Федеральное агентство по образованию РФ

ГОУ ВПО Нижегородский государственный университет им. Н.И. Лобачевского

Факультет Вычислительной математики и кибернетики

Кафедра Математического обеспечения ЭВМ

УЧЕБНЫЙ КУРС

«Объектно-ориентированный анализ и проектирование»

для подготовки по направлению «Информационные технологии»

проект «Система бронирования такси»

СТРУКТУРА ПРОЕКТА

Выполнили:

Студент группы 85М21 Лазарев Антон

Студент группы 85М21 Погорельский Илья

Студент группы 85М22 Васильев Евгений

Студент группы 85М22 Патрушев Андрей

Нижний Новгород

2015

Оглавление

1. Рамки проекта
   1.1. Матрица компромиссов проекта
   1.2. Вехи проекта
   1.3. Сметы проекта
   1.4. План-график проекта
2. Роли и ответственности
   2.1. Знания, умения и навыки
   2.2. Структура команды
3. Протоколы проекта
   3.1. Управление конфигурацией
   3.2. Управление изменениями
   3.3. Управление внедрениями
   3.4. Достижение качества проекта
   3.5. Рабочая среда проекта

1. Рамки проекта

1.1. Матрица компромиссов проекта

	Фиксируется	Согласовывается	Принимается
Ресурсы			✓
Время	✓		
Возможности		✓	

Оценка времени.

Период завершения работ над проектом взят с 17 марта по 12 мая 2015 года, итого:

- Календарных дней: 55;
- Из них рабочих: 36

Следовательно, проект необходимо завершить за 36 рабочих дней.

Оценка возможностей, которые необходимо реализовать во время проекта. Требуется разработать следующие программные решения для реализации проекта:

- База данных заказов и таксистов;
- Программное ядро системы, занимающееся обработкой заказов и уведомлениями всех задействованных в нём лиц;
- Web-сайт для возможности создания заказов пользователями;
- Раздел сайта, предназначенный для взаимодействия таксистов с программным ядром системы.

Оценка ресурсов.

- 4 разработчика;
- 4 ноутбука, соответственно;
- Команда мотивирована;
- “Бюджет”.

Было принято придерживаться следующей стратегии:
Зафиксировав сроки, мы согласовываем объем функциональности решения и принимаем результирующие затраты ресурсов.

1.2. Вехи проекта

1. Утверждение концепции проекта
   Постановка задачи; удовлетворение интересов заказчика; выявление и постановка требований; бизнес-сторона проекта; согласованность со стратегией заказчика.
2. Утверждение планов проекта
   Формирование и утверждение план-графика; бюджет; создание условий для повышения эффективности команды.
3. Разработка
   Разработка кода приложения; архитектура; пользовательский дизайн; документация, тестирование;
4. Подтверждение продукта
   Внедрение, документация.

1.3. Сметы проекта

1) Заработная плата разработчикам:
   Если брать среднюю заработную месячную программистов разработчиков с небольшим опытом работы от 27 000 руб. в месяц, получаем, что один рабочий день стоит 1174 руб., следовательно бюджет на заработную плату разработчиков составляет:
   36*1174*5 = 169 056 руб.
2) Хостинг для будущего веб сайта:
   Для дальнейшего использования реализованного проекта в целях приобретения коммерческой прибыли требуется приобретение услуг хостинга, для размещения баз данных и сайта, используемого в проекте для взаимодействия с клиентами. Был выбран http://hosting.nic.ru/tariff302.shtml сроком на год.

Промежуточные итоги: 169 056 + 4890 = 173 946 руб.

3) Регистрация домена:

Сайт во всемирной сети не может поддерживаться без доменного имени. По умолчанию домен регистрируется на год, была выбрана компания, в которой приобретался хостинг (http://www.nic.ru/dns/domain/ru.html).

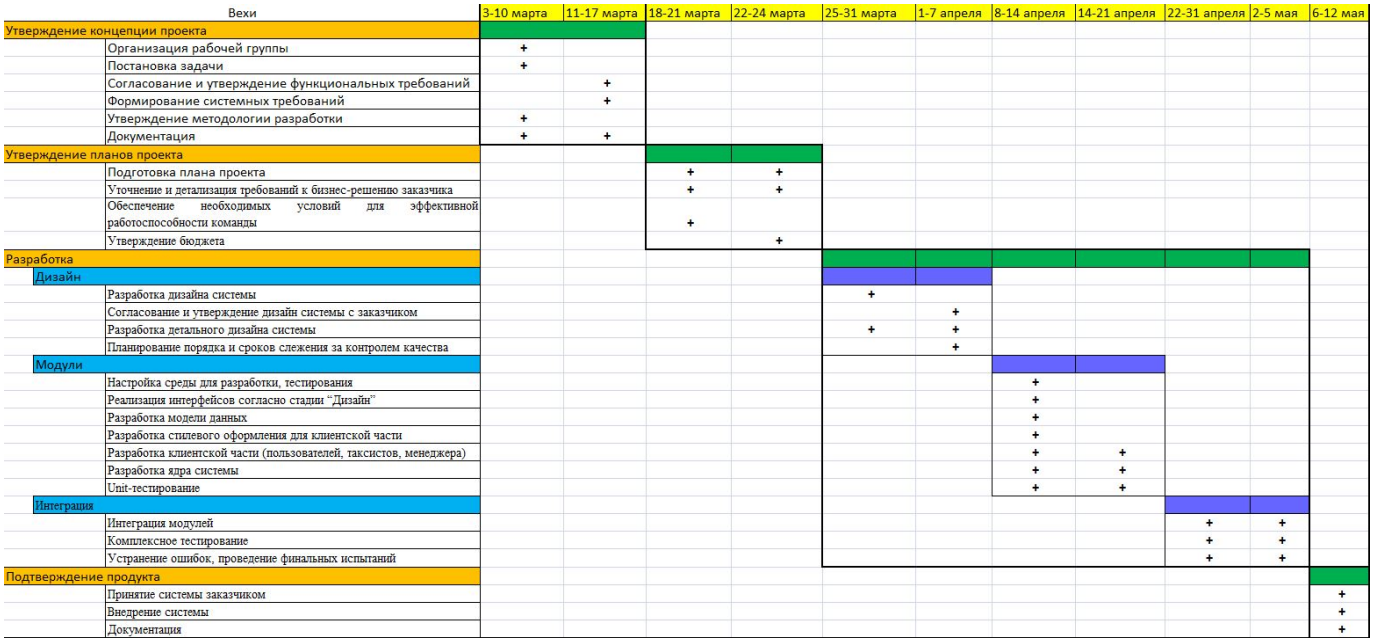
Промежуточные итоги: 173 946 + 600 = 175 546 руб.

4) Поддержка системы:

После успешной сдачи текущего проекта и запуска его для получения прибыли проектная команда прекращает свою работу над ним. В случае если заказчику потребуется техническая поддержка проекта на временной основе, то она будет оплачиваться отдельной услугой в расчёте 315 000 руб. * N, где N - количество необходимых людей из команды разработчиков. Услуги поддержки оформляются отдельным договором и по умолчанию сроком на год.

1.4. План-график проекта

План-график



Вехи	3-10 марта	11-17 марта	18-21 марта	22-24 марта	25-31 марта	1-7 апреля	8-14 апреля	14-21 апреля	22-31 апреля	2-5 мая	6-12 мая
Утверждение концепции проекта											
Организация рабочей группы	+										
Постановка задачи	+										
Согласование и утверждение функциональных требований		+									
Формирование системных требований		+									
Утверждение методологии разработки	+										
Документация	+	+									
Утверждение планов проекта											
Подготовка плана проекта			+	+							
Уточнение и детализация требований к бизнес-решению заказчика			+	+							
Обеспечение необходимых условий для эффективной работоспособности команды			+								
Утверждение бюджета				+							
Разработка											
Дизайн											
Разработка дизайна системы					+						
Согласование и утверждение дизайн системы с заказчиком						+					
Разработка детального дизайна системы					+	+					
Планирование порядка и сроков слежения за контролем качества						+					
Модули											
Настройка среды для разработки, тестирования							+				
Реализация интерфейсов согласно стадии "Дизайн"							+				
Разработка модели данных							+				
Разработка стилевого оформления для клиентской части							+				
Разработка клиентской части (пользователей, таксистов, менеджера)							+	+			
Разработка ядра системы							+	+			
Unit-тестирование							+	+			
Интеграция											
Интеграция модулей									+	+	
Комплексное тестирование									+	+	
Устранение ошибок, проведение финальных испытаний									+	+	
Подтверждение продукта											
Принятие системы заказчиком											+
Внедрение системы											+
Документация											+

2. Роли и ответственности

2.1. Знания, умения и навыки

Для успешного выполнения проекта все члены проекта должны обладать следующими знаниями:

- Знание русского языка. Проект разрабатывается на территории РФ, заказчик и состав команды разработчиков - русскоговорящие.
- Стаж работы в проектах, написанных на языке C# на платформе .NET - не менее года. Сроки разработки программы не позволяют выделить время на обучение программированию.
- Члены команды должны уметь пользоваться системами контроля версий, уметь работать по схеме «Fork + Pull».
- Члены команды должны обладать умением работать в команде. Основные критерии: умение налаживать конструктивный диалог; умение управлять своими эмоциями и абстрагироваться от личных симпатий/антипатий; аргументированно убеждать коллег в правильности предлагаемого решения.

Дополнительные умения для проект-менеджера:

- Высшее образование.
- Умение руководить проектной командой.
- Опыт руководства (в рамках проектных команд).
- Отличные коммуникативные навыки.

Дополнительные умения для архитектора ПО:

- Наличие практического опыта в коммерческих проектах в компании в роли ведущего разработчика или проектировщика программного продукта.
- Опыт проектирования архитектуры программного обеспечения.
- Навыки в области определения архитектурного шаблона/парадигмы и разбиения его на технические подсистемы/слои/компоненты/модули.
- Знание основных промышленных стандартов (HTTP, SSL, HTML, WSDL/SOAP, Rest, Json и другие).

Дополнительные умения для тестировщика:

- Опыт организации и проведения различных видов тестирования.
- Знание инструментов и библиотек для автотестирования.
- Опыт написания тестов.

Дополнительные умения для верстальщика:

- Хорошее знание HTML, CSS.
- Наличие художественного вкуса и творческого мышления.
- Знания композиции, формообразования, сочетания цветов.

2.2. Структура команды

Обязанности проект-менеджера:

- Согласование требований проекта с заказчиком;
- Согласование сроков выполнения проекта заказчиком;
- Организация реализации проектов в соответствии с планом работ;
- Проведение совещаний и планирование по ходу выполнения проектов.

Обязанности архитектора системы:

- Разработка ключевых технических сценариев взаимодействия компонентов;
- Определение протоколов взаимодействия компонентов;
- Создание модели данных и определения интерфейсов;
- Подбор технических средств и шаблонов для реализации подсистем;
- Выбор средств исполнения.

Обязанность верстальщика системы:

- Разработка дизайна HTML-страницы;
- Разработка макета HTML-страницы;
- Обработка фотографий и картинок;

Обязанности тестировщика:

- Моделирование ситуаций, которые могут возникнуть в условиях эксплуатации программного обеспечения;
- Контроль и верификация Unit-тестов;
- Разработка тест-кейсов;
- Участие в проведении опытных эксплуатаций программных продуктов.

Проект-менеджером команды выбран Васильев Евгений.
Архитектором системы назначен Погорельский Илья.
Менеджером по качеству назначен Погорельский Илья.
Верстальщиком назначен Лазарев Антон.
Тестировщиком назначен Патрушев Андрей.
Разработчиками назначены: Васильев Евгений, Погорельский Илья, Лазарев Антон, Патрушев Андрей.

3. Протоколы проекта

3.1. Управление конфигурацией

Система управления проектами и задачами - Redmine:

- диаграмма Ганта;
- календарь;
- управление файлами, документация;
- учет времени;
- интеграция с Git;
- добавление задач, отслеживание статуса задач;
- система уведомлений

При внесение нового изменения/задачи/баги менеджер проекта создает новую задачу. Добавляет ей тему, приводит описание требований, присваивает приоритет, статус, назначает на исполнителя, указывает дату исполнения и дедлайн, указывает время на задачу. Затем исполнитель, на которого была назначена задачу изменяет ее статус (In progress) и начинает ее выполнение. В ходе работы над задачей у исполнителя могут возникнуть вопросы, которые он оформляет с помощью Redmine в виде примечаний к задаче (Updated). Если исполнитель считает, что задача завершена или требуется дополнительное обсуждение, он снова изменяет статус (Resolved). В этом случае, менеджер проекта принимает решение о том, удовлетворяет ли задача поставленным требованиям. Если удовлетворяет, задача изменяет статус и считается закрытой (Closed). В ином случае, после необходимых обсуждений, исполнитель продолжает работу над задачей до момента появления новых вопросов или завершения задачи. После выполнения требуемых задач (со статусом Closed).

