

Содержание

Введение.....	2
1. Исследовательский раздел.....	4
1.1 Что такое управление проектами.....	4
1.2 Какие факторы ведут к успеху.....	5
1.3 Обзор методологии <i>Scrum</i>	7
1.4 Применение системы управления проектами на основе методологии скрам.....	9
2. Специальный раздел.....	13
2.1 Диаграммы прецедентов.....	13
2.2 Диаграммы взаимодействий.....	22
2.3 Диаграмма классов.....	30
3. Технологический раздел.....	33
3.1 Архитектура.....	33
3.2 Технологии, используемые в проекте.....	36
Заключение.....	40
Список литературы	

Введение

Для того чтобы понять, насколько актуальна тема управления проектами необходимо дать определение термина «проект».

Проект (от лат. *projectus* — брошенный вперед, выступающий, выдающийся вперед) — замысел, идея, образ, воплощенные в форму описания, обоснования, расчетов, чертежей, раскрывающих сущность замысла и возможность его практической реализации. [1]

Проект — это работы, планы, мероприятия и другие задачи, направленные на создание уникального продукта (устройства, работы, услуги). [1]

Таким образом под определение проекта попадает практически любая деятельность человека, характеризующаяся такими признаками как:

- 1) ограниченность во времени;
- 2) конкретность, достижимость результата;
- 3) наличие плана действий;

Естественно желать, чтобы деятельность человека была успешной, то есть чтобы проекты завершались в срок и без превышения требований к ресурсам. Однако даже в современном мире далеко не все проекты заканчиваются успешно. Многие даже не доходят до своего логического завершения, а некоторые вообще тянутся годами, потребляя как временные так и человеческие ресурсы.

Особенно остро данная проблема чувствуется в области разработки программного обеспечения. Так, в исследовании *Standish Group* под названием «*Chaos Report*» от 1995 года говорится, что в США в год тратится более 250 миллиардов долларов на реализацию примерно 175 000 программных проектов. Средняя стоимость разработки проекта для крупной кампании составляет более 2 000 000 долларов, для компании средних размеров более 1 000 000 долларов, для небольшой компании — порядка полумиллиона долларов. При этом большинство проектов проваливаются. *Standish Group* выяснила, что работа над

					1304.115190.000 ПЗ	Лист
						2
Изм.	Лист	№ докум.	Подпись	Дата		

31,1% проектов прекратится еще до их логического завершения, 52,7% проектов будут стоить на 89% дороже изначальной стоимости. Так в 1995 году частные и государственные компании США потратили 81 миллиард долларов на программные проекты, разработка которых не будет завершена, 59 миллиардов долларов на программные проекты, завершенные с сильным опозданием. При этом речь не идет о сверхсложных системах. Провальные проекты в большинстве своем также просты как рядовая система учета заказов. [2]

Несмотря на то, что данному исследованию уже 20 лет, опыт автора подсказывает, что ситуация в сфере разработки программных продуктов за прошедшее время не сильно изменилась. Многие проекты все также затягиваются или проваливаются. Следовательно обозначенная тема будет актуальна до тех пор, покуда будут существовать проекты.

Целями данной работы является исследование различных методологий управления разработкой программных продуктов, нахождение оптимальной методологии для управления разработкой программных продуктов в сфере решений для управления и поддержки бизнеса, разработка автоматизированной информационной системы на основе выбранной методологии для упрощения, рационализации и стандартизации взаимодействия между заказчиком разработки и командой, реализующей программный продукт, оптимизации работы команды.

В данной работе решаются вопросы взаимодействия между разработчиками, конечными пользователями, заказчиками и спонсорами разработки программного продукта, вопросы эффективной организации команды разработки, рассматриваются полезные практики, увеличивающие шансы на успех при разработке программного продукта. Также анализируются факторы, наличие которых приводят к задержке выпуска программного продукта, перерасходам, низкой эффективности команды разработки.

					1304.115190.000 ПЗ	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

1. Исследовательский раздел.

В данном разделе охватываются вопросы о том что такое проект, какова специфика программных проектов. Также освещаются факторы успеха программных проектов и рассматриваются подходы к автоматизации управления разработкой программного обеспечения.

1.1 Что такое управление проектами?

Управление проектом — это управление процессом его реализации. В свою очередь реализация проекта — это комплекс мер, дел и действий, направленных на достижение целей проекта. Таким образом, управление проектом - это управление комплексом мер, дел и действий, направленное на достижение целей проекта.[3]

Довольно любопытен тот факт, что при всей понятности приведенного выше определения, не у всех организаций получается производить грамотное управление проектом. Кажущаяся простота данного процесса является обманчивой. Это также верно и для программных проектов. Даже при условии наличия грамотного менеджера проекта, достаточного количества ресурсов, адекватной оценке сроков возможен срыв проекта из-за неповоротливости команды, из-за проблем с коммуникациями между заказчиком проекта и командой разработки. Так же может произойти следующее — проект разработан, протестирован и сдан вовремя без перерасходов по финансам, однако им никто не пользуется. Очевидно, что в данной ситуации обратная связь между командой разработки и конечным пользователем была очень слабой или ее вообще не было. Такой проект также нельзя считать успешным по причине его убыточности для заказчика. Следовательно в контексте управления разработкой программных продуктов данное определение необходимо дополнить. Автор в своем исследовании сформулировал следующее определение.

					1304.115190.000 ПЗ	Лист
						4
Изм.	Лист	№ докум.	Подпись	Дата		

Управление проектами — это комплекс мер, дел и действий направленный на достижение целей проекта путем создания комфортных условий работы для исполнителей проекта, установление и поддержка качественной и своевременной связи между исполнителями, заказчиком и конечным пользователем.

Как видно, в данном определении фигурируют заказчик, исполнитель проекта и конечный пользователь. По мнению автора эти действующие лица можно считать тремя китами в управлении разработкой программного продукта. Без заказчика не возникнет идеи проекта, без исполнителя его некому будет реализовывать, а если не спрашивать мнения конечного пользователя о реализуемом проекте, то есть риск сделать проект, который будет нравиться только заказчику или только исполнителю.

1.2 Какие факторы ведут к успеху проекта?

Как у любого другого процесса у разработки программного обеспечения имеются факторы как ведущие к успеху, так и влекущие к провалу. Рассмотрим, какие причины приводят к успеху разработку программного продукта.

Для того чтобы понять причины следствия необходимо определить, что именно мы считаем успехом разработки программного продукта?

Автор выделяет следующие критерии, при наличии которых можно считать разработку программного продукта успешной:

1. Проект уложился в бюджет;
2. В проекте реализована вся заявленная функциональность;
3. Проект выполнен в срок;
4. Все функции программного продукта выполняются по указанным в документации сценариям при любых действиях пользователя;

Рассмотрим каждый пункт подробнее и выясним, как можно добиться такого результата при работе над проектом.

					1304.115190.000 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Проект способен уложиться в бюджет при условии, что оценка его стоимости (учитывая погрешности) произведена адекватно, в процессе работы над проектом не добавляется новых "быстрых" задач, которые накапливаются как снежный ком и отвлекают исполнителя от основной работы. Адекватная оценка стоимости проекта может произвести только эксперт с опытом реализации подобных проектов. Однако в некоторых случаях оценка не устраивает заказчика и он пытается снизить цену, что приводит либо к урезанию возможностей, либо к увеличению сроков.

Перейдем к рассмотрению реализации заявленной функциональности. Самое главное в этом пункте - должен быть составлен список конкретных задач по проекту. При этом задачи должны быть поставлены правильно. Они должны быть количественно измеримы, достижимы, актуальны.

В любом проекте во время реализации список задач имеет свойство раздуваться, дополняться "быстрыми" задачами. Так же старые задачи изменяются и дополняются новыми требованиями. Чтобы функциональность реализовывалась в полном объеме следует либо составлять конечный список задач в начале проекта и ни в коем случае ничего не менять до окончания проекта, либо работать над проектом итеративно, реализуя небольшую часть задач за короткий промежуток времени и производя переоценку приоритетов в конце каждой итерации. Первый способ весьма неэффективен при разработке средних и больших проектов. Невозможно учесть все требования в самом начале проекта. Если руководствоваться таким принципом, то заказчик постарается собрать в одном техническом задании все возможные требования, несмотря на то, что эти требования могут противоречить друг другу или быть просто ненужными для конкретно этого проекта. Очевидно, что такой проект затянется.

Таким образом второй способ ведения проекта - итеративный - наиболее эффективен в плане реализации всей необходимой функциональности. Данный метод также хорошо поддерживает изменения в требованиях.

					1304.115190.000 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Четвертый пункт подразумевает наличие достаточного опыта в проектировании программ у разработчиков и четкость, конкретность поставленных заказчиком задач. Если будут соблюдены два этих условия, то реализуемая функция будет выполняться только по тем сценариям, которые будут указаны в документации.

1.3 Обзор методологии *Scrum*.

Приведенные выше требования к работе над программным проектом реализуются в семействе гибких методик разработки среди которых особенно выделяется методология *Scrum*.

Scrum (от [англ. scrum](#) «толкучка») — методология управления проектами, активно применяющаяся при разработке информационных систем для [гибкой](#) разработки программного обеспечения. *Scrum* чётко делает акцент на качественном контроле процесса разработки. Кроме управления проектами по разработке ПО, *Scrum* может также использоваться в работе команд поддержки программного обеспечения (*software support teams*), или как подход управления разработкой и сопровождением программ.

Скрам (*Scrum*) — это набор принципов, на которых строится процесс разработки, позволяющий в жёстко фиксированные и небольшие по времени итерации, называемые спринтами (*sprints*), предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определён наибольший приоритет. Возможности ПО к реализации в очередном спринте определяются в начале спринта на этапе планирования и не могут изменяться на всём его протяжении. При этом строго фиксированная небольшая длительность спринта придаёт процессу разработки предсказуемость и гибкость.

Спринт — итерация в скраме, в ходе которой создаётся функциональный рост программного обеспечения. Жёстко фиксирован по времени. Длительность

					1304.115190.000 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

одного спринта от 2 до 4 недель. В отдельных случаях, к примеру согласно *Scrum* стандарту *Nokia*, длительность спринта должна быть не более 6 недель. Тем не менее, считается, что чем короче спринт, тем более гибким является процесс разработки, релизы выходят чаще, быстрее поступают отзывы от потребителя, меньше времени тратится на работу в неправильном направлении. С другой стороны, при более длительных спринтах команда имеет больше времени на решение возникших в процессе проблем, а владелец проекта уменьшает издержки на совещания, демонстрации продукта и т. п. Разные команды подбирают длину спринта согласно специфике своей работы, составу команд и требований, часто методом проб и ошибок. Для оценки объема работ в спринте можно использовать предварительную оценку, измеряемую в очках истории. Предварительная оценка фиксируется в бэклоге проекта. На протяжении спринта никто не имеет права менять список требований к работе, внесённых в бэклог спринта.

Бэклог проекта — это список требований к функциональности, упорядоченный по их степени важности, подлежащих реализации. Элементы этого списка называются «пожеланиями пользователя» (*user story*) или элементами бэклога (*backlog items*). Бэклог проекта открыт для редактирования для всех участников скрам процесса.

Бэклог спринта — содержит функциональность, выбранную владельцем проекта из Бэклога проекта. Все функции разбиты по задачам, каждая из которых оценивается скрам-командой. Каждый день команда оценивает объем работы, который нужно проделать для завершения спринта.

Диаграмма, показывающая количество сделанной и оставшейся работы. Обновляется ежедневно с тем, чтобы в простой форме показать подвижки в работе над спринтом. График должен быть общедоступен.

Существуют разные виды диаграммы:

- диаграмма сгорания работ для спринта — показывает, сколько уже задач сделано и сколько ещё остаётся сделать в текущем спринте;

- диаграмма сгорания работ для выпуска проекта — показывает, сколько уже задач сделано и сколько ещё остаётся сделать до выпуска продукта (обычно строится на базе нескольких спринтов);

Методология скрам необходима но не достаточна для успешного управления разработкой ПО. Еще одним немаловажным фактором будет наличие автоматизированной информационной системы управления проектами, основанной на данной методологии.

Рассмотрим, почему возникает необходимость в такой системе. Если ее нет, то вся работа по записи, оценке и распределению пользовательских историй будет вестись вручную в электронных документах, в тетрадях или на досках. Ко всему прочему часто требуется наглядная отчетность по выполненным пользовательским историям, по выполненным задачам. Заказчик желает видеть прогресс не только в виде прироста возможностей его продукта, но и в виде бумажных отчетов. А рисовать диаграммы сгорания задач от руки или в каком-либо графическом редакторе также не очень удобно. Поэтому оптимальным решением будет переложить всю рутинную работу по хранению, отслеживанию пользовательских историй, формированию отчетов на автоматизированную информационную систему.

1.4 Применение системы управления проектами на основе методологии скрам.

Система управления проектами на основе методологии скрам позволит поддержать внедрение самой методологии на каждом уровне работы в организации. Т.к. сама методология концентрируется на взаимодействии между людьми, то и система будет делать упор на взаимодействие, на более тесное общение, не упуская однако и значимости управленческой составляющей.

Современные технологии позволяют создавать возможности для интерактивного общения внутри команды, такие как корпоративный чат или видео конференции. Данная возможность будет весьма полезна, если все члены

					1304.115190.000 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

команды не могут разместиться в одном месте.

Система учета задач позволит формировать стандартизированные отчеты по текущему проекту, оформлять графики и представлять другую необходимую информацию. А подсистема шаблонов позволит гибко настраивать нужные возможности.

Система отслеживания задач позволит определить на какой стадии находится каждая конкретная задача. Также система управления проектами сможет самостоятельно рассчитывать коэффициент полезного действия команды, помогая таким образом объективно оценивать количество задач, которое возможно реализовать за одну итерацию.

Таким образом можно сделать вывод, что необходимо либо найти систему управления программными проектами, удовлетворяющую поставленным условиям, либо разработать ее самостоятельно.

1.5 Обоснование необходимости разработки.

На данный момент существует достаточно много систем управления программными проектами. Данные системы используются как для управления разработкой ПО на заказ, так и для внутренних разработок компаний. Вследствие ограниченности времени автором рассматривались далеко не все программные решения, призванные стандартизировать и автоматизировать разработку программных проектов. Список продуктов-аналогов проекта, разрабатываемого в данном дипломе приведен в таблице 1.

Таблица 1.1

Системы управления проектами

Система управления проектами	Поддерживаемые модели жизненного цикла проекта	Основные возможности
<i>Atlassian Jira</i>	Водопад, <i>SCRUM</i> ,	Отслеживание ошибок, моделирование и управление процессами, планирование задач, совместная работа разработчиков,
<i>MS Project</i>	Водопад, метод кратчайшего пути	Диаграмма Ганта, планирование задач, управление ресурса

Мегаплан	Водопад, водопад с обратными связями	Планирование задач, ведение клиентской базы и базы сотрудников, управленческий и финансовый учет
Basecamp	Водопад с обратной связью.	Планирование задач, разграничение доступа к задачам и проектам,, визуализация прогресса в виде истории
Wrike	Водопад с обратной связью	Планирование, приоритезация задач, совместная работа, визуализация загрузки работников и выполнения задач с помощью диаграммы Ганта
Spider Projects	Водопад с обратной связью	Планирование и приоритезация задач, управление ресурсами, визуализация загрузки на диаграмме Ганта, широкие возможности по работе с ресурсами.
Asana	В о д о п а д , в о з м о ж н а итеративная разработка	Планирование задач, подключение тэгов, назначение задачам сроков.
Redmine	В о д о п а д , в о з м о ж н о итеративная разработка	Планирование задач, создание wiki страниц, формирование отчетов.

Как видно, все приведенные системы реализуют одни и те же функции. Многие имеют свои специфические возможности, однако при ближайшем рассмотрении автор пришел к следующим выводам. Большинство приведенных систем не имеют русскоязычного интерфейса. Исключения - «Wrike», «Redmine», «Spider Project». Подавляющее большинство не поддерживает методологию *SCRUM* (за исключением «Atlassian Jira»), по которой ведется разработка программных продуктов в компании автора. Ни у одной системы нет возможности взаимодействия между членами группы разработки или между разработчиками и заказчиками в реальном времени в виде чата или видеоконференции. Данный аспект немаловажен в том случае, если члены группы разработки находятся далеко друг от друга и не могут ежедневно собираться на *SCRUM*-митинги в одном месте. Конечно, для устранения данного недостатка можно использовать стороннее ПО, однако намного удобнее было бы иметь встроенную в систему возможность связи в реальном времени с функцией создания и изменения расписаний конференций.

Далее, в компании, для которой автор разрабатывает данный проект использую методику *Kanban*, для отслеживания стадий проекта и реализации

принципа «точно в срок». На данный момент указанная методология реализована только в *Atlassian Jira*, в остальных же системах данный функционал реализован либо в виде расчета кратчайшего пути и построения диаграммы Ганта, что не совсем подходит под концепцию *Kanban*, либо не реализован вообще.

Наконец вопрос цены. *Asana*, *Wrike*, *Basecamp* предоставляют возможность бесплатного использования данных систем для ограниченного количества пользователей. Количество пользователей варьируется от пяти до двадцати.. Если системами пользуются большее количество человек, то приходится платить ежемесячную абонентскую плату. Другие системы могут предоставляться в демо-версиях с урезанными возможностями либо на пробный период в течении тридцати дней.

«*Spider Project*» является лишь настольной системой и не имеет функций совместной работы над проектом.. «*Atlassian Jira*» нужно разворачивать на собственном сервере.

Ввиду перечисленных выше недостатков существующих систем автором предлагается разработать собственную систему управления проектами, которая реализовывала бы все необходимые возможности планирования задач, отслеживания стадий выполнения проектов, совместной работы над проектами,, формирования отчетов, декларируемые методологиями *SCRUM* и *Kanban*. Разработка собственной системы также позволит сэкономить средства на оплачивание лицензий сторонних систем, а при условии реализации продаж может приносить прибыль.

Далее рассматривается способ реализации данной системы, описываются ее возможности и выбираются технологии, требуемые для ее реализации.

					1304.115190.000 ПЗ	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

2. Специальный раздел.

В данном разделе приводятся диаграммы, описывающие систему моделей программного продукта, разрабатываемого в данном дипломном проекте. Диаграммы описаны на языке *UML*.

2.1 Диаграммы прецедентов.

Как было описано выше: среди пользователей разрабатываемого программного продукта выделяются две ключевые роли: *product owner* - владелец продукта - который отвечает за постановку задач и их приоритезацию, а также за приемку готового продукта в конце каждого спринта; собственно команда разработки, которая отвечает за разработку программного продукта и тестирование. Следовательно моделирование будет производиться с точек зрения этих двух ролей. Ниже приведены диаграммы прецедентов, и их описания моделирующие взаимодействия между разрабатываемой системой и каждой из указанных ролей.

					1304.115190.000 ПЗ	Лист
						13
Изм.	Лист	№ докум.	Подпись	Дата		

Точка зрения: *Product Owner*.

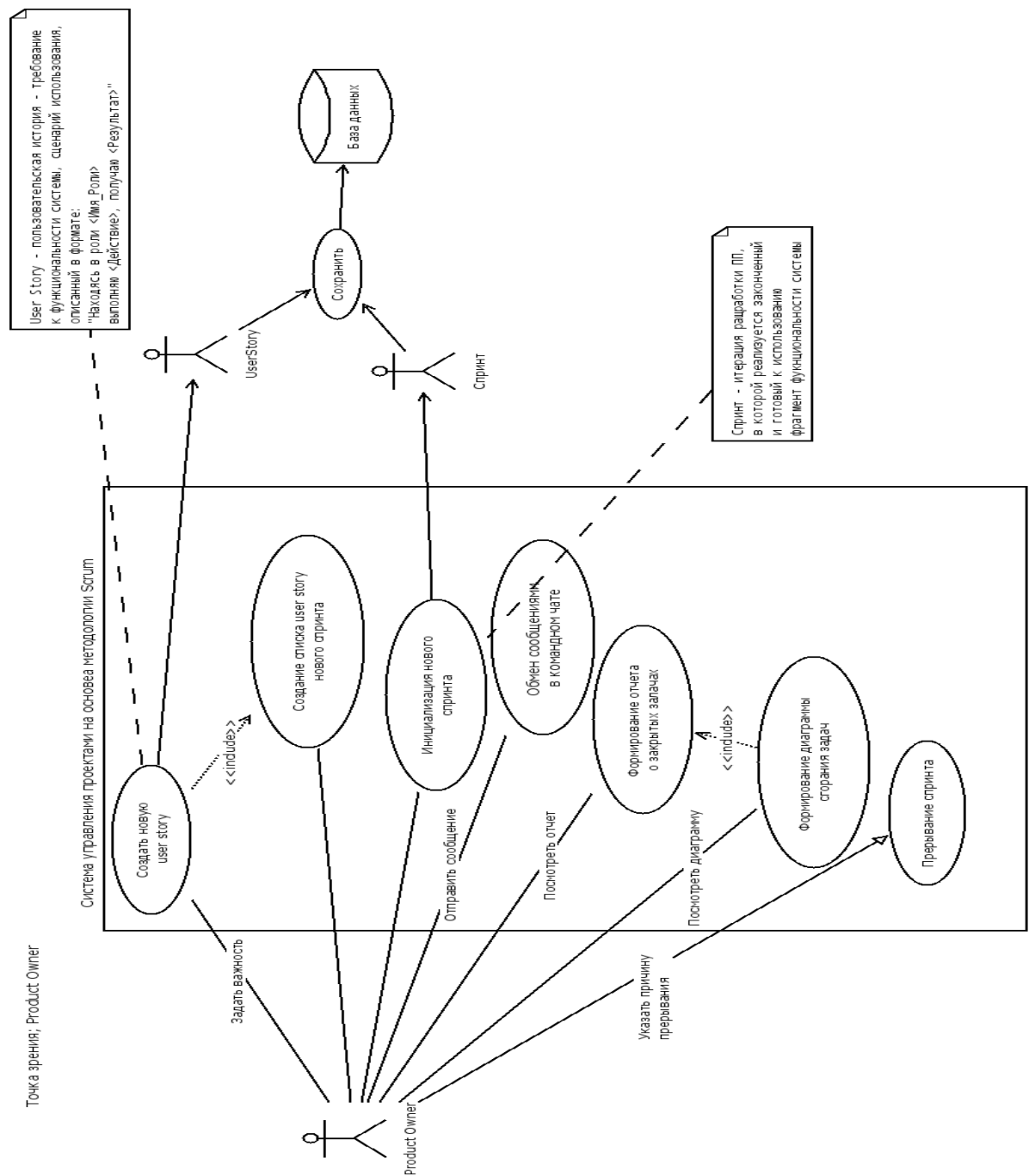


Рисунок 2.1., диаграмма прецедентов, точка зрения *Product Owner*

Описание:

1. Создать новую *user story*.

Действующие лица: *product owner*

Предусловия; пользователь находится на странице создания *user story*;

Основной сценарий;

- а) пользователь нажимает кнопку «Создать новую историю»
- б) Система предоставляет форму ввода данных
- в) Пользователь вводит необходимые данные и отправляет форму
- г) Система создает новую *user story*;

2. Создание списка *user story* для нового спринта.

Действующие лица: *product owner*

Минимальное условие: емкость спринта определена разработчиками.
(Прим. Емкость спринта — количество задач, которое команда разработки способна выполнить за один спринт. Емкость спринта определяется в очках историй. Очки историй — относительная величина применяемая для оценки сложности одной задачи относительно другой. Может изменяться в пределах от нуля до бесконечности.)

Предусловия; пользователь создал новый спринт и перешел на страницу его редактирования;

Основной сценарий;

- а) Система отобразила бэклог (список *user story*) проекта и и текущего спринта;
- б) Пользователь переносит наиболее важные задачи из бэклога проекта в бэклог спринта;
- в) Пользователь вводит необходимые данные и отправляет форму
- г) Система запрещает перенос, когда емкость спринта заполнена;

3. Инициализация нового спринта

Действующие лица: *product owner*;

Минимальное условие: Пользователь имеет права на создание спринта;

Предусловия: Предыдущий спринт окончен;

Основной сценарий;

- а) Пользователь нажимает кнопку «Создать новый спринт»
- б) Система создает черновик спринта и предоставляет форму для ввода цели спринта, даты начала, набора бэклога спринта;
- в) Пользователь вводит необходимые данные и отправляет форму
- г) Система создает новый спринт;

Альтернативные сценарии:

- в.1) Не введена цель спринта
 - а.1.1 Система выдает сообщение о том что необходимо ввести цель;
 - а.1.2 Переход к шагу (б)
- в.2) Не введена дата начала спринта
 - а.2.1 Система устанавливает текущую дату как дату начала спринта;
 - а.2.2 Переход к шагу (г)

4. Обмен сообщениями в командном чате.

Действующие лица: *product owner*;

Предусловия: пользователь находится на странице чата;

Основной сценарий;

- а) Пользователь вводит сообщение и отправляет форму;
- б) Система отправляет сообщение в чат;

Альтернативные сценарии:

- а.1) Пользователь не ввел сообщение
- а.2) Система бездействует;

5. Формирование диаграммы сгорания задач.

Действующие лица: *product owner*;

Предусловия: Пользователь находится на странице спринта;

Основной сценарий;

- а) Пользователь нажимает кнопку «Показать отчет»;
- б) Система выводит диаграмму сгорания задач в составе общего отчета;

6. Формирование отчета по выполненным задачам.

Действующие лица: *product owner*;

Предусловия: Пользователь находится на странице спринта;

Основной сценарий;

- а) Пользователь нажимает кнопку «Показать отчет»;
- б) Система выводит отчет по выполненным задачам;

7. Прерывание спринта..

Действующие лица: *product owner*;

Минимальное условие: у пользователя должны быть права на прерывание спринта;

Предусловия: Пользователь находится на странице спринта;

Основной сценарий:

- а) Пользователь нажимает кнопку «Прервать спринт»;
- б) Система выводит предупреждение;
- в) Пользователь продолжает прерывание;
- г) Система выводит форму ввода причины прерывания
- д) Пользователь вводит причину прерывания и отправляет форму;
- е) Система прерываем спринт и сохраняет отчет о прерывании;

Альтернативные сценарии:

- г.1) Пользователь не вводит причину
 - г.1.1) Система выдает сообщение об ошибке
 - г.1.2) Переход к шагу (г)

Точка зрения: разработчик



Рисунок 2.2, диаграмма прецедентов, точка зрения разработчика

Описание:

Большая часть функций аналогична функциям владельца продукта, поэтому здесь описываются только те, которые не были указаны в предыдущей диаграмме.

1. Написать отчет о выполненной задаче.

Действующие лица: разработчик;

Предусловия: пользователь находится на странице просмотра задачи;

Основной сценарий;

- а) Пользователь нажимает кнопку "Завершить задачу";
- б) Система выводит форму для ввода отчета;
- в) Пользователь вводит отчет и отправляет форму;
- г) Система сохраняет отчет и выводит сообщение об успешном сохранении;

Альтернативные сценарии:

- в.1) Пользователь не ввел отчет
- в.2) Система выдает сообщение о необходимости отчета;
- а.3) Переход к шагу б;

2. Показать КПД команды.

Действующие лица: разработчик;

Предусловия: пользователь вошел в систему;

Основной сценарий;

- а) Система выводит в информационном окне КПД команды в очках историй.

3. Создать Kanban процесс.

Действующие лица: разработчик;

Предусловия: пользователь находится на странице создания Kanban процесса;

Основной сценарий;

- а) Система отображает форму создания Kanban процесса;
- б) Пользователь вводит название и описание процесса;
- в) Пользователь создает стадии процесса;

- г) Пользователь нажимает кнопку "Сохранить";
- д) Система сохраняет процесс и его стадии и выводит сообщение об успешном сохранении;

Альтернативные сценарии:

- б.1) Пользователь не ввел названия процесса
- б.2) Система выдает сообщение о необходимости названия;
- б.3) Переход к шагу а;
- г.1) Пользователь не создал ни одной стадии для данного процесса;
- г.2) Системы выводит сообщение о необходимости создания хотя бы одной стадии;
- г.3) Переход к шагу в;

4. Создание стадии Kanban процесса.

Действующие лица: разработчик;

Предусловия: пользователь находится на странице создания или редактирования Kanban процесса;

Основной сценарий;

- а) Система отображает список стадий процесса на форме редактирования Kanban процесса;
- б) Пользователь нажимает кнопку "Добавить стадию";
- в) Система создает новую стадию и отображает форму ее редактирования;
- г) Пользователь вводит название и описание стадии и нажимает кнопку "Созранить";
- д) Система сохраняет стадию процесса и выводит сообщение об успешном сохранении;

Альтернативные сценарии:

- г.1) Пользователь не ввел названия стадии процесса
- г.2) Система выдает сообщение о необходимости названия;

г.3) Переход к шагу в;

5. Добавление user story к стадии Kanban процесса.

Действующие лица: разработчик;

Предусловия: пользователь находится на странице создания или редактирования user story;

Основной сценарий;

- а) Система отображает список стадий процесса на странице редактирования user story;
- б) Пользователь выбирает нужную стадию и нажимает кнопку "Прикрепить";
- в) Система прикрепляет user story к выбранной стадии и выполняет сохранение;

Альтернативные сценарии:

- б.1) Пользователь не выбрал стадию и нажал кнопку "Прикрепить"
- б.2) Система выдает сообщение о необходимости выбора стадии;
- б.3) Переход к шагу а;

На основе приведенных диаграмм прецедентов можно строить диаграммы взаимодействия по каждой отдельной функции.

Диаграммы взаимодействия.

1. Создание пользовательской истории.

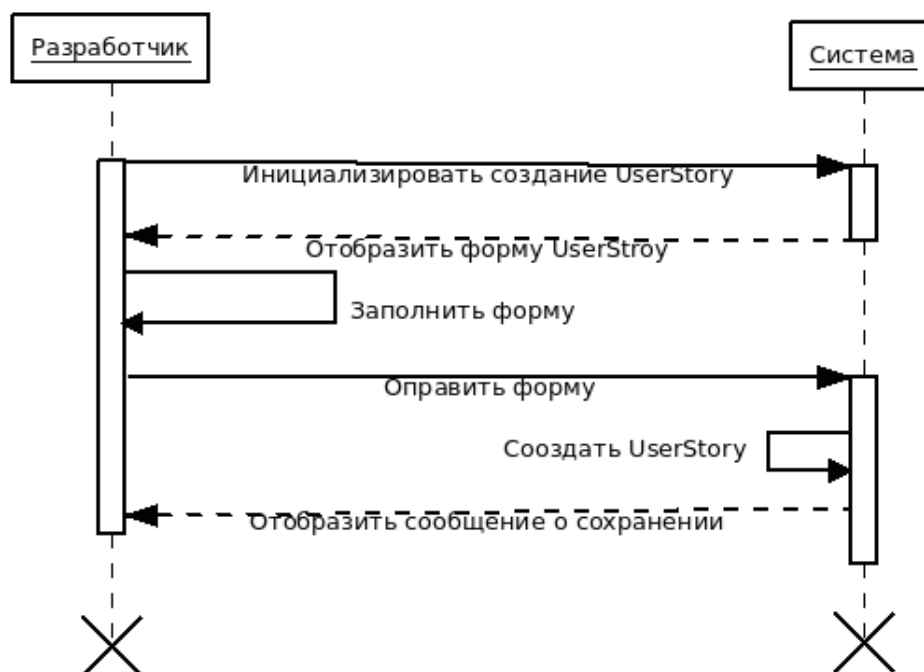


Рисунок 2.3, лиаграмма взаимодействия функции "создание пользовательской истории"

2. Инициализация нового спринта.

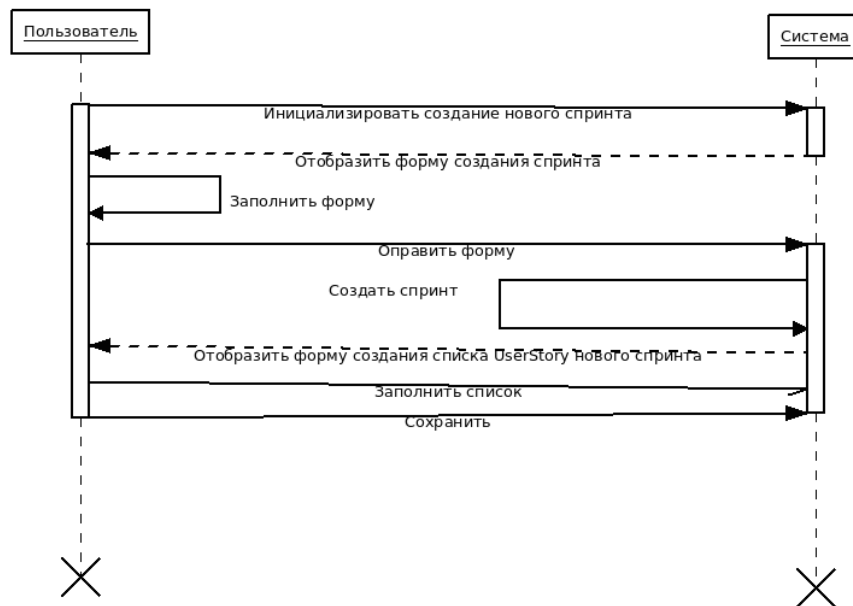


Рисунок 2.4, диаграмма взаимодействия функции "Инициализация спринта"

3. Создание списка пользовательских историй нового спринта.

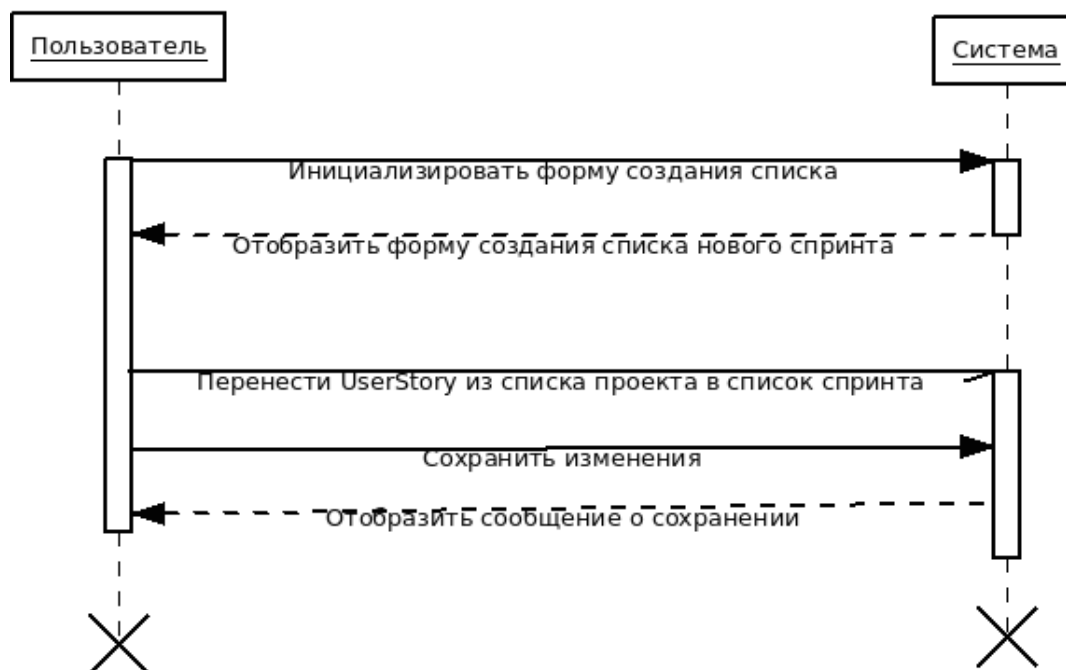


Рисунок 2.5, диаграмма взаимодействия функции "Создание списка пользовательских историй нового спринта"

4. Прерывание спринта.

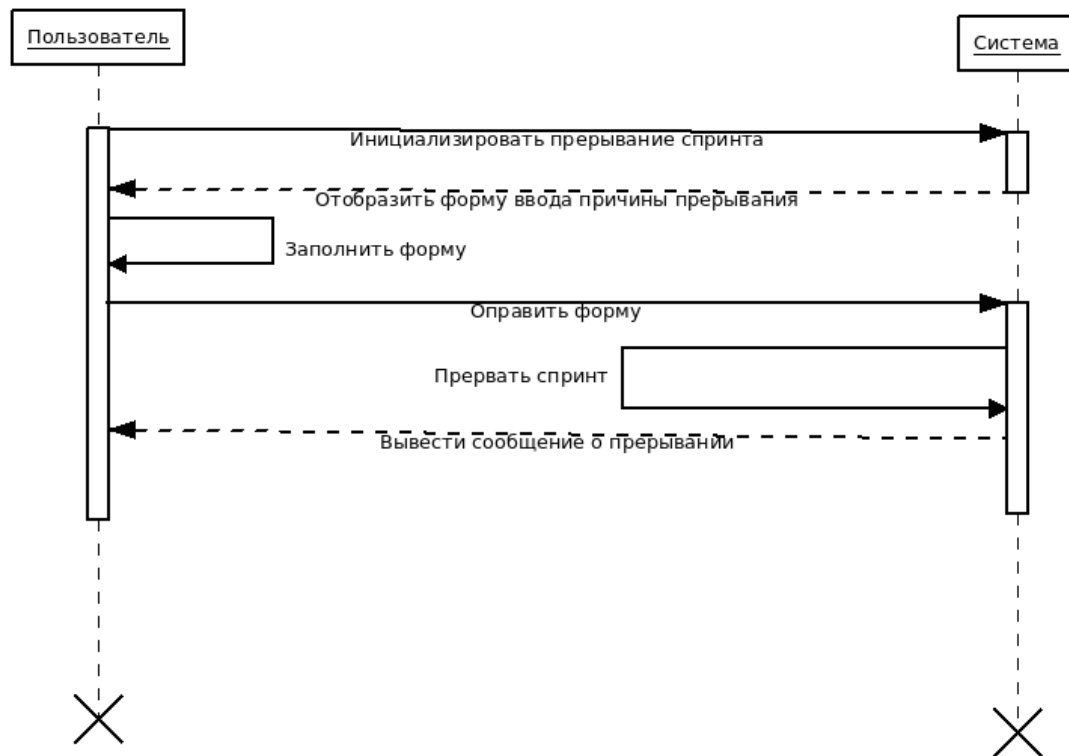


Рисунок 2.6, диаграмма взаимодействия функции "прерывание спринта"

5. Формирование отчета о закрытых задачах.

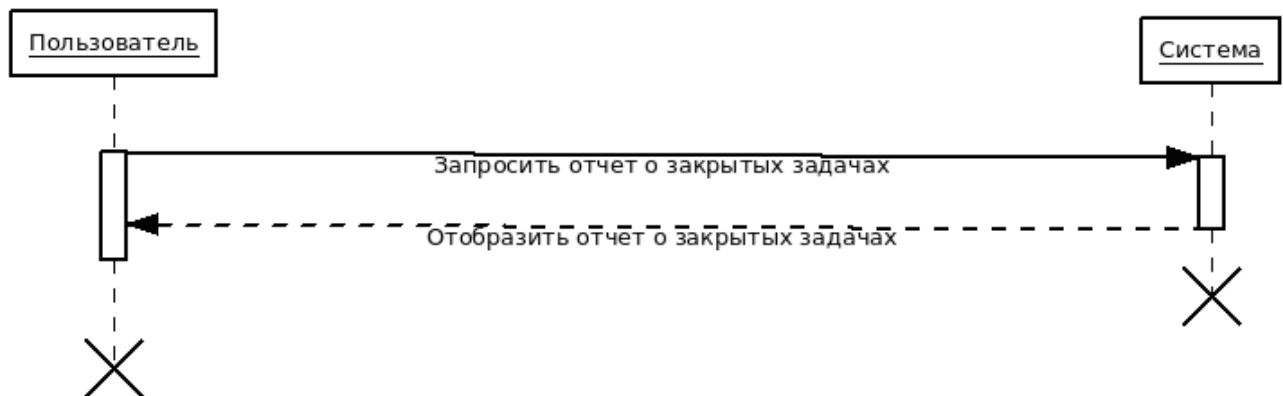


Рисунок 2.7, диаграмма взаимодействия функции "формирование отчета о закрытых задачах"

6. Написание отчета о закрытой задаче

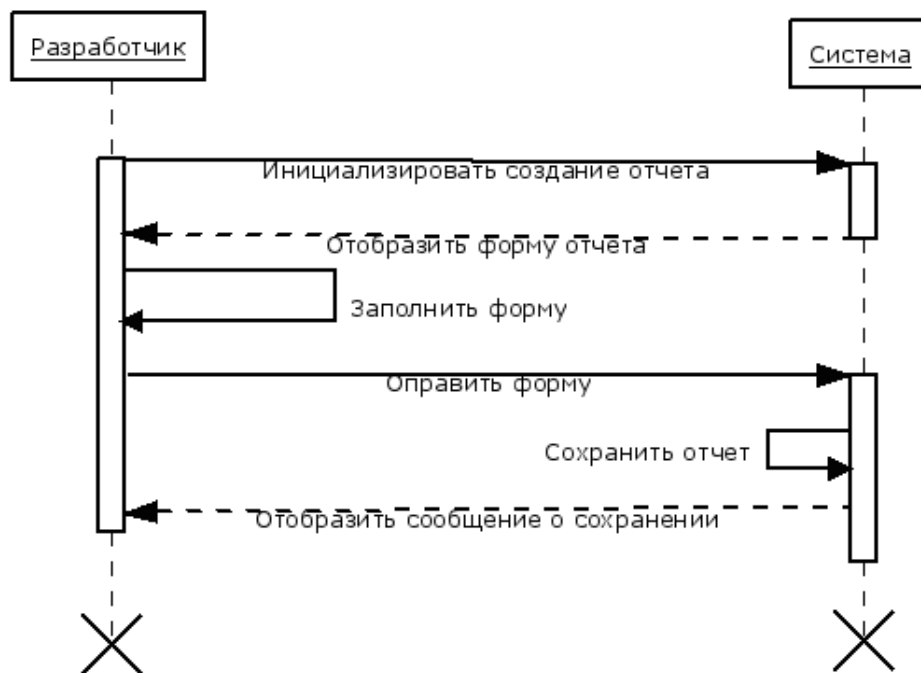


Рисунок 2.8, диаграмма диаграмма взаимодействия функции "Написание отчета о закрытой задаче"

7. Обмен сообщениями в командном чате.

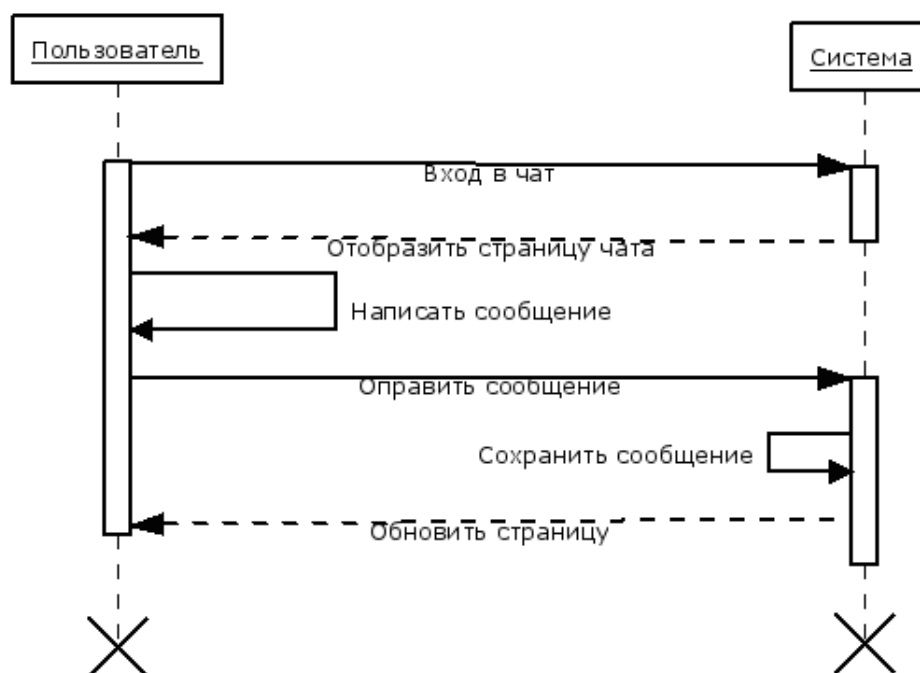


Рисунок 2.9, диаграмма взаимодействий функции "обмен сообщениями в командном чате"

8. Формирование диаграммы сгорания задач.

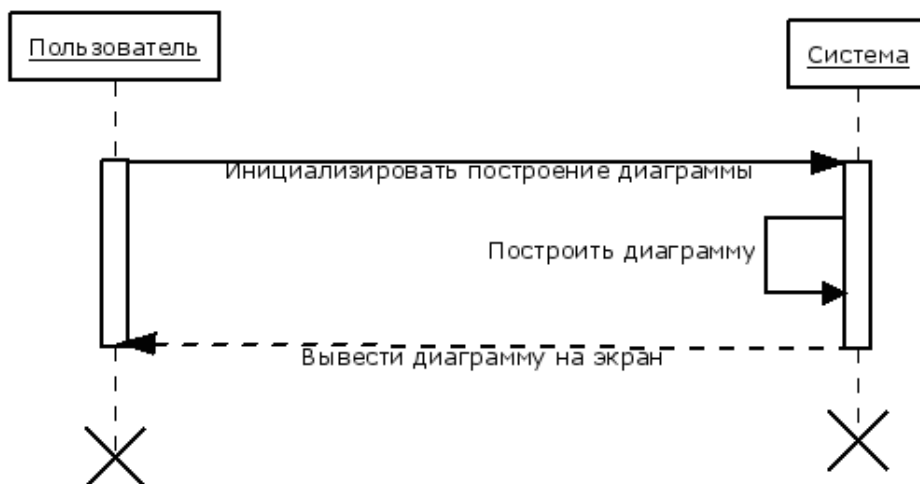


Рисунок 2.10, диаграмма взаимодействий функции "формирование диаграммы сгорания задач"

9. Создание Kanban процесса

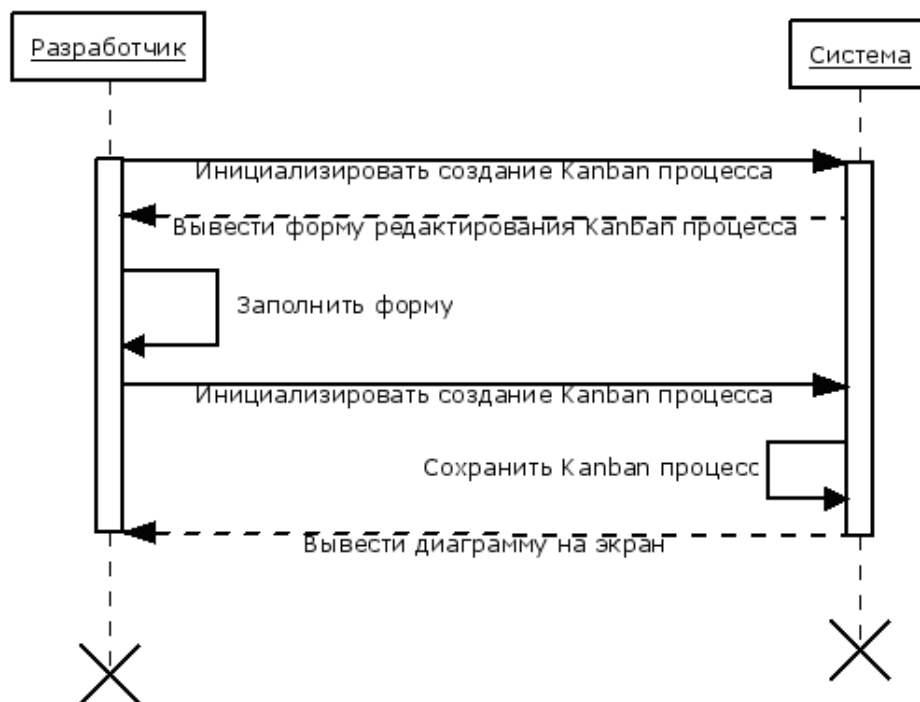


Рисунок 2.11, диаграмма взаимодействия функции "создание kanban процесса"

10. Создание стадии Kanban процесса.

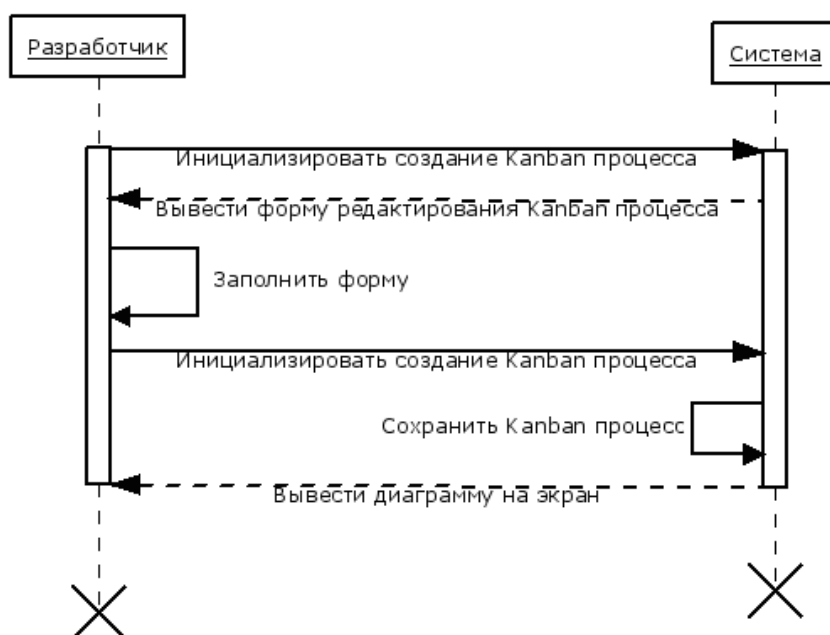


Рисунок 2.12, диаграмма взаимодействия функции "создание стадии Kanban процесса"

11. Прикрепление пользовательской истории к стадии Kanban процесса

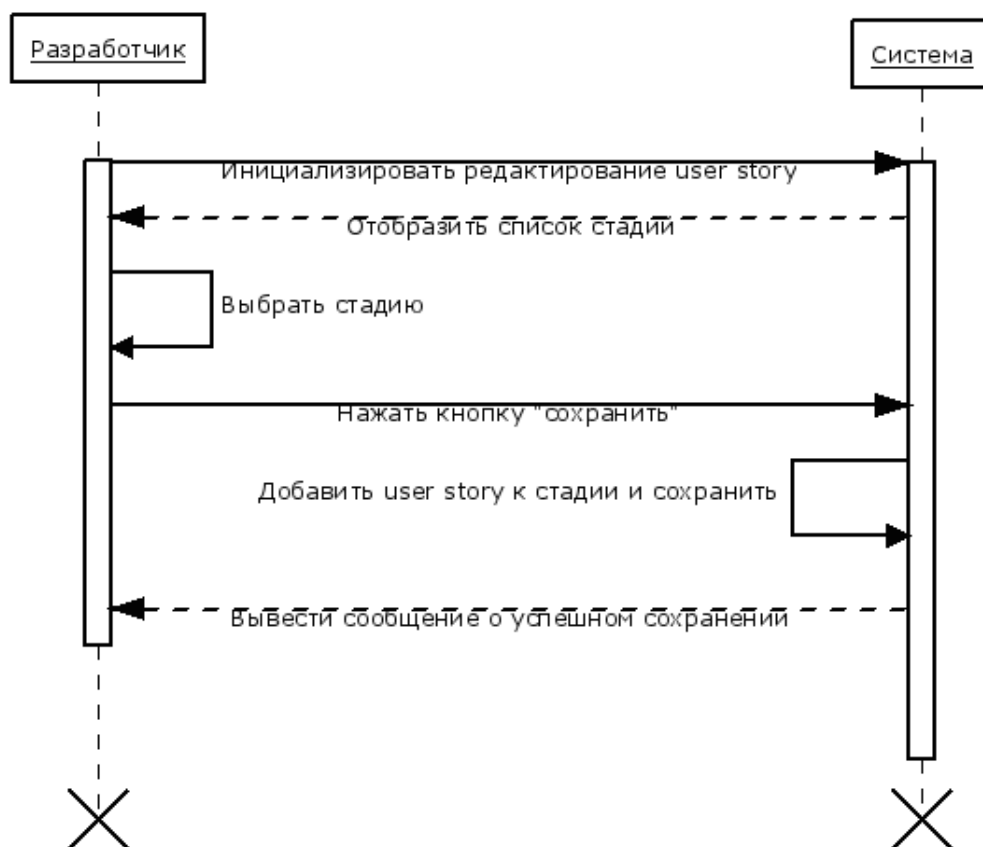


Рисунок 2.13, диаграмма взаимодействия функции "прикрепление пользовательской истории к стадии Kanban процесса"

Приведенные выше диаграммы взаимодействия позволяют выделить сущности, характеризующие предметную область и смоделировать их взаимоотношения на диаграмме классов.

Диаграмма классов.

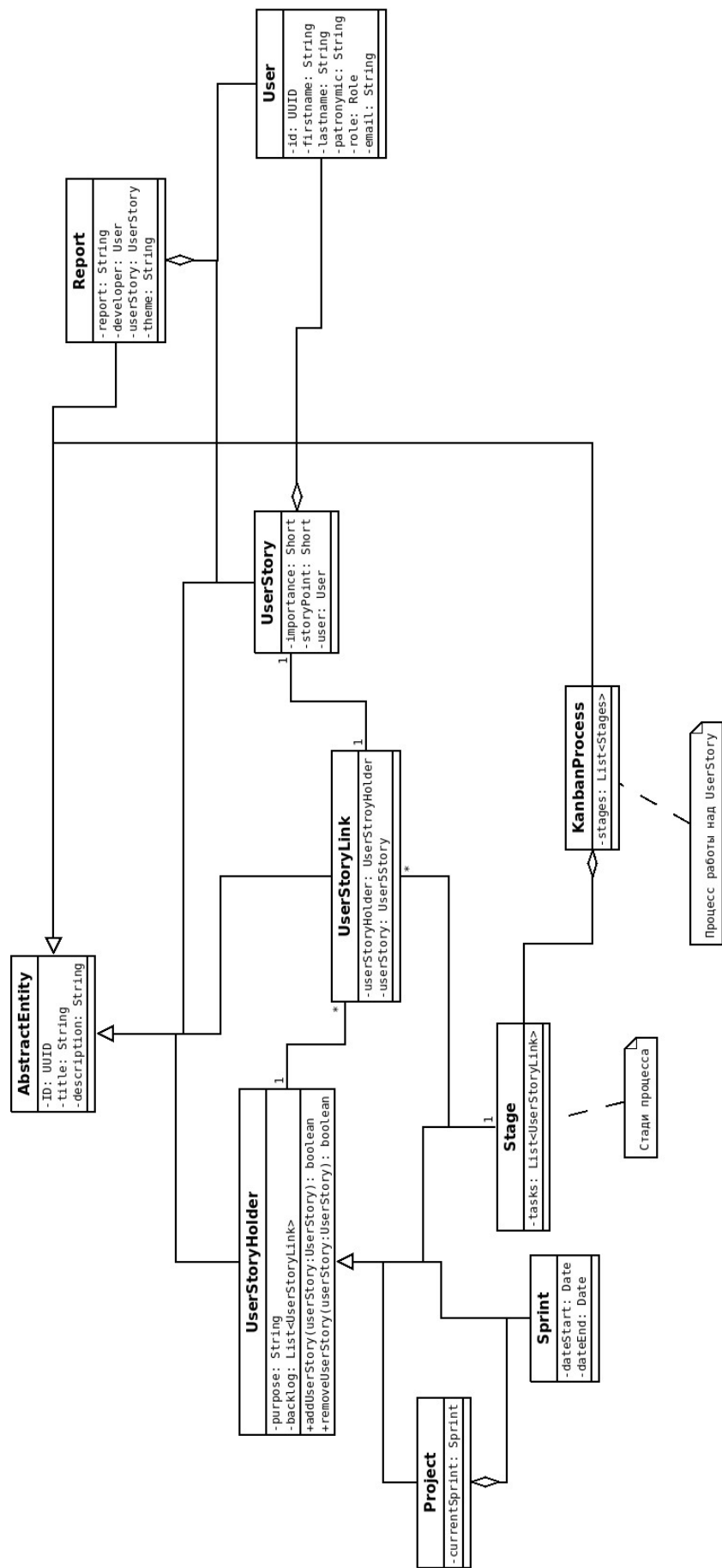


Рисунок 2.14, диаграмма классов

Описание диаграммы классов.

AbstractEntity.

Назначение: представляет собой абстрактную сущность, инкапсулирует общие для всех классов системы поля.

UserStory.

Наследует AbstractEntity.

Назначение: представляет собой сущность пользовательской истории, инкапсулирует ее поля. Является объектом предметной области.

UserStoryHolder.

Наследует AbstractEntity.

Назначение: представляет собой абстрактный класс-контейнер для UserStory. Инкапсулирует общую для всех контейнеров UserStory логику.

UserStoryLink.

Наследует AbstractEntity.

Назначение: связывает объекты классов UserStory и UserStoryHolder.

Report.

Наследует AbstractEntity.

Назначение: представляет собой класс отчета разработчика о выполнении требований UserStory.

User.

Назначение: класс описывает пользователя в контексте системы.

Project.

Наследует UserStoryHolder.

Назначение: класс описывает сущность проекта в контексте системы. Является объектом предметной области.

Sprint.

Наследует UserStoryHolder.

Назначение: класс описывает сущность спринта в контексте системы. Является объектом предметной области.

Stage.

Наследует UserStoryHolder.

Назначение: класс описывает стадию работы над UserStory (разработка/тестировани/приемка и др.) в контексте системы. Является объектом предметной области.

KanbanProcess.

Наследует: AbstractEntity.

Назначение: описывает процесс работы над UserStory, в контексте системы. Является контейнером для объектов класса Stage.

Далее идет технологический раздел, в котором приведено описание и обоснование выбора используемых в проекте технологий.

					1304.115190.000 ПЗ	Лист
						32
Изм.	Лист	№ докум.	Подпись	Дата		

3. Технологический раздел.

В данном разделе рассматривается архитектура разрабатываемого программного продукта, а также обосновывается выбор технологий, используемых для реализации парогаммных компонентов приложения.

3.1 Архитектура.

Невозможно переоценить, насколько важен выбор правильной архитектуры при проектировании программного приложения. Архитектура должна отвечать таким требованиям как простота, гибкость и расширяемость. Иными словами приложения должно быть устроено достаточно просто, при этом необходимо, чтобы оно имело достаточно широкий диапазон настроек, а механизм расширения позволял наращивать функционал не затрагивая уже существующие модули. Системы.

Т.к. подразумевается, что пользователь получает доступ к приложению через сеть Интернет необходимо использовать клиент-серверную архитектуру. Однако также нужно решить, на какие логические части необходимо разбить систему для того чтобы реализовать гибкость настройки и взаимозаменяемость компонентов. Для выполнения поставленной задачи было принято решение использовать шаблон модель-вид-контроллер. Данный шаблон разделяет приложение на три части, одна из которых — модель — инкапсулирует модель предметной области и сервисы для взаимодействия с базой данных. Вторая часть шаблона — вид — отвечает исключительно за представление данных пользователю, а также за отслеживание событий, таких как нажатие на кнопку и информирование о произошедшем событии контроллера. Контроллер в свою очередь инкапсулирует поведение системы, ее бизнес логику. Он является связующим звеном между видом и моделью. Контроллер обращается в модель для того чтобы получить данные, подготавливает их и формирует представление (вид), заполняя его данными, полученными из модели. В обратную сторону,

					1304.115190.000 ПЗ	Лист
						33
Изм.	Лист	№ докум.	Подпись	Дата		

контроллер получает сигнал от представления, о том что произошло некоторое событие, производит необходимые действия, заданные разработчиком и записывает и передает данные в модель для записи их в базу данных. Схема работы приложения представлена на рисунке 1.

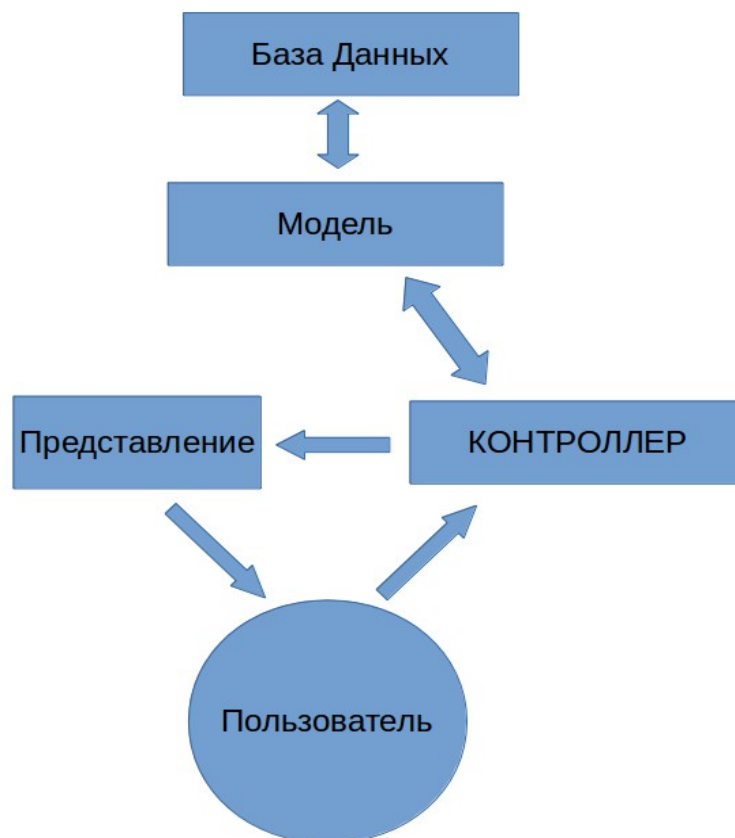


Рисунок 3.1 , архитектура программного продукта.

Если с контроллером и представлением все понятно, то модель стоит рассмотреть поближе. Модель представляет из себя цельный модуль, содержащий в себе несколько слоев абстракции.

Первый уровень — уровень предметной области. На данном уровне хранятся все сущности описываемой предметной области, а также связи между ними. Иными словами это уровень модели предметной области.

Далее идет инфраструктурный уровень — репозиторий, - который отвечает за взаимодействие с базой данных — за сохранение, извлечение, создание и

удаление экземпляров сущностей из базы. Данный уровень собственно и является промежуточным слоем абстракции между базой данных и контроллером. На рисунке 2 показана детализированная архитектура приложения.

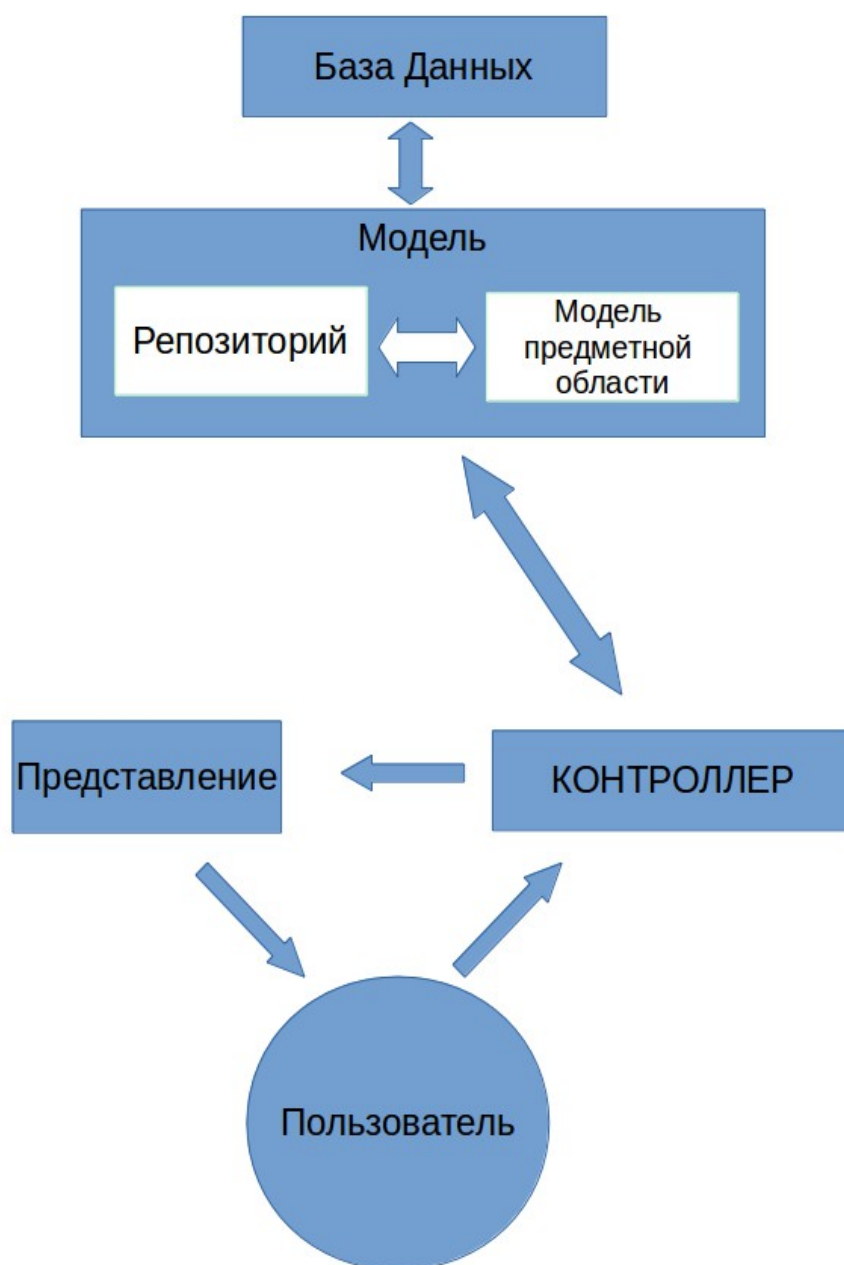


Рисунок 3.2, детализированная архитектура.

3.2 Технологии, используемые в проекте.

Многие языки программирования поддерживают описанную выше архитектуру приложений модель-вид-контроллер. Однако в данном проекте подразумевается разработка веб-приложения, поэтому круг средств разработки несколько снижается.

Рассмотрим три самых популярных на данный момент языка программирования для разработки веб-приложений; *Java*, *C#*, *php*.

Сравнение будет производиться по следующим критериям:

- 1) Скорость работы и производительность;
- 2) Кроссплатформенность;
- 3) Поддерживаемые возможности;

Скорость работы и производительность.

С точки зрения конечного пользователя одно и то же приложение на разных языках программирования на одном и том же оборудовании будет работать с одной и той скоростью. Однако при повышении нагрузки с 10 одновременно работающих пользователей хотя бы до 100 уже можно будет увидеть разницу в производительности. Так при большой нагрузке *php* перестает справляться с требуемыми функциями, замедление работы приложения уже видно невооруженным глазом. К сожалению, за свою простоту и нестрогую типизацию *php* платит низкой производительностью. В этом отношении компилируемые и строго типизированные *Java* и *C#* намного превосходят конкурента. Данные языки ориентированы на разработку корпоративных приложений, то есть приложений, которые априори будут подвергаться высокой нагрузке, поэтому их производительность на порядок превосходит *php*, который создавался для быстрой разработки простых веб сайтов. Результаты относительного сравнения по данной характеристике представлены в таблице 3.1. Сравнение производилось по десятибальной шкале.

					1304.115190.000 ПЗ	Лист
						36
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 3.1

Сравнение языков программирования по производительности

	<i>C#</i>	<i>Java</i>	<i>php</i>
Производительность	10	10	5

Кроссплатформенность.

Технологии разработки веб-приложений развиваются с колоссальной скоростью и сейчас рассматриваемые языки программирования поддерживаются на всех популярных операционных системах. Однако есть некоторые различия. *Php* изначально распространяемый под лицензией *GPL* имеет реализации от производителя как для *Windows*, *Linux*, *Mac OS X*. *Java* в свою очередь также представляет собой открытый стандарт, но распространяется в двух реализациях — свободная от *open source* сообщества и официальная от компании-производителя *Oracle*. Обе реализации распространяются бесплатно.

C # работает на платформе *.NET* — альтернативе платформы *Java*, разработанной компанией *Microsoft*. Официальных реализаций платформы *.NET* для *Linux* *Microsoft* не выпускает, а среды разработки являются платными. Тем не менее существует свободная реализация платформы *.NET* и языка *C#* для *Linux*. Эта платформа называется *Mono*. На данный момент платформа разрабатывается компанией *Xamarin*. Поставляемая среда разработки — *Xamarin Studio* - реализована лишь для ОС *Windows* и *Mac OS X*.

Результаты относительного сравнения по данной характеристике представлены в таблице 3.2. Сравнение производилось по трехбальной шкале.

Таблица 3.2

Сравнение языков программирования по кроссплатформенности

	<i>C#</i>	<i>Java</i>	<i>php</i>
Кроссплатформенность	2	3	3

Поддерживаемые возможности.

C# относительно молодой язык, поэтому для него написано еще не так много библиотек и каркасов приложений, как для *Java* или для *php*. *Java* поддерживает несколько технологий разработки пользовательского интерфейса для веб приложения — стандарты JSP (JavaServer Pages) — компилируемые на стороне сервера страницы, JSF (JavaServer Faces) — технология создания пользовательского интерфейса, реализующая компонентную модель, библиотека Google Web Toolkit и основанный на ней каркас приложений *Vaadin* позволяют создавать пользовательские интерфейсы веб-приложений исключительно на языке *Java* — иными словами все веб приложение пишется на *Java*. C# на данный момент предоставляет только технологию ActiveServer Pages, сходную с *JSP*. *Php* в свою очередь позволяет смешивать *html* разметку с собственно *php*-кодом, получая таким образом весьма запутанные страницы, отвечающие за представление. Такой подход считается весьма ущербным, т. к. предполагает возможность реализации всей бизнес-логики приложения в компонентах, отвечающих за представление, что является неправильным и даже вредным, хотя и позволяет разработать приложение быстро. Также *php* предоставляет шаблонизатор *twig* для создания страниц представления. Данный шаблонизатор сходен по работе с *JSP* и предоставляет более идеологически верный способ представления данных.

Еще один недостаток *php* по сравнению с C# и *Java* — отсутствие возможности хранить состояние. В то время как C# и *Java* способны в течении сессии хранить рабочую информацию в оперативной памяти, *php*, являясь по сути языком сценариев, работает по модели запрос-ответ. То есть информация хранится только в базе данных, а в оперативную память она попадает только на время выполнения сценария. Это является очень существенным недостатком в случае, когда нужно реализовывать многошаговые процедуры в веб-приложении. Результаты относительного сравнения по данной характеристике представлены в таблице 3.3. Сравнение производилось по пятибальной шкале.

					1304.115190.000 ПЗ	Лист
						38
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 3.3

Сравнение языков программирования по поддерживаемым возможностям

	<i>C#</i>	<i>Java</i>	<i>php</i>
Возможности	4	5	3

Из приведенного выше сравнения можно сделать вывод, что язык программирования *Java* является наилучшим выбором для реализации данного программного проекта, т.к. обладает широким набором поддерживаемых возможностей, работает на всех популярных платформах, средства разработки бесплатны, *Java* де факто является стандартом корпоративной разработки приложений.

В качестве сервера приложений выбирается *Apache Tomcat*. Выбор мотивирован тем, что данный сервер является самым легковесным, простым в использовании и надежным. Он бесплатен и прост в настройке. Также хорошо держит высокую нагрузку и обладает встроенным балансировщиком.

В качестве системы управления базой данных выбирается СУБД *MySQL*. Данная СУБД является одной из самых популярных в силу производительности, стабильности работы и простоты использования. Все функции и возможности *MySQL* полностью задокументированы, по данной СУБД выпущено большое количество литературы, в том числе русскоязычной. *MySQL* поддерживает большое количество движков таблиц, в том числе *MyISAM*, *InnoDB* и т.д. Репликация *MySQL* отлично протестирована и используется на таких сайтах как *Yahoo Finance*, *Mobile.de*, *Slashdot*. Создано большое количество программного обеспечения для управления и разработки баз данных с помощью *MySQL*. Это такие решения как *phpMyAdmin* и *MySQL Workbench*.

Заключение.

В данном дипломном проекте были рассмотрены факторы, влияющие на успешность управления разработкой программного обеспечения, были рассмотрены возможности увеличения эффективности управления разработкой программного обеспечения, в связи с чем получили освещение различные методологии управления разработкой программных продуктов. На основе проведенного анализа автор пришел к выводу, что оптимальной методологией является *Scrum*, относящаяся к семейству гибких методологий разработки.

Помимо того, что в процессе анализа проблемы неэффективности управления разработкой программного обеспечения была определена оптимальная методология, также была выявлена необходимость разработки программного продукта, позволяющего производить управление разработкой в рамках указанной методологии.

Далее был разработан проект программного продукта, включающий в себя *UML* схемы и сценарии взаимодействия пользователя с системой. Были выбраны и обоснованы технологии, используемые для реализации программного продукта.

					1304.115190.000 ПЗ	Лист
						40
Изм.	Лист	№ докум.	Подпись	Дата		

Список литературы

1. Б. Райзберг, Л. Лозовский, Е. Стародубцева: Современный экономический словарь;
2. *The Standish Group: «Chaos Report»*, 1995;
3. Заренков В.А.: Управление проектами: Учебное пособие. - 2-е издание - М.: Изд-во АСВ; СПб.: СПбГАСУ, 2006
4. Хенрик Книберг, Маттиас Скаррин: *Scrum* и *Kanban*. Выжимаем максимум.: C4Media Inc. 2010
5. Кент Бек: Экстремальное программирование: разработка через тестирование: БИБЛИОТЕКА ПРОГРАММИСТА: 2003

					1304.115190.000 ПЗ	Лист
						41
Изм.	Лист	№ докум.	Подпись	Дата		

