2 Обстоятельства, вызванные не зависящими от воли Сторон событиями, которых добросовестная Сторона не могла избежать или последствия которых она не могла устранить, считаются случаями, которые освобождают от ответственности, если они произошли после заключения Договора и препятствуют его полному или частичному исполнению.

9.3 Случаями непреодолимой силы считаются следующие события: война, военные действия, массовые беспорядки, забастовки, эпидемии, природные катастрофы, а также акты органов власти, влияющие на выполнение обязательств Сторон, и все другие аналогичные события и обстоятельства.

9.4 Сторона, пострадавшая от действия непреодолимой силы, обязана известить другую Сторону заказным письмом или иным доступным ей способом сразу же после наступления форс-мажорных

обстоятельств и разъяснить, какие меры необходимы для их устранения, но в любом случае не позднее _____ (_____[32]) календарных дней после начала действия непреодолимой силы.

9.5 Несвоевременное уведомление об обстоятельствах непреодолимой силы лишает соответствующую Сторону права на освобождение от ответственности по причине указанных обстоятельств. Обстоятельства непреодолимой силы должны быть подтверждены документально компетентными органами.

9.6 Если указанные обстоятельства продолжаются более _____ (_____[33]) месяцев, каждая Сторона имеет право инициировать досрочное расторжение настоящего Договора. В этом случае, Стороны производят взаиморасчеты на основании двустороннего Акта, подписанного Сторонами.

10. ПРАВА ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

10.1. Исключительное право на результат работ (разработанное ПО) в полном объеме без ограничений принадлежит Заказчику с момента подписания Сторонами Акта сдачи-приемки работ.

10.2. Исполнитель гарантирует, что при разработке ПО по Соглашению не будут нарушены авторские, смежные и любые иные права третьих лиц.

В случае если к Заказчику будут предъявлены претензии (требования, иски) со стороны третьих лиц по поводу нарушения их прав в результате использования Заказчиком полученного от Исполнителя ПО, Исполнитель по получении извещения от Заказчика обязуется выступить на стороне Заказчика, оказать всемерное содействие Заказчику при урегулировании таких претензий, в том числе взять на себя обязанность по подготовке и проведению досудебных переговоров и переписки с такими третьими лицами, а впоследствии (в том случае если Заказчик будет вынужден в силу вступившего в силу решения суда или если по согласованию с Исполнителем будет признано приемлемым возместить ущерб третьих лиц во внесудебном порядке) возместить Заказчику в полном объеме выплаченные Заказчиком третьим лицам денежные средства, все связанные с нарушением прав третьих лиц судебные издержки Заказчика и иные расходы, а также уплатить Заказчику штраф в размере _____ (_____[34]) %, включая НДС, от подлежащей возмещению суммы.

Возмещение и выплата штрафа производится Исполнителем не позднее _____ (_____[35]) рабочих дней со дня получения соответствующего требования от Заказчика.

Если по решению суда Заказчик не может пользоваться ПО или иным результатом услуг, или в случае если Исполнитель и/или Заказчик желает прекратить текущее использование ПО из соображений устранения нарушения прав третьего лица, Исполнитель обязан незамедлительно без дополнительной оплаты со стороны Заказчика заменить программное обеспечение, являющееся предметом претензий третьих лиц, таким образом, чтобы права третьих лиц не нарушались.

11. УВЕДОМЛЕНИЯ

11.1. Все уведомления, извещения и сообщения в связи с выполнением Договора должны быть оформлены в письменном виде на **русском языке** и могут быть направлены с помощью средств факсимильной связи, электронной почтой, заказной или курьерской почтой, с подтверждением факта их получения, по фактическим адресам Сторон, приведенным в Статье 15 Договора, либо по адресу, указанному ниже для соответствующей Стороны, либо по иному адресу, о котором любая из Сторон может уведомить другую Сторону.

11.2. Информация считается полученной Сторонами:

11.2.1 в случае направления с помощью средств факсимильной связи или по электронной почте – в дату, указанную в подтверждении о получении Стороной-получателем факсимильного сообщения или сообщения электронной почты, имеющемся у Стороны-отправителя;

11.2.2 в случае направления заказной или курьерской почтой – на дату, указанную в подтверждении о вручении отправления Стороне-получателю, имеющемуся у Стороны-отправителя.

11.3. Исполнитель обязуется регулярно представлять Заказчику отчет о ходе выполнения работ по Договору в формате ведения учета у Заказчика.

12 ПОРЯДОК РАССМОТРЕНИЯ СПОРОВ

12.1. Все споры между Сторонами, возникшие в ходе исполнения Договора, подлежат рассмотрению в Арбитражном суде г. Москвы.

13 ЗАКЛЮЧИТЕЛЬНЫЕ ПОЛОЖЕНИЯ

13.1. При изменении адреса, реквизитов или уполномоченных (ответственных) лиц Сторон данная Сторона обязуется уведомить об этом другую Сторону незамедлительно, но в любом случае не позднее _____ (_____[38]) календарных дней. До получения Стороной уведомления о таких изменениях исполнение Договора этой Стороной, совершённое с использованием имеющихся у неё сведений, считается надлежащим.

13.2. Договор составлен в ____ (____)[39] экземплярах, которые подписываются обеими Сторонами и имеют одинаковую юридическую силу, ____ (____) экземпляр ____ (____) – для Исполнителя и ____ (____) – для Заказчика.

14 СПИСОК ПРИЛОЖЕНИЙ

- 14.1. Приложение. Техническое задание.
- 14.2. Приложение. План-график выполнения работ.
- 14.3. Приложение. График платежей.
- 14.4. Приложение. Форма Акта сдачи-приемки работ.
- 14.5. Приложение. Классификация дефектов и сроки их устранения.
- 14.6. Приложение. Форма Протокола несоответствий.
- 14.7. Приложение. Форма Дополнительного Соглашения.

15 АДРЕСА, РЕКВИЗИТЫ И ПОДПИСИ СТОРОН

Исполнитель:

Местонахождение: _____
Тел.: : _____, факс: : _____
БИК : _____,
Счет : _____,
Кор/счет : _____
в : _____,
ОКПО : _____, ОКВЭД : _____,
КПП : _____, ИНН : _____,
ОГРН : _____

От Исполнителя:

Должность _____
ФИО _____
подпись
М. П.

Заказчик:

Местонахождение: : _____
Тел.: : _____, факс: : _____
БИК : _____,
Счет : _____,
Кор/счет : _____
в : _____,
ОКПО : _____, ОКВЭД : _____,
КПП : _____, ИНН : _____,
ОГРН : _____

От Заказчика:

Должность _____
ФИО _____
подпись
М. П.

Приложение 5

к Договору № _____
от " ____ " _____ 20 ____ г.

Классификация дефектов и сроки их устраниния

Классификация дефектов

Критические	Определение
K_1	Дефект, вызывающий повреждение или разрушение операционной системы.
K_2	Дефект, вызывающий повреждение структуры базы данных или потерю данных в определенных таблицах.
K_3	Дефект, делающий невозможным дальнейшую работу или запуск программы.
K_4	Дефект, вызывающий зависание программы или компьютера, а также вызывающий критическую ошибку ОС.
K_5	Дефект, после проявления которого, невозможно дальнейшее использование какой-либо функциональности.
K_6	Не реализованная функциональность.
K_7	Дефект, вызывающий нарушение информационной безопасности
Важные	
B_1	Дефект, проявляющийся только после определенной последовательности действий, после проявления которого, затруднено дальнейшее использование какой-либо функциональности.

B_2	Дефект, проявляющийся часто, не имеющий четкой последовательности действий к нему приводящей, не вызывающий эффектов, описанных в K_1 и K_4, после проявления которого, затруднено дальнейшее использование какой-либо функциональности.
B_3	Дефект, вызывающий непредвиденное использование ресурсов, не указанных в Техническом задании.
B_4	Искаженный внешний вид пользовательского интерфейса в Web проектах при использовании версий Web-browser'ов, указанных в Техническом задании.
Средние	
C_1	Появление неправильных сообщений или отсутствие требуемых.
C_2	Искаженный внешний вид пользовательского интерфейса, который затрудняет работу пользователя, но оставляет возможность работы с программой.
C_3	Дефект, проявляющийся редко, не имеющий четкой последовательности действий к нему приводящей, не вызывающий эффектов, описанных в K_1 и K_4.
Незначительные	
H_1	Искаженный внешний вид пользовательского интерфейса, который затрудняет работу пользователя. Для примера – ошибки правописания, неточная прокрутка и т. д. Для Web проектов такие дефекты классифицируются по B_4.
H_2	После деинсталляции программы остаются файлы и записи в реестре или конфигурационных файлах.
H_3	Другие дефекты.

- [1] Указать краткую характеристику программного обеспечения (например: прикладного, системного и т. п.)
- [2] Указать название программного обеспечения
- [3] Перечислить составляющие компонент (например: модули, технические задания, документация и другие артефакты проектирования).
- [4] Указать вид носителя
- [5] Указать цифрами и прописью количество экземпляров
- [6] Указать цифрами и прописью количество возвращаемых экземпляров.
- [7] Указать цифрами и прописью
- [8] Указать цифрами и прописью
- [9] Указать цифрами и прописью
- [10] Указать проценты цифрами и прописью (размер неустойки должен быть экономически обоснован и стимулировать Исполнителя на надлежащее исполнение обязательств по Договору)
- [11] Указать проценты цифрами и прописью, но не менее 10 %
- [12] Указать проценты цифрами и прописью (размер неустойки должен быть экономически обоснован)
- [13] Указать проценты цифрами и прописью, но не более 10 %
- [14] Указать проценты цифрами и прописью (размер неустойки должен быть экономически обоснован и стимулировать Исполнителя на надлежащее исполнение обязательств по Договору)
- [15] Указать временной интервал (например: день (календарный, рабочий), час)
- [16] Указать проценты цифрами и прописью, но не менее 10 %
- [17] Указать проценты цифрами и прописью (размер неустойки должен быть экономически обоснован и стимулировать Исполнителя на надлежащее исполнение обязательств по Договору)
- [18] Указать временной интервал (например: день (календарный, рабочий), час)
- [19] Указать проценты цифрами и прописью, но не менее 10 %
- [20] Указать проценты цифрами и прописью (размер неустойки должен быть экономически обоснован и стимулировать Исполнителя на надлежащее исполнение обязательств по договору)
- [21] Указать временной интервал (например: день (календарный, рабочий), час)
- [22] Указать проценты цифрами и прописью
- [23] Указать проценты цифрами и прописью (размер неустойки должен быть экономически обоснован и стимулировать Исполнителя на надлежащее исполнение обязательств по договору)

- [24] Указать временной интервал (например: день (календарный, рабочий), час)
- [25] Указать проценты цифрами и прописью
- [26] Указать цифрами и прописью (срок не должен быть менее 12 месяцев).
- [27] Указать проценты цифрами и прописью, но не более чем 20 %
- [28] Указать срок цифрами и прописью. Срок должен составлять не менее 5 лет.
- [29] Указать цифрами и прописью
- [30] Указать цифрами и прописью
- [31] Указать цифрами и прописью
- [32] Указать цифрами и прописью
- [33] Указать цифрами и прописью
- [34] Указать проценты цифрами и прописью (размер штрафа должен быть экономически обоснован)
- [35] Указать цифрами и прописью
- [36] Указать ФИО, тел., факс, e-mail представителя (представителей) Заказчика
- [37] Указать ФИО, тел., факс, e-mail представителя (представителей) Исполнителя
- [38] Указать цифрами и прописью
- [39] Указать цифрами и прописью (для всего пункта)
- [40] Указать в строках столбца номера этапов (если работы выполняются без разбиения на этапы, столбец не заполняется)
 - [41] Перечислить в строках столбца выполняемые на этапе работы (если работы выполняются без разбиения на этапы – указать: «работы по разработке программного обеспечения «_____» (указать наименование ПО)»
 - [42] Указать в строках столбца даты начала работ
 - [43] Указать в строках столбца даты завершения работ
 - [44] Указать в строках столбца наименования документов, служащих основанием для завершения работ (например: Акт сдачи-приемки работ, Акт сдачи-приемки работ по этапу (заключительному этапу))
 - [45] Указать цифрами и прописью количество экземпляров
 - [46] Указать входящие в комплект: техническую документацию и других материалы и документы, предусмотренные ТЗ
 - [47] Указать цифрами и прописью количество возвращаемых экземпляров
 - [48] Указать цифрами и прописью
 - [49] Указать цифрами и прописью
 - [50] Указать в столбце наименование платежа (например: аванс, оплата работ по этапу, окончательный расчет и т. д.)
 - [51] Указать в столбце дату платежа
 - [52] Указать в столбцах суммы (сумма аванса за выполняемые работы не может превышать 50 % от стоимости работ)
 - [53] Указать основание для проведения платежа (например: счет на авансовый платеж, Акт сдачи-приемки работ, Акт сдачи-приемки работ по этапу (заключительному этапу)). Основанием для авансового платежа является счет, выставленный Исполнителем, в остальных случаях (закрытие этапа, сдача-приемка выполненных работ) – соответствующий Акт, подписанный Сторонами.
 - [54] Для Акта сдачи-приемки работ по промежуточному этапу данный абзац исключить
 - [55] Указать в каком объеме (например: полном, неполном; в последнем случае в строке «претензии Заказчика к выполненным работам» указать конкретно, что выполнено, а что – нет)
 - [56] Указать в какие сроки (например: установленные, досрочно, с просрочкой; в последнем случае в строке «претензии Заказчика к выполненным работам» указать, с какой именно (количество дней))
 - [57] Указать с каким качеством (например: надлежащим, ненадлежащим; в последнем случае в строке «претензии Заказчика к выполненным работам» указать, в чём именно выражается нарушение обязательств по качеству)
 - [58] Указать конкретные претензии Заказчика
 - [59] Указать срок цифрами и интервал времени (дни, часы) – применить ко всему столбцу

ЛАБОРАТОРНАЯ РАБОТА № 2

Разработка UML модели проекта в Visual Studio 2013. Использование Visual Studio 2013

Цель работы: получить навык создания на основе UML модели в системе **Visual Studio 2013** и изучить особенности создания на основе UML кода на языке C# .

Продолжительность работы – 4 часа.

Содержание

1. Модель жизненного цикла программы	25
2. Моделирование структуры программы на языке UML	26
3. Пример генерации кода.....	31
4. Пример доработки кода	36
5. Разработка и реорганизация кода: рефакторинг.....	40
6. Порядок выполнения лабораторной работы.....	43
7. Вопросы.....	43

1. Модель жизненного цикла программы.

Одна из наиболее важных возможностей интегрированной среды Visual Studio 2013 – поддержка всех этапов жизненного цикла разработки программного проекта.

Концепция жизненного цикла разработки состоит из следующих этапов:

Выработка требований и целей (Для больших проектов – этап разработки архитектуры системы) – формулировка технических, маркетинговых, эксплуатационных и других требований будущей программной системы;

Спецификация – формализованное, полное, точное и внешнее описание программной системы (термин внешнее в данном случае понимается описание того, "ЧТО, а не КАК", т.е. элементы и проект реализации в спецификацию не входят);

Проектирование (дизайн) – разработка подробного проекта системы, включая иерархию модулей, входные и выходные данные, информационные потоки, представление данных, основные алгоритмы; частью спецификации и проектирования является моделирование – построение формальной модели проекта;

Реализация (кодирование) – разработка программного кода системы на выбранном для реализации языке программирования;

Верификация – проверка корректности реализации программной системы, которая на практике в большинстве случаев выполняется путем тестирования (прогона набора тестов), либо путем формальной верификации – формального доказательства того, что реализация системы соответствует формальной спецификации, выполненной на каком-либо языке спецификаций.

Выпуск (релиз) программной системы для пользователей;

Сопровождение программной системы – исправление ошибок, обучение пользователей, ответы на их вопросы, реализация расширений функциональности системы по требованию пользователей.

2. Моделирование структуры программы на языке UML

В Visual Studio 2013, как и во многих других современных интегрированных средах, имеется поддержка моделирования структуры программы на языке UML. Данный язык моделирования может быть использован на ранних этапах разработки проекта.

Язык UML позволяет спроектировать иерархию классов в абстрактных терминах, представить ее в виде модели, а модель – в виде диаграммы. Затем по диаграмме может быть сгенерирован код на выбранном языке (например, на C#). Таким образом, использование языка UML позволяет перейти от этапа моделирования и проектирования к этапу реализации. Сгенерированный код может затем использоваться как основа для последующей реализации проекта.

Однако автоматически сгенерированный код не является законченной программой. В коде достаточно подробно описана структура данных, иерархия классов и параметры методов класса. Но методы представляются в виде заглушек и требуют доработки до работающей программы.

Тем не менее, такая возможность позволяет сэкономить время, затрачиваемое на описание классов, что уже само по себе не так уж и мало.

В качестве примера рассмотрим генерацию простой UML-модели, визуализацию этой модели и генерацию по ней кода на языке C#. Для создания UML-моделей используется специальный вид проекта Modeling Project.

Создадим проект для моделирования. В среде VS 2013 выберем в главном меню File / NewProject и вид проекта Modeling Project (Рис. 1).

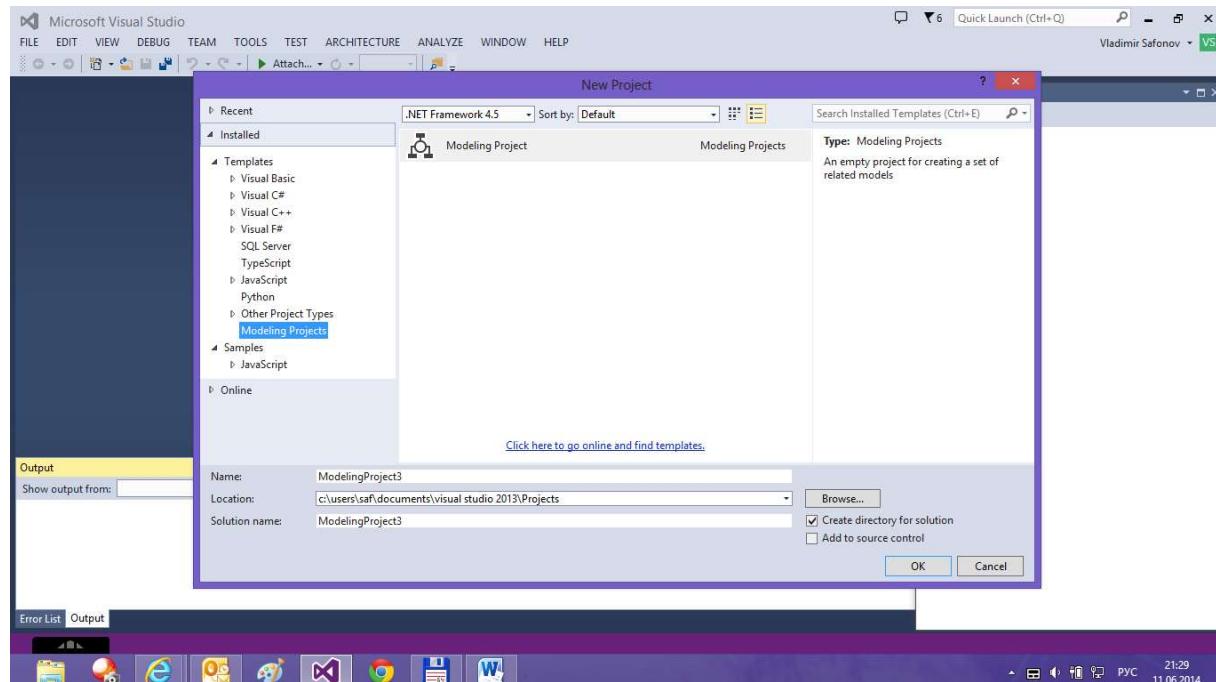


Рис. 1. Создание проекта для построения UML-модели

Для добавления к проекту новой UML-диаграммы выберем пункт меню Architecture, который специально предназначен для отображения архитектуры программы в виде UML-модели, и в нем пункт Add New Diagram (Рис. 2).

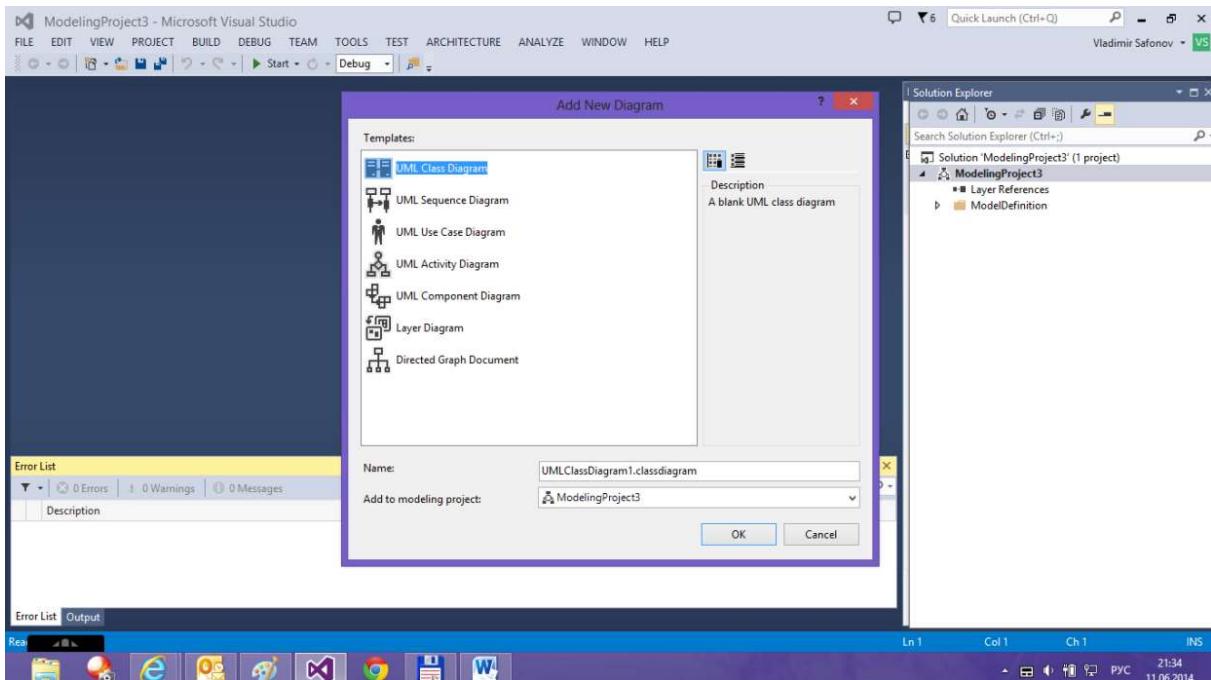


Рис. 2. Выбор вида новой UML-диаграммы

По умолчанию это новая диаграмма классов (UML Class Diagram), как наиболее часто используемая. Однако в этом пункте меню могут быть созданы и другие виды диаграмм:

UML Sequence Diagram – диаграмма последовательности;

UML Use Case Diagram – диаграмма использования;

UML Activity Diagram – диаграмма активности;

UML Component Diagram – диаграмма компонент;

Layer Diagram – диаграмма уровней;

Directed Graph Document – диаграмма, изображающая документ в виде

ориентированного графа.

Создадим диаграмму использования (Use case) (Рис. 3, 4).

