МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

09.04.03 Прикладная информатика
(код и наименование направления подготовки)

Информационные системы и технологии корпоративного управления
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Разработка методики проектирования программного обеспечения по технологии Agile»

Студент К.В. Бугакова
(И.О. Фамилия) (личная подпись)

Научный руководитель к.т.н., О.В. Аникина
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

Оглавление

Введение

В современном мире всё чаще разработка ПО для телекоммуникационных операторов ведется на основе гибких моделей. Данный выбор обосновывается тем, что для того, чтобы сделать продукт конкурентоспособным, заказчику приходится оперативно изменять требования к продукту, подстраивая его под требования рынка. Изменения в требованиях влекут за собой быстрые изменения в дизайне, коде, перетестирование продукта. Если проект разрабатывается по методологии Waterfall, это может привести к значительному изменению сроков выдачи продукта и увеличению бюджета, недовольству заказчика и потере прибыли из-за того, что похожую идею другая компания смогла воплотить быстрее. В связи с этим, команды и заказчики выбирают для разработки своего продукта гибкую модель, которая позволяет им выдавать работающий функционал раз в 2-3 недели, быстро реагировать на изменения рынка и максимально быстро находить и устранять дефекты.

Актуальность темы исследования обусловлена необходимостью создания методики проектирования программного обеспечения для телекоммуникационного проекта, сочетающей в себе современные гибкие практики и обеспечивающей быстрый и эффективный переход от каскадной модели разработки проекта на гибкую, а также инкрементальную поставку продукта и повышение производительности команды от релиза к релизу.

Объектом исследования является процесс проектирования программного обеспечения.

Предметом исследования является создание методики и модели проектирования программного обеспечения.

Целью данной работы является разработка методики и модели проектирования программного обеспечения для телекоммуникационного оператора.

Цель работы определила необходимость решения следующих **задач:**

- изучить исследования специалистов, связанные с моделями проектирования ПО и переходом с каскадной модели проектирования на гибкую;
- произвести обзор существующих гибких моделей проектирования ПО, определить их преимущества и недостатки в разрезе телекоммуникационных проектов;
- произвести анализ популярных гибких практик, определить, какие из них могут быть эффективно использованы на проектах, целью которых является поставка ПО телекоммуникационному оператору;
- предложить шаги для перехода с каскадной модели проектирования ПО для телекоммуникационного оператора на модель с использованием гибких практик;
- проанализировать традиционную модель проектирования ПО на телекоммуникационном проекте (на примере реального проекта для европейского телекоммуникационного оператора);
- разработать гибкую модель проектирования ПО, которая сможет обеспечить высокую эффективность деятельности проекта, создающего программное обеспечение для телекоммуникационного оператора;
- произвести оценку эффективности разработанной модели и доказать гипотезу.

Гипотеза исследования состоит в предположении, что применение разработанных в рамках данного диссертационного исследования методики и модели проектирования ПО для телекоммуникационного оператора обеспечит повышение объема выполненных проектных работ без изменения ресурсов, увеличит скорость поставки продукта, сделает процесс разработки более прозрачным и позволит интегрировать заказчика в процесс разработки.

Основные **методы исследования**: теоретический анализ, классификация, систематизация, объектно-ориентированный подход к моделированию систем управления, статистический анализ.

Новизна исследования заключается в разработке нового алгоритма перехода с каскадной модели проектирования ПО на гибкую с учетом современных особенностей и требований к проектированию ПО для телекоммуникационных операторов, а также создании гибкой модели на основе предложенного алгоритма.

Практическая значимость данного исследования состоит в возможности практического применения предлагаемых методики и модели проектирования ПО для телекоммуникационного оператора, которые способствуют повышению объема работ без изменения ресурсов, увеличению скорости поставки продукта, позволяют сделать процесс разработки более прозрачным и интегрировать заказчика в процесс разработки.

Методологическую базу исследования составляют работы, опубликованные зарубежными и отечественными авторами, такими как Дж. Сазерленд, Р. Коул, Л. Криспин, Дж. Грэгори, Л. Адкинс, Ю. Аппело, Дж. Грин, Э. Стеллман, А. Ганноченко и др.

Основные этапы исследования: исследование проводилось с 2018 по 2020 годы в несколько этапов:

На первом этапе формулировалась тема исследования и выполнялся сбор и анализ информации по теме исследования, также проводилась постановка цели, задач, предмета, объекта и гипотезы исследования.

На втором этапе осуществлялся анализ существующих гибких моделей проектирования ПО определено, какие практики могут быть эффективно применены на проектах, занимающихся созданием ПО для телекоммуникационного оператора, были предложены шаги для перехода с каскадной на гибкую модель проектирования ПО, а также проводилось написание и публикация научных статей по теме исследования.

