МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ПЕДАГОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И
ТЕХНОЛОГИЧЕСКОГО ОБРАЗОВАНИЯ**
Кафедра информационных технологий и электронного обучения

Основная профессиональная образовательная программа
Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль) «Технологии разработки программного обеспечения»
форма обучения – очная

Курсовая работа

«Итеративная модель в управлении программным проектом по созданию электронного образовательного ресурса»

Обучающегося 3 курса

Лабырина Матвея Сергеевича

Научный руководитель:
Кандидат физико-математических наук,
доцент кафедры ИТиЭО
Жуков Николай Николаевич

Санкт-Петербург
2020

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3
Глава 1. Основыне требования к итоговому продукту 4
Глава 2. Обзор методологий разработки 5
2.1 Классические модели разработки 6
2.3 Гибкие модели разработки 8
2.3 Итоговый выбор методологии 11
Глава 3. Проектирование Use-case диаграммы и этапов разработки 14
3.1 Use-case диаграмма 14
3.2 Проектирование этапов разработки 15
Глава 4. Выбор средств и инструментов программирования 16
ЗАКЛЮЧЕНИЕ 17
СПИСОК ЛИТЕРАТУРЫ 18

ВВЕДЕНИЕ

Управление программным проектом – сложная задача, требующая не только понимание того, как должен выглядеть итоговый продукт, какая функциональность должна быть реализована и т.д., но и понимание того, какая методология управления разработки программным проектом подходит в данном случае, какие сервисы использовать для управления разработкой, какие выбрать языки и фреймворки и т.д. Грамотный выбор этих элементов значительно увеличивает эффективность работы команды, а также качество итогового продукта.

В данной работе будут проанализированы многие нюансы процесса разработки и подобраны самые оптимальные и подходящие решения.

1 Основные требования к итоговому продукту

Требования по функциональным характеристикам

Продукт (веб-сайт) должен обеспечивать возможность выполнения перечисленных ниже функций:

1. Возможность регистрации и аутентификации в свой аккаунт при помощи логина и пароля.
2. Возможность просматривать учебные материалы (видео, текстовые материалы, ссылки на ресурсы и т.д.).
3. Добавление учебных материалов (преподавателями).
4. Коммуникация как между студентами, так и преподавателей со студентами.
5. Загружать студентами результаты выполненных работ в виде файлов.

2 Обзор методологий разработки.

При разработке какого-либо программного продукта рано или поздно нужно будет сделать выбор: каким образом создавать продукт. От выбора методологии зависит, будем ли мы жестко планировать все этапы разработки и выполнять их шаг за шагом, работать короткими итерациями, с возможностью отследить результативность и вовремя внести правки в прототип или же разбить работу на несколько не зависящих друг от друга частей, и для каждой из них выделять отдельную команду. Моделей и методологий разработки продукта десятки, и каждая из них имеет свои преимущества и недостатки.

Модель разработки продукта — это то, через какие стадии он проходит во время создания и что происходит на каждой из них. Если речь идет о разработке программного обеспечения, то, как правило, проект проходит через шесть основных фаз:

- планирование**.** Определяется, что надо сделать первым делом и какие проблемы надо решить. Ставятся цели, выясняется, какие ресурсы понадобятся для реализации проекта. Изучается рынок и конкуренты, прорабатываются альтернативные варианты разработки продукта;

- анализ системы**.** Определяются и документируются требования конечного пользователя системы;

- дизайн**.** Определяются элементы системы, ее компоненты, уровень безопасности, архитектура, интерфейсы, типы данных. Разрабатывается примерный вид будущего продукта;

- разработка и внедрение**.** К началу этой стадии дизайн уже завершен, наступает очередь разработки. Пишется код, настраивается система под определенные требования и функции. К концу фазы система готова к установке и запуску;

- тестирование**.** Проверяется, получилось ли в итоге то, что требовалось, или же результаты работы оказались другими. Тестируется продукт автоматизированными тестами, командой, предлагается поработать с системой потенциальным пользователям. Определяются дефекты и недостатки в работе системы;

- поддержка системы. Подготовка и выпуск обновлений, оценка производительности системы, замена/деактивация устаревших компонентов.