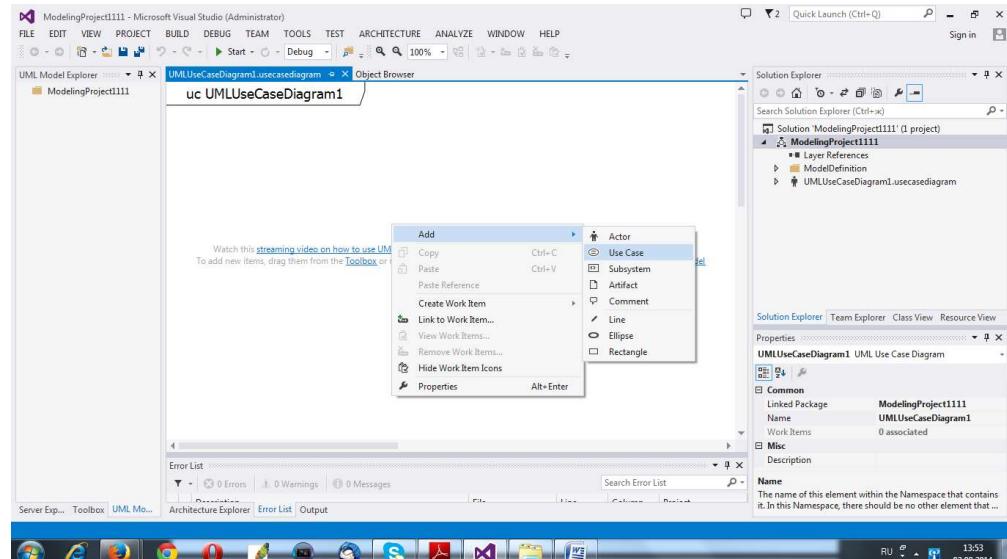


Рис. 3. Создание новой диаграммы использования (Use Case Diagram)

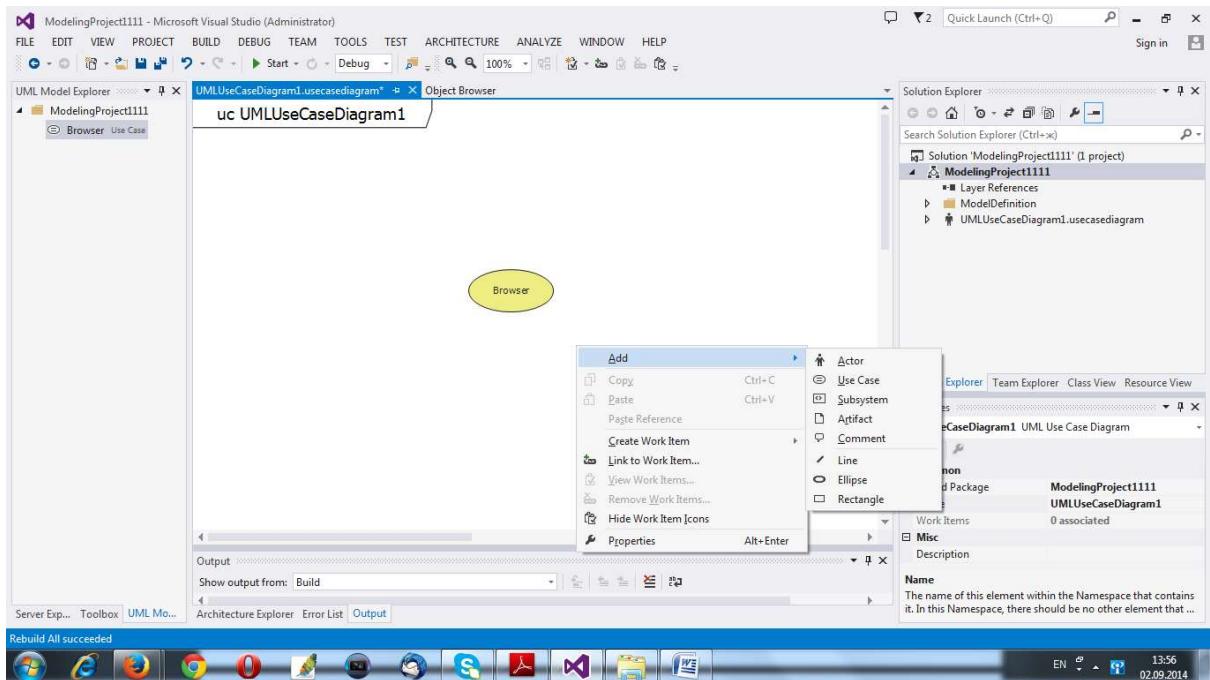


Рис. 4. Создание новой диаграммы использования (Use Case Diagram)

Создадим актеров, прецеденты, комментарии и установим связи между элементами (Рис. 5).

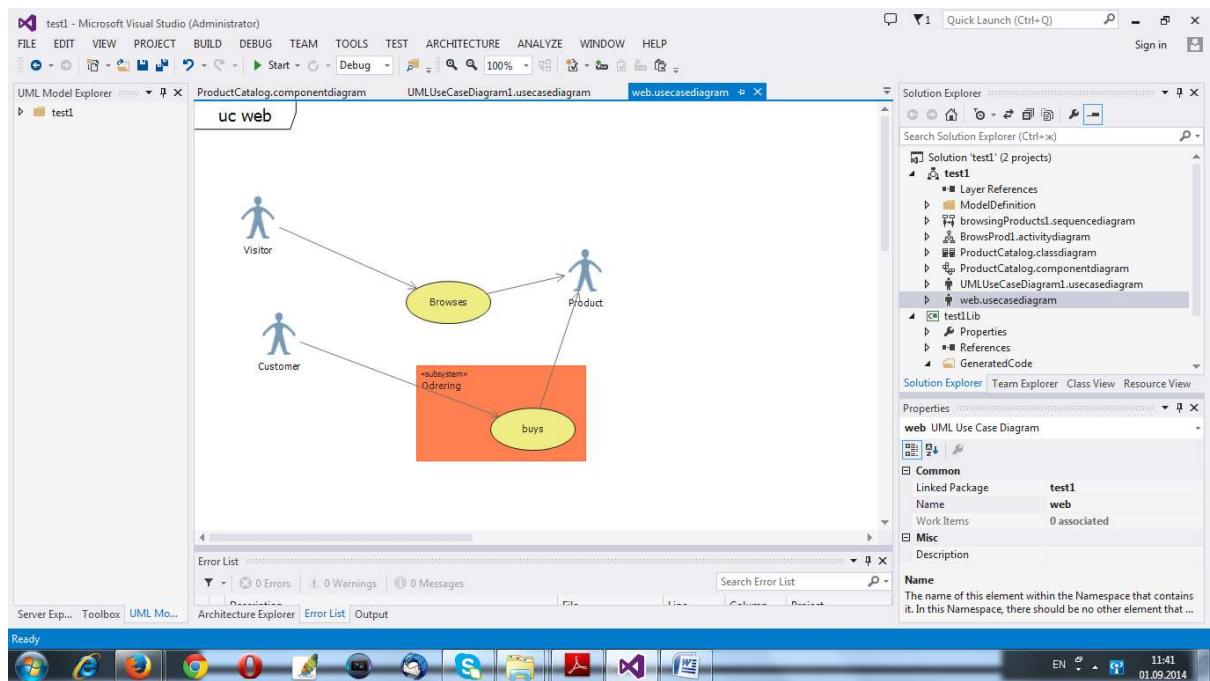


Рис. 5. Создание актеров, прецедентов, комментариев и связей между элементами

Построим диаграмму активностей (Activity Diagram) (Рис 6, 6а, 6б).

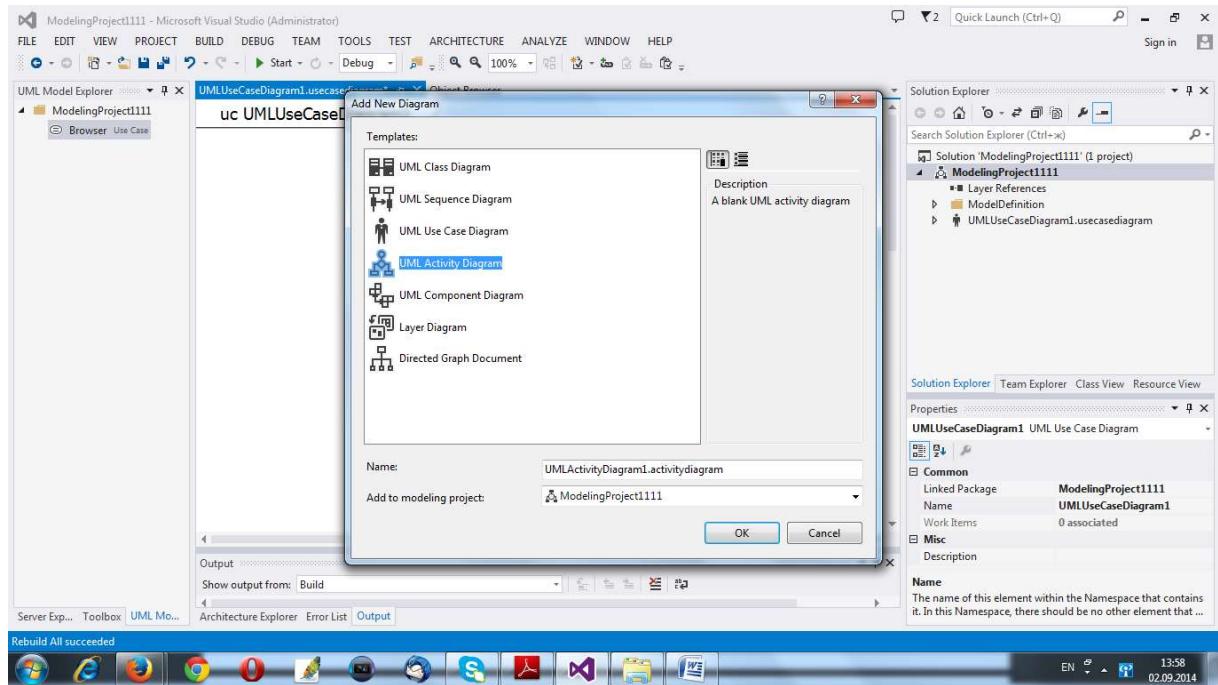


Рис. 6. Построение диаграммы активностей Activity Diagram.

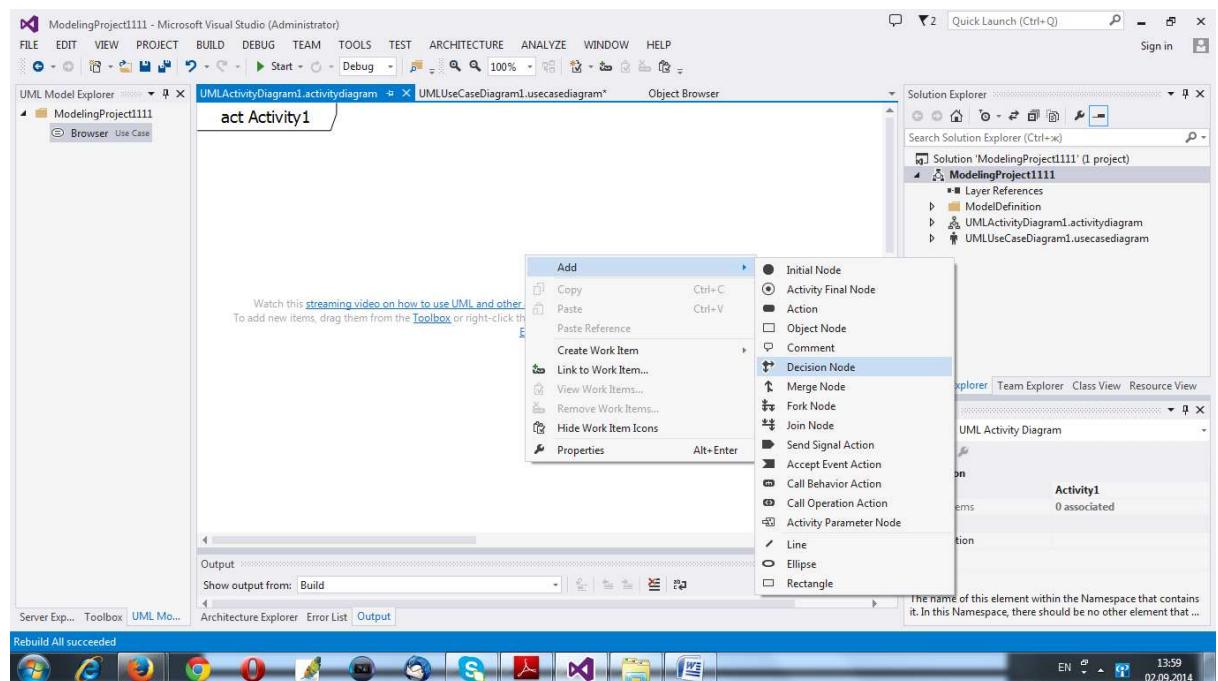


Рис. 6а. Выбор элементов диаграммы активностей.

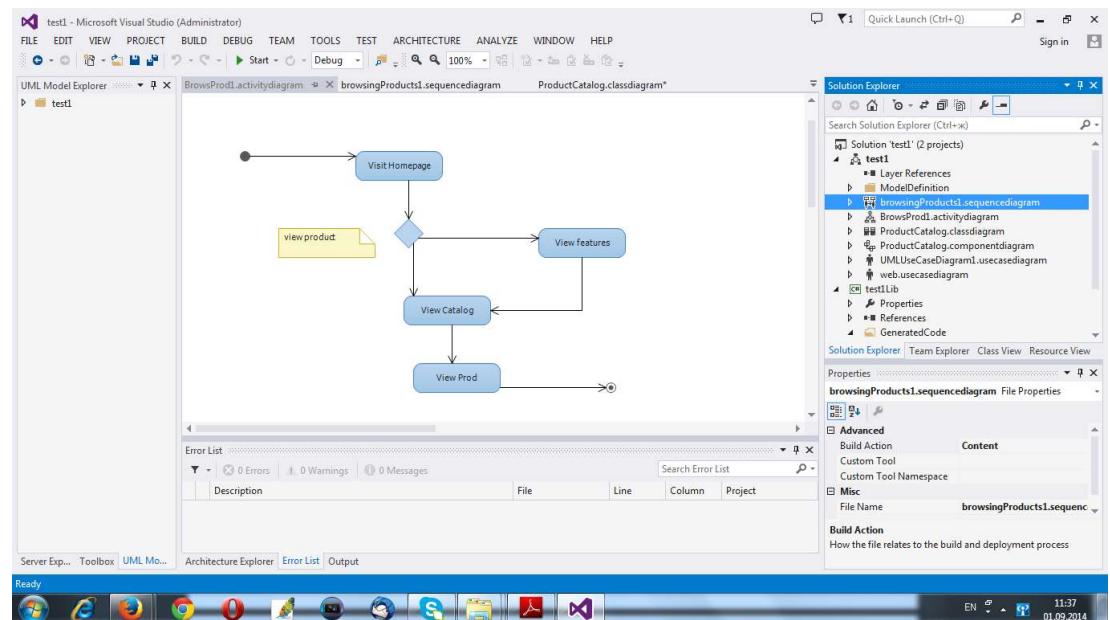
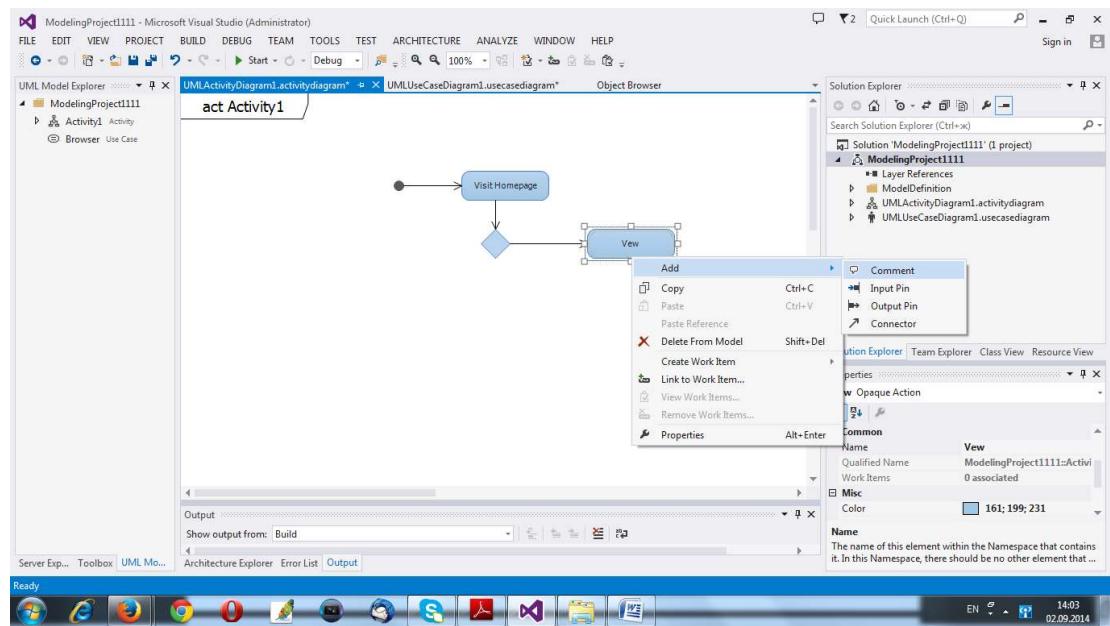


Рис. 66. Диаграмма активностей.

Построение диаграммы последовательностей (Sequence Diagram) (Рис. 7.).

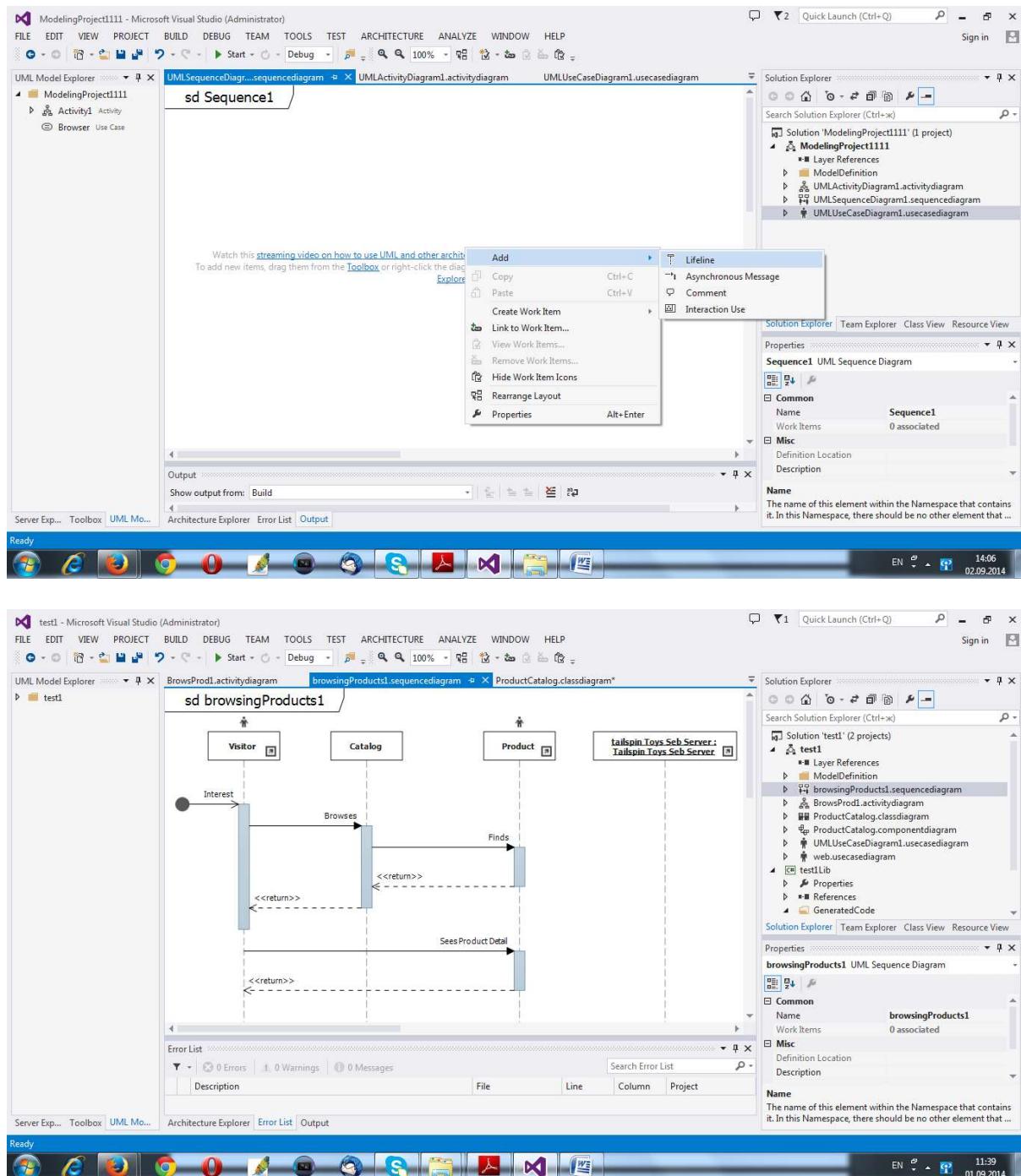


Рис. 7. Построение диаграммы последовательностей (Sequence Diagram).

3. Пример генерации кода из диаграммы UML

В качестве примера создадим UML диаграмму для построения простой консольной программы преобразования строковых переменных.

Перейдем к созданию описания некоторой персоны, включающее в себя имя, отчество и фамилию. Далее имитируем ввод этих данных пользователем и произведем

отображение этих данных на консоль, затем проведем операцию преобразования имени и отчества в инициалы и снова отобразим эту информацию на консоль.

Создаем новую диаграмму классов, пока – пустую (Рис. 8):

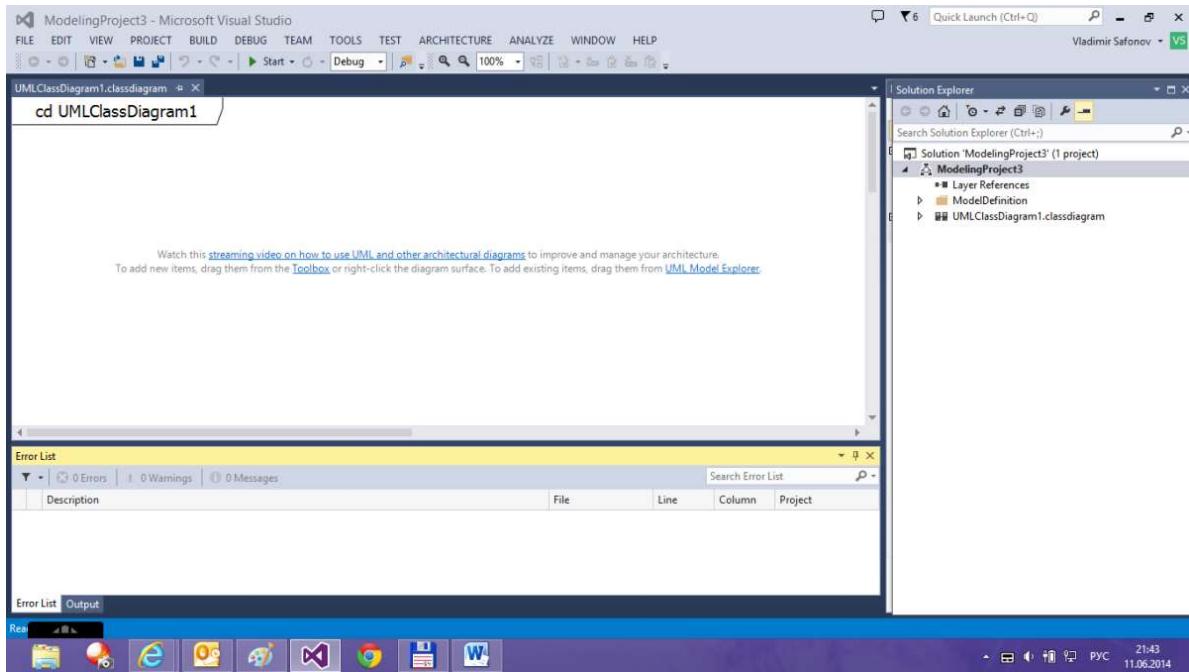


Рис. 8. Создание новой диаграммы классов

Для добавления к диаграмме класса выбираем в контекстном меню пункты Add / Class (Рис. 9):

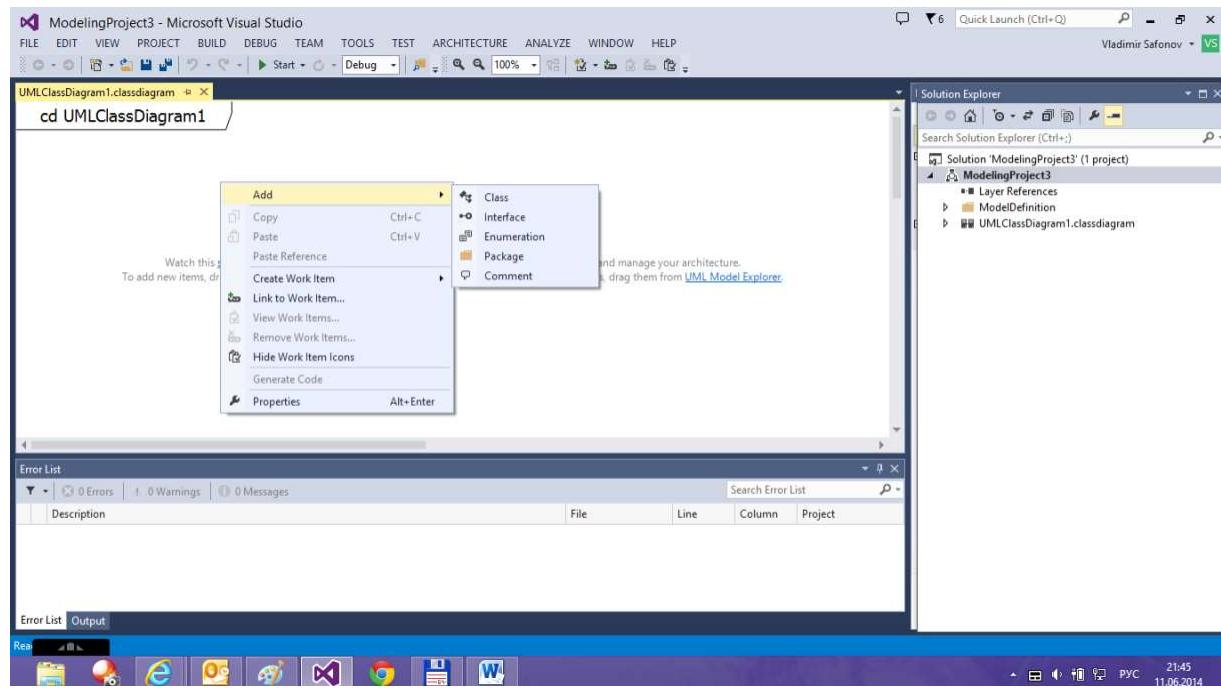


Рис. 9. Добавление к диаграмме нового класса

Класс добавляется с именем Class1 по умолчанию. По терминологии UML, класс состоит из атрибутов и операций (см. рис. 10).

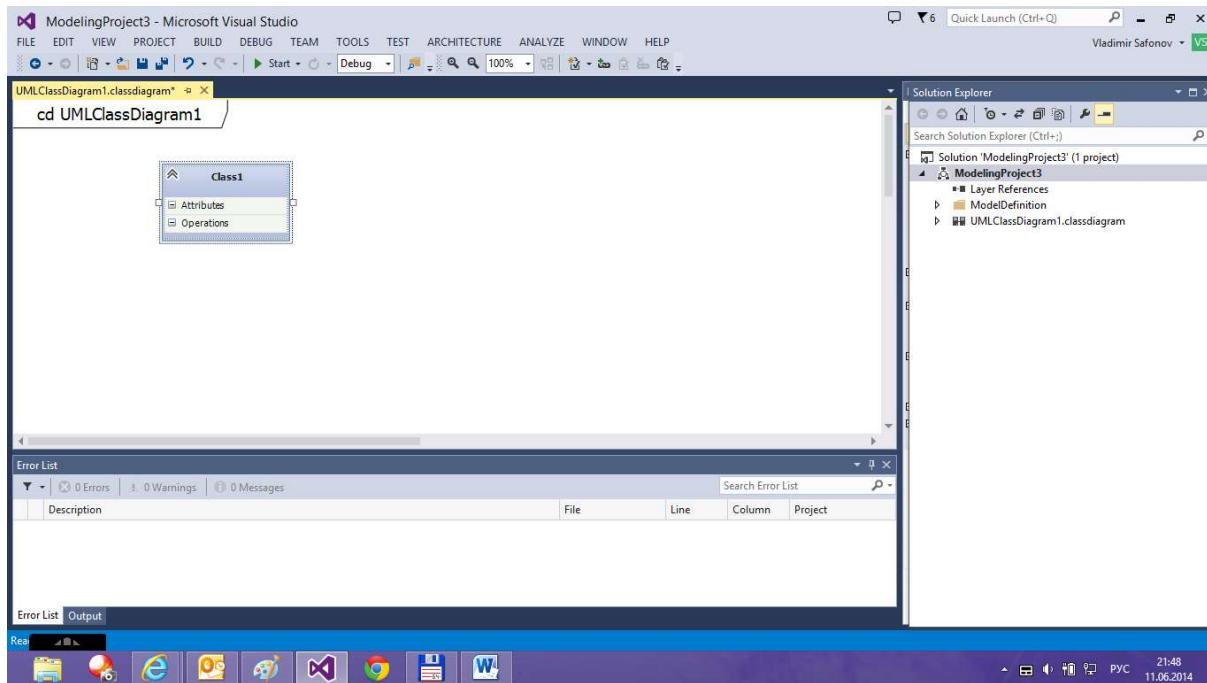


Рис. 10. Класс с атрибутами и операциями

Создадим два класса: Programm и Person. Класс Programm содержит описание основной программы, Person – описание объекта класса и методов класса. (Рис. 11.)

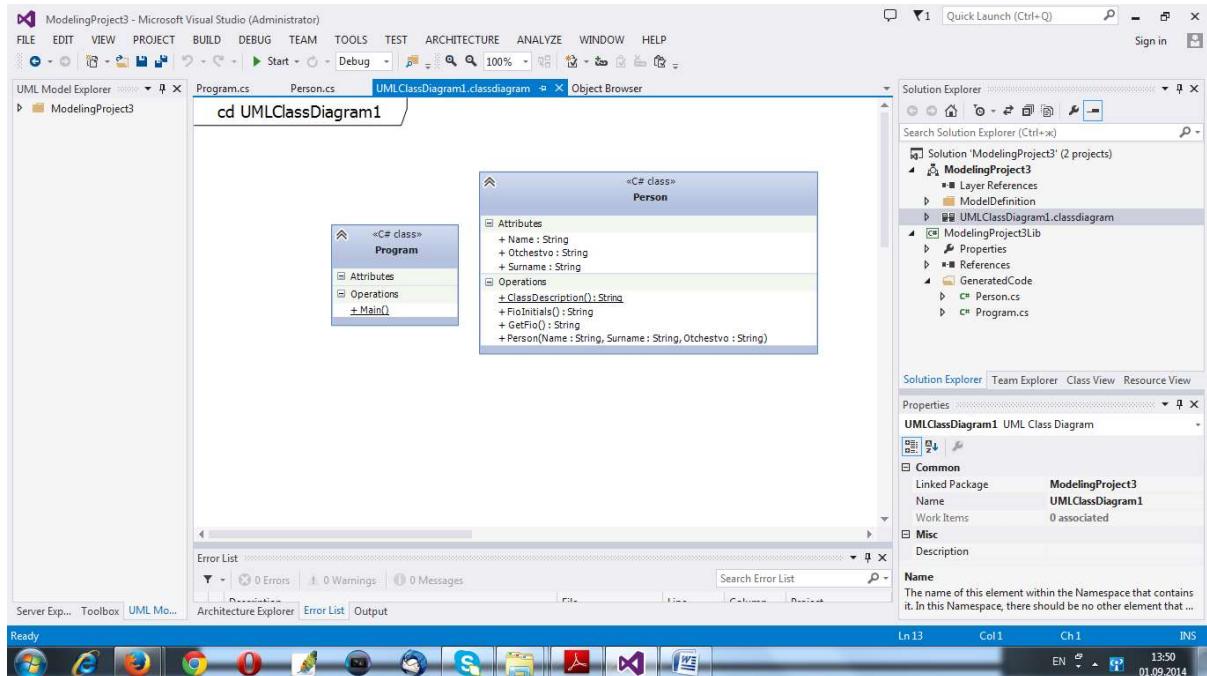


Рис. 11. Создаем два класса с атрибутами и операциями.

Так как цель лабораторной работы носит ознакомительный характер с возможностями Visual Studio, мы не будем усложнять себе задачу созданием большого количества классов и установлением различных видов связи между ними, хотя такая возможность существует (Рис. 12).

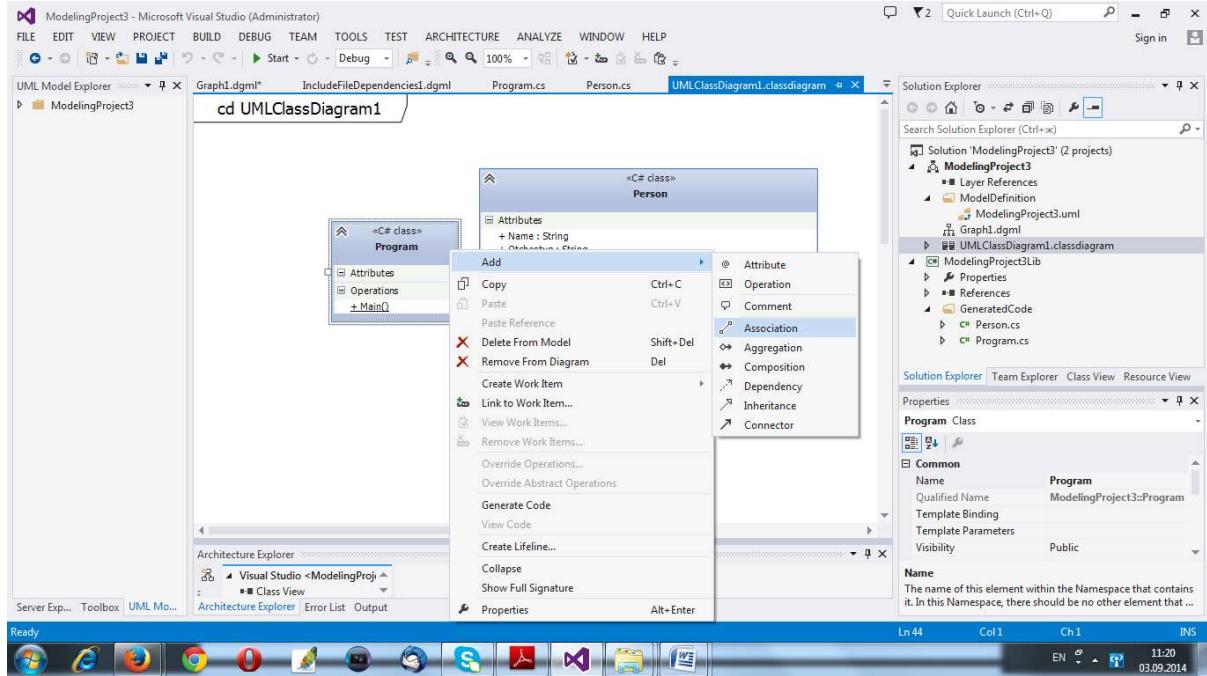


Рис. 12. Контекстное меню класса.

Теперь можно сгенерировать код, который используем при дальнейшей разработке. В контекстном меню выбираем пункт Generate code (Рис. 13):

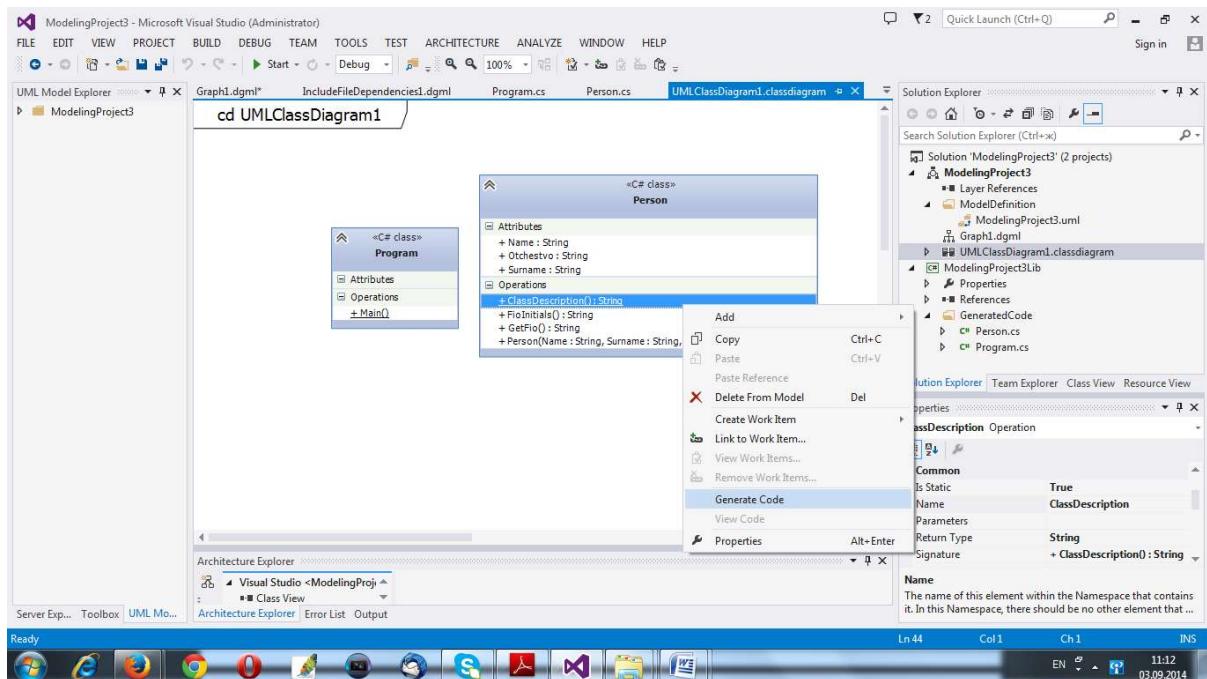


Рис. 13. Выбор операции создания кода.

Генератор кода просит уточнить, по какому шаблону будет происходить генерация. Выбираем шаблон кода для класса (Рис. 14):

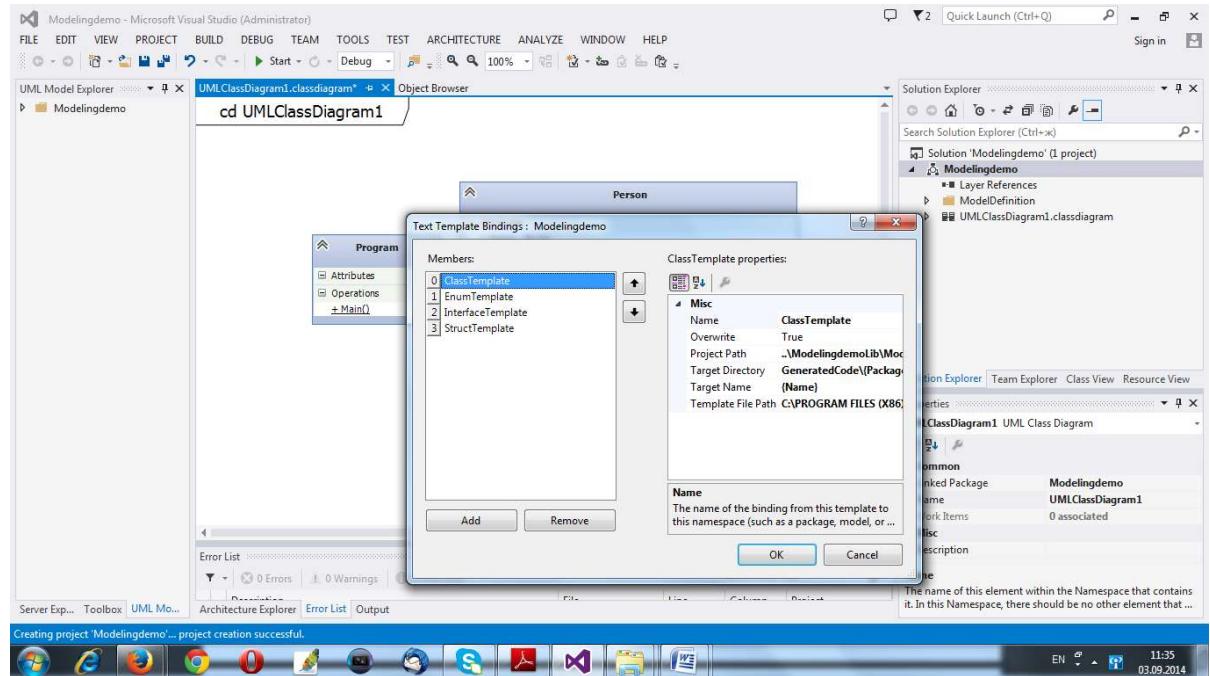


Рис. 14. Выбор шаблона кода для класса при генерации кода по UML-диаграмме

После генерации кода в Solution Explorer появляются два новых пункта – файлы на языке C# Person.cs и Program.cs (Рис. 15):

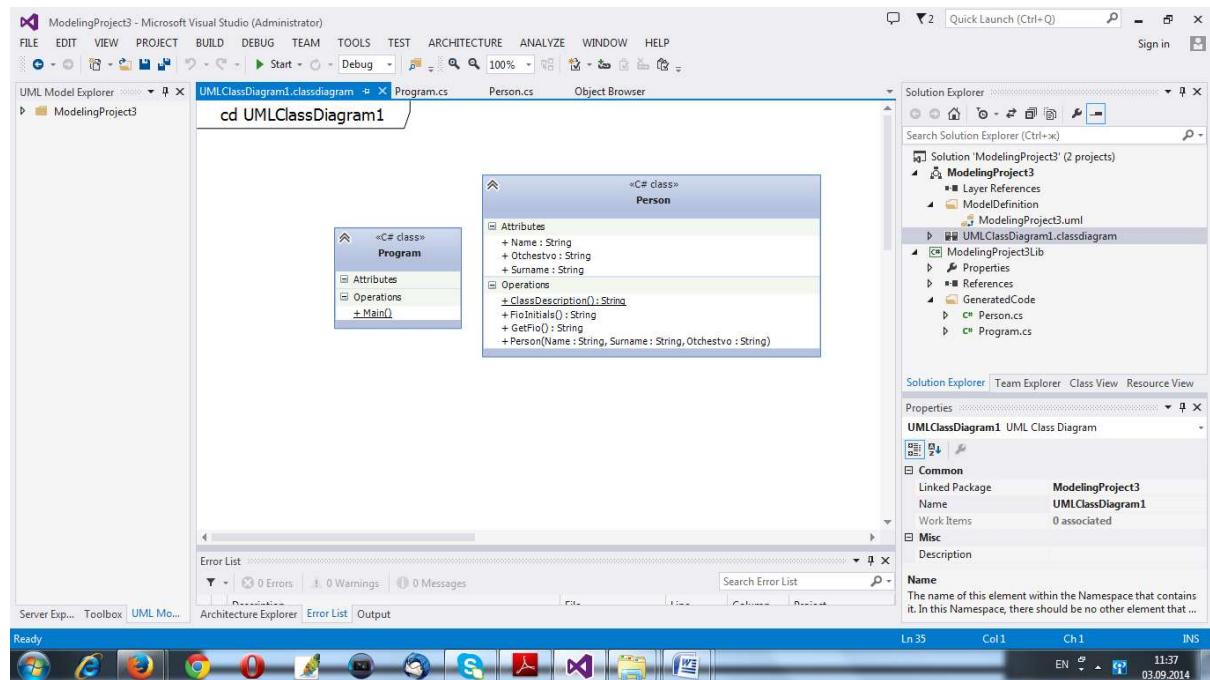


Рис. 15. Завершение генерации кода по UML-диаграмме: генерация двух файлов на C#

Фрагмент файла Person.cs показан на рис. 16.

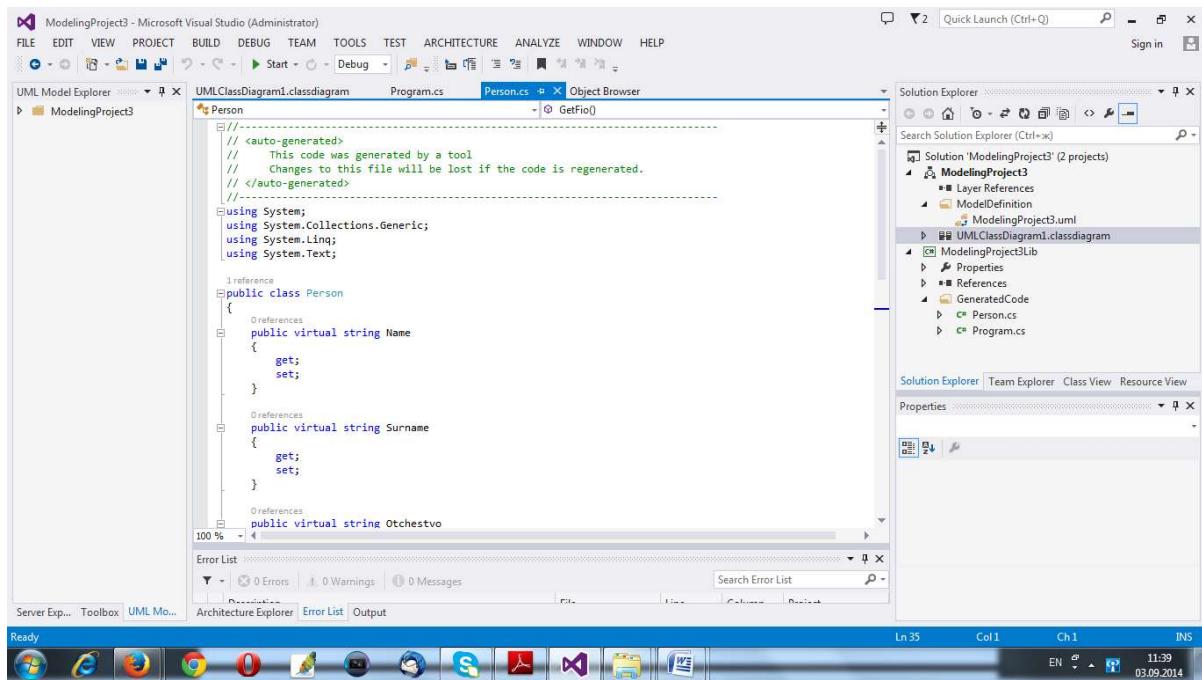


Рис. 16. Сгенерированный файл Person.cs

Атрибуты класса реализованы в виде свойства (property) с методами get и set.

Заглушки методов – в виде виртуальных методов с реализацией в виде генерации исключения, например:

```
public virtual string GetFio()
{
    throw new System.NotImplementedException();
}
```

Теперь сгенерированные файлы можно использовать при последующей разработке.

Созданная модель играет весьма важную роль: это отражение в проекте результата раннего этапа разработки – моделирования и проектирования. При необходимости изменить проект, изменения могут быть сделаны в UML-диаграммах, по которым исходные коды генерируются автоматически.

4. Пример доработки кода

Автоматически сгенерированный код имеет вид:

```
//-----
//<auto-generated>
// This code was generated by a tool
// Changes to this file will be lost if the code is regenerated.
//</auto-generated>
//-----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```

public class Person
{
    public virtual string Name
    {
        get;
        set;
    }

    public virtual string Surname
    {
        get;
        set;
    }

    public virtual string Otchestvo
    {
        get;
        set;
    }

    public virtual string GetFio()
    {
        throw new System.NotImplementedException();
    }

    public virtual string FioInitials()
    {
        throw new System.NotImplementedException();
    }

    public static string ClassDescription()
    {
        throw new System.NotImplementedException();
    }

    public Person(string Name, string Surname, string Otchestvo)
    {
    }
}

//-----
// <auto-generated>
// This code was generated by a tool
// Changes to this file will be lost if the code is regenerated.
// </auto-generated>
//-----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Program
{
    public static void Main()

```

```

    {
        throw new System.NotImplementedException();
    }
}

```

В автоматически созданной структуре необходимо уточнить описание атрибутов и методов:

```

//-----
//<auto-generated>
// This code was generated by a tool
// Changes to this file will be lost if the code is regenerated.
//</auto-generated>
//-----

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Person
{
    private string _name = "";
    public virtual string Name
    {
        get{ return _name; }
        set { _name = value; }
    }

    private string _surname = "";
    public virtual string Surname
    {
        get { return _surname; }
        set { _surname = value; }
    }

    private string _otchestvo = "";
    public virtual string Otchestvo
    {
        get { return _otchestvo; }
        set { _otchestvo = value; }
    }

    public virtual string Fio
    {
        get
        {
            string fio = Surname + " " + Name + " " + Otchestvo;
            return fio;
        }
    }

    public virtual string FioInitials
    {
        get
        {
            string fio = Surname + " " + Name.Substring(0, 1) + ". " + Otchestvo.Substring(0, 1) + ". ";
        }
    }
}

```

```

        return fio;
    }
}

public static string ClassDescription
{
    get
    {
        return "Класс Person. Хранит данные о человеке.";
    }
}

public Person(string name, string surname, string otchestvo)
{
    Name = name;
    Surname = surname;
    Otchestvo = otchestvo;
}

}

//-----
//<auto-generated>
// This code was generated by a tool
// Changes to this file will be lost if the code is regenerated.
//</auto-generated>
//-----

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Program
{
    public static void Main(string[] args)
    {
        // создаем два экземпляра класса человек с разными фио
        Person person1 = new Person("Пушкин", "Александр", "Сергеевич");
        Person person2 = new Person("Гончарова", "Наталья", "Николаевна");

        System.Console.WriteLine(person1.Fio);
        System.Console.WriteLine(person1.FioInitials);
        System.Console.WriteLine(person2.Fio);
        System.Console.WriteLine(person2.FioInitials);
        System.Console.WriteLine(Person.ClassDescription);
        System.Console.WriteLine(Person.ClassDescription);
    }
}

```

Как видим, задача не сводится к чисто механическим действиям и требует серьезных навыков программирования. И, тем не менее, автоматическая генерация кода снимает с программиста часть рутинных процессов чем существенно ускоряет процесс программирования.

5. Разработка и реорганизация кода: рефакторинг

Особое место при разработке и модификации кода занимает рефакторинг – систематическая модификация и улучшение существующего кода, без коренного изменения его семантики, с помощью автоматических преобразований, осуществляемых средой. Другими словами, это способ приведения кода в порядок, при котором шансы появления новых ошибок в коде минимальны.

Изменения при рефакторинге кода вносятся пошагово, мелкими операциями. Переименовать функцию, разбить ее на несколько подфункций, изменить сигнатуру метода – все эти операции легко выполнить так, чтобы не занести в код ошибок. Однако совокупный эффект ряда таких элементарных операций может быть весьма значительным – от упрощения структуры кода, до полного изменения архитектуры разрабатываемой программы.

Между тем, проведение рефакторинга вручную – достаточно утомительное занятие. Например, чтобы изменить порядок следования параметров в методе класса, мало изменить сигнатуру метода – необходимо пройти по всему коду программы и модифицировать все вызовы данного метода. Изменение тривиальное, но требует активного "копипастинга".

Среда Visual Studio впервые обзавелась такими средствами в версии VS2005. Разработчики, использующие C# и J#, получили в свое распоряжение следующий набор инструментов:

- Rename – переименование имени переменной, метода, класса и т.п. с автоматическим обновлением всех ссылок на это имя в коде;
- Extract method – оформление выделенной части кода в новый, отдельный метод;
- Encapsulate field – создание свойства, скрывающего выбранную переменную-член класса;
- Extract interface – создание интерфейса на основе списка методов класса;
- Promote local variable to parameter – вынесение локальной переменной в параметр метода;
- Remove/Reorder parameters – удаление параметров метода и изменение порядка их следования с автоматическим обновлением всех ссылок в коде на данный метод.

В качестве примера применим рефакторинг для нашей программы- изменим имя метода. Задача состоит в том, чтобы изменить имя метода и в его определении, и во всех его использованиях. В общем случае, если проект достаточно велик, вручную выполнять решение подобной задачи весьма неудобно. Почти наверняка при этом какие-либо использования метода будут забыты, и к ним придется возвращаться уже после получения ошибок при компиляции (сборке) проекта.

Применим рефакторинг к созданной нами программе (Рис. 17):

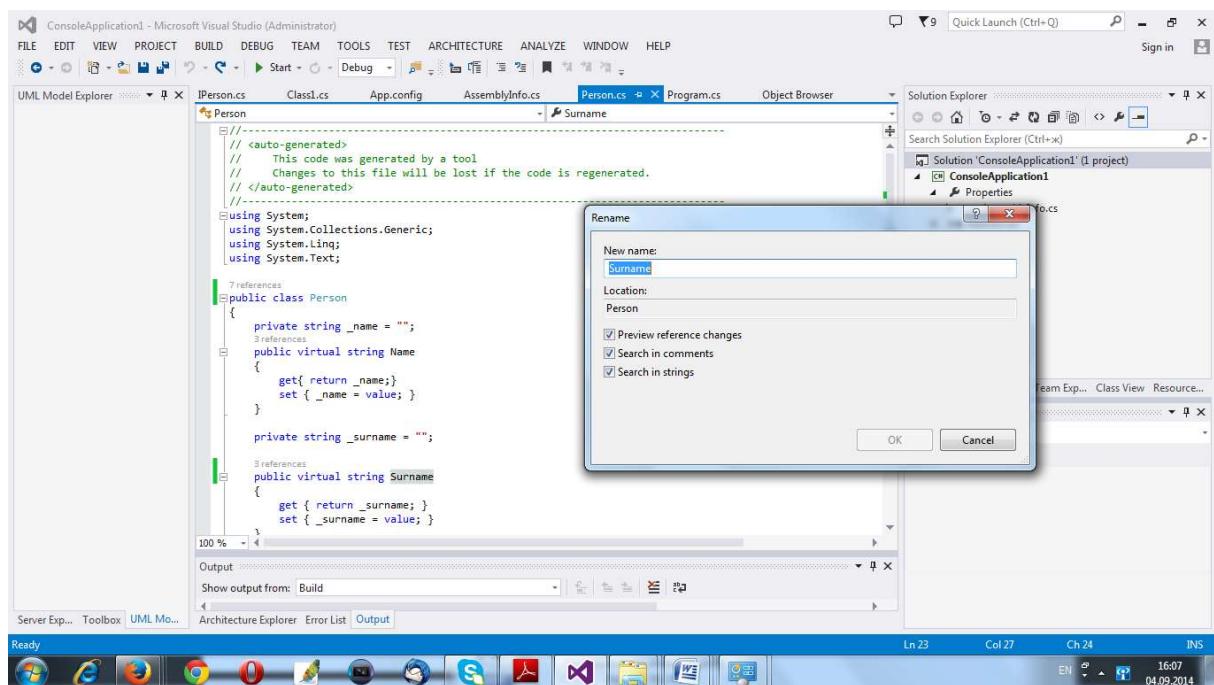
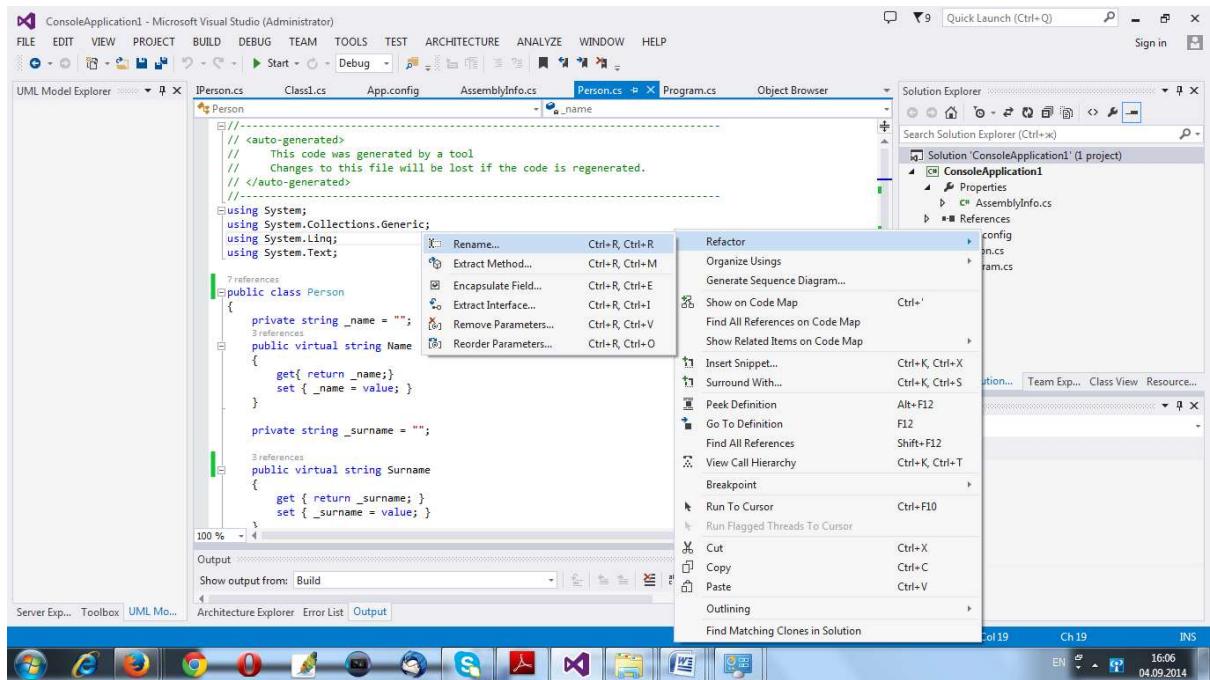


Рис. 17. Рефакторинг имени

В качестве примера изменим имя Surname на Surname1. В окне Preview Changes отобразятся все замены имени по тексту (Рис. 18):

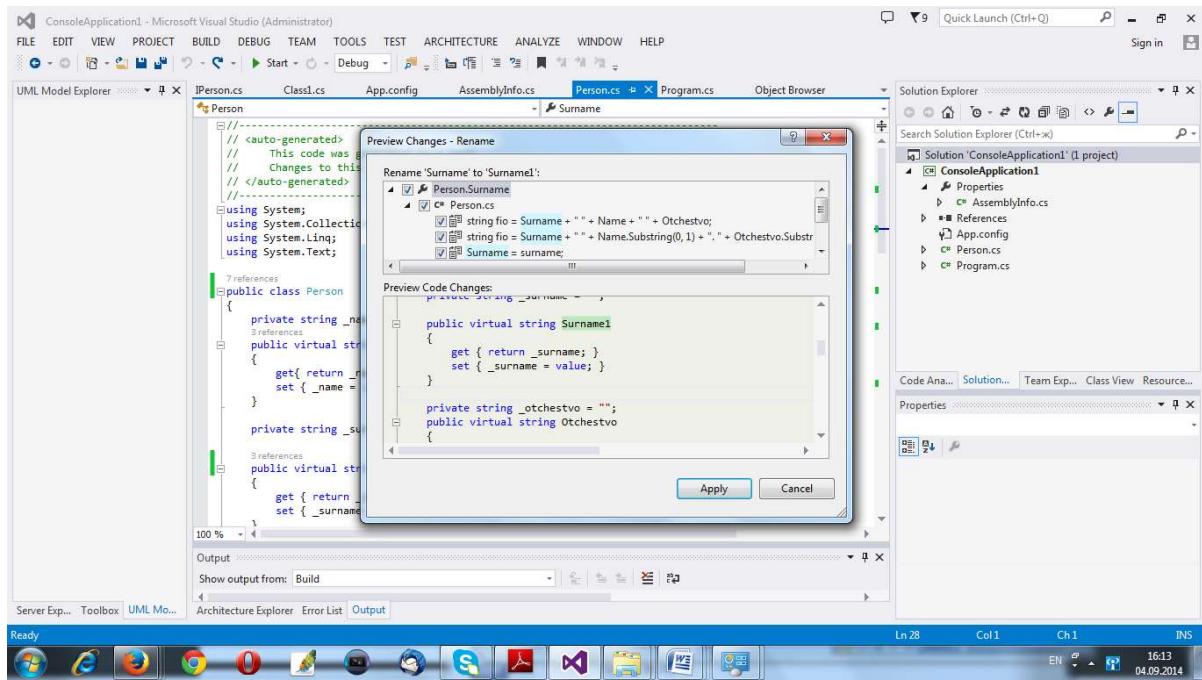


Рис. 18. Отображение изменений в тексте программы

Поскольку все предлагаемые изменения нас устраивают, нажимаем Apply.
На рис. 19 показан результат рефакторинга.

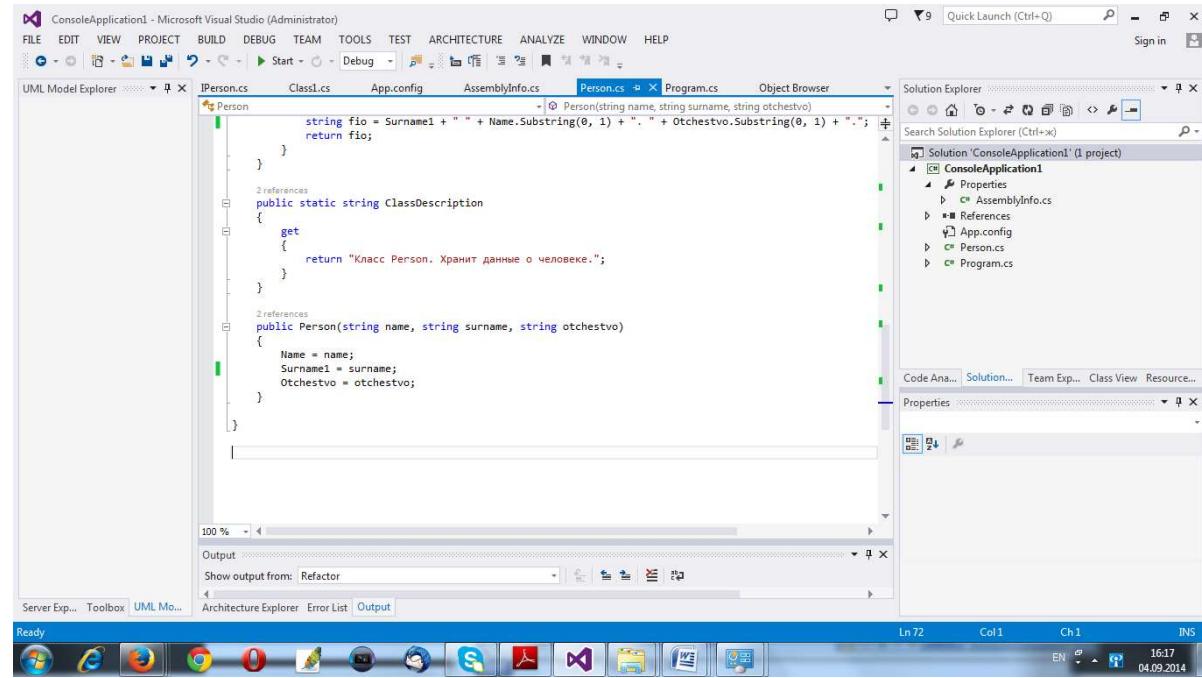


Рис. 19. Результат рефакторинга – имя метода изменено

Чтобы убедиться в правильности выполненных средой преобразований, выполним запуск программы.

Реализованный набор инструментов, конечно же, упростил жизнь разработчикам, однако он далеко не идеален. Из недостатков можно отметить:

- отсутствие поддержки C++ и VB;
- достаточно медленная работа;
- набор средств рефакторинга минимален, ряд полезных инструментов в нем отсутствует;
- интерфейс далеко не самый эргономичный и требует активного использования мышки.

7. Порядок выполнения лабораторной работы.

1. Изучить теоретические сведения, приведенные в описании лабораторной работы, письменно ответить на контрольные вопросы, создать и выполнить рассмотренный в лабораторной работе пример.
2. Запустить систему Microsoft Visual Studio 2013.
3. Создать новый проект: выбрать команду главного меню File | New | Project, Modeling Project. Далее набрать имя проекта Lab1Project в поле Name, с помощью кнопки Browse указать в поле Location папку на сетевом диске Н для хранения проекта и нажать кнопку OK. Для добавления к проекту новой UML-диаграммы выберем пункт меню Architecture и вид создаваемой диаграммы.
4. Далее в соответствии с индивидуальным заданием (Лабораторная работа №1) создать Use-case диаграмму, диаграмму классов, диаграмму последовательностей, диаграмму использования, диаграмму активностей.
5. По созданной диаграмме классов произвести генерацию кода. Доработать код до работающей программы, используя текст из описания лабораторной работы. Сохранить на сетевом диске Н проект приложения командой главного меню File | Save All.
6. Провести рефакторинг.

Защита лабораторной работы заключается в предъявлении преподавателю полученных результатов (на экране монитора), демонстрации полученных навыков и ответах на вопросы преподавателя.

9. Вопросы.

1. Какие этапы жизненного цикла разработки программного проекта поддерживает среда Visual Studio 2013.
2. Какие UML-диаграммы можно построить с использованием Visual Studio 2013.
3. Каковы возможности автоматической генерации кода в Visual Studio 2013.
4. Рефакторинг в Visual Studio 2013. Его возможности и ограничения.

ЛАБОРАТОРНАЯ РАБОТА №3.

Выбор платформы и декомпозиция проекта. Основы использования системы контроля версий Git.

Цель работы: получить навык выработки требований к программно-аппаратной платформе разрабатываемого ПО, проведения процесса декомпозиции задачи, освоить работу в системе контроля версий.

Продолжительность работы – 4 часа.

Содержание

1. Понятие "платформы" при разработке ПО	44
2. Декомпозиция	48
3. Система управления версиями.....	52
4. Порядок выполнения лабораторной работы.....	71
5. Вопросы.....	72

1. Понятие платформы

В информационных технологиях под термином «платформа» в широком смысле обычно понимается совокупность следующих компонентов:

- аппаратного решения;
- операционной системы (ОС);
- прикладных программных решений и средств для их разработки.

В более узком смысле выделяют следующие виды платформ:

Программная платформа – это совокупность операционной системы, средств разработки прикладных программных решений и прикладных программ, работающих под управлением этой операционной системы

Прикладная платформа – это средства выполнения и комплекс технологических решений, используемых в качестве основы для построения определенного круга прикладных программ

Аппаратная платформа (hardware) – это совокупность совместимых аппаратных решений с ориентированной на них операционной системой

Понятие «аппаратная платформа» связано с решением фирмы IBM о выработке и утверждении единого стандарта на основные комплектующие персонального компьютера. Стандарт получил название *платформа IBM PC-совместимых ПК*.

В настоящее время фирма IBM чаще используют понятие «платформа Wintel», подразумевая под этим сочетание микропроцессора фирмы Intel с операционной системой Windows. Микропроцессор при этом рассматривается как основа аппаратной платформы, которая определяет архитектуру персонального компьютера, т. е. его тип и характеристики.

Однако термин Wintel не совсем точно определяет понятие платформы, так как открытая архитектура современных IBM-совместимых персональных компьютеров позволяет собирать их из комплектующих, изготавливаемых различными фирмами-производителями, включая и микропроцессоры, которые в настоящее время выпускаются не только фирмой Intel, но и Advanced Micro Devices (AMD), Cyrix Corp.

и др. Кроме того, IBM-совместимые ПК могут работать не только под управлением операционной системы Windows, но и под управлением других операционных систем.

Платформа IBM-совместимых компьютеров включает в себя широкий спектр самых различных персональных компьютеров: от простейших домашних до сложных серверов.

Кроме платформы IBM-совместимых ПК достаточно широкое распространение получила *платформа Apple*, представленная довольно популярными на Западе компьютерами Macintosh.

Специалисты по компьютерной истории отдают приоритет в создании ПК именно компании Apple. С середины 70-х г. эта фирма представила несколько десятков моделей ПК — начиная с Apple I и заканчивая современным iMac, — и уверенно противостоит корпорации IBM.

Существует два основных варианта решения проблемы совместимости компьютерных платформ:

1. *Аппаратные решения* — это специальные платы, несущие на себе дополнительные процессор, оперативную память и видеопамять другой аппаратной платформы. Фактически они представляют собой отдельный компьютер, вставленный в существующий ПК. Его, как и обычный компьютер, можно оснастить любой операционной системой по выбору пользователя и соответствующим программным обеспечением. При этом можно легко переключаться между двумя операционными системами, обмениваться между ними файлами и выполнять другие операции, причем производительность обеих систем остается высокой и они не влияют друг на друга, так как практически не имеют разделяемых ресурсов, кроме мыши, клавиатуры и монитора. Основным недостатком таких плат является их высокая стоимость, хотя и несколько меньшая, чем отдельного ПК.

2. *Программные решения* — это специально написанные программы-эмулаторы, позволяющие запустить программное обеспечение, разработанное для персональных компьютеров одного типа, на другом ПК.

Прикладные решения и средства их разработки

Средства разработки прикладных решений — это очень важная часть платформы персонального компьютера. От гибкости, богатства, удобства и надежности этих средств зависит популярность платформы. Платформа без средств разработки приложений под неё перестает существовать.

Все поставщики платформ поставляют и средства разработки прикладных решений в той или иной форме. Производители операционных систем предлагают всевозможные компиляторы и интерпретаторы, системы управления базами данных, системы организации взаимодействия (например, электронная почта). Конечно, решения для популярных операционных систем предлагают не только фирмы-создатели, но и другие фирмы-разработчики.

Для платформ, у которых возможности осуществления разработки решений непосредственно на них ограничены (например, для сотовых телефонов), производители предлагают средства разработки, функционирующие под популярной и мощной операционной системой (Windows, Linux). В дополнение к этим средствам предлагается эмулятор целевой платформы, на котором можно отладить решение, не используя целевую платформу непосредственно.

В настоящее время набирают популярность решения, обеспечивающие независимость разрабатываемых прикладных решений не только от аппаратной составляющей платформы, но и от операционной системы. Самые популярные решения подобного рода — Java и .Net.

Основная идея этих платформ состоит в создании «виртуальной машины» — специального программного комплекса, функционирующего на конкретной аппаратной платформе и на конкретной операционной системе. Прикладную программу обрабатывает виртуальная машина, которая преобразует «виртуальные команды» в команды конкретной программно-аппаратной платформы. В итоге получается, что программа для виртуальной машины функционирует на множестве связок «аппаратная часть—операционная система» без переделки. Единственное условие — наличие виртуальной машины для конкретного программно-аппаратного решения. Самая распространенная аппаратно-независимая платформа — Java.

Существует определенный класс программных продуктов — конструкторов, использование которых ограничено какой-либо предметной областью. Эти продукты реализуют не только базовую функциональность, но и гибкие средства создания решений в определенной области деятельности. Такие программные продукты зачастую называются прикладными платформами.

Под прикладной платформой понимаются среда исполнения и набор технологических решений, используемых в качестве основы для построения определенного круга приложений. Фактически приложения базируются на нескольких платформах, образующих многослойную среду. При этом важно, что платформа предоставляет разработчику определенную модель, как правило, изолирующую его от понятий и подробностей более низкоуровневых технологий и платформ.

Ключевым качеством прикладной платформы является достаточность ее средств для решения задач, стоящих перед бизнес-приложениями. Это обеспечивает хорошую согласованность всех технологий и инструментов, которыми пользуется разработчик. Другой важный момент — стандартизация. Наличие единой прикладной платформы для большого количества прикладных решений способствует формированию общего «культурного слоя», включающего и людей (программистов, аналитиков, пользователей), и методологию (типовые структуры данных, алгоритмы, пользовательские интерфейсы). Опираясь на этот «культурный слой», разработчик тратит минимум усилий на поиск необходимого решения практически в любой ситуации, начиная от включения в проект нового специалиста и кончая реализацией какой-либо подсистемы бизнес-приложения по типовой методологии.

Типичный представитель специальных прикладных платформ — система «1С: Предприятие». Сама по себе система является гибким, настраиваемым под нужды конкретного предприятия конструктором, предоставляющим разработчику решения, «более прикладные» методы и средства по сравнению с традиционными языками программирования, т. е. такая платформа представляет собой набор различных механизмов, используемых для автоматизации экономической деятельности и не зависящих от конкретного законодательства и методологии учета.

Существуют комплексные прикладные системы масштаба корпораций, которые являются основой для надежного ведения крупного бизнеса, так называемые ERP-системы (Enterprise Resource Planning Systems). Эти системы также являются прикладной платформой, гибко настраиваемой в своей предметной области.

Критерии выбора платформы

Выбор платформы представляет собой сложную задачу, которая состоит из двух частей:

1. Определение сервиса, который должен обеспечиваться платформой
2. Определение уровня сервиса, который может обеспечить данная платформа

Существует несколько причин, в силу которых достаточно сложно оценить возможности платформы с выбранным набором компонентов, которые включаются в систему:

- подобная оценка прогнозирует будущее: предполагаемую комбинацию устройств, будущее использование программного обеспечения, будущих пользователей;
- конфигурация аппаратных и программных средств связана с определением множества разнородных по своей сути компонентов системы, в результате чего сложность быстро увеличивается;
- скорость технологических усовершенствований аппаратных средств, функциональной организации системы, операционных систем очень высокая и постоянно растет. Ко времени, когда какой-либо компонент широко используется и хорошо изучен, он часто рассматривается как устаревший.
- доступная потребителю информация об аппаратном обеспечении, операционных системах, программном обеспечении носит общий характер. Структура аппаратных средств, на базе которых работают программные системы, стала настолько сложной, что эксперты в одной области редко являются таковыми в другой.

Выбор той или иной платформы и конфигурации определяется рядом критериев. К ним относятся:

1. Отношение стоимость-производительность.
2. Надежность и отказоустойчивость.
3. Масштабируемость.
4. Совместимость и мобильность программного обеспечения.

Отношение стоимость-производительность. Появление любого нового направления в вычислительной технике определяется требованиями компьютерного рынка. Поэтому у разработчиков компьютеров нет одной единственной цели.

Майнфрейм — это электронно-вычислительная машина, относящаяся к классу больших ЭВМ с высокой производительностью, поддерживающая многопользовательский режим работы для решения специализированных задач.

Майнфрейм или суперкомпьютер стоят дорого, т.к. для достижения поставленных целей при проектировании высокопроизводительных конструкций приходится игнорировать стоимостные характеристики.

Другим крайним примером может служить низкостоимостная конструкция, где производительность принесена в жертву для достижения низкой стоимости. К этому направлению относятся персональные компьютеры. Между этими двумя крайними направлениями находятся конструкции, основанные на отношении стоимость-производительность, в которых разработчики находят баланс между стоимостными параметрами и производительностью. Типичными примерами такого рода компьютеров являются мини-компьютеры и рабочие станции.

Надежность и отказоустойчивость. Важнейшей характеристикой аппаратной платформы является надежность. Повышение надежности основано на принципе предотвращения неисправностей путем снижения интенсивности отказов и сбоев за счет применения электронных схем и компонентов с высокой и сверхвысокой степенью интеграции, снижения уровня помех, облегченных режимов работы схем, обеспечение тепловых режимов их работы, а также за счет совершенствования методов сборки аппаратной части персонального компьютера.

Введение отказоустойчивости требует избыточного аппаратного и программного обеспечения. Структура многопроцессорных и многомашинных систем приспособлена к автоматической реконфигурации и обеспечивает возможность продолжения работы системы после возникновения неисправностей. Понятие надежности включает не только аппаратные средства, но и программное обеспечение. Главной целью повышения надежности систем является целостность хранимых в них данных.

Отказоустойчивость – это свойство вычислительной системы, которое обеспечивает возможность продолжения действий, заданных программой, после возникновения неисправностей.

Масштабируемость должна обеспечиваться архитектурой и конструкцией компьютера, а также соответствующими средствами программного обеспечения.

Добавление каждого нового процессора в действительно масштабируемой системе должно давать прогнозируемое увеличение производительности и пропускной способности при приемлемых затратах. В действительности реальное увеличение производительности трудно оценить заранее, поскольку оно в значительной степени зависит от динамики поведения прикладных задач.

Возможность масштабирования системы определяется не только архитектурой аппаратных средств, но и зависит от заложенных свойств программного обеспечения. Простой переход, например, на более мощный процессор может привести к перегрузке других компонентов системы. Это означает, что действительно масштабируемая система должна быть сбалансирована по всем параметрам.

Совместимость и мобильность программного обеспечения. В настоящее время одним из наиболее важных факторов, определяющих современные тенденции в развитии информационных технологий, является ориентация компаний-поставщиков компьютерного оборудования на рынок прикладных программных средств. Это объясняется прежде всего тем, что для конечного пользователя в конце концов важно программное обеспечение, позволяющее решить его задачи, а не выбор той или иной аппаратной платформы. Переход от однородных сетей программно-совместимых компьютеров к построению неоднородных сетей, включающих компьютеры разных фирм-производителей, в корне изменил и точку зрения на саму сеть: из сравнительно простого средства обмена информацией она превратилась в средство интеграции отдельных ресурсов — мощную распределенную вычислительную систему, каждый элемент которой лучше всего соответствует требованиям конкретной прикладной задачи.

Этот переход выдвинул ряд новых требований:

Во-первых, такая вычислительная среда должна позволять гибко менять количество и состав аппаратных средств и программного обеспечения в соответствии с меняющимися требованиями решаемых задач.

Во-вторых, она должна обеспечивать возможность запуска одних и тех же программных систем на различных аппаратных платформах, т. е. обеспечивать мобильность программного обеспечения.

В-третьих, эта среда должна гарантировать возможность применения одних и тех же человеко-машинных интерфейсов на всех компьютерах, входящих в неоднородную сеть.

2. Декомпозиция

Выделение классов и объектов – одна из самых сложных задач объектно-ориентированного проектирования, которая осуществляется в процессе декомпозиции ключевых абстракций программной системы.

Декомпозиция занимает центральное место в объектно-ориентированном анализе и проектировании программного обеспечения. Под объектно-ориентированной декомпозицией понимается процесс разбиения системы на части, соответствующие объектам предметной области. Правильное разделение системы на составные части, представление предметной области в виде набора объектов, или разбиение программы на модули является почти такой же сложной задачей, как выбор правильного набора абстракций.

Разработчики программного обеспечения активно пользуются следующим правилом при выделении модулей программ: «Особенности системы, подверженные изменениям, следует скрывать в отдельных модулях; в качестве межмодульных можно использовать только те элементы, вероятность изменения которых мала». Также необходимо отметить, что поскольку модули служат элементарными и неделимыми блоками программы, которые могут использоваться повторно, это должно учитываться при распределении классов и объектов по модулям.

С проблемой декомпозиции исследователи ранее сталкивались при проектировании и анализе сложных административных, хозяйственных, производственных систем. В этих процессах очень часто непосредственное изучение объекта в целом как системы невозможно из-за его сложности. В этих случаях приходится расчленять объект на конечное число частей, учитывая связи между ними, характеризующие их взаимодействие. Здесь и начинается интерпретация исследуемого объекта как сложной системы, а его частей – как подсистем.

Если некоторые подсистемы оказываются все еще чрезмерно сложными, каждая из них разделяется (с сохранением связей) на конечное число более мелких подсистем. Процедура разделения подсистем продолжается до получения таких подсистем, которые в условиях данной задачи будут признаны достаточно простыми и удобными для непосредственного изучения. Эти подсистемы, не подлежащие дальнейшему расчленению, назовем элементами сложной системы.

Таким образом, в общем случае сложная система представляется как многоуровневая конструкция из взаимодействующих элементов, объединяемых в подсистемы различных уровней. Разделение системы на элементы в общем случае может быть выполнено неоднозначным образом и является в высшей степени условным.

Естественно возникает вопрос о правилах разделения сложных систем или декомпозиции в случае разработки объектно-ориентированной модели для задач, поставленных для заданной предметной области.

Синтез и анализ в изучении сложных систем

Разделение или декомпозиции применяется в случаях, когда рассматривается существующая система или проектируется новая. При этом применяются два фундаментальных понятия системологии: анализ (как было сформулировано – рассмотрение) и синтез (проектирование) систем. Задачи анализа определяются как изучение свойств и поведения системы в зависимости от ее структуры и значений параметров, исходя из заданных свойств системы; задачи синтеза сводятся к выбору структуры и значений параметров, исходя из заданных свойств системы.

В настоящее время не существуют формальных методов синтеза сложных систем. По-видимому, это в принципе невозможно, так как сложная система синтезируется для условий, которые невозможно описать конкретными математическими моделями в силу непрерывно изменяющегося многообразия условий. Поэтому повсеместно на практике применяется синтез через анализ, когда задается исходя из практических соображений некоторая структура системы, затем она анализируется, затем изменяются ее параметры или модифицируется структура, затем осуществляется анализ и так далее до достижения необходимого результата.

Другими словами, при разработке достаточно сложного программного обеспечения решается задача синтеза методами синтеза через анализ. С одной стороны, задача создания новой структуры или задача изобретения новых абстракций является

творческой, с другой стороны, существуют общие правила, помогающие в этом процессе.

Обобщенное правило декомпозиции

Зададим в общем виде функционал эффективности системы в виде следующего выражения: $A = F(a_1, \dots, a_n)$, где A – показатель, отражающий качество реализации основного назначения системы (например, некоего системного качества или эмергентного свойства), a_1, \dots, a_n – параметры, от которых зависит данное системное качество (они также могут быть функционально зависимыми от других параметров).

В качестве пояснения назначения функционала рассмотрим задачу увеличения высоты полета самолета. Пусть A – высота полета самолета. Первый вопрос, на который должен ответить человек, решающий данную задачу – от чего зависит высота полета? Для простоты рассмотрим два составляющих элемента самолета: двигатель, крылья, и, кроме того, саму систему – самолет. Выделим свойства, оказывающие влияние на заданное системное свойство, например, в двигателе – мощность (пусть это a), в крыльях – площадь (c), а в самолете его вес (b).

Значит, для увеличения системного качества A нужно рассмотреть возможности синтеза такой структуры рассмотренных элементов при которой значения показателей a, b, c приведут к требуемому результату. Это очевидный пример синтеза через анализ.

Конечно же, для увеличения высоты полета самолета необходимо рассматривать более сложный комплекс показателей. И более того, нельзя забывать о возможной функциональной зависимости данных параметров и о влиянии других (неучтенных) параметров. Вполне может оказаться так, что один из неучтенных параметров оказывает большее влияние на анализируемое системное качество A .

Таким образом, в общем виде правило декомпозиции заключается в выполнении следующих основных этапов:

1. Определение системных(ого) свойств(а), значимых (ого) для решения поставленной задачи. В примере – высота полета самолета.
2. Поиск составных частей системы и их свойств, оказывающих решающее влияние на выделенные системные свойства. В примере – двигатель (мощность), крылья (площадь), самолет (вес).

Декомпозиция может продолжаться до тех пор, пока не будут выделены достаточно простые составляющие рассматриваемой системы и её подсистемы. Здесь «простота» – является субъективным показателем, то есть каждый аналитик определяет сам для себя как долго необходимо выполнять разбиение системы и ее подсистем на составляющие элементы. Выделенные элементы и их упрощенное описание в виде набора свойств, необходимых для решения поставленной задачи представляет собой не что иное как абстракцию.

Особенности проектирования программных систем

Необходимо отметить, что сами по себе объекты не представляют интереса: только в процессе взаимодействия объектов реализуется система. Поведение системы определяется сочетанием состояний объектов, которые свою очередь задаются некоторым состоянием агрегируемых объектов. Поэтому на ранних стадиях проектирования внимание проектировщика сосредотачивается на внешних проявлениях системы, что позволяет создать ключевые абстракции и механизмы.

Такой подход создает логический каркас системы: структуру классов, представляющий объекты реальной системы в заданной абстракции. На последующих фазах проектирования (включая и реализацию – где происходит распределение классов

по модулям), внимание переключается на внутреннее поведение ключевых абстракций и механизмов, а также их физическое представление. Принимаемые в процессе проектирования решения задают архитектуру системы, архитектуру процессов и архитектуру модулей.

Архитектура – это логическая и физическая структура системы, сформированная всеми стратегическими и тактическими проектными решениями. Хорошей архитектуре присущие следующие основные черты:

- многоуровневая система абстракции;
- модульность;
- простота.

Многоуровневая система абстракции основывается на иерархии классов, максимально взаимосвязанных по горизонтали и минимально по вертикали. Каждый уровень абстракции «сотрудничает» по вертикали с другим посредством четкого интерфейса с внешним миром и основываются на столь же хорошо продуманных средствах нижнего уровня. На каждом уровне интерфейс абстракции строго ограничен от реализации. Реализацию можно изменять, не затрагивая при этом интерфейс. Изменяясь внутренне, абстракции продолжают соответствовать ожиданиям внешних клиентов.

Модульность – это свойство программной системы, которая была разделена на связные и слабо зацепленные между собой модули, реализующие заданные абстракции разрабатываемой системы. Модуль – это единица кода, служащая строительным блоком физической структуры системы; программный блок, который содержит объявления, выраженные в соответствие с требованиями языка и образующие физическую реализацию части или всех классов и объектов логического проекта системы. Как правило, модуль состоит из интерфейсной части и реализации.

Как уже отмечалось выше, связность – это степень взаимодействия между элементами отдельного модуля (а для объектно-ориентированной декомпозиции еще и отдельного класса или объекта). Наиболее желательной является функциональная связность, при которой все элементы класса или модуля тесно взаимодействуют в достижении определенной цели.

Для построения системы должен использоваться минимальный набор неизменяемых компонент; сами компоненты должны быть по возможности стандартизованы и рассматриваться в рамках единой модели. Применительно к объектно-ориентированному проектированию такими компонентами являются классы и объекты, отражающие ключевые абстракции системы, а единство обеспечивается соответствующими механизмами реализации.

Архитектура считается простой, если она не содержит ничего лишнего, а общее поведение системы достигается общими абстракциями и механизмами. Необходимо отметить, что не существует единственно – верного способа классификации абстракции и проектирования архитектуры.

Системологическое описание проектирования программных систем

На основе изложенных особенностей проектирования можно говорить о проектирование как о процедуре синтеза программной системы на основе анализа реальной системы. По результатам проектирования программная система представляет собой взаимосвязанный комплекс подсистем, где каждая подсистема в терминах объектно-ориентированного проектирования является модулем, представляемым либо объектом, либо группой взаимосвязанных объектов.

Здесь каждый объект является экземпляром заданной абстракции. Структурой системы являются устойчивые взаимосвязи между подсистемами. Деление системы на подсистемы и объекты является условным и зависит от объема и сложности самой системы.

Программная система имеет системные качества, или эмергентные свойства, отражающие основное ее назначение или основные функции, отсутствующие у входящих в неё подсистем и являющиеся интегральным результатом согласованного функционирования подсистем. Подсистемы, входящие в программную систему, также обладают системными качествами, или эмергентными свойствами, являющимися основным функциональным назначением подсистем. Агрегация одной подсистемы в другую формирует иерархию подсистем. Сочетание состояний объединенных подсистем формирует новые эмергентные свойства каждого из объединений.

Системологический подход, который, несомненно, лежит в основе объектно-ориентированного анализа, позволяет установить наиболее общие закономерности проектирования программных систем, представляющих реальные объекты и/или системы физического мира. Целенаправленное использование выявленных закономерностей при исследовании одних систем позволяет повысить эффективность анализа и синтеза других систем.

Так, в любом творческом процессе разработки системы есть общие похожие закономерности. В начале работы разработчик обосновывает необходимость, цель и варианты решений, на основе анализа существующих материальных и/или знаковых систем. Анализ системы начинается с изучения ее внешних проявлений и выделения системных или эмергентных свойств. Данные свойства формируют ключевые подсистемы, на основе которых формируется структура системы, обычно представляемая в виде иерархии подсистем. Такой подход создает логический каркас системы, задаваемый множеством связей между подсистемами и их свойствами, которые в результате формируют общее поведение системы и её системные свойства.

Следующий этап – синтез внутренней реализации каждой из выделенных подсистем. Здесь важно помнить, что любая подсистема должна иметь четко определенное и неизменное в процессе эксплуатации множество элементов управления (в терминах объектно-ориентированного проектирования – интерфейс) её поведением. Такая реализация подсистем позволяет влиять на эмергентные свойства системы путем замены одних внутренних реализаций выбранных подсистем на другие.

3. Система управления версиями

Система управления версиями (от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (*Software Configuration Management Tools*). Систем контролия версий существует довольно много, наиболее распространенные : RCS, CVS, Subversion, Aegis, Monoton, Git, Bazaar, Arch, Perforce, Mercurial, TFS.

В настоящий момент две наиболее используемые системы контроля версий — Git и Subversion (SVN). Принципиальная разница: GIT распределенная, а SVN — нет. Другими словами, если есть несколько разработчиков работающих с репозиторием у каждого на локальной машине будет ПОЛНАЯ копия этого репозитория. Разумеется есть и где-то центральная машина, с которой можно клонировать репозиторий. Это напоминает SVN. Основной плюс в том, что если вдруг у вас нет доступа к интернету, сохраняется возможность работать с репозиторием. Потом только один раз сделать синхронизацию и все остальные разработчики получат полную историю.

GIT сохраняет метаданные изменений, а SVN целые файлы. Это экономит место и время. Система создания branches, versions и прочее в GIT и SVN отличаются значительно. В GIT проще переключаться с ветки на ветку, делать слияние между ними. Недостаток Git — приватные репозитории являются платными, а в SVN есть возможность бесплатно создавать приватные репозитории с ограничением по количеству участников. Тем не менее, на данный момент более популярен Git, а также его форки, такие как, например, GitLab. Поэтому в данной лабораторной работе будет рассмотрена система управления версиями Git.

Установка.

Для установки необходимо скачать файл устанавливающий GIT систему на ваш компьютер. Ссылка для скачивания: <http://git-scm.com/downloads>

Итак, что же представляет из себя репозиторий? Это папка, хранящая весь ваш проект. Репозиторий есть локальный (на вашем компьютере) и удаленный (на сервере). Именно с удаленного сервера в дальнейшем берется вся информация для выпуска конечного продукта. Изменения в репозиторий разработчики добавляют с помощью системы ветвления. В качестве примера мы разработаем простейшую программу с использованием базовых возможностей Git.

В данной лабораторной работе будут приведены примеры работы со стандартным графическим интерфейсом git gtk, а также для каждой команды будут приведены аналоги для консоли. Зачем это нужно? Порой разработчикам, в частности веб-разработчикам, приходится вносить изменения в файлах на удаленном компьютере, поэтому для экономии ресурсов компьютера и более быстрой работы используется подключение через консоль. Вы увидите, что в ряде случаев использование консольной версии Git оказывается более удобным нежели его графическая оболочка. Поэтому под каждым скриншотом будет соответствующая консольная команда. Работать будем с английской версией поскольку большинство информации в сети именно на английском, в случае возникновения проблем решение найти будет проще.

Установите git. Создайте пустую папку проекта. Затем внутри этой папки нажмите правую кнопку мыши и выберете пункт Git GUI Here. Для открытия консоли — Git Bash Here (Рис. 1, 2).

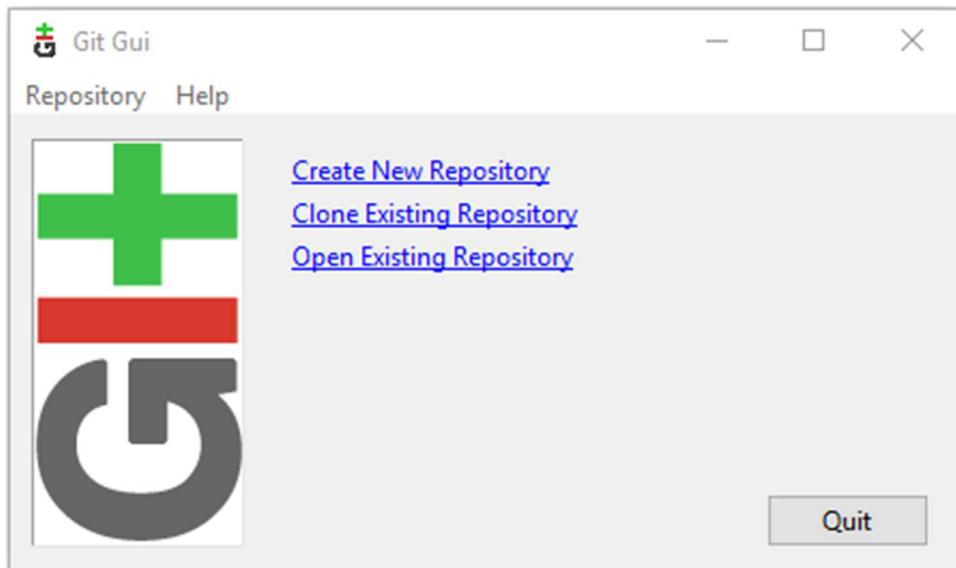


Рисунок 1. Создание пустой папки проекта

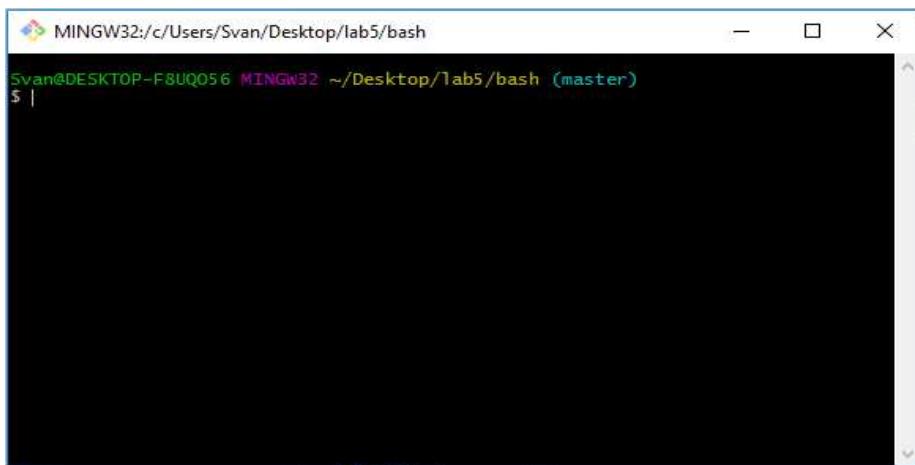


Рисунок 2. Создание пустой папки проекта на консоли

Нажмите Create New Repository, и через кнопку Browse выберите текущую папку.
В ней создастся пустой репозиторий Git (Рис 3.).

Консоль: **git init**

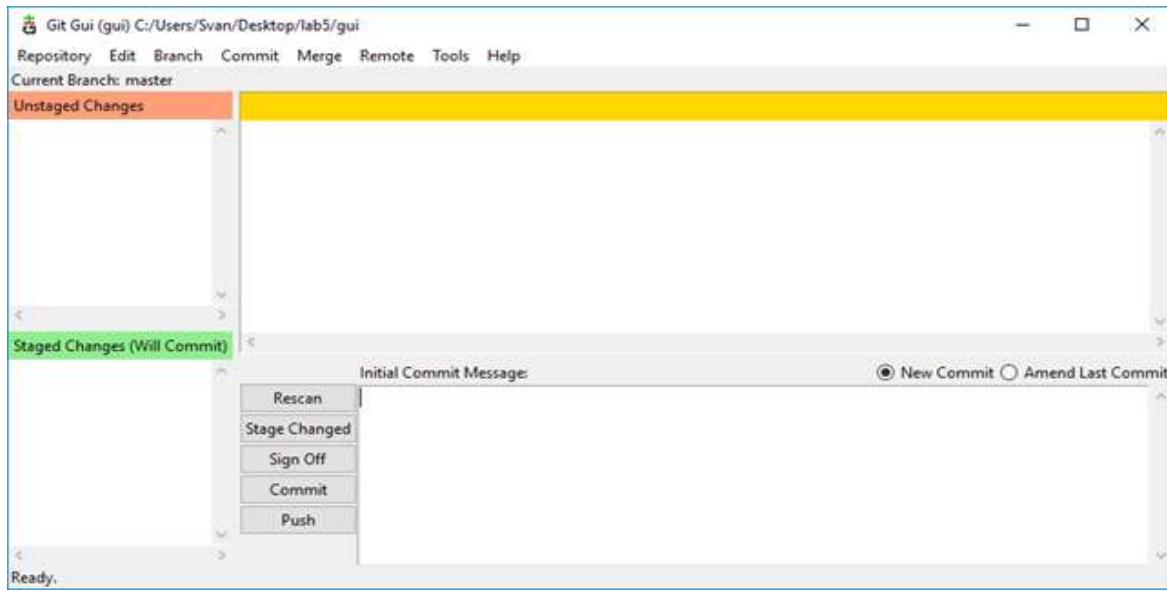


Рисунок 3. Создание пустого репозитория

В настоящий момент мы находимся на ветке master. Эта ветка по умолчанию создается в Git. Добавим подпись к нашему репозиторию чтобы остальные разработчики знали кто вы. Для этого выберем пункт Edit – Options. В нашем случае (Рис. 4) удобно будет сделать универсальную подпись для всех репозиториев, поэтому заполним соответствующие поля в правой колонке (User Name и Email Address).

На консоли:

```
git config -- global user.name "Student"
git config -- global user.email yourmail@yourmail.ru
```

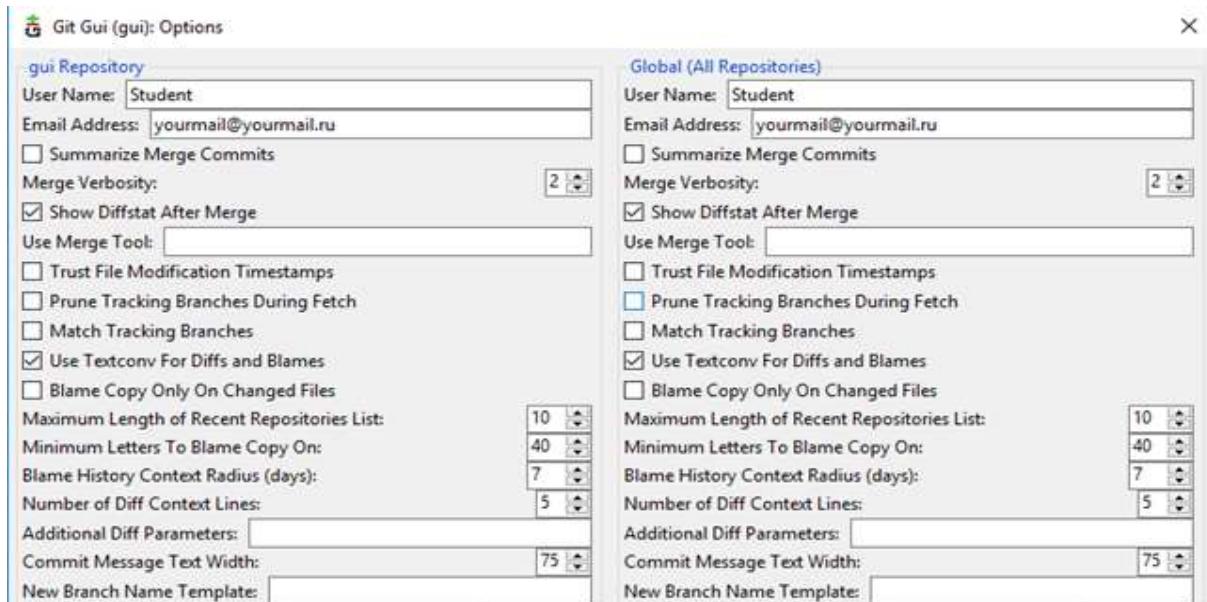


Рисунок 4. Создание подписей репозитория

Добавим в нашу папку файл main.cpp со следующим содержанием:

```
#include <iostream>
int main() {
    std::cout << "Hello world!\n";
    system("pause");
    return 0;
}
```

Нажмем кнопку Rescan и увидим появившийся файл в колонке Unstaged Changes. В этой колонке будут появляться все измененные файлы в репозитории (Рис. 5).

Консоль: git status

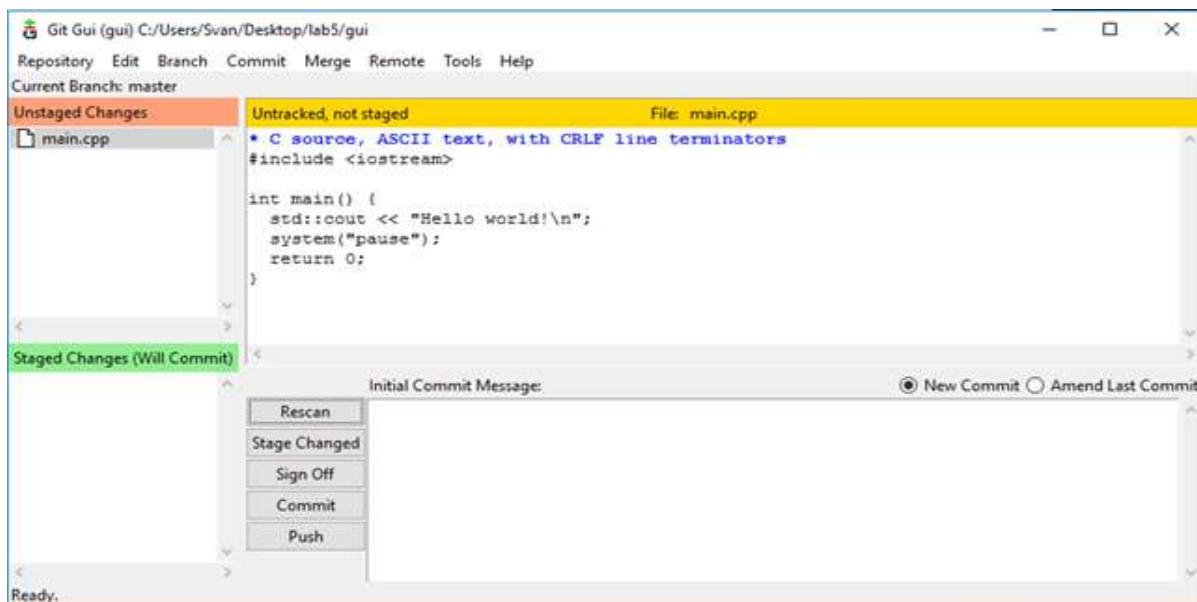


Рисунок 5. Добавление в папку файла main.cpp

Однако не всегда бывает надо запомнить изменения всех файлов. Чтобы Git добавил этот файл в список сохраняемых (в индекс) нужно нажать на иконку слева от названия файла. Файл переместится в зеленую зону Staged Changes.

На консоли:

```
git add (если нужно добавить все файлы в индекс).
git add filename (если нужно добавить конкретный файл).
```

Сохраним в репозитории наше первое изменение, иначе — сделаем коммит. Для этого в окне Initial Commit Message добавим описание коммита и нажмем кнопку Commit (Рис. 6). На консоли: **git commit -m "init"**

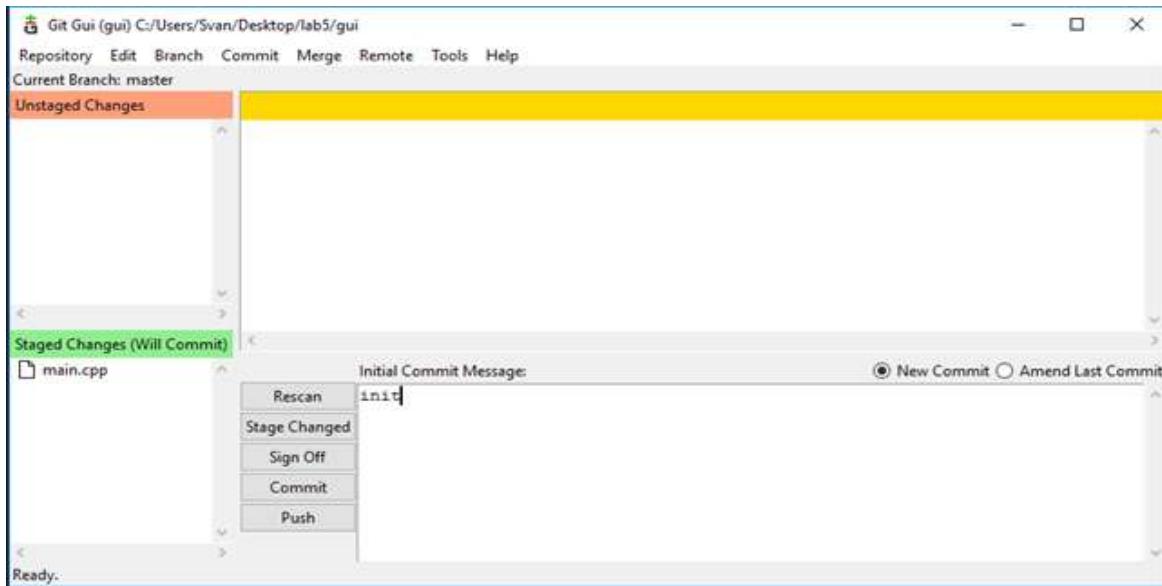


Рисунок 6. Первое изменение – коммит

Изменения сохранены. Но работать в ветке master небезопасно, поскольку там может оказаться неисправный код. Поэтому мы создадим ветку dev, в которой будет содержаться бета-версия нашей программы, и ветку student, в которой мы и будем работать. На практике разработчики используют для работы свои ветки, потом делают слияние в ветку dev, продукт из ветки dev тестируется и только после этого ветка dev вливается в master. Далее мы это рассмотрим. А пока создаем ветку dev. Для этого выбираем Branch – Create. (консоль: **git checkout -b dev**). Аналогично создадим ветку student (консоль: **git checkout -b student**) (Рис. 7).

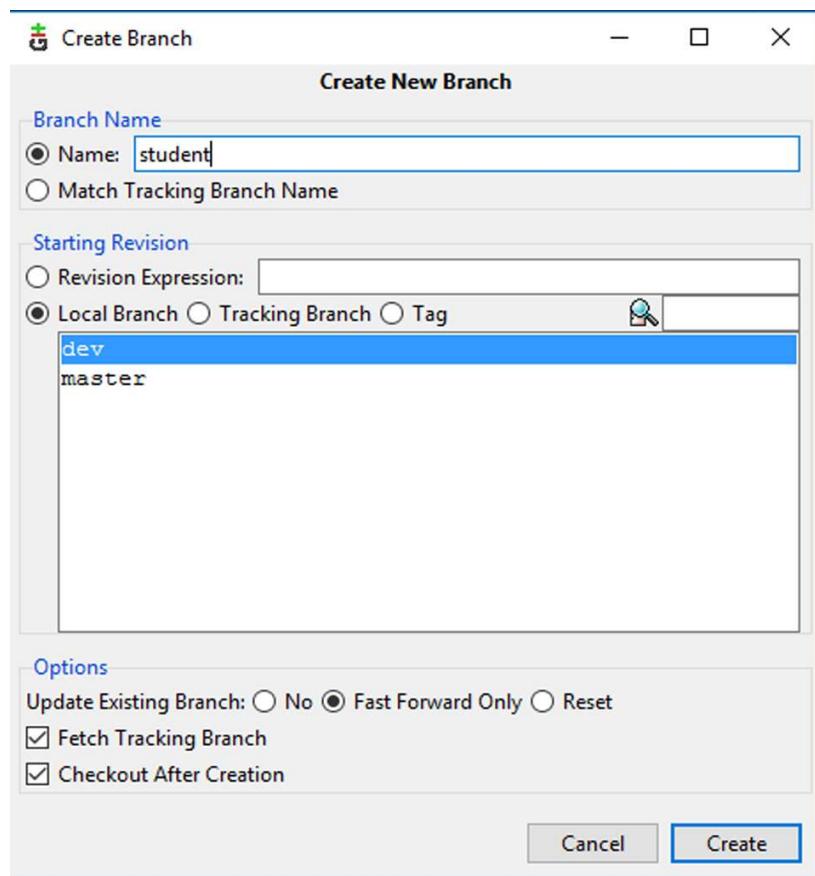
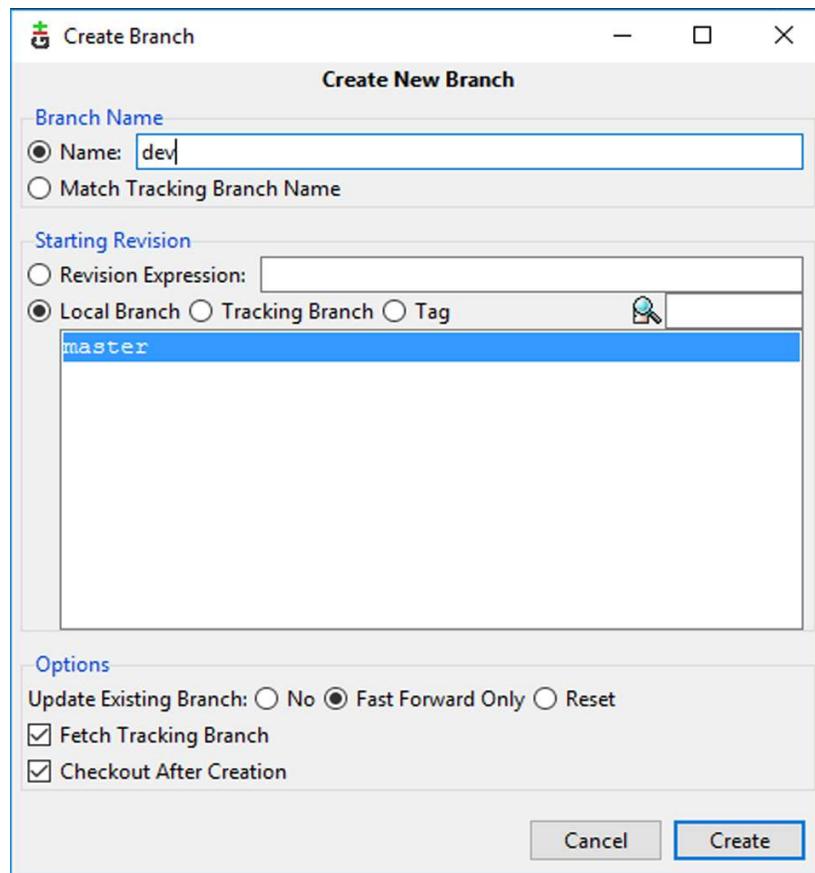


Рисунок 7. Создание рабочих веток dev и student

Теперь, находясь в рабочей ветке, мы можем спокойно вносить изменения в нашу программу (Рис. 8). Изменим main.cpp:

```
#include <iostream>
using namespace std;
int main() {
    char name[20];
    cout << "Enter your name: ";
    cin >> name;
    cout << "Hello, " << name << "!\n";
    system("pause");
    return 0;
}
```

Нажимаем Rescan
(Консоль: **git status**)

И видим:

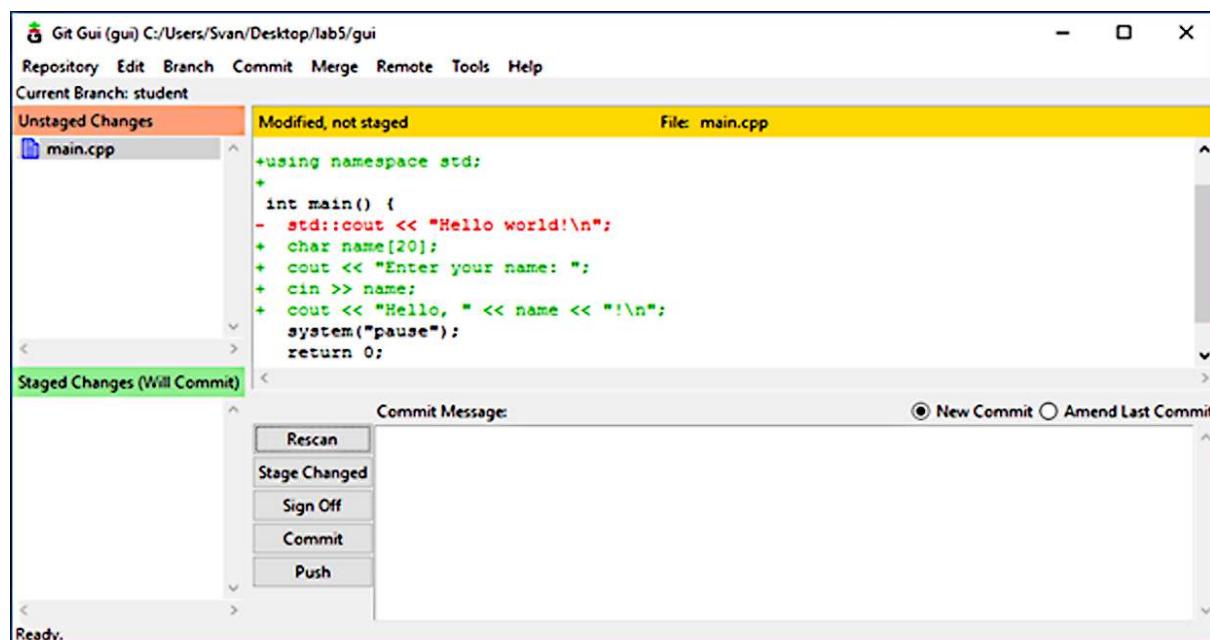


Рисунок 8. Внесение изменений в текст программы

Система заметила наши изменения и отобразила их красным цветом. Добавим файлы в индекс и сделаем коммит (Рис. 9).

Консоль:

```
git add .
git commit -m "added student name input"
```

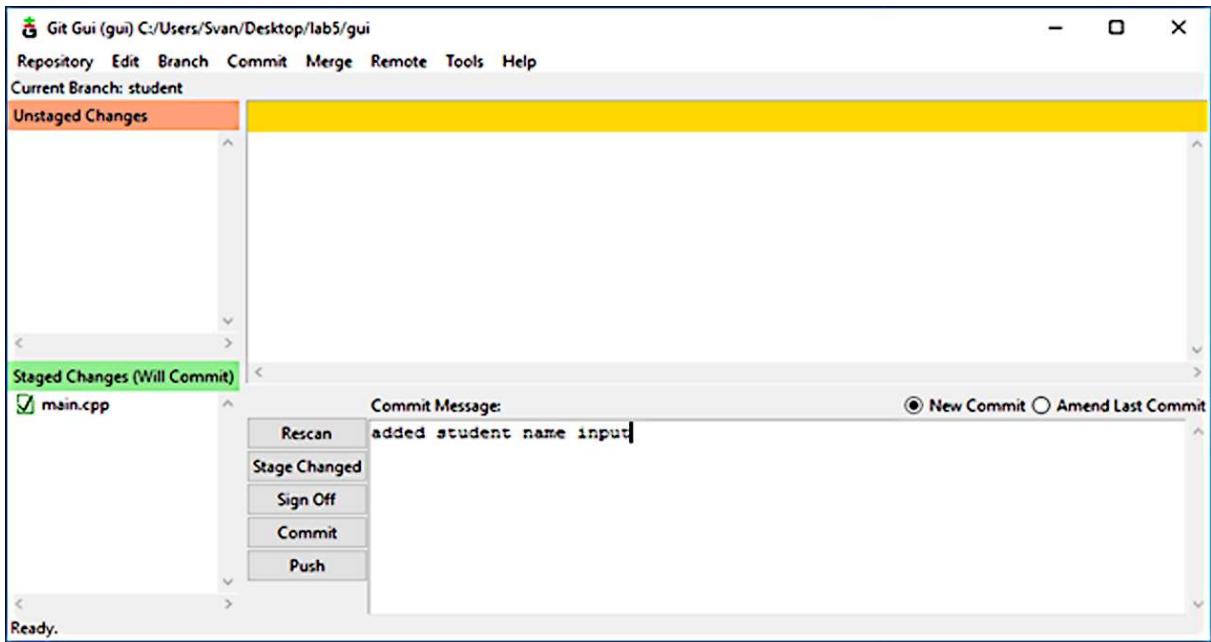


Рисунок 9. Делаем коммит

Внесем еще изменения в main.cpp:

```
#include <iostream>
using namespace std;
int main() {
    char name[20];
    cout << "Enter your name: ";
    cin >> name;
    cout << "Hello, " << name << "\n";
    cout << "Nice to see you learning Git\n";
    system("pause");
    return 0;
}
```

Аналогично сделаем коммит с описанием "added greeting Git info"

Консоль:

```
git add .
git commit -m "added greeting Git info"
```

Представим, что тестовую ветку dev сейчас смотрит клиент и хочет чтобы кроме приветствия миру выводилось еще и пожелание хорошего дня. Что ж, желание клиента – закон, переходим на ветку dev и меняем содержимое файла main.cpp. Для этого выбираем Branch – Checkout и выбираем ветку dev (Рисунок 10).

Консоль:

```
git add .
git commit -m "added having a nice day wish"
```

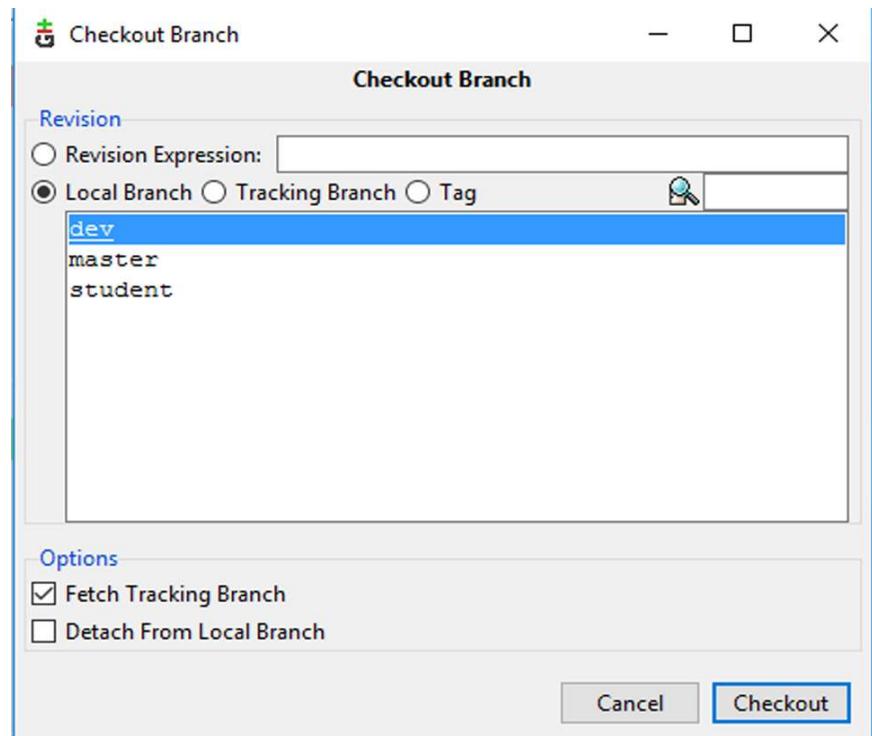


Рисунок 10. Выбор ветки dev для внесения изменений

Git предоставляет возможность посмотреть всю историю коммитов, иначе – лог. Для этого выбираем Repository – Visualize All Branch History (Рис. 11).

Консоль: `git log --graph --all`

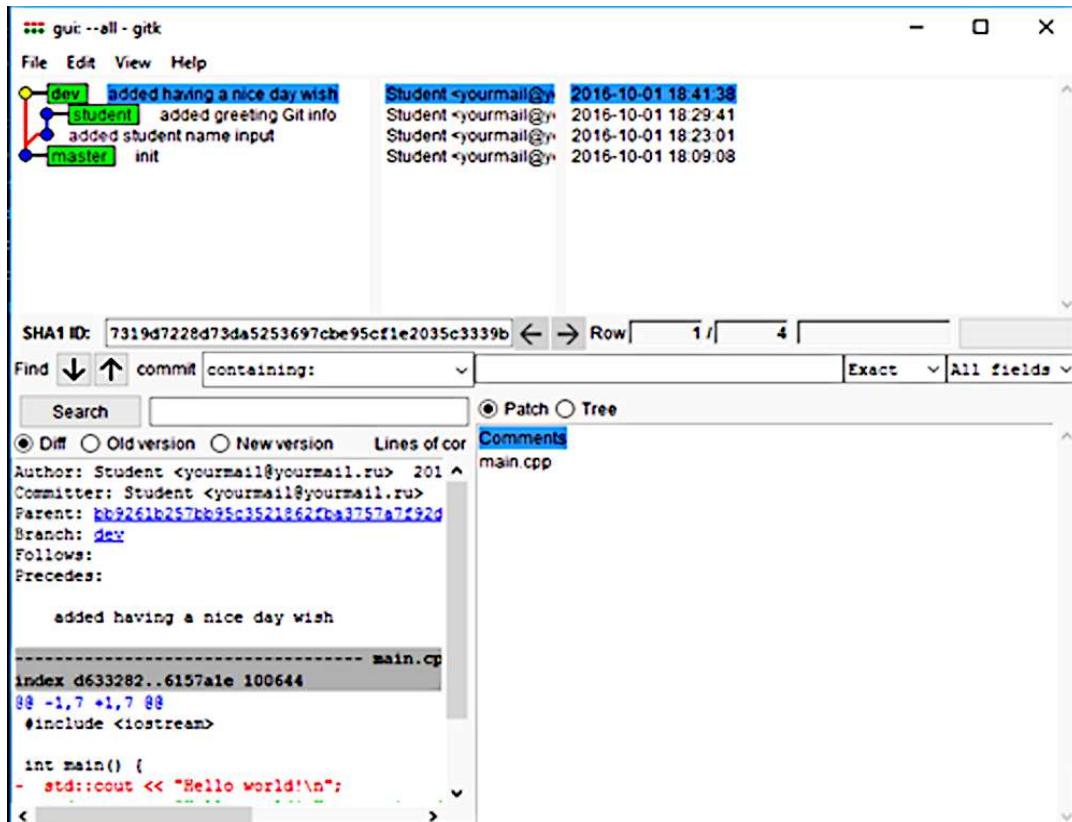


Рисунок 11. История коммитов

Попробуем теперь влить изменения с ветки student в ветку dev, другими словами – сделать merge в dev из student. Выбираем Merge – Local Merge (Рис.12)

Консоль: git merge student

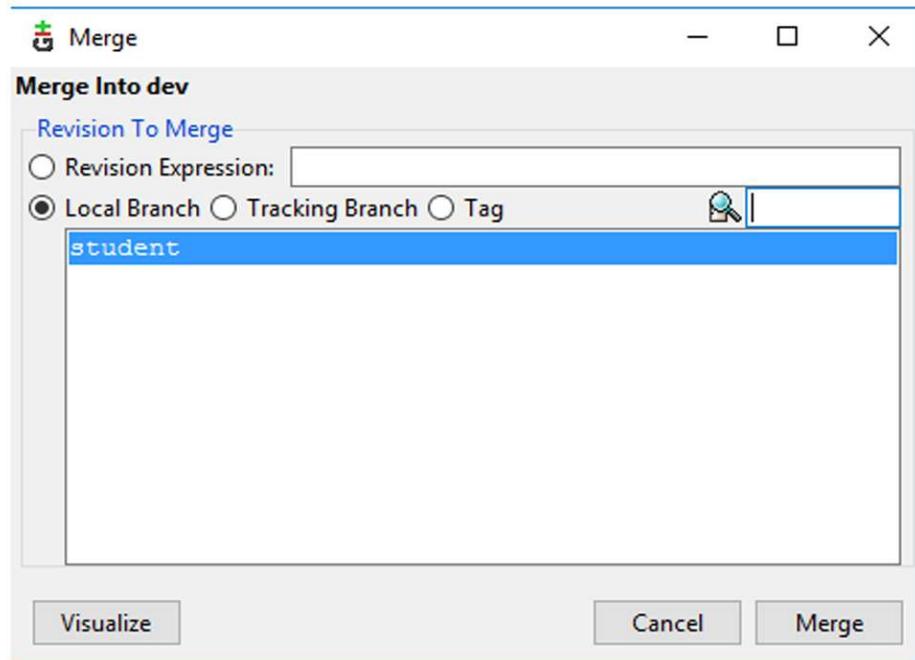


Рисунок 12. Делаем merge в dev из student

Но при попытке выполнить слияние возник конфликт (Рис. 13):

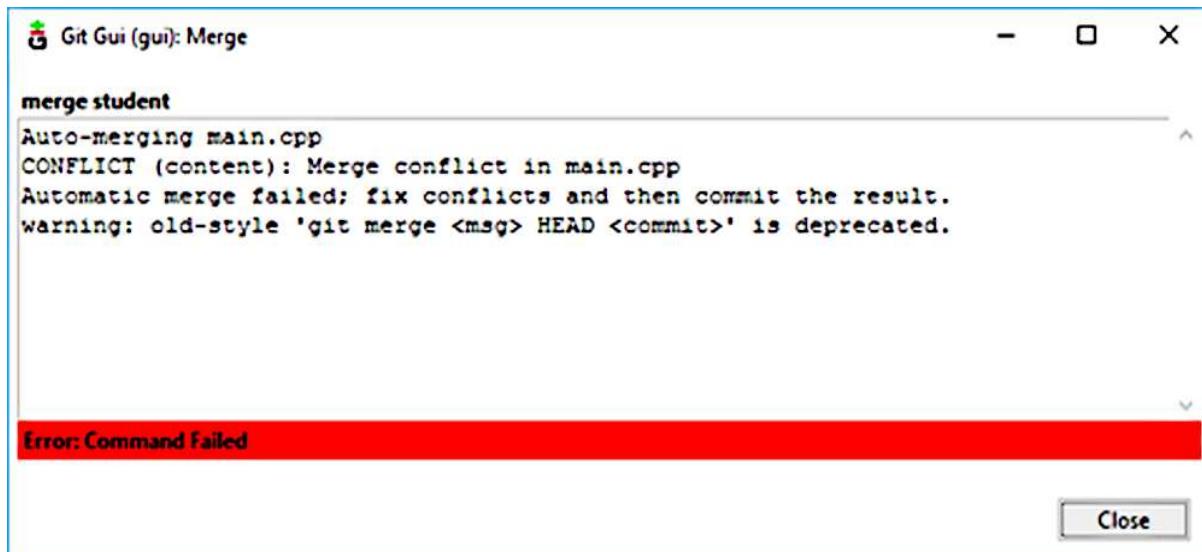


Рисунок 13. При попытке выполнить слияние возник конфликт

Тут мы дошли до самой частой ошибки разработчиков. Дело в том, что мы изменили файл main.cpp сначала в ветке student, а потом в ветке dev никак не синхронизировав изменения между собой. Поэтому Git не знает какие изменения

оставлять, а какие удалять. Этот конфликт нужно решить. Для этого откроем main.cpp и заменим его содержимое следующим:

```
#include <iostream>
using namespace std;
int main() {
    char name[20];
    std::cout << "Hello world! Have a nice day!\n";
    cout << "Enter your name: ";
    cin >> name;
    cout << "Hello, " << name << "!\n";
    cout << "Nice to see you learning Git\n";
    system("pause");
    return 0;
}
```

Мы решили конфликт, добавив вручную изменения из ветки student в наш файл, находящийся на ветке dev. Теперь нужно сделать коммит, дающий понять Git'у, что конфликт решен (Рис. 14).

Консоль: **git commit -m "conflict resolved"**

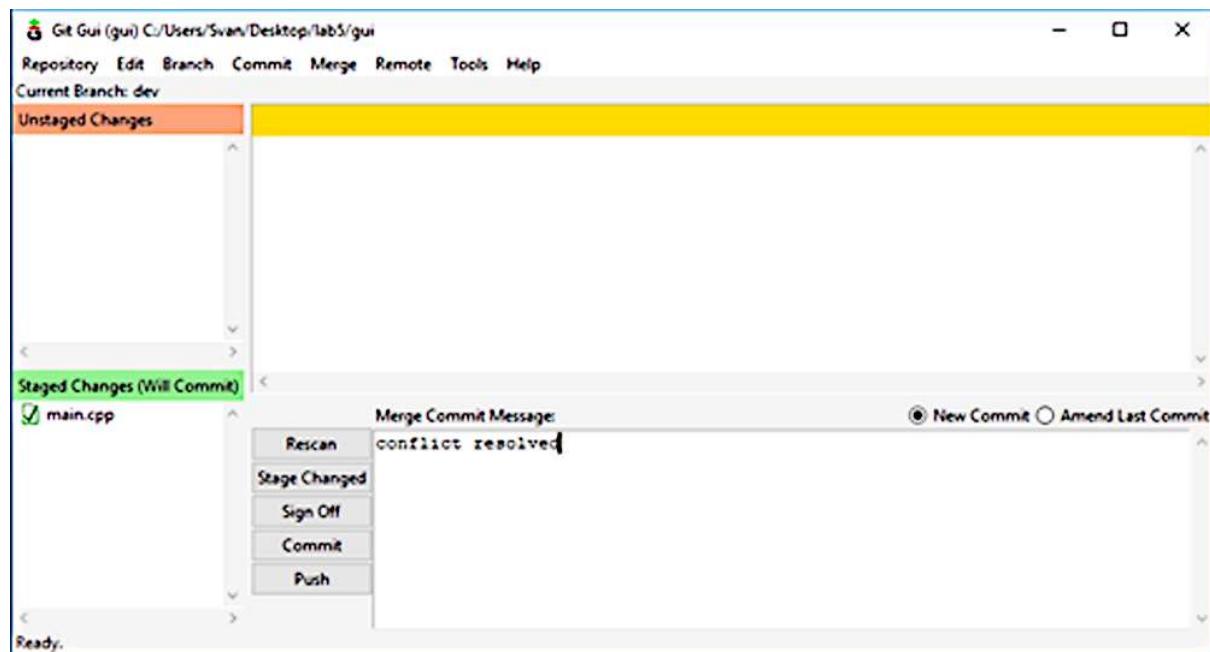


Рисунок 14. Делаем коммит, дающий понять Git'у, что конфликт решен

После этого делаем повторное слияние веток dev и student. На этот раз все проходит успешно. Можем проследить это на нашем дереве коммитов (Рис. 15).

Консоль: **git log --graph --all**

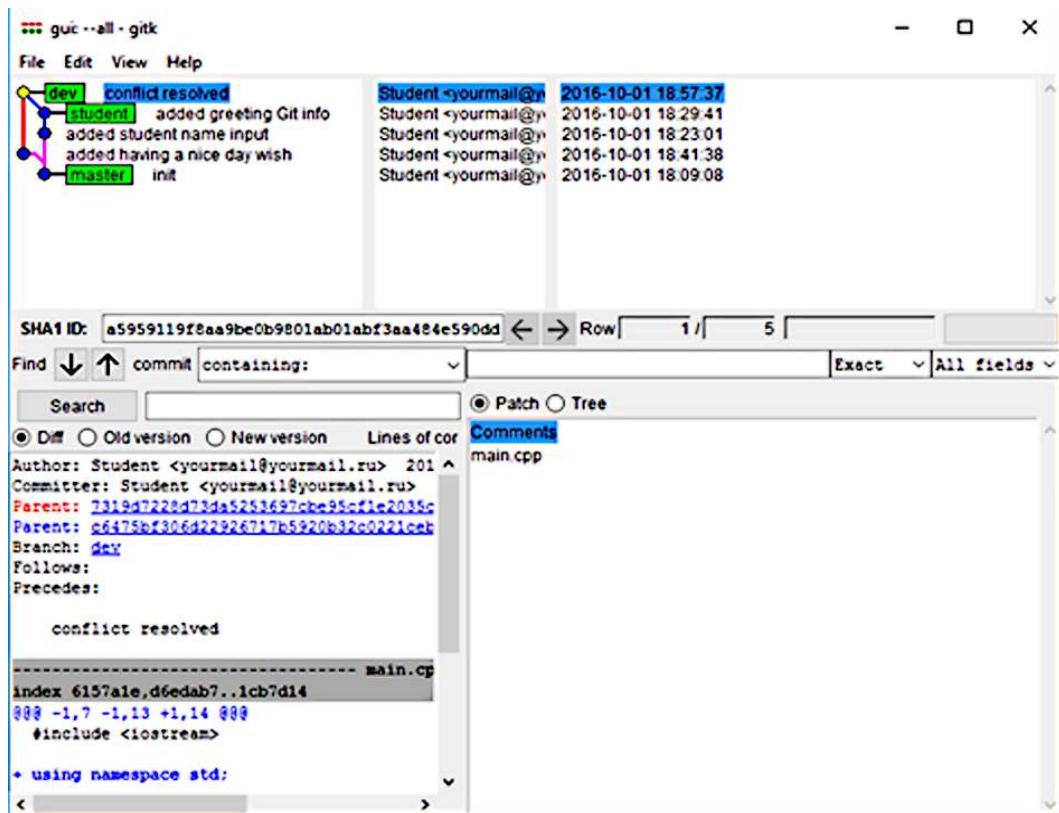


Рисунок 15. Повторное слияние веток dev и student

Проблем не замечено, можем сливать ветку dev в ветку master. Для этого переходим в ветку master через Branch – Checkout и после этого Merge – Local Merge

Консоль:

git checkout master

git merge dev

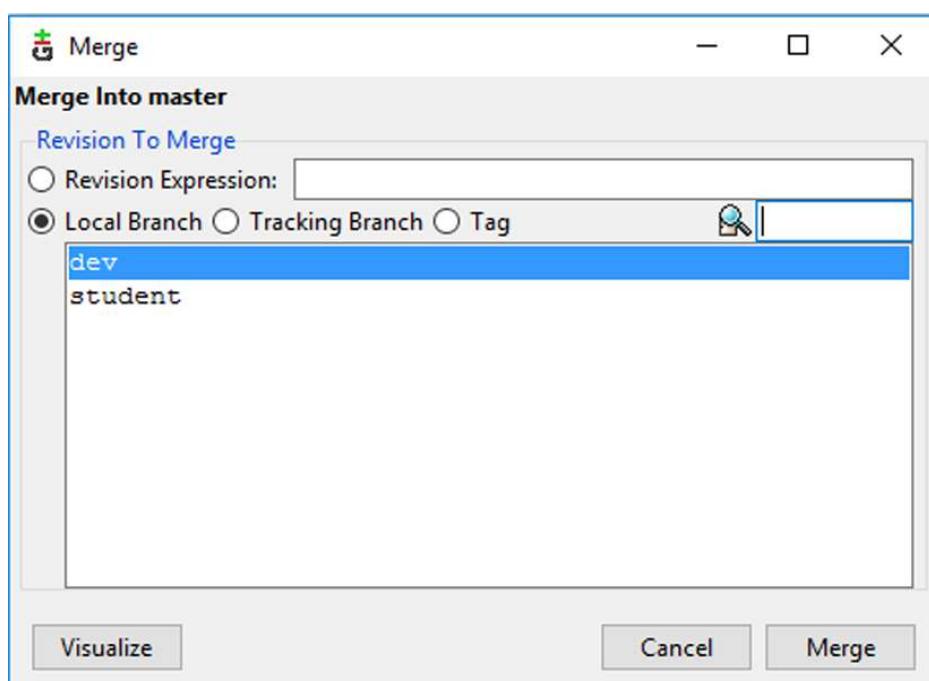


Рисунок 16. Ветка dev – в ветку master

Половина работы проделана. Теперь нужно вылить весь наш локальный репозиторий на удаленный. Для этого нужно перейти на github.com, создать или войти в свою учетную запись. После этого создать репозиторий, выбрав соответствующий пункт в верхнем меню (Рис. 17).



Рисунок 17. Создать удаленного репозитория

Создав новый репозиторий, вы попадете на стартовую страницу, где есть интересующая нас ссылка HTTPS (Рис. 18).

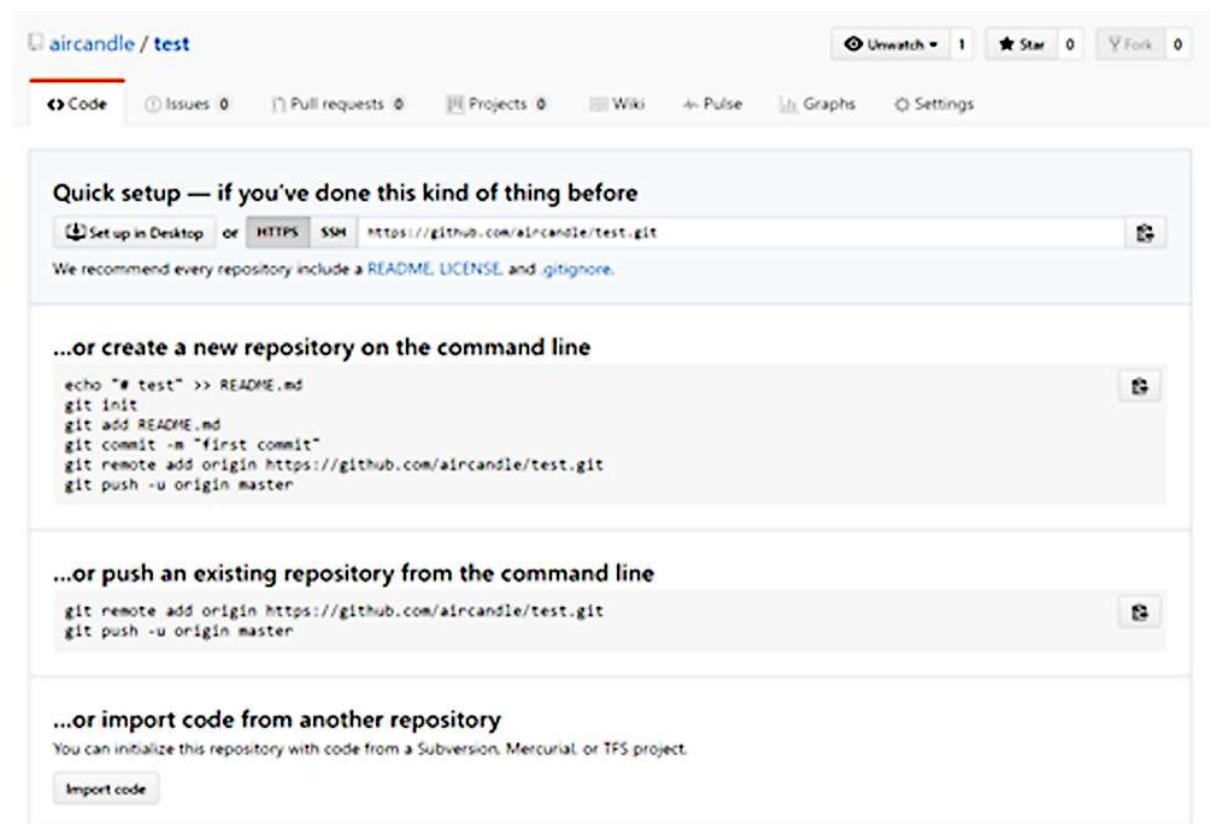


Рисунок 18. Стартовая страница со ссылкой HTTPS

Копируем эту ссылку и возвращаемся в Git GUI. Выбираем пункт Remote – Push, выделяем все наши ветки, вставляем в строку Arbitrary Location ссылку к нашему удаленному репозиторию и жмем кнопку Push (Рис. 19).

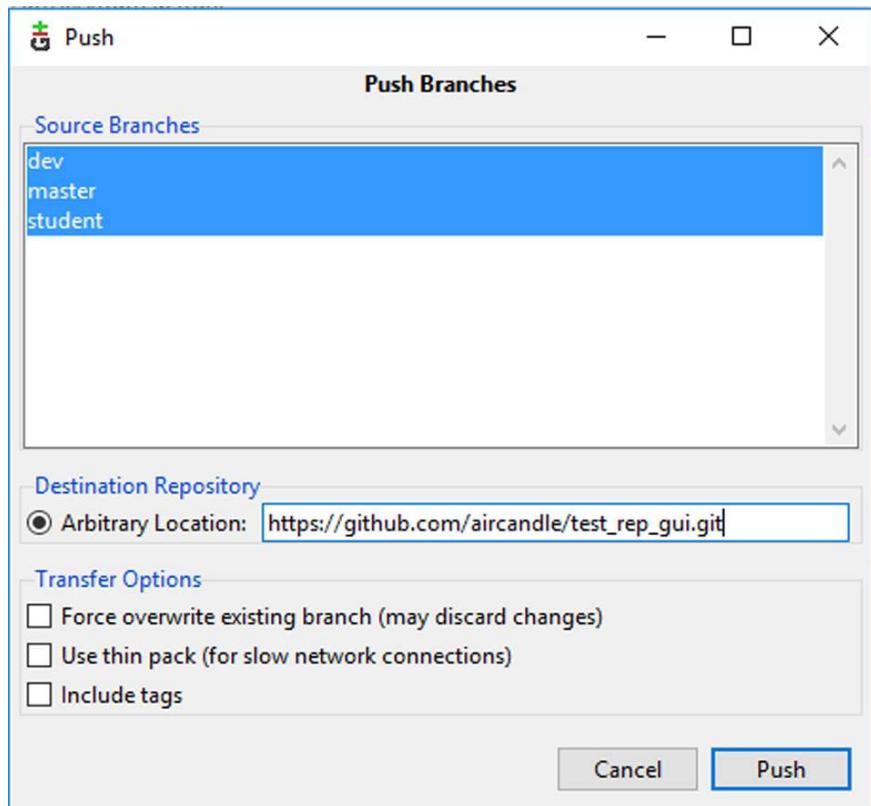


Рисунок 19. Вставляем ссылку на удаленный репозиторий

На практике удаленный репозиторий обычно добавляется в список, но мы это сделаем позже. А пока дожидаемся сообщения об успешной отправке нашего локального репозитория (Рис. 20).

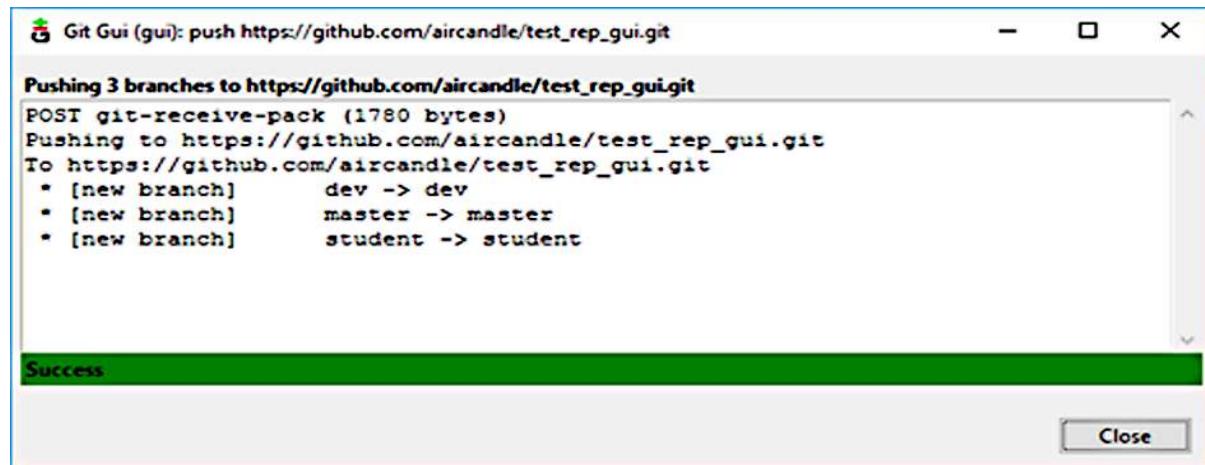


Рисунок 20. Отправка на удаленный репозиторий

Для проверки можем перейти в браузере на `github` и обновить страницу с репозиторием. Мы увидим, что все наши ветки и коммиты успешно перенесены.

Но обычно мы далеко не всегда первыми создаем репозитории, намного чаще возникает необходимость подключиться к уже существующему. Поэтому сейчас мы рассмотрим случай, когда у нас имеется только удаленный репозиторий и нам необходимо внести в нем правки. Для этого перейдем в директорию, в которой будет расположена папка с содержимым нашего проекта. Важный момент: если вы работаете

именно с Git GUI, то вы НЕ ДОЛЖНЫ создавать папку для вашего проекта. Git GUI создает ее самостоятельно, это особенность данного графического клиента. Если же вы предпочтете работать с консолью, то все понятнее и проще: создаете папку с проектом, там щелчок правой кнопкой мыши – Git Bash Here и спокойно выполняете все необходимые команды.

В нашем случае возникла необходимость изменить файл main.cpp, поскольку мы указали пространство имен и std::cout нужно заменить на cout.

Итак, в нужной папке кликаем правой кнопкой мышки – Git GUI Here. Выбираем пункт Clone Existing Repository, открывается диалоговое окно (Рис. 21).

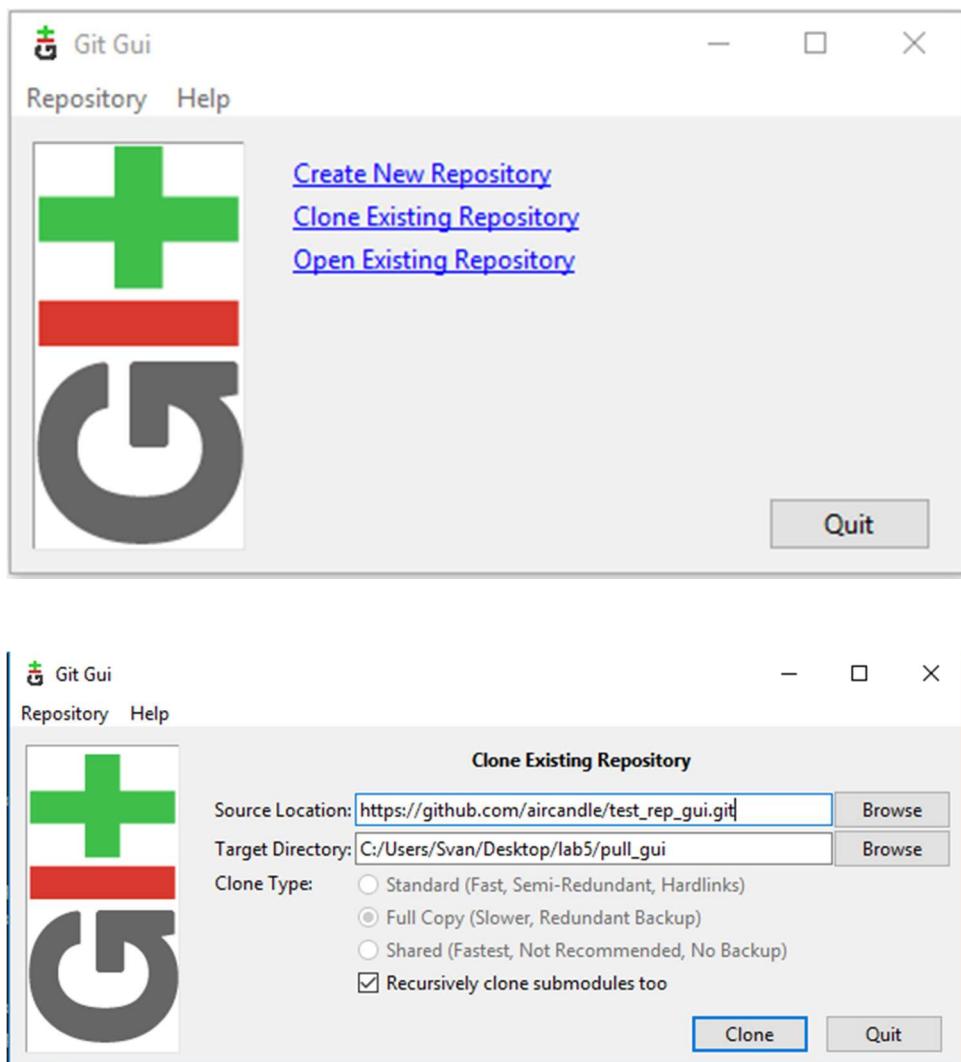


Рисунок 21. Размещение на удаленный репозиторий

В строку Source Location вставьте ссылку на удаленный репозиторий, в поле Target Directory через кнопку Browse выберите текущую папку. Как только путь к папке появится в строке вручную добавьте путь к новой НЕСУЩЕСТВУЮЩЕЙ папке с вашей программой. Например, здесь Git GUI вызывался из папки lab5, вручную необходимо было дописать /pull_gui. После этого нажимаем кнопку Clone. Git подгрузит все данные из удаленного репозитория, создаст локальный репозиторий и синхронизирует локальный с удаленным.

Консольный аналог:

```
git init  
git remote add origin https://github.com/aircandle/test\_rep\_gui.git  
git pull
```

Мы автоматом попадем на ветку master. Но, как мы помним, в этой ветке коммитить небезопасно, как и в ветке dev. Поэтому переходим в Branch – New Branch (Рис. 22).

Консоль:

```
git checkout master  
git checkout dev  
git checkout student  
git pull origin dev
```

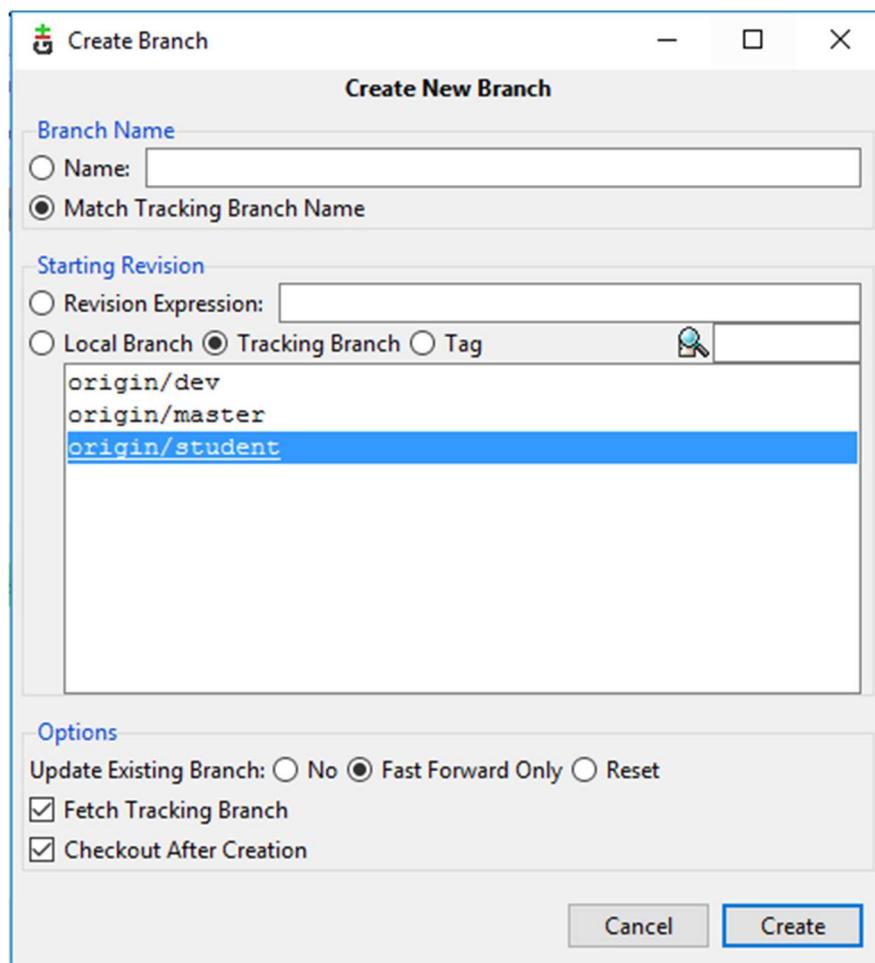


Рисунок 22. Переходим в Branch – New Branch

Вместо Local Branch ставим Tracking Branch, таким образом мы перейдем на локальную ветку, которая связана с удаленной. Если же мы оставим Local Branch, то создастся обычная локальная ветка.

Итак, мы на ветке student, меняем файл main.cpp:

```
#include <iostream>  
using namespace std;  
int main() {
```

```

char name[20];
cout << "Hello world! Have a nice day!\n";
cout << "Enter your name: ";
cin >> name;
cout << "Hello, " << name << "!\n";
cout << "Nice to see you learning Git\n";
system("pause");
return 0;
}

```

Делаем Rescan, добавляем файл в индекс и коммитим с сообщением "fixed namespaces". В консоли это будет так:

```

git add .
git commit -m "fixed namespaces"

```

Далее выполняем уже знакомые операции: переходим на dev, делаем merge с веткой student. Далее аналогично влияем изменения из dev в master

```

git checkout dev
git merge student
git checkout master
git merge dev

```

После этого осталось только синхронизировать удаленный репозиторий с локальным. Для этого мы добавим удаленный репозиторий в список чтобы дальше было удобнее с ним работать. Выбираем Remote – Add (Рис. 23).

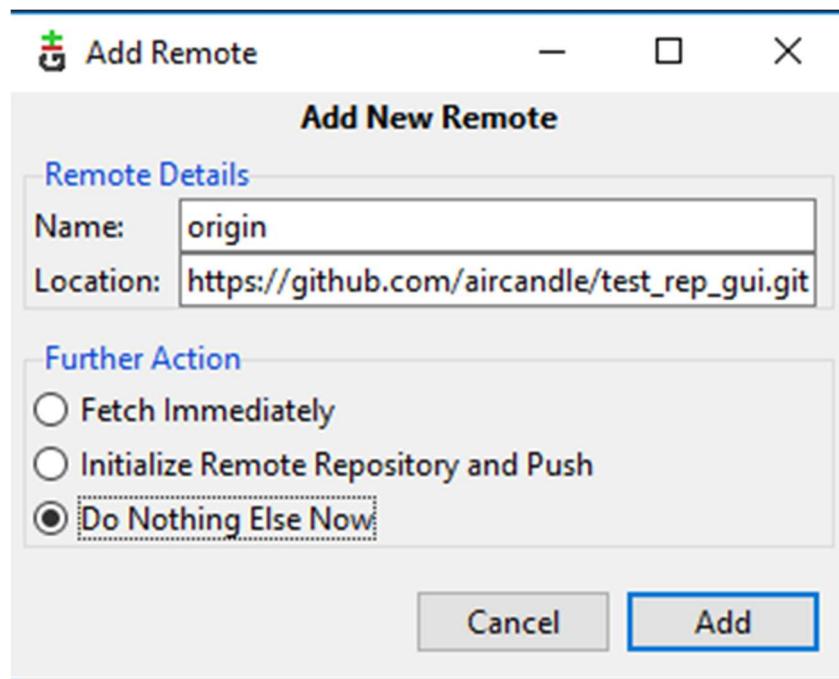


Рисунок 22. Синхронизация локального и удаленного репозиториев

В поле Name вводим сокращенное название для нашего репозитория. Поскольку, как уже говорилось ранее, Git GUI является лишь оболочкой для консольной версии, то

Name используется для более удобного и быстрого способа выполнить команду git push (в консольной версии мы ввели название репозитория на этапе git remote add). Название можно выбрать любое, но негласным стандартом считается название origin. В поле Location вставляем ссылку к нашему удаленному репозиторию, выбираем пункт Do Nothing Else Now и жмем кнопку Add.

Теперь выполняем Remote – Push, выделяем все ветки и жмем Push (рис. 23).

Консоль git push

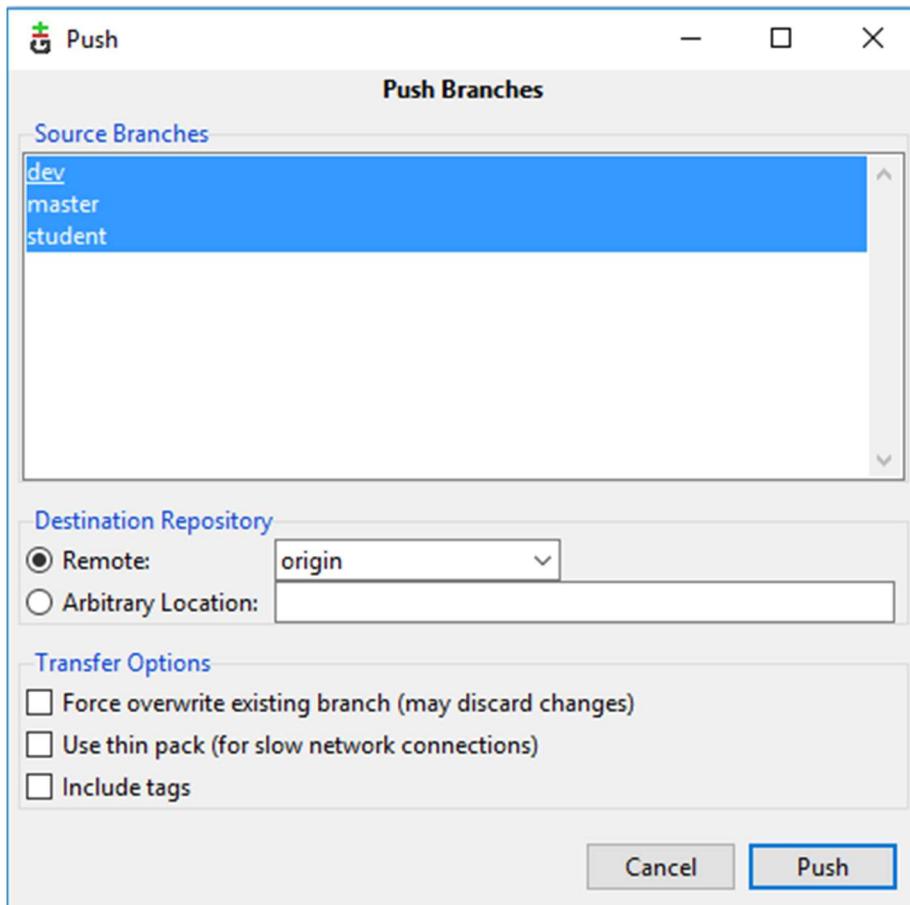


Рисунок 23. Синхронизация локального и удаленного репозиториев

Изменения вылиты на сервер. Посмотрим итоговое дерево коммитов (Рис. 24).

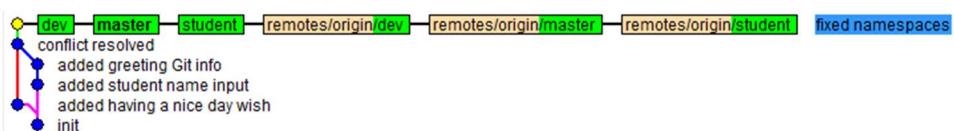


Рисунок 24. Синхронизация локального и удаленного репозиториев

Все ветки находятся в одном месте и синхронизированы между собой.
На рис. 25 показано как выглядит дерево коммитов на консоли.

```
MINGW32:/c/Users/Svan/Desktop/lab5/pull_bash
* commit 63185818d6de89373d0081def945f5294918e9ab
Author: Student <yourmail@yourmail.ru>
Date:   Sat Oct 1 20:40:54 2016 +0300

    fixed namespaces

* commit 2657bd9c12374761305336563527da6d369e7f2a
Merge: 788909a b3cb1f6
Author: Student <yourmail@yourmail.ru>
Date:   Sat Oct 1 18:55:42 2016 +0300

    conflict resolved

* commit b3cb1f6e9708d51aeac6cb0a0601e5e437a9c7a3
Author: Student <yourmail@yourmail.ru>
Date:   Sat Oct 1 18:28:22 2016 +0300

    added greeting Git info

* commit a18ed7457a96da8c6b1fe8c8078d68ce0382102b
Author: Student <yourmail@yourmail.ru>
Date:   Sat Oct 1 18:20:51 2016 +0300

    added student name input

* commit 788909ad38a97254b458c428ddf34b6d61464c18
Author: Student <yourmail@yourmail.ru>
Date:   Sat Oct 1 18:39:12 2016 +0300

    added having a nice day wish

* commit 7549aa14504ee90d6303a1dd499743ec9444bba9
Author: Student <yourmail@yourmail.ru>
Date:   Sat Oct 1 18:05:15 2016 +0300
:|
```

Рисунок 24. Вид дерева коммитов на консоли

P.S. Если вы захотите в дальнейшей разработке использовать графический клиент для Git, то лучше использовать другие клиенты, нежели Git GUI. Например, достаточно удобны клиенты GitHub for Windows/Mac или же SourceTree. Git GUI для полноценной разработки не самое удобное решение, без возможности решать конфликты, к тому же некоторый функционал консольной версии в нем не реализован, например git pull. Кстати говоря о консоли, в некоторых случаях, например, для клонирования существующего репозитория, консольный git понятнее и удобнее. Поэтому рекомендуется использовать графические клиенты главным образом для решения конфликтов и наглядности при слиянии веток.

4. Порядок выполнения лабораторной работы.

1. Составить перечень технологий, используемых для реализации варианта задания
2. Обосновать выбор данных технологий и удобство их использования
3. Декомпозировать разрабатываемую систему, описать модули, необходимые для реализации ПО согласно заданию.
4. Сформулировать набор задач, необходимых для реализации ПО согласно варианту задания, определить порядок выполнения и приоритет каждой из них.
5. Если система контроля версий Git не установлена, то установить ее (параметры оставить по умолчанию).
6. Запустить Git GUI (или консоль). Создать новый репозиторий. Добавить в папку репозитория файлы. Зафиксировать состояние репозитория (выполнить commit).
7. Внести изменения в файлы. Зафиксировать новое состояние репозитория.
8. Создать новую ветку 1. Внести в нее изменения и зафиксировать их.
9. Переключиться на ветку мастера. Внести в нее изменения и зафиксировать их.

10. Продемонстрировать слияние веток.
11. Просмотреть дерево изменений веток (историю).
12. Создать удаленный репозиторий.
13. Загрузить на него свой проект.
14. Обеспечить доступ к нему всем членам команды.
15. Скачать каждым членом команды файлы с удаленного репозитория, произвести изменения и совершить слияние.

Защита лабораторной работы заключается в предъявлении преподавателю полученных результатов (на экране монитора), демонстрации полученных навыков и ответах на вопросы преподавателя.

5. Вопросы.