На третьем этапе была предложена гибкая модель проектирования ПО для телекоммуникационного оператора, проведен анализ эффективности модели и сформулированы выводы о полученных результатах исследования.

На защиту выносятся:

- Шаги для перехода с каскадной модели проектирования ПО для телекоммуникационного оператора на модель с использованием гибких практик.
- Требования к гибкой модели проектирования ПО для телекоммуникационного оператора.
- Предложенная в ходе исследования гибкая модель проектирования ПО для телекоммуникационного оператора.
- Анализ эффективности разработанной модели.

По теме исследования опубликована 1 статья, результаты исследования были представлены на V международной научно-практической конференции (школы-семинара) молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук» [3].

Диссертация состоит из введения, трех глав, заключения и списка используемой литературы.

В первой главе рассматриваются современное состояние проблемы, перехода с каскадной на гибкую модель проектирования ПО.

Во второй главе проанализированы существующие гибкие модели проектирования ПО. В разрезе телекоммуникационных проектов рассмотрены основные практики гибких методологий, позволяющие сделать процесс разработки ПО более быстрым, качественным и прозрачным.

В третьей главе произведена разработка модели проектирования ПО для телекоммуникационного оператора, позволяющая сделать процесс разработки более прозрачным, повысить скорость команды и объемы работ, а также включить заказчика в работу.

В заключении приводятся результаты исследования.

Работа изложена на 98 страницах, включает 23 рисунка, 11 таблиц, 33 источника используемой литературы.

Глава 1 Анализ проблемы перехода с каскадной модели проектирования программного обеспечения на гибкую

1.1 Анализ существующих моделей жизненного цикла программного обеспечения

Жизненный цикл программного обеспечения подразумевает собой непрерывный процесс, который начинается с момента появления идеи о создании программного обеспечения и заканчивается его полным изъятием из эксплуатации. Модель жизненного цикла ПО, в свою очередь описывает этапы работы и последовательность действий и задач на протяжении всего жизненного цикла ПО.

Основными моделями жизненного цикла ПО являются: каскадная модель и итерационная модель, разработанные У. Ройсом и Т. Гилбом соответственно и дополненные рядом других учёных и практиков. На основе моделей жизненного цикла ПО был разработан ряд методологий создания ПО, которые получили широкое распространение по всему миру [17].

В 1970 году У. Ройс описал концепцию каскадной модели жизненного цикла ПО и раскритиковал её, как модель, требующую значительных доработок. Однако, данная модель в то время, а также и по сей день является весьма популярной среди компаний-разработчиков программного обеспечения. На английском языке название данной модели звучит как Waterfall model (водопадная модель), что обуславливается тем, что каждый последующий этап работ как бы вытекает из предыдущего. Каскадная модель подразумевает, что каждый последующий этап стартует только после полного завершения предыдущего этапа. То есть невозможно приступить к написанию кода до того, как дизайн окончательно не сформирован, также как нельзя внедрить продукт до окончательного завершения его тестирования.

Методология, построенная на базе данной модели, носит соответствующее название – методология Waterfall [26]. Поэтапную схему каскадной модели можно увидеть на рисунке 1.1.

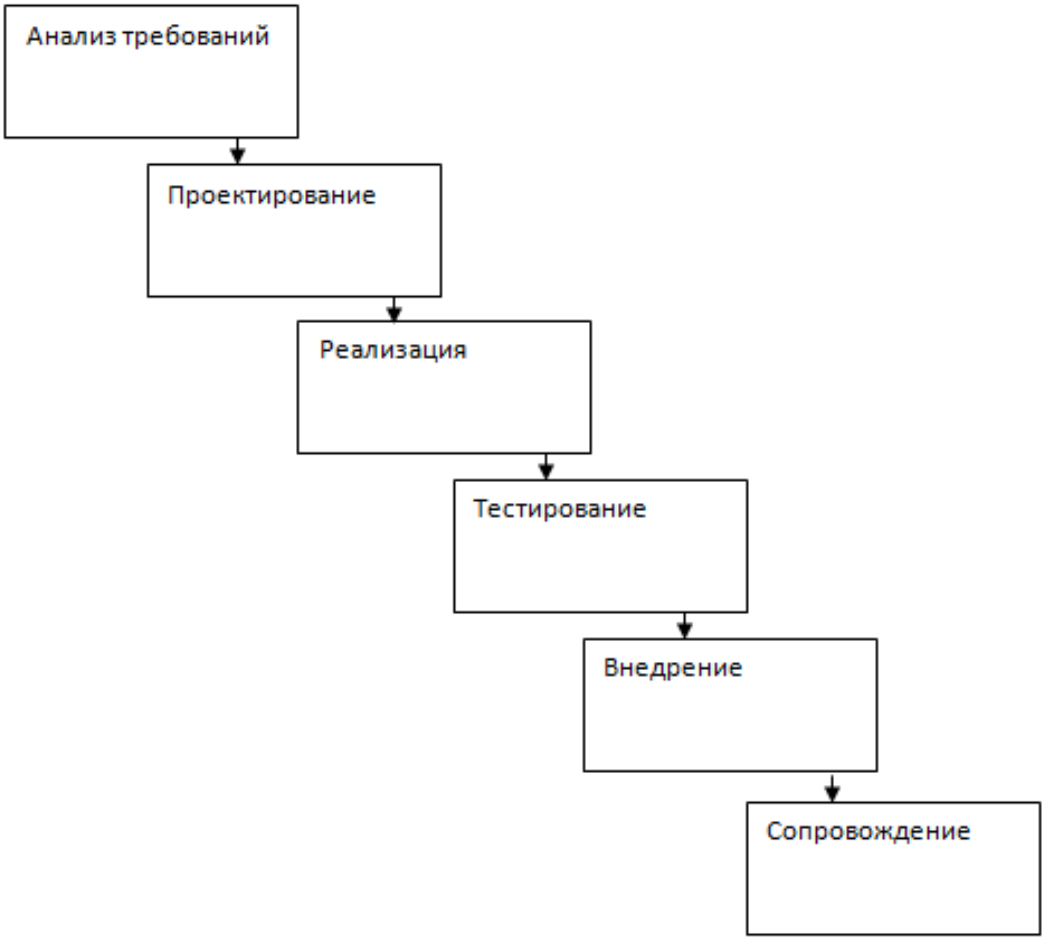

Рисунок 1.1 - Каскадная модель

У. Ройс определил, что каскадная модель в большинстве случаев используется тогда, когда заказчик чётко представляет себе, что за продукт ему необходим, и предельно ясно и прозрачно формирует набор требований, передаваемых исполнителю. Главным образом, модель применяется в проектах, где задействовано большое количество людей и, зачастую, команд, члены которых могут находится в разных локациях. Данная модель достаточно понятна и легка в исполнении, а также позволяет планировать ресурсы и сроки, но самым большим её минусом является то, что к завершению этапа написания дизайна и началу этапа старта разработки нет

возможности изменить начальные требования, так как это приведёт к сдвигу всех возможных сроков и увеличению временных и денежных ресурсов.

В своём традиционном представлении, предложенном У. Ройсом, каскадная методология предполагает разработку продукта на шести этапах. На этапе *анализа требований* происходит сбор и обработка начальных требований заказчика, планируются дальнейшие этапы работ, определяется количество необходимых ресурсов, сроки работ, стоимость проекта, а также анализируются все возможные риски. Ключевыми фигурами на данном этапе являются бизнес аналитики, однако, также к работе могут быть привлечены QA Leads и Dev Leads, иными словами, группа опытных разработчиков и тестировщиков, способных определить все возможные риски, дать свою оценку касаемо времени и ресурсов, которые будут потрачены на реализацию требований. После того, как список требований к программному продукту сформирован, начинается стадия *проектирования*. На данном этапе, как правило, требования заказчика уже невозможно изменить. Основываясь на списке требований, бизнес аналитики создают документацию, которая подробно описывает, каким образом требования должны быть реализованы программистами. Итак, к завершению данного этапа и началу этапа *разработки* команда разработчиков получает необходимую для написания программы документацию. Все документы являются конечными и не могут быть изменены (подразумеваются глобальные изменения в дизайне). Когда программа написана, код отправляется на проверку команде тестировщиков. Однако, в том случае, если над проектом работают несколько команд разработчиков, которые создают различные компоненты программы, перед тем, как продукт будет предоставлен на тестирование специалистам отдела контроля качества, производится интеграция отдельных компонентов программы. На стадии *тестирования* находятся и устраняются все недочёты, которые были пропущены на предыдущих этапах, это могут быть как ошибки в реализации, так и ошибки в документации (подразумеваются ошибки, не требующие

значительных изменений в дизайне или коде). Зачастую тестирование разбивается на несколько этапов в соответствии с типом тестирования. Обычно, основными видами тестирования в рамка методологии waterfall являются: функциональное тестирование, регрессионное тестирование, нефункциональное тестирование (все виды тестирования производительности, тестирование установки и др). Помимо внутреннего тестирования может также проводиться тестирование на стороне заказчика. Как правило, это происходит в том случае, если реализованный проект должен интегрироваться с программами и системами, созданными другими разработчиками. В таком случае заказчик может пригласить отдельную группу тестировщиков из другой компании для проверки End-to-end работы функционала. Следующий этап – этап *внедрения* программного обеспечения. Как правило, на данном этапе представители компании-разработчика разворачивают продукт на серверах заказчика, устраняют дефекты, возникшие при установке релиза (новой версии продукта, которая устанавливается на окружение заказчика) и обучают представителей заказчика работе с установленным ПО. На этапе *сопровождения* или, как его ещё называют, *поддержки* команда разработчиков и тестировщиков занимается устранением production defects (дефектов, которые найдены конечными пользователями ПО). В том случае, если заказчик хочет добавить к существующему ПО новый функционал, формируются новые требования к change requests (запросу на изменение). Далее эти изменения либо реализуются в рамках тех же этапов, которые описаны выше, либо компания реализует все изменения, основываясь на гибких методологиях [5, 13, 26].

Каскадная методология обладает большим количеством преимуществ:

- все сроки и дэдлайны заранее определены: это означает, что заказчик получит продукт точно в срок;
- методология достаточно понятна и проста в использовании: это означает, что методология может использоваться командами, которые ранее не работали с данной методологией;

- требования определены на начальном этапе работ и не подлежат изменениям: это означает, что с самого старта проекта работа над продуктом детализирована и прозрачна;
- заранее ясен бюджет проекта, и выход за рамки бюджета маловероятен: этот факт является огромным плюсом, как для заказчиков, так и для команды разработчиков, так как позволяет первым заранее спрогнозировать, какая сумма денег будет потрачена на проект и готовы ли они потратить такую сумму, и позволяет вторым определить, согласны ли потратить время и рабочие ресурсы, получив взамен данную сумму;
- с методологией Waterfall проект лёгок в управлении и контроле: это значит, что менеджеры имеют абсолютно точное представление о том, что происходит на каждом этапе работ, успевает ли команда со сроками, много ли найдено дефектов, как быстро они устраняются, а это, в свою очередь, позволяет им быстро и точно отчитываться перед высшим менеджментом своей компании, а также перед представителями заказчика.
- методология может использоваться территориально распределёнными компаниями: это означает, что у исполнителя есть возможность привлекать к работе фрилансеров, а также работников в разных локациях.

Однако, У. Ройс и Ф. Брукс определили следующие минусы:

- работа на каждом этапе должна быть выполнена очень тщательно и с минимум ошибок, так как возвращение к любому из завершенных этапов не предусмотрено данной методологией: это означает, что в том случае, если будет допущена крупная ошибка, это может повлечь за собой эскалации со стороны заказчика, бесплатное или крайне затратное переделывание продукта, расторжение договора с заказчиком;

- требования должны быть сформулированы предельно чётко и ясно: это означает, что к работе следует привлекать опытных бизнес аналитиков и дизайнеров, чтобы абсолютно исключить вероятность неправильно сформулированного дизайна;
- изменившиеся потребности пользователей, которые могут вызвать изменения в продукте в ходе его разработки, не могут быть учтены и приняты к исполнению: это означает, что существует риск предоставления клиенту неконкурентоспособного продукта;
- заказчик не вовлечён в работу на этапах разработки проекта, а значит, что он увидит продукт только на этапе релиза: в том случае, если требования были поняты неправильно, это приведёт к тому, что заказчик будет совершенно недоволен заказом [2, 12, 21, 26].

Таким образом, можно сделать вывод, что каскадную методологию лучше всего использовать для реализации самой первой версии продукта при наличии чётко сформулированных требований от заказчика и конкретных бюджетных и временных рамок. Также данная методология подходит для проектов с жёстким контролем со стороны менеджмента. Дальнейшие изменения в программе, а также поддержка продукта могут осуществляться с применением гибких методологий.

Другая модель жизненного цикла ПО, рассматриваемая в данной работе, – итеративная, она по своей сути является альтернативой каскадной модели. Т. Гибл называл её «Эволюционной моделью», так как итеративная модель предполагает собой разбиение всего процесса создания ПО на некое количество стадий, которые в свою очередь состоят из всех этапов, предполагаемых каскадной моделью. Цель каждой стадии (итерации) – предоставить готовый к эксплуатации программный продукт, которым может пользоваться заказчик. На следующей стадии исполнитель выдает более новую версию продукта в соответствии с новыми требованиями заказчика. Таким образом, программный продукт развивается, а, значит, эволюционирует с каждой новой итерацией. Данный подход позволяет не

только постоянно изменять разрабатываемый продукт в соответствии с требованиями заказчика, но и также даёт разработчикам возможность улучшить код программы или исправить незамеченные ранее ошибки. На основе данной модели и были разработаны гибкие или Agile методологии создания программного обеспечения [25].

Исследованию гибких методологий ПО посвящены труды многих зарубежных и отечественных авторов, таