IT-проекты

Длинно о главном V.3.0

Этот документ для тех, кто решился выбрать IT—проект в качестве индивидуальной выпускной работы. Или для тех, кто хочет просто скоротать вечерок.

30.9.2017

Оглавление

IT-проекты

§0 – Проектный замысел

Уважаемые друзья, если вы открыли этот документ, значит, либо вы уже твёрдо решили, что в ближайшие полтора года вы будете делать проект в сфере IT, либо вы находитесь в процессе выбора. Ну или вам просто любопытно, что, в общем, тоже неплохо.

Цель этого документа - дать вам ту информацию, которая поможет ***успешно пройти все этапы*** выполнения проекта. Ну или, что то же самое, как ни странно, угадать, какие у вас возникнут ***вопросы*** и по возможности дать на них как можно более информативные ***ответы***. Или, что, опять же, одно и то же - создать ***общую систему понятий*** для вас, преподавателей и всех тех, кто будет принимать участие в оценивании вашего проекта.

* * *

Итак, IT-проект. Думаем, слово "проект" вы уже слышали раз двести и вряд ли здесь имеет смысл давать какое-то определение. Хотя, с другой стороны, чтобы у вас не возникло ощущение, что мы относимся ко всему легкомысленно, приведём цитату из документа под названием “Национальный стандарт Российской Федерации ГОСТ Р ИСО 21500-2014 – Руководство по проектному менеджменту (англ. ISO 21500:2012 Guidance on project management)”:

“Проект состоит из уникального набора процессов. Процессы состоят из координируемых и контролируемых работ с датами начала и окончания, которые выполняются для достижения целей проекта. Достижение целей проекта требует получения определенных результатов, отвечающих конкретным требованиям. При реализации проекта могут действовать множество ограничений, включая описанные в подразделе 3.11.

Несмотря на возможное сходство каждый проект уникален. Проект может отличаться:

- получаемыми результатами;
- составом влияющих на проект заинтересованных лиц;
- используемыми ресурсами;
- существующими ограничениями;
- особенностями использования процессов проектного менеджмента для получения результатов. ”[1]

[1] Национальный стандарт Российской Федерации ГОСТ Р ИСО 21500-2014 – Руководство по проектному менеджменту. М.: Стандартинформ, 2015. 52 с.
URL: http://protect.gost.ru/document.aspx?control=7&id=198457

В этой цитате многое (если не всё) может показаться достаточно очевидным, но именно то, что здесь указано, и будет определять структуру вашей деятельности в ближайший год. Однако, понимая, что всё, что здесь сказано про проект, относится к проекту любого типа, мы, не медля, перейдём сразу к проектам нашего типа и, отбросив строгость ГОСТа, скажем, что проект - это процесс и результат работы над программным продуктом, который, среди прочего, характеризуется: 1) функциональностью - то есть, способностью решать задачи определённого круга пользователей, 2) законченностью - то есть, непосредственно к сроку сдачи пользователи, для которых он предназначен, могут начать с ним работать, 3) определённой степенью сложности (об этом подробнее будет сказано дальше, пока что отметим, что этот пункт нужен для отсечения *Hello World*-проектов).

* * *

Функциональность - как уже было сказано, вы пишете проект для решения определённого круга проблем, которые могут возникнуть у определённых людей. Иными словами, вы пишете *настоящий* (возьмём на себя смелость употреблять это слово без всяких кавычек) проект, а не рассуждаете на тему, каким могло бы быть то или иное приложение (сайт и т. п.), если бы вы сели его писать. Никакого сослагательного наклонения - вы садитесь и пишете проект от начала до конца.

К моменту дедлайна у "заказчика" (а тут кавычки и несколько секунд позже вы поймёте, почему) проект должен быть уже *полностью готов к работе*. Что это значит?

Если у вас ***сайт***, это значит, что сайт полностью должен быть залит на хостинг, а также предоставлен внешним тестировщикам для возможность протестировать его локально (говоря проще - залейте сайт на GitHub).

Если у вас ***мобильное приложение***, то оно в идеале оно должно присутствовать в AppStore / Google Play, но, если с этим возникнут разного рода сложности, то этот пункт можно смягчить до предоставления полностью готового проекта (в терминах той среды разработки, которой вы пользуетесь), который, опять же, будет протестирован локально.

Если же у вас ***десктоп-приложение***, то здесь всё просто: код и установочные файлы должны быть предоставлены внешним тестировщикам к определённому дню.

Если привести какие-то аналогии, то нам в данном случае нравится покупка квартиры. Представьте себе, что некто говорит вам, что в квартире, которую вы присмотрели, можно жить с момента покупки. Если оно, действительно , так - то всё хорошо, если же потом выясняется, что в ней нужно перестелить полы, поменять трубы, проводку и установить ванну, то продавец, мягко скажем, вас обманул.

Резюме: на защиту выносится полностью готовый продукт, требующий только установки и настройки параметров со стороны пользователя (конечно, если последнее предусмотрено самой концепцией проекта).

* * *

Возможно, вы уже спрашиваете себя или Вселенную: хм, а кто такой ***заказчик***? Есть два варианта: заказчиком может быть реальный человек (родственник, друг, деловой партнёр, кто угодно), который попросил вас написать приложение/сайт/что угодно для своих задач. Он ставит перед вами задачу и, если она проходит этап предзащиты, вы приступаете к работе. В данном случае нам бы очень хотелось иметь возможность общения с вашим заказчиком (мы не будет отрывать его от дел и забот каждую неделю, конечно, речь идёт о разумной консультации).

Но, может так случиться, что к вам никто не обратится. Пожалуйста, в этом случае, не выдумывайте заказчиков. Не говорите нам, что ваш троюродный дедушка заказал вам игру-стрелялку, если, в действительности, у вас нет троюродного дедушки или он есть, но он просил вас написать ему игру-головоломку. Вместо этого будьте готовы поразмышлять на тему, какой могла бы быть ***целевая аудитория*** у вашего проекта. Целевая аудитория – это те люди, которые могли бы заинтересоваться вашим проектом и со временем стать заказчиками, например, обновлённой версии вашего продукта с дополнительным функционалом.

* * *

Что не является проектом?

А почему, собственно, мы задаёмся этим вопросом? Дело в том, что, как вы уже догадались, однозначно определить, что такое проект - попросту невозможно, а когда определение дать невозможно, полезно уточнить, чем не является то, о чём мы говорим.

1) Естественно, не является проектом результат чужой работы. Мы не будем говорить, что будет в случае, если кто-то попадётся на плагиате. И так понятно, что ничего хорошего.

Тем не менее, отметим важный момент: поскольку весь код с нуля, понятное дело, вы писать не будете, вам нужно будет в документации к проекту указать, какие именно библиотеки, фреймворки и т. п. вы используете. Это разумно, это не нарушает авторские права и отдаёт дань уважения тем, кто создал этот код до нас.

2) Если ваш проект представляет собой ***сайт***, то не будет засчитываться сайт, собранный на одной из CMS - будь то *WordPress*, *Joomla*, *Drupal* и тому подобное. Всё это важные вещи, но, как нам кажется, в этих проектах особо нет (мягко говоря) программирования, а есть сборка модели из конструктора. Вместо CMS изучайте фреймворки. Их много, с ними интересно, вы получите отличный опыт чтения документации и погружения во что-то новое (если, конечно, вы до этого ни на чём не писали).

3) Не может быть представлен к защите проект, который трудно или даже невозможно протестировать в аудитории. Допустим, ваш проект предусматривает работу с геолокацией и

отслеживанием координат общественного транспорта, например, автобусов. Как мы узнаем, что ваш проект, действительно, работает с геолокацией автобусов и работает *правильно*? Увы, отправиться в пешую прогулку по Москве вслед за автобусами у нас возможности не будет. Или, допустим, вы написали приложение, которое каждые сутки выполняет какую-то функцию. А защита длится всего 10-15 минут. В общем, думаем, намёк понятен. *Функционал приложения должен демонстрироваться в отведённое на защиту время в обыкновенной лицейской среде.* Здесь же упомянем проекты, которые требуют необычной (нетипичной для нашего Лицея) экспертной оценки. Допустим, вы написали переводчик с русского языка на кхмерский. Подозреваем, в Лицее на кхмерском не говорит никто, и не факт, что мы сумеем найти быстро и к нужному времени эксперта в этой области. Поэтому обращаем ваше внимание на то, что, ***если ваш проект требует экспертной оценки со стороны, пожалуйста, согласуйте это с нами заранее***. ***Если у вас возникли сомнения, что такая оценка понадобится, пожалуйста, проконсультируйтесь с нами***.

4) Не является проектом и тот код, где вы выступили только пользователем каких-то библиотек, а сами только "прикрутили" к ним интерфейс. Пример: допустим, вы написали приложение, которое подсчитывает число котиков на фотографии, использовав для этого *машинное обучение-глубокое обучение-нейросети* и всё то, что сегодня гремит во всех медиа. Если же вы при этом нейросети и алгоритмы обучения взяли из библиотек, а ваша работа свелась к написанию интерфейса загрузки фотографий и вызову функции *countCats('file.jpg')* (мы утрируем, конечно, но вы понимаете), то это не может быть признано проектом.

Значит ли это, что нельзя считать котиков посредством нейросети? Нет. Но будьте готовы:

1. ***писать*** нейросеть и все алгоритмы ***сами***,
2. ***понимать*** математику, которая за ней стоит.

В отличие от кхмерского языка тут проблем с поиском экспертов для разговора с вами на эту тему не будет - найдём.

Разумеется, это не значит, что нужно вообще всё писать с нуля. Но должен быть разумный баланс между использованием своего кода и чужого, который достигается на этапе обсуждения проектной заявки.

В общем и целом это касается не только желающих считать нейрокотиков: ***будьте готовы понимать***, как работает то, чем вы пользуетесь. Если, например, вы выводите какой-то список в алфавитном порядке, будьте готовы поговорить про алгоритмы сортировки, если работаете с очень большими числами - то поболтаем о том, как числа представлены к оперативной памяти и т. п.

5) Не могут быть, к сожалению, допущены к защите всевозможные фреймворки, библиотеки, компиляторы, интерпретаторы, парсеры и другие программные блоки, предназначенные для узкого круга лиц. Мы ни в коей мере не относимся к этому коду пренебрежительно - напротив (домов без строительного материала быть не может), но ваш проект должен быть понятен широкому кругу пользователей. Возможно, ваш проект использует написанные вами

фреймворки и парсеры - это здорово, но в любом случае это его не главная задача и сводиться к написанию оных он не может.
6) Странные проекты. Хм, а что это такое? Мы, конечно, думаем, что таких проектов у нас не будет, но и обойти их своим вниманием мы тоже не можем. Представьте, что вы написали проект, который просит пользователя ввести число, а затем выводит столько кружочков, сколько было указано. Проект всем понятен, доступен, сами вы всё про него знаете и можете отчитаться за каждую запятую и точку с запятой в нём. Возникает, однако, один простой вопрос: зачем? Какую задачу решает вывод двадцати кружочков на экран? Кому это нужно?

7) Даже если бы вы нас убедили, что проект с кружочками сделает счастливым многих людей, он бы не прошёл по критерию сложности. Поскольку этот раздел оказался достаточно длинным, давайте передохнём, а потом продолжим.

* * *

Пример с кружочками довольного искусственный, но, допустим, вы решили написать программу, которая считает корни квадратного уравнения. Это полезно, вы понимаете, как происходят вычисления, но... *это слишком просто*. Для решения непосредственно задачи вычисления корней нужно несколько строк кода. Даже если вы никогда не изучали программирование до этого, вычислить корни квадратного уравнения вы сможете уже на втором-третьем занятии. Здесь нет ни сложной математики, ни трудоёмкого программирования, ни концептуально трудноосмыслимых понятий.

Тут можно спросить (мы поймём, если вы будете спрашивать с ехидной интонацией): а как вы будете измерять сложность проекта? Ответ: никак. Проект это не стол, его линейкой не измерить. Аршином общим тоже. Именно для этого нужен *этап предзащиты и согласования проектной заявки*: в процессе обсуждения мы сможем прийти к общему знаменателю и решить, как усложнить ваш проект, если заявка выглядит слишком простой, или, наоборот, предупредить вас о том, что стоящая перед вами задача потребует от вас слишком много временны*х* затрат.

* * *

Первый раздел нашего документа подошёл к концу. Что нужно сделать сейчас? Совет: изложите "вольным стилем" в одном абзаце суть вашего продукта и посмотрите, насколько ваша задумка соответствует перечисленным выше требованиям.

Эти требования вам могут показаться очень строгими. На самом деле, они нужны для того, чтобы отсечь крайности. Если вам кажется, что приведённые выше требования слишком несправедливы или не могут применяться к вашему проекту, пожалуйста, сообщите нам об этом.

Мы готовы к диалогу, а наша задача - не запрещать вам что-то, а помогать "доводить до ума" вашу заявку так, чтобы на защите ни у кого не возникало вопроса, а что же, собственно, вы защищаете, где тут проект, где тут работа головы и сердца и т. п.

§1 – Подход к разработке программного продукта

Когда ваш замысел пройдёт всевозможные согласования и утверждения, он превращается уже в рабочую идею, которую вы будете реализовывать, используя определённую *методологию разработки программного обеспечения* – объектно-ориентированный подход (см. Википедию, статью “Объектно-ориентированное программирование”[2]).

Если вы уже интересовались разработкой ПО (программного обеспечения) или просто время от времени вы просматриваете IT-новости, вы как минимум один раз слышали всякие магические слова типа *Agile*, *Scrum*, *XP* и т. п. На фоне этого объектно-ориентированный подход может показаться чем-то устаревшим, особенно, если учесть, что он был популярным уже в 1980-ые годы. Это ощущение довольно обманчиво и мы попробуем объяснить, почему мы остановились именно на этом подходе:

- Как правило, все современные подходы к разработке ПО – это подходы для управления *командами*, а ваш проект – *индивидуальный*.
- Все современные подходы подразумевают, что требования заказчика могут меняться налету и даже буквально вот-вот перед релизом. В нашем случае мы бы имели право написать вам за 3 дня до дедлайна что-то типа: “А давайте в вашей чудесной игре будет не 2 персонажа, а 4, мы решили, что так лучше, и уровней лучше не 5, а 10. И да, нам нужны также версии под Mac OS и Linux”. Будем откровенны: даже мы не настолько жестоки.
- ОО-подход позволяет хорошо спланировать ваш проект и написать понятную документацию.
- Разработка любого ПО требует понимания того, как пишется ООП-код, знания всех основных понятий; почти любой современный язык программирования поддерживает классы и объекты. Понимание всего этого – абсолютный must, так же как умение считать от 0 до 10 для изучения математики.

Итак, переходим непосредственно к этапам планирования, которые вам предстоит пройти:

- Требования к приложению и аппаратному обеспечению
- Пользовательские сценарии (user scripts)
- Определение основных объектов и их взаимодействия
- Построение классовой диаграммы (UML diagrams)

§2 – Требования: soft and hard

На данном этапе вам предстоит сформулировать функциональные требования и требования к вычислительным мощностям, то есть, если совсем просто, требования к софту и к “железу”, или,

[2] Что? Ссылка на Википедию?? Как возможно??? Отвечаем: во-первых, наш документ - это не исследование, это гайд для дальнейшей работы, во-вторых, Википедия – не проблема сама по себе, *проблема* – если вы, не владея терминологией и пониманием ООП, *ограничитесь только Википедией*. Мы подскажем в конце документа, что почитать по ООП.

ещё проще, определить, что должно делать ваше ПО и какие минимальные требования к тому гаджету или компьютеру, на котором будет функционировать ваше ПО, вы как разработчик выдвигаете.

Функциональные требования

Если коротко: это довольно подробное описание того, что именно ваше ПО должно делать (must do) и в каком виде оно будет существовать (web, mobile, desktop).

Пример 1: сайт онлайн-магазина *должен* предоставлять пользователям возможность регистрации, совершения покупок, управления заказами (согласование условия доставки, отмена заказов, сортировка списка заказов по сумме, просмотр истории заказов), рейтингования товаров. Также сайт *должен* по каждой позиции в заказе предоставлять рекомендацию к покупке двух других товаров.

Пример 2: игровое приложение *должно* давать возможность пользователю зарегистрироваться под своим именем и, управляя игровым персонажем, пройти 10 игровых уровней. Порядок прохождения уровней *должен* определяться пользователем, однако, при этом, он должен соответствовать определённым требованиям. В игровом мире *должно* жить не менее 10 видов врагов, 5 боссов; *должно быть* не менее 5 видов оружия, не менее 7 видов артефактов. Игра должна предоставлять возможность мультиплеера и возможность покупки оружия за деньги.

Q: Насколько подробной должна быть заявка, которую вы отправляете через соответствующую Гугл-форму (информацию о том, как отправить заявку, вы получите позже через сайт)?
A: Ответ на этот вопрос трудно дать в количестве слов, предложений и т. п. Однако, из двух примеров ясно, что эти требования отражают основной функционал программы, который требуется пользователю от начала до конца пользовательского цикла.
Если функциональные требования будут недостаточны, мы в процессе обсуждения с вами выработаем дополнительные, так, чтобы они выглядели “солидно”.

Q: Нужно ли указывать все требования?
A: Нет, только основные. Какие из них основные, определяется предназначением ПО. Очевидно, что главная функция онлайн-магазина – возможность совершать покупки, а игры – проходить уровни или иным образом развлекаться/обучаться.

Q: Могут ли требования меняться со временем?
A: В сторону увеличения функционала – однозначно, да, на любом этапе, но рассчитывайте свои компетенции и ресурсы правильно. В сторону уменьшения/изменения – короткий ответ: после определённого момента – нет. В определённый момент времени (естественно, об этом будет объявлено заранее и перед этим ваши идеи пройдут этап обсуждения) **написанные вами функциональные требования будут зафиксированы** и они станут вашими обещаниями сделать то-то и то-то. Они лягут в основу того **чеклиста**, который комиссия будет использовать для оценивания вашего проекта. Об этом мы поговорим позже, но, говоря простыми словами, мы будем смотреть, сделали ли вы то, что действительно обещали.

Требования к аппаратному обеспечению

Их вы тоже указываете в заявке и здесь всё просто: когда вы хотите скачать приложение, вы обычно смотрите, пойдут ли они на вашем компьютере/телефоне/планшете. Для этого вы читаете recommended system requirements. Вы должны перечислить ***минимальные требования***:

Для десктоп-приложений:

- Процессор, тактовая частота
- RAM
- Место на жёстком диске
- Видеокарта, размер RAM
- Операционная система и её минимальная версия
- Особые требования (например, ПО не будет работать с видеокартами Radeon или ПО требует минимум 16 Гб ОЗУ) с обоснованиями, почему они выдвигаются

Для мобильных приложений:

- Операционная система (iOS/Android) и её минимальная версия
- Диагональ экрана/тип устройства: здесь необходимо указать, если, например, вы пишете приложение под iOS, будет ли оно работать на iPhone, iPad mini, iPad, iPad Pro, iPod Touch
- Оперативная память, необходимая для комфортной работы
- Минимум свободного места в памяти планшета
- Будет ли возможность работать с картой памяти (для Android)
- Особые требования с обоснованиями

Для веб-приложений (параметры хостинга):

- PHP-версия
- Тип базы данных и версия
- Дисковое пространство
- Лимит оперативной памяти
- Лимит трафика
- Особые требования с обоснованиями

§3 – Пользовательские сценарии (User scripts)

Этот раздел очень похож на предыдущий, фактически, это зеркальное отражение функциональных требований.

В описании функциональных требований нет самого главного: пользователя. Мы описываем эти требования так, как будто программа всё делает самостоятельно, без получения указаний пользователя. Данный раздел компенсирует это и позволяет *более детально* продумать то, что должна делать ваша программа.

Под пользовательским сценарием мы будем понимать желание пользователя осуществить некое действие и последовательность шагов, которые он для этого должен совершить.

Пример 1:

Действие: выбор товара в магазине

Сценарий:

- Поиск товара
- Добавление нужного товара в корзину
- Реакция на запрос “Желаете узнать о рекомендованных для Вас товарах?”
- Реакция на запрос “Желаете узнать информацию о выгодных предложениях?”
- Переход к оформлению заказа

В этом примере обратим внимание на несколько моментов:

- Действие не является излишне общим (хочу купить товар) или излишне конкретным (хочу ответить на предложение о скидках, сценарий: подвести курсор к кнопке “ОК” и нажать левую клавишу мыши 1 раз
- Некоторые шаги сценария могут быть выделены в отдельные действия, например, поиск товара, как правило, включает в себя простой поиск по введённому запросу или расширенный поиск с настройкой параметров поиска (категория товара, ценовой диапазон и т. п.)

Пример 2:

Действие: атака в приближающегося врага

Сценарий:

- Пользователь тапает в зону экрана, отвечающую за выстрел
- Система проверяет, имеется ли у игрока оружие
- Система проверяет, имеется ли у игрока амуниция
- Если да, то осуществляется выстрел
- Если игрок попал во врага, то система проверяет, сколько у врага осталось жизненной энергии
- Отрабатываются сценарии, в зависимости от того, осталось ли у врага жизненная энергия

Чем, помимо содержания, отличается этот пример от предыдущего? 1) Здесь упоминается система (на самом деле – это ваша приложение, сайт, программа, ПО, называйте, как хотите).

2) Здесь нелинейная структура, поэтому у нас есть несколько “если”.

Q: Сколько нужно таких сценариев?
A: 3-7.

Q: Должны ли они перечислять все возможные действия пользователя?
A: Конечно, нет. Для защиты достаточно 3-7, как было сказано выше, но – чем тщательнее вы спланируете вашу работу сейчас, тем меньше вам придётся “опять всё переделывать, да что ж такое”.

§4 – Определение основных объектов и их взаимодействия

После того, как вы напишете требования к вашему ПО и пользовательские сценарии, самое время подумать о том, как всё это реализовано программно. В самом деле, это уже 10-ая страница документа, а ещё ни слова не было сказано о, собственно, самом программировании! Сейчас – самое время, пожалуй.

Итак, вы уже понимаете в определённой степени, что будет делать пользователь и как будет реагировать на это программа. Ваша следующая задача – понять, какие классы и объекты будут в вашем коде.

Строго говоря, на этом этапе мы уже не можем ничего вам предписывать, давать какие-либо инструкции и т. п. Здесь начинается исключительно ваша работа, но, если вам трудно определиться с тем, какие объекты будут в вашем проекте, можно поступить следующим образом:

- Вернуться к пользовательским сценариям
- Выделить в текстах сценариев имена существительные/номинативные группы
- Убрать повторы и то, что по тем или иным причинам не будет объектом

Вернёмся к одному из наших примеров.

Действие: выбор товара в магазине

Сценарий:

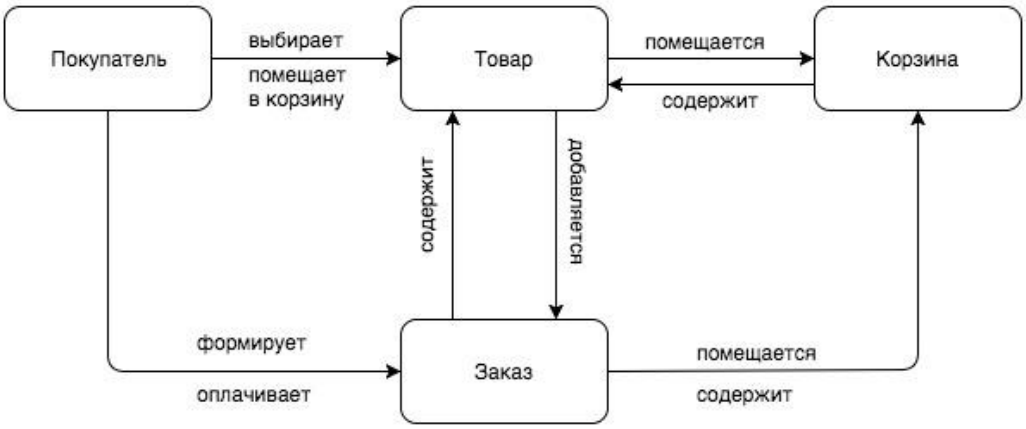
- Поиск товара
- Добавление нужного товара в корзину
- Реакция на запрос “Желаете узнать о рекомендованных для Вас товарах?”
- Реакция на запрос “Желаете узнать информацию о выгодных предложениях?”
- Переход к оформлению заказа

Выделяем все существительные (или существительные с прилагательными, так как это говорит нам о классах и атрибутах), получаем список, в котором, на наш взгляд (повторяем: здесь, как и на любом этапе, нет правильного или неправильного решения, есть более или менее подходящие, при этом на данном этапе, как говорят по-английски, *you call the shots* – всё в ваших руках), объектами могут быть:

- товар
- корзина
- запрос
- покупатель

Думаем, вы уже заметили, что покупателя тут нет, но, очевидно, что здесь весь сценарий подразумевает выполнение всех действий покупателем (строго говоря, то, что все действия всегда выполняются пользователем – факт совершенно неочевидный).

Следующий этап – диаграмма отношений объектов:



Наверное, вы заметили, что часть этих отношений идентична, например, между корзиной и товаром и между корзиной и заказом. Есть ли здесь противоречие? Конечно, нет: в каждый конкретный момент времени у покупателя в корзине есть *n* товаров (от 0 до, условно говоря, бесконечности, строго говоря – от 0 до количество товаров, которое есть на складе магазина!), они составляют заказ, который тоже находится в корзине (да и самих заказов может быть несколько, но мы тут не усложняем).

Для чего нужна эта диаграмма? Ну, во-первых, ~~это красиво~~ она позволяет продумать *все возможные* (с точки зрения, назовём это так, здравого смысла и/или практики) отношения между объектами, а, во-вторых – она подталкивает вас к очень важному вопросу: *все ли отношения должны быть запрограммированы?* Ведь каждая стрелочка на данной диаграмме – это тоже программный код! Но это пока ещё в теории. А вот нужно ли программно выражать эти *все* отношения или нет? – Ответить на этот вопрос можете только *вы*. И когда вы ответите на этот вопрос, вы перейдёте к последнему (слышу голоса: эй, давно пора, 11 страниц текста!) этапу: к классовым диаграммам.

§5 – Классовые диаграммы (UML diagrams)

Этот этап планирования похож на предыдущий, а отличия состоят в следующем:

- Вы используете UML-диаграмму классов (см. Википедию, статьи “Диаграмма классов” и UML) (предыдущий этап – диаграмма в свободной форме)
- На этих диаграммах будут не объекты, а классы с упомянутыми атрибутами и методами, а также отношения классов.

Язык UML вы изучаете самостоятельно.

Q: Сколько должно быть объектных диаграмм и классовых диаграмм на каждом из этих двух этапов?
A: На каждый пользовательский сценарий вы делаете одну объектную диаграмму и одну диаграмму классов
Q: Могут ли потом быть изменения?
A: Да, но они должны быть обоснованными и их объём не должен приближаться к 100%.

Согласитесь, если вы полностью меняете внутреннюю структуру проекта, значит вы не очень эффективно реализовали этап планирования (тут нам подсказывают: подсунули халтуру, чтобы не пропустить дедлайны, но мы знаем, что среди вас таких нет) и выглядит всё это подозрительно.

Ещё раз по поводу всех изменений: мы прекрасно понимаем, что любой серьёзный проект нельзя спланировать раз и навсегда за 2 месяца, а потом реализовывать его ещё 8, не внося изменений. Однако, перед всеми нами стоят следующие задачи:

- Обеспечить чёткое планирование работы
- Обеспечить как можно более однозначно определённые критерии оценивания
- Дать вам такие инструменты работы, которые позволят хотя бы наполовину (а планирование – это как минимум половина любого дела) выполнить работу к середине срока, а не делать всё за три ночи до защиты.

§6 – Чекпоинты

Чекпоинт – это промежуточный дедлайн, к которому у вас должен быть определённый результат. Таким образом, у нас получаются следующие чекпоинты (которые имеют дедлайны):

- требования
- пользовательские сценарии
- объектные диаграммы и классовые диаграммы
- показ кода #1 – можно только код
- показ кода #2 – часть скомпилированного кода или объяснить, почему пока рано
- предоставление готового проекта на тестирование за 1 месяц до защиты

К последним двум этапам вы пишете небольшую пояснительную записку, в которой указываете, что именно вы представите к рассмотрению.

§7 – Стиль кода

Одним из критериев экспертной оценки вашего проекта будет соответствие определённым требованием к оформлению кода. Умение работать с теми или иными правилами написания – важный навык для программиста. Несмотря на то, что среди программистов, пишущих на том или ином языке, есть свои соглашения, часто IT-компании вводят свои стандарты оформления кода (например, здесь вы можете ознакомиться со стандартами Google: https://google.github.io/styleguide/).

Наши стандарты значительно проще и, в отличие от стандартов крупных корпораций, охватывают не так много аспектов: комментарии, пробелы и отступы, именование переменных.

Общее правило такое: ***вы можете пользоваться любым стандартом, при условии, что он не противоречит тем правилам, которым мы просим вас соответствовать.***

§8 – Экспертная оценка

За этим несколько возвышенным словосочетанием стоит оценка, которую выставляет вам один из членов комиссии, обладающий статусом эксперта. Критерии оценки будут указаны ниже, однако, перед этим необходимо отметить, что эксперт делает следующее:

- после чекпоинта, на котором вы предоставляете объектные и классовые диаграммы, фиксирует функционал вашего проекта и составляет чек-лист;
- после того, как вы отправляете проект на тестирование, он проверяет наличие этого функционала в вашем продукте;
- выставляет баллы по критерию "экспертная оценка продукта";
- пишет небольшую характеристику вашему продукту, в которой указывает общее впечатление от вашего проекта, глубину технической проработки, возможные недостатки

Критерии экспертной оценки (4 балла в сумме):

Дедлайны (ЭК-1):

- соблюдены 4-6 технических дедлайнов – 2 балла
- соблюдены 2-3 технических дедлайна – 1 балл
- соблюдены 0-1 технических дедлайнов – 0 баллов

Мы думаем, что к моменту чтения этого документа слово "дедлайн" уже прочно вошло в вашу жизнь, поэтому комментарии тут излишни, кроме одного *строгого примечания*: если вы ***не предоставляете готовый продукт за месяц до защиты***, комиссия оставляет за собой право *снизить итоговую оценку* на 1 балл.

Стиль кода (ЭК-2):

- требования к стилю кода соблюдены или имеются небольшие отклонения – 2 балла
- имеются значительные отклонения от требований – 1 балл
- требования к оформлению кода игнорируются – 0 баллов

Потенциальное развитие проекта (ЭК-3):

- создатель понимает, как можно усовершенствовать продукт – 1 балл
- варианты развития продукта не предложены – 0 баллов

Помимо экспертных критериев необходимо обратить внимания на 2 критерия из общей части, которые формально экспертными не являются, но баллы по которым будут выставляться именно экспертами (см. сводную таблицу критериев).

Продукт (критерий D)

Этот критерий можно назвать, наверное, самым главным, поскольку именно он определяет, насколько полно вы реализовали тот функционал, который вы пообещали сделать в заявке. Более подробно с этим критерием можно ознакомиться в общей таблице критериев, здесь же мы только сделаем довольно строгое примечание к нему:

Строгое примечание: в случае 0 баллов по данному критерию оценка не может быть выше "3" по пятибалльной системе; комиссия также вправе поднимать вопрос о выставлении неудовлетворительной оценки с последующей академической задолженностью по ИВР. Подобная практика позволит реализовать простую максиму "сделал другой продукт – провалил проект". Разумеется, если функционал вашего продукта будет шире, чем вы заявили, то никаких проблем с этим не будет. Главное – не подменяйте то, что вы задумали, чем-то другим.

Ответы на *экспертные* вопросы на защите (см. критерий E):

Как известно, любая ИВР заканчивается презентацией (как её делать узнает тот, кто дочитает этот документ до конца), а каждая презентация – вопросами комиссии. Все вопросы можно условно разделить на вопросы общего характера и узко-технические. Ожидается, что вы не только создадите продукт, но и будете ориентироваться в некоторых аспектах теории и практики его создания, будете владеть некоей терминологической базой (мы не будем сейчас писать список вопросов тут, будет, скорее, список тем для разговора, появится он перед защитой и будет зависеть от того, какой именно у вас проект).

Уже сейчас, однако, можно сказать, что экспертные вопросы на защите затрагивают специальные вопросы по программированию, программной структуре проекта, принципам написания кода, понятиям программирования и т. п.

Как обычно, в таких случаях – *строгое примечание*: в случае 0 баллов по данному критерию итоговая оценка по пятибалльной системе может быть снижена на 1 балл. Если вы не можете ответить ни на один вопрос или если ваши ответы излишне пространны, невольно закладываются всевозможные нехорошие подозрения. Надеемся, у нас не возникнет необходимости пользоваться этим правилом.

§9 – Тестирование

Процедура тестирования проходит следующим образом:

- За 1 месяц до защиты вы предоставляете полностью готовый проект ***(ссылку на GitHub-репозиторий)***.
- К этому моменту у тестировщиков на основе вашей документации (функциональные требования и т. п.) будет чеклист тех функций, которые должны быть реализованы в вашем проекте.

- Вы ***обязательно*** пишете ***алгоритм тестирования***, в котором указываете:
  1) IDE, в которой проходит тестирование;
  2) необходимые SDK или иные файлы / модули / библиотеки, которые надо установить (понятно, что иногда установка IDE подразумевает установку SDK, например, IDEA и Java, но, тем не менее, мы всё равно просим указать, что именно должно быть установлено);
  3) пункты меню-команды, которые нужно выбирать в IDE. Это для тех, кто, возможно, не сразу сориентируется в IDE. Поверьте, вы заинтересованы в том, чтобы тестирование вашего кода прошло без всяких багов.
- Тестировщики смотрят, реализованы ли эти функции, просматривают код, выставляют оценки по критериям ЭК-1, ЭК-2, ЭК-3.
- В случае каких-то сбоев, зависаний, непонятной работы тестировщик связывается с вами и вы обговариваете эти проблемы. В ваших интересах быть на связи регулярно, поскольку если ваш проект не запустится, а вы пропадёте, мы не сможем его оценить по критерию критерию “Продукт”.
- В случае отсутствия каких-то функций, заявленных в документации проекта, тестировщик может вам указать на эти недочёты, вы можете их исправить и вернуть ваш проект на повторное оценивание, однако, ***повторное оценивание никому не гарантируется и не обязательно для тестировщика***. Это, скорее, ***жест доброй воли***. Подобное положение вещей обусловлено не вредностью характеров комиссии и даже не стремлением приучить вас к дедлайнам – просто на момент тестирования все могут быть очень заняты, чтобы обращаться к каждому проекту по несколько раз.
- Если всё проходит нормально, то вы записываете ***демо-ролик*** вашего проекта.

§10 – Демо-ролик

Демо-ролик представляет собой скрин-каст на 1-2 минуты, в котором вы показываете основные принципы работы с вашим продуктом. Чтобы было совсем понятнее: если ваш проект – игра, вы играете в неё в течение 1-2 минут, если сайт – вы показываете, что можно делать на этом сайте и т. п.; плюс, демонстрируете нам всяческие ”фишки” вашего проекта. Этот ролик не должен быть техническим: не стоит нам показывать скрины из IDE или структуру кода – это рекламный, “продающий” ролик, цель которого – показать, какой крутой у вас проект. Заинтересуйте им комиссию.

§11 — Презентация

Ничего необычного здесь нет. Она состоит из:

- **Введения** - 1) проблемное поле, в котором работает ваш проект, 2) потенциального/реального заказчика, 3) что для вас как программиста интересно в проблеме заказчика и её решении, что показалось вам любопытным, 4) знанием и навыками в работе с какими технологиями вы не обладали до работы над проектом и чему научились в процессе (выдумывать тут не нужно: если вы все языки и фреймворки знали до начала работы, если вы в процессе работы над проектом ничего нового не узнали, это не делает ваш проект хуже, просто так и скажите, что вы были крутым уже до начала работы).

- **Функционала/образа продукта** – какие вы задачи поставили перед собой как подрядчик, какие функции для своего проекта вы определили и что *сделано – не сделано*.
- **Этапов работы над проектом -** (правильно понимаете, да: расскажите за 2 минуты, как вы работали целый год, ага)
- **Показа демо-ролика**
- **Ответов на вопросы комиссии**
- **Ответов на вопросы эксперта**
- **Очень короткого отзыва эксперта** о вашей работе
- **Аплодисментов, цветов, автографов….**

§12 — Коммуникация

Для того, чтобы мы могли просматривать и скачивать ваши проекты, вам необходимо изучать, как пользоваться Git/GitHub. Поскольку ваши проекты не коллективные, а индивидуальные, то вам не придётся изучать эту систему досконально - только самые базовые навыки по созданию, инициализации, добавлению репозиториев. Вы так же должны уметь их коммитить и пушить. Несмотря на возможную несимпатичность этих терминов, использование Git/GitHub – это необходимый навык для любого программиста.

§13 — Библиография

В заключение хотели бы отметить, что ничего нового в этом документе мы не предложили: мы воспользовались советами профессионалов, почерпнуть которые можно в источниках, указанных ниже. Некоторые из их – англоязычные. Быть программистом и не знать английский сегодня просто невозможно, так что возражений не принимаем. А вот если будут какие-то вопросы по пониманию того, что говорится и пишется – с радостью на всё ответим.

1. Lynda.com course: Foundations of Programming: Object-Oriented Design.
2. М. Вайсфельд. Объектно-ориентированное мышление.
3. Lynda.com course: Foundations of Programming: Design Patterns.
4. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования.
5. Г. Буч, И. Якобсон, Д. Рамбо. Введение в UML от создателей языка.
6. Lynda.com course: GitHub for Web Designers (James Williamson, 2014).

Список будет пополняться. To be continued, как говорится.

§14 — Дедлайны

На тот случай, если вы прочитали, всё поняли, но не совсем уяснили, что когда необходимо сдавать, мы подготовили таблицу с дедлайнами как для тех, кто будет защищаться в первую волну, так и для тех, кто защищается во вторую.

Важно: защита в первую волну не есть репетиция защиты или некий второй шанс: если вы, например, получаете 3 в первую волну, то исправить оценку, доработав проект и защитившись во вторую волну, **невозможно**. **Пожалуйста, рассчитывайте свои силы и время**.

Защита *в мае 2018 года*:

Событие	**Дата**	Как подать заявку
1) Заявка (hard и soft)	ноябрь, 2017	Заполнить Гугл-форму
2) Пользовательские сценарии	*декабрь, 2017*	Загрузить файл на ГитХаб
3) Объектные диаграммы и диаграммы UML	*декабрь, 2017*	Загрузить файл на ГитХаб
4) Показ когда №1	31 января, 2018	ГитХаб
5) Показ когда №2	20 марта, 2018	ГитХаб
6) Готовый продукт	1 апреля, 2018	ГитХаб

Защита *в ноябре 2018 года*:

Событие	**Дата**	Как подать заявку
1) Заявка (hard и soft)	ноябрь, 2017	Заполнить Гугл-форму
2) Пользовательские сценарии	январь, 2018	Загрузить файл на ГитХабе
3) Объектные диаграммы и диаграммы UML	февраль, 2018	Загрузить файл на ГитХабе
4) Показ когда №1	март, 2018	ГитХаб
5) Показ когда №2	май, 2018	ГитХаб
6) Готовый продукт	15 октября	ГитХаб

Комментарии:

1) Для того, чтобы отправить заявку, необходимо заполнить Гугл-форму: https://docs.google.com/forms/d/e/1FAIpQLSeaKr_hI8-AoUnH52g09FrDBdU9PKrOrBBL2tqF7GNLSU9FEA/viewform

2) Начиная со второго дедлайна, вам необходимо иметь аккаунт и репозиторий вашего проекта на ГитХабе. Ссылку на ГитХаб впишите в Гугл-таблицу: https://docs.google.com/spreadsheets/d/1tZLzfZxyBZiaathrgVQolkKvnWdnu-YDu-W5stesZ6w/edit?usp=sharing

3) пользовательские сценарии должны быть в корне репозитория; файл должен называться *ivanov_user_scripts*, где первая часть имени файла – ваша фамилия

4) объектные диаграммы и UML-диаграммы в должны быть в корне репозитория; файл должен называться ivanov_diagrams, где первая часть имени файла – ваша фамилия

Авторы документа:
Купцов А. А., Сёмина С. Е., кафедра исследовательской и проектной деятельности

Контакты: akuptsov.hse@gmail.com https://vk.com/alekscooper