Redmine является центральным связующим звеном между менеджером проекта и командой разработчиков. Отвечает за создание новых задач и отслеживание изменений в ходе их выполнения, используется для учета времени, затраченного на выпуск очередного релиза, построения календарных планов и диаграммы Гантта; уменьшаются затраты на коммуникации внутри команды проекта, повышается качество и скорость разработки.

Средство связи - Skype

Раз в 8-10 дней проводятся Skype-meetings и Planing.

- Skype-meetings - на них каждый из участников проекта отчитывается по проделанной работе, говорит какие у него на данный момент blokers и то, что он планирует сделать (то, что не успел за данный спринт).
- Planning - формируется список приоритетный задач на данный спринт (3-5 дней). В список задач входят:
    - баги, выявленные в предыдущем спринте;
    - новые задачи из новой product story;
    - изменения - новые требования, которые за предыдущий спринт сформировал заказчик

  Product story имеет ряд задач. Каждая задач разбивается на подзадачи: ряд задач разработки, [верстка], тестирование, review, acception (продуктовый код). На каждую подзадачу путем “игры в покер” утверждается время. Каждая из подзадач назначается на определенного разработчика (по собственному желанию или исходя из имеющегося опыта). Из суммы времени подзадач формируется итоговое время задачи.

  На один спринт не назначается больше задач, чем команда способна решить. Время работоспособности команды вычисляется по формуле: countDevelopers*workTime*countDays.

Система контроля версий - Git.

В качестве web графической оболочки используется GitHub. В качестве desktop графической оболочки используется SourceTree.

Релизная ветка - ветка под названием master.

Разработчик, вносящий изменения обязан:

1. Создать отдельную ветку (для каждой задачи своя ветка)
2. Сделать Fetch (извлечь обновления на свой локальный репозиторий)
3. Сделать merge с веткой master’a.
    a. Если возникли конфликты, то устранить их.
4. Внести необходимые изменения. Сделать серию commit.
5. Сделать Push изменений
    a. перед push делаем pull. Если есть новые изменения в ветке master, то делаем merge с master’ом. При возникновении конфликтов решаем их;
    b. push делается, соответственно, только своей ветки.
6. Сделать Pull Request своей ветки

Для любых изменений:

- добавление новой задачи;
- добавление баги;

- добавление задачи-изменения;
- исправление стилевой ошибки

создается новая ветка.
Для внесения исправлений, выявленных в code review новая ветка не создается. Исправления вносят в рамках текущей ветки.

7. После того, как Pull Request был сделан - его проверяют остальные разработчики на предмет code review, code smells.

8. Как только данный Pull Request соберет два approve, данная ветка merge с веткой master.

Документооборот

- для описания требований конкретных задач/багов используется функционал (раздел описания, wiki) системы управление проектами и задачами - Redmine.
- для описания списка требований, рамок, видений, протоколов и прочее используется функционал Google Docs.

3.2. Управление изменениями

Внесение изменений со стороны разработчиков:

- разработка. Вносятся локальные изменения, делается pull request. Остальные разработчики делают code review. Как только данные изменения собиают два approve со стороны разработчиков - изменения merge с релизной веткой;
- принятие решения о внесении изменений возникает после Skype-meeting'ов, ShowCase'ов.
- средство создания задач-изменений - Redmine.
- решение принимается коллективно - командой разработчиков.

Внесение изменений со стороны заказчика:

- для внесения изменения заказчик, непосредственно, связывается с Product Owner'ом проекта.
- средство связи - Skype конференция.
- форма запроса.

  При формировании нового требования заказчик высылает пакет документов:

  - описание требования (назначение, суть);
  - последовательность действий, показывающая работоспособность требования;
  - интерфейс/визуализация/демо-версия;
  - [стилевое оформление кода]

  Заказчик с Product Owner путем разговора обсуждают функциональность, возникшие вопросы, blokers, временные рамки, финансы.

  Также, новые требования отображаются в смете.

3.3. Управление внедрениями

В качестве инструмента подготовки релиза используется Git.

1. Создать две ветки: master и release.
    - master - ветка, с которой merge все pull request, которые собрали два approve. Каждый commit в master - это merge одной задачи. За один sprint количество commits зависит от количества задач и багов.
    - release - ветка, с которой merge ветка master. Одной release версии соответствует ровно один commit.
2. Из release ветки собирается финальное решение.

Тестирование системы происходит на локальных компьютерах разработчиков.
В качестве системы непрерывной интеграции используется CruiseControl. В качестве системы контроля качества кода используется система Sonar.

Внедрение проекта:
Программный продукт представляет собой веб-сайт: клиентская часть, серверная, база данных.

Веб-сайт разворачивается на сервере на IIS (Intertnet Information Server).

1. Установить IIS дляWindows OS.
2. Положить папку проекта, например, по пути C:\Web\site.
3. Зайти в IIS, добавить новый сайт.
4. Ввести имя сайта, расположение веб-сайта (C:\Web\site), указать номер порта.
5. Зайти в Applications Pools и изменить на v4.0

3.4. Достижение качества проекта

Менеджером проекта по качеству назначен Погорельский И.И.

Качество проекта складывается из пяти составляющий:

1) удовлетворение ожиданиям заказчика;
2) проектирование системы, паттерны, принципы;
3) покрытие системы тестами, интеграция - минимизация багов;
4) проверка качества кода;
5) стилевое оформление.

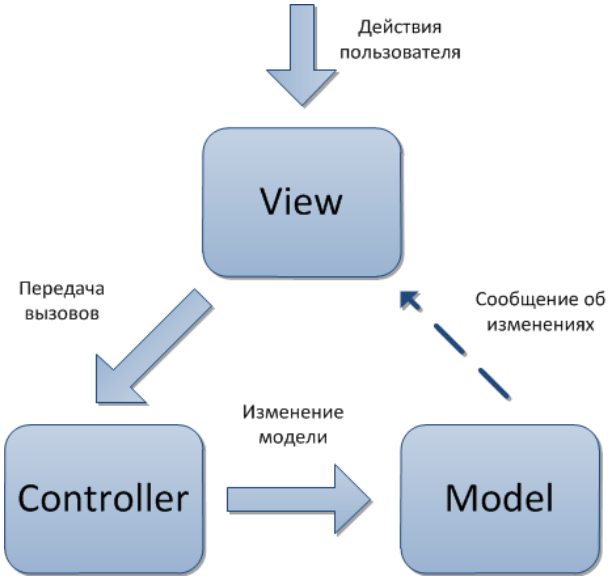
1. Удовлетворение ожиданиям заказчика:

- частая своевременная поставка релизов (раз в неделю);
- гибкость принятия новых требований;
- демонстрация продукта в конце каждого релиза;
- скорость работы
    - Максимальное время от поступления заказа в систему до назначения её на водителя должно составлять не более минуты;
    - Уведомления клиенты и водителю о состоянии заказа должны отправляться в течении 30 секунд после изменения заказа;
    - Положение и статусы водителей обновляются раз в 5 минут;
- производительность
    - высокая пропускная способность - единовременная обработка 1000 заказов;
    - Местоположение водителя и клиента определяются с точностью до 75 метров.
- надежность
    - 97% обеспечение работоспособности системы;
- интуитивно понятный интерфейс
- техническая поддержка
    - быстрое реагирования на решение возникшей проблемы
- набор необходимой документации

2. Проектирование системы (паттерны, принципы):

В данной системе применяются следующие паттерны и принципы:

- MVM (Model-View-Controller) - модель приложения (Model), пользовательский интерфейс (View) и взаимодействие с пользователем (Controller).



- разработка и тестирование Model (не зависит от View и Controller);
- повторное использование Model к различным View (сохранение реализации Model);
- повторное использование View - изменение действий пользователей не затрагивая View;
- к Controller можно привязать несколько View;
- автоматическое модульное тестирование;
- разделение ответственности

- SOLID принципы
  - Принцип единой ответственности - каждый класс/функция имеет единую ответственность (разделение обработки заказов от отображения заказов).
  - Принцип открытости/закрытости - изменение поведения сущностей без изменения их исходного кода (интерфейсы, добавление новых ролей, добавление новых форматов хранилищ данных).
  - Принцип Лисков - поведение наследников не должно противоречить поведения заданному в базовом классе.
  - Принцип разделения интерфейса - классы должны знать только о тех элементах, которые они используют (независимость классов “таксисты” и “клиенты”).
  - Принцип инверсии зависимостей - абстракции не должны зависеть от деталей, детали должны зависеть от абстракций (Процесс заказа -> Клиент. Решение: Клиент <- Интерфейс процесса заказа -> Процесс заказа).
- Builder - поэтапное построение объектов с контролем результатов выполнения каждого из этапов.
  - контроль создания более сложных объектов;

    - возможность получения разных представлений некоторых данных.
- SingleTon - единственный экземпляр, предоставление к нему доступа извне, запрещение созданий нескольких экземпляров одного и того же типа (ведение журнала).
    - всегда доступен. Время жизни от запуска до завершения программы.
    - доступен из любой части программы
- Prototype - создает новые объекты с помощью прототипов (без параллельной иерархии, лучше Factory Method).
    - система независима от создания новых объектов и порождения типов объектов;
    - создание объектов, точные классы которых известны только на стадии выполнения программы.
- Abstract Factory - создание семейства/группы взаимосвязанных объектов.
    - Независимость системы от процесса создания новых объектов и от типов порождаемых объектов.
    - Исключение возможности одновременного использования объектов из разных семейств в одном контексте.
- Adapter - программная обертка над существующими классами, преобразование их интерфейсов к виду, пригодному для последующего использования (есть система расчет стоимости для рублей, а система будет использоваться в США. Нужен адаптер преобразующий рубли в доллары).
    - создание класса - интерфейс пользователя UIAdapter
    - создание производного класса Adapter, реализующий интерфейс UIAdapter. В Adapter есть ссылка на экземпляр RubleAdapter. Паттерн Adapter использует эту ссылку для перенаправления клиентских вызовов в RubleAdapter.
- Observer - зависимость "один-ко-многим" между объектами так, что при изменении состояния одного объекта все зависящие от него объекты уведомляются и обновляются автоматически.
    - Инкапсулирует главный (независимый) компонент в абстракцию,а изменяемые (зависимые) компоненты в иерархию.
    - Решение проблемы с монолитностью системы - классы слабо связанные (или повторно используемые).

3. Покрытие системы тестами, интеграция

Система разрабатывается по **TDD технологии** - разработка через тестирование

- короткие циклы разработки;
- красный тест - зеленый тест - красный тест;
- создание модульных тестов, определяющих требования к кода непосредственно перед написанием кода;
- проверка корректности;
- формирование представления о том, что на самом деле необходимо пользователю;
- “долой лишний код” - писать только тот код, который необходим для прохождения теста.

В качестве системы **непрерывной интеграции** используется CruiseControl.

- быстрое выявления проблем интеграции и их устранение;
- немедленный прогон модульных тестов для свежих изменений;
- постоянное наличие работающей версии (для демонстрации);

4. Проверка качества кода

1) **Рефакторинг -** процесс улучшения написанного ранее кода путем изменения его внутренней структуры, без влияния на внешнее поведение.

- улучшение композиции кода;
- улучшение понимания структуры кода;
- помогает найти ошибки;
- устранение code smells;
- дешевые модификации

В качестве регулярного рефакторинга используется система **Sonar.**

2) Review - проверка содержимого кода перед добавление его к готовому продукту. Перед тем как merge новой функциональности в release ветку:
    - другие разработчики проверяют содержимое кода. Если все корректно, то ставят свой approve;
    - прогон модульных тестов;
    - выявление ошибок;
    - рекомендации по стилевому оформлению, построения структуры

5. Стилевое оформление

В качестве **статического анализатор C# кода** на предмет соответствия стилю используется система **StyleCop**.

- повышение читабельности кода;
- простота, логичность и понятность кода;
- набор стандартов и правил, возможность добавления собственных, исходя из требований заказчика

В качестве **анализатора JS кода** используется анализатор **JSLint**.

3.5. Рабочая среда проекта

- самоорганизация рабочего места;
- локальные компьютеры разработчиков;
- доступ в интернет;
- open source средства разработки
    - MVS 2012
    - Git
    - SourceTree
    - Sonar
    - StyleCop
    - Redmine
- средство связи - Skype
- личные встречи - аудитории университета