Это классическая схема, которая изменяется в зависимости от того, каким образом идет разработка, а также от специфики работы команды, размера бюджета и особенностей продукта. Именно здесь и наступает момент выбора модели разработки. Они подразделяются на классические и гибкие. Для начала проведём краткий обзор основных моделей и после определимся с выбором.

2.1 Классические модели разработки

Классические модели предполагают акцент на последовательности, сроках, конечных требованиях к продукту. Далее будут кратко описаны основные классические модели.

Code-and-Fix

Это одна из самых старых моделей разработки: она очень проста и подойдет небольшим проектам, где команда невелика, нет особых конфликтов, все знают, что нужно сделать и имеете представление, как это сделать.

Как работает Code-and-Fix: Команда понимает, какой продукт должен получится в итоге и начинает его создавать, на ходу находя баги и неточности и исправляя их.

Преимущества этого метода в его простоте. Максимум времени тратится непосредственно на работу с кодовой базой. Из недостатков: исправление одной ошибки может привести к тому, что сломается вся система. Так же такой подход не подходит для больших проектов, в которых очень важна грамотная архитектура продукта.

Waterfall

Это классическая жесткая модель: есть план и бюджет. Все этапы выполняются строго друг за другом.

Разработка продукта идет ступенями: изучаются требования, проектируется продукт, разрабатывается, тестируется и затем поддерживается его работа. Новый этап не начинается, пока не завершится предыдущий. Ошибки, допущенные на этапах изучения требований и проектирования, очень сложно исправить — как в финансовом плане, так и в техническом — и это самая большая проблема этой модели.

Водопадная модель хороша при разработке продукта, очень похожего на какой-либо другой: в таком случае вы заранее знаете какие проблемы могут возникнуть, сколько времени уйдёт на тот или иной эта и т. д. Поскольку проект — это не всегда разработка абсолютно нового продукта, иногда это создание усовершенствованной версии существующего решения, то каскадная модель может оказаться подходящим вариантом при выборе модели разработки. Главное – правильно рассчитать силы команды, бюджет и время.

V-Model

V-Model — улучшенная версия водопадной модели. В отличие от последней V-model предполагает тщательную проверку и тестирование продукта уже на ранних стадиях разработки — оба процесса идут параллельно. При переходе на следующий этап разработки предусмотрен контроль всего, что было сделано до этого. Все найденные ошибки устраняются, и только затем наступает новая фаза работы над продуктом (Рис. 1).

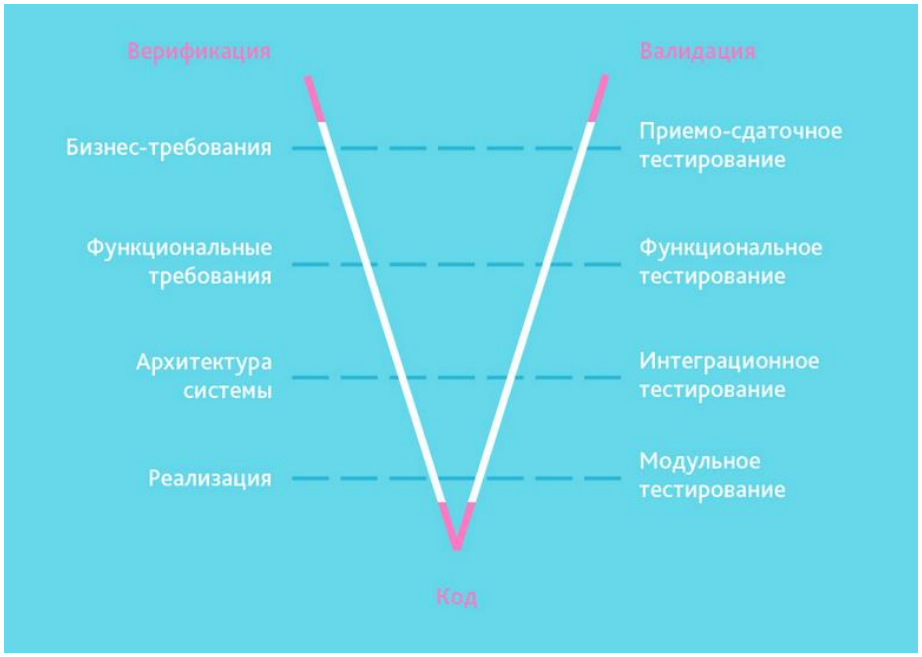

Рис. 1

Тестирование начинается еще на стадии написания требований, для каждой фазы разработки предусмотрен свой тест-план. Кроме того, уже во время проверки текущего уровня идет разработка стратегии тестирования для следующего. При создании тестов определяются ожидаемые результаты тестирования, указываются критерии входа и выхода для каждого этапа работы над продуктом.

Инкрементальная модель

Эта модель разработки дает возможность делать продукт по частям — инкрементам. Каждая часть представляет собой готовый фрагмент итогового продукта, который в идеале не переделывается. Получается несколько циклов разработки — своеобразный “мультиводопад”. Каждый цикл делится на модули, каждый модуль — на фазы: определение требований, проектирование, написание кода, внедрение, тестирование.

Такая модель позволяет проработать риски и вывести базовую версию продукта на рынок, когда весь запланированный функционал может быть еще в стадии проектирования. Небольшие проекты могут ее использовать, если у них есть общее понимание, каким должен быть их продукт в итоге.

Спиральная модель

Эта модель схожа с инкрементной, однако имеет существенную отличительную особенность — детальную проработку рисков. Спиральная модель применяется для ведения критически важных проектов, где неудача приведет к закрытию компании. Таким образом, маленькие проекты вполне могут ее применять, ведь существование проекта напрямую связано с тем, понравится ли его продукт рынку.Разработка идет по спиралям, каждая из которых состоит из четырех основных этапов: планирования, анализа рисков, конструирования, оценки результатов.

Эту модель хорошо использовать для работы над продуктом проекта в том случае, если вы еще не определились с конечными требованиями к нему, но знаете, что в любом случае они достаточно сложные. В там случае, работая итерациями и планируя риски, можно благополучно выпустить продукт на рынок и затем постепенно его дорабатывать.

Итеративная модель

Для того, чтобы начать работать с этой моделью, не нужно иметь все требования и спецификации. Продукт создается таким образом, что в первую очередь вы создаете базовый работающий функционал. Затем с каждой итерацией вы совершенствуете его, добавляя новые функции (Рис. 2).

Итеративную модель можно сравнить с картиной, когда сначала рисуется некий набросок, исходя из которого можно увидеть, что будет изображено. С каждым шагом добавляются новые цвета и переходы. В итоге получается готовая картину. А в случае небольшого проекта — готовый продукт. Модель подходит для проекта, который хочет как можно быстрее выйти на рынок и привлечь клиентов.

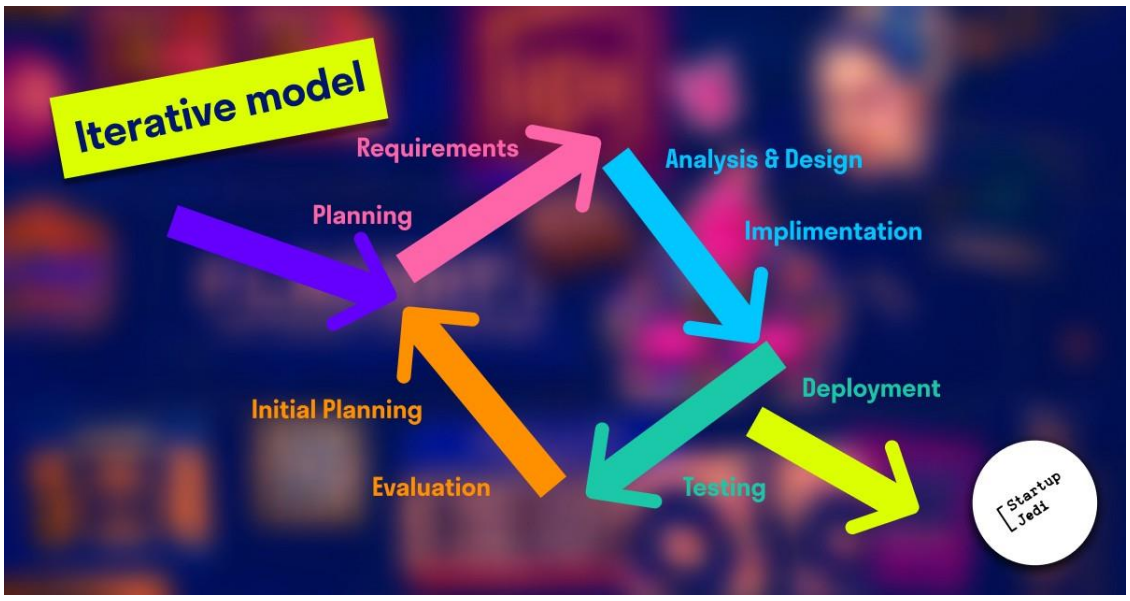

Рис. 2

2.2 Гибкие модели разработки

Agile (agile software development, от англ. agile – проворный) – это семейство «гибких» подходов к разработке программного обеспечения. Такие подходы также иногда называют фреймворками или agile-методологиями. Гибкими моделями разработки пользуется большинство ИТ-проектов. Благодаря им возможно получать определенный результат после каждой итерации, позволяет вносить изменения в первоначальное видение продукта

практически в любой момент работы — исходя из того, какой функционал больше нужен пользователям продукта.

Минусы гибких моделей — готовый продукт может на рынок так никогда и не выйти, команда постоянно будет заниматься его усовершенствованиями, дополнениями, тем временем бюджет может закончиться. Плюсы — чтобы начать работать над продуктом не нужно иметь детальное представление о том, что желательно получить в конце. Не нужно иметь весь бюджет и просчитывать все риски. Agile имеет множество вариаций и фреймворков, которые будут перечислены далее.

Kanban

Метод базируется на концепции бережливого производства, основанной на стремлении к устранению всех потерь — временных, производственных, логистических, качественных.

Особенность Kanban — задачи должны выполняться точно в срок, нагрузка между командой распределяется равномерно. На практике это выглядит следующим образом. Каждая задача по проекту описывается в отдельной карточке и добавляется на доску — виртуальную или настоящую. Карточка и доска — неотъемлемые элементы Kanban. Все задачи, которые необходимо сделать, собраны в специальной колонке, условно, она может называться “сделать”/ “to do”. Исполнитель выбирает задачу и перемещает в колонку “в процессе” / “in progress”. Когда задача сделана, она попадает в соответствующую колонку “готово” / “done”. На практике колонок может быть гораздо больше, чем три. К примеру, колонки на доске могут выглядеть так: “обсуждается” (backlog), “согласовано” (ready), “кодируется” (coding), “тестируется” (testing), “подтверждается” (approval) и “сделано” (done).

Scrum

Одна их самых популярных методологий Agile. В отличие от канбан, у скрама гораздо больше элементов — различные митинги (от ежедневных пятиминутных, до планирований спринтов, демо), четкое разделение по ролям. Кроме того, разработка подразделяется на спринты — которые длятся от недели до четырех недель и заканчиваются выпуском части продукта.

В переводе с английского scrum — это драка либо схватка вокруг мяча. Термин пришел из регби и означает специфическую игровую ситуацию, в которой участники команд смыкаются в три линии с каждой стороны, когда в игру вводится мяч после нарушения правил; задача игроков — выиграть за счет совместных усилий команды.

Почти так же применяется методология Scrum при разработке ПО: команда кросс-функциональных специалистов совместно реализует проект. Важно, чтобы это были профессионалы с высокой мотивацией, инициативные и готовые вносить изменения как в ход реализации проекта, так и в продукт.

Рассмотрим пример применения Scrum:

1. На этапе формирования продукта команда решает, кто будет исполнять роль product owner — человека, который отвечает за связь команды с потребителем и инвесторами (если инвесторы будут интересоваться ходом разработки продукта).

2. Product owner формирует список пожеланий к продукту, собирает первичную информацию от возможных пользователей и затем формирует бэклог продукта — список задач, выполнение которых в конце концов приведет вас к выходу на рынок.

3. Команда определяет размер спринта — периода, в конце которого команда должна сделать какой-то рабочий кусок продукта, и выбирает задачи для первого спринта из бэклога.

4. В течение первого спринта вы отслеживаете качественные и количественные характеристики своей работы. Неотъемлемая часть скрама — ежедневные короткие (5–10 минут) митинги, в течение которых каждый из участников команды рассказывает, что он планирует сделать за день, делится возникающими сложностями или, наоборот, успехами.

5. Ход выполнения задач отслеживается по скрам-доске, на которой все задачи двигаются от условной позиции “сделать” до “выполнено”.

6. По завершении спринта команда демонстрирует выполненную часть работы и собирает обратную связь — от членов команды, клиентов, в т.ч. потенциальных.

7. Цикл спринтов повторяется до того момента, пока продукт не будет полностью завершен.

Экстремальное программирование (XP)

eXtreme Programming, экстремальное программирование, XP — гибкая методология разработки, которая появилась в конце 90-х годов прошлого столетия. Авторы взяли лучшие, на их взгляд, практики гибкой разработки и усилили их до максимума — отсюда и слово “экстремальный” в названии.

В отличие от канбанф и скрама, которые можно применять в самых разных проектах и бизнесах, а также в организации личных дел, XP применяется исключительно в разработке программных продуктов. В рамках экстремального программирования выделяются четыре процесса: кодирование, тестирование, дизайн, слушание. Если описать эту методологию несколькими словами, то ее характеризуют оперативность, высокое качество, командная работа.

Особенностью XP являются некоторые практики, самая известная из которых — парное программирование. Суть его заключается в том, что два разработчика одновременно работают над кодом для одной функции продукта: сначала один пишет, а второй наблюдает и исправляет ошибки, затем они меняются местами. Таким образом, в процессе создания кода есть два альтернативных решения, на каждом этапе выбирается лучшее. Парное программирование работает по принципу: одна голова — хорошо, а две лучше.

Другая особенность экстремального программирования заключается в том, что сначала готовятся тесты, и только потом — код. При этом тесты пишут сами программисты. Тестирование позволяет исправить большинство ошибок на стадии создания кода.

Третья особенность — коллективное владение кодом: каждый программист в команде имеет доступ к коду продукта и может вносить в него изменения. В том случае, если изменения привели к некорректной работе системы, исправить все должен тот программист, который внес эти изменения.

Экстремальное программирование предполагает также работу в рамках небольших релизов — от одного дня до месяца. При этом чем короче релизы, тем лучше качество продукта. Наконец, интеграция новых частей в систему происходит так быстро как это возможно. Как только тесты показали, что функция работает корректно, она интегрируется в систему.

2.3 Итоговый выбор

Жизненный цикл проекта при итерационной разработке разбит на последовательность итераций, каждая из которых, по сути, является проектом в миниатюре, то есть включает в себя все процессы разработки ПО (сбор и анализ требований, составление спецификаций, непосредственную реализацию, тестирование и запуск), но в рамках одной итерации разрабатывается не весь проект, а только его версия или отдельная часть. Т.е. на каждом этапе разработки будет иметься рабочая модель проекта, хоть и с различными недочётами и недоработками. В случае с выбранным проектом это нужно и важно – можно будет получать фидбэк с самых первых итераций продукта.

Как правило, цель каждой итерации — это получение версии ПО, включающей в себя как новые или переработанные возможности, реализованные в ходе текущей итерации, так и функциональность всех предыдущих итераций. Результат же финальной итерации содержит всю требуемую функциональность продукта.

Бюджет и сроки, необходимые для реализации финальной версии обычно изначально не устанавливаются (как и в нашем случае), так как не определяется общий объём работ и требования формируются по ходу реализации.

Рис.3

Для демонстрации преимуществ итеративного подхода можно сравнить его с Waterfall на примере автомобиля:

В случае с «водопадом» сначала описываются требуемые характеристики автомобиля, затем по этим требованиям разрабатывается проектная документация. После составления проектной документации собираются отдельные узлы автомобиля и происходит их взаимная интеграция. Результат сборки тестируется на соответствие проектной документации и после это созданный автомобиль передается заказчику. Все эти этапы занимают достаточно продолжительное время, а пригодный для использования продукт заказчик получает только в самом конце.

В случае с итеративным подходом всё несколько иначе. Изначально ставится задача разработки транспортного средства. И результатом первой итерации может быть вариант такого транспортного средства — например, самокат. Для него не нужен двигатель внутреннего сгорания и собрать его можно в десятки раз быстрее, чем автомобиль. Да, самокат проигрывает автомобилю по очень многим характеристикам, но он всё же более эффективен для передвижения, чем хождение пешком. Результатом второй итерации может быть уже самокат с электродвигателем. На третьей итерации — у самоката могут быть увеличены колеса, и он превратится в электровелосипед. На четвертой — электровелосипед может быть оснащён ДВС и станет мотоциклом.

По сути, с каждой итерацией повышаются функциональные возможности. И пока команда, работающая по «водопада» ждут готовность создаваемого автомобиля, команда с итерационным подходом уже пользуются транспортным средством. И вполне может быть, что получившийся в итоге мотоцикл — более правильный бизнес-результат.

Каковы же основные преимущества итеративной модели разработки? Снижение рисков — раннее обнаружение конфликтов между требованиями, моделями и реализацией проекта; организация эффективной обратной связи проектной команды с потребителем, создание продукта, реально отвечающего его потребностям; быстрый выпуск минимально ценного продукта;

Все эти преимущества отлично подходят для нашего проекта, т. к. мы не знаем точно, как должен выглядеть итоговый продукт, но имеются представления об основной функциональности и общей направленности продукта.

3 Проектирование Use-case диаграммы и этапов разработки

3.1 Use-case диаграмма

Use-case диаграммы нужны для описания функциональности и поведения, позволяя тем самым наглядно рассматривать проектируемую или уже существующую систему. Для нашего проекта эта диаграмма нужна для более полного понимания, какую систему нужно построить. Поскольку в проекте используется итеративная модель разработки, то со временем Use-case диаграмма может меняться и дорабатываться.

Для начала построена достаточно простая диаграмма, отвечающая основным пунктом технического задания (Рис. 3).

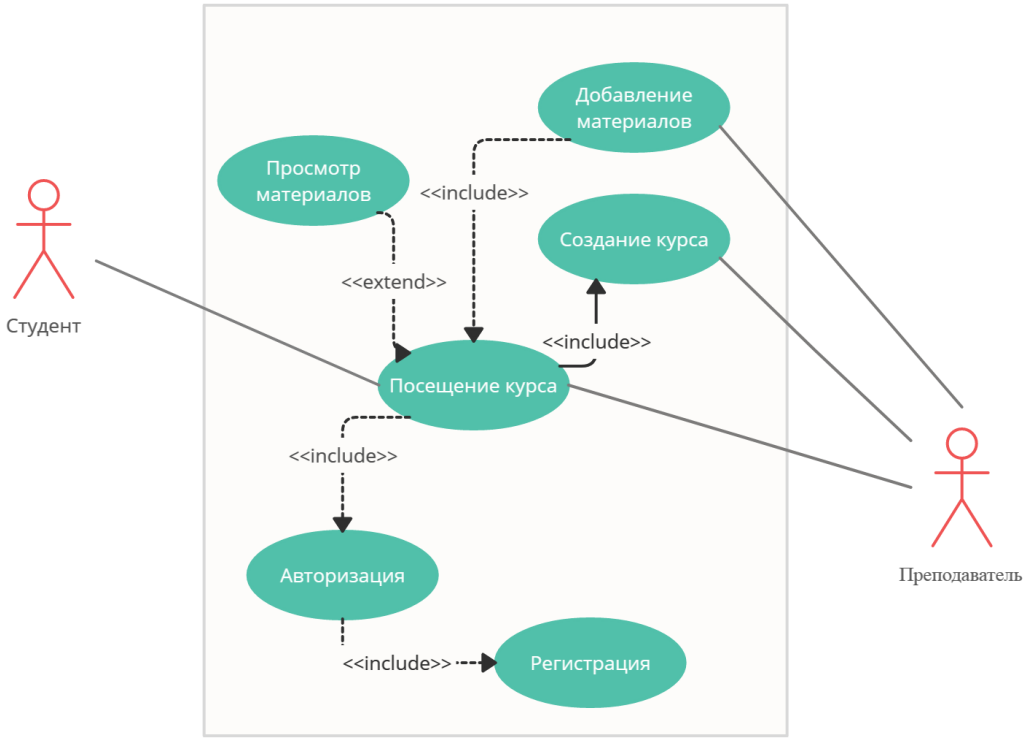

Рис. 3

Как можно увидеть из диаграммы, существует два вида пользователей: студенты и преподаватели. Им обоим для пользования ресурсом требуется сначала авторизоваться (а также зарегистрироваться, если это не было сделано ранее), после чего можно просматривать учебные материалы. Но преподаватели, в отличие от студентов, так же могут создавать новые курсы и добавлять контент в уже созданные ими курсы (будь то новый раздел, видеозапись, описание задания, форум, pdf-файл или же ссылка на другой ресурс).

3.2 Проектирование этапов разработки

Поскольку нами выбрана итеративная модель разработки, работа над проектом будет разбита на циклы. На [Рис. 4] изображён один цикл разработки, длящийся 4 недели.

Рис. 4

На диаграмме видно, что цикл разбит на 4 основных этапа. На первом этапе строятся и фиксируются планы на текущую итерацию. Выдвигаются определённые требования, которые должны быть достигнуты в конце текущего цикла. Как правило на первых итерациях строится основной «костяк» проекта, а не последующий наращивается новый функционал и модернизируется старый. Вторым этапом идёт дизайн и проектирование. На этом этапе продумывается архитектура, структура и внешний вид разрабатываемой части приложения (в нашем случае сайта). Так же если стоит цель сделать новый модуль, то продумывается каким образом он будет взаимодействовать с остальными элементами сайта. Третий этап посвящён непосредственной реализации того, что продумывалось в предыдущих двух пунктах. Так же во время разработки проводится тестирование для выявления багов и неточностей работы. Последним этапом идёт обзор проделанной работы за текущий цикл. Проводится анализ всех пройденных этапов и делаются соответствующие выводы (например, что в следующей итерации надо большее время уделить тестированию или же более точно составлять требования на первом этапе).

Таким образом каждая итерация разработки не только добавляет новый компонент проекта (или же улучшает старый), но и повышает эффективность последующих итераций благодаря анализу каждого витка. После прохождения цикла нововведения отправляются на рабочую кодовую базу.

4 Выбор средств и инструментов программирования

Для создания клиентской части сайта будет использована классическая связка HTML как язык разметки, CSS как язык стилей и JavaScript для программирования поведения. Будет использоваться ванильный JS без каких-либо фреймворков для того, чтобы сайт был максимально производительным. В дальнейшем можно будет переписать на нужный фреймворк по мере надобности.

Для написания серверной части будет использован серверный фреймворк Django языка Python в связке с СУБД MySQL. MySQL взята благодаря достаточно быстрой и простой разработке а также поддержке.

Для общения был выбран мессенжер telegram, поскольку изначально проект небольшой и функциональности telegram достаточно. В дальнейшем возможен переход на slack. В нём же будет происходить планирование разработки и различные организационные моменты.

В качестве системы контроля версий выбран Git в связке с сервисом GitHub т.к. на данный момент это является стандартным и лучшим решением. Также появляется возможность выложить кодовую базу сайта в открытый доступ и принимать помощь от сообщества.

ЗАКЛЮЧЕНИЕ

Были рассмотрены основные модели разработки программного обеспечения и был сделан выбор в пользу итеративной модели. Была создана Use-case диаграмма и диаграмма Ганта, спроектирована структура циклов разработки. Был сделан выбор инструментов программирования и языков для создания продукта.

СПИСОК ЛИТЕРАТУРЫ

1. Портал «Профессионал управления проектами» http://www.pmprofy.ru
2. Грекул В.И., Коровкина Н.Л., Куприянов Ю.В. Методические основы управления ИТ-проектами.
3. Прсоветов Г. И. Управление рисками: задачи и решения: Учебно-практическое пособие / Г.И. Просветов. М. : Альфа-Пресс, 2008.
4. https://qalight.com.ua/baza-znaniy/iterativnaya-model-iterative-model/
5. Статья «Ещё раз про семь основных методологий разработки» https://habr.com/ru/company/edison/blog/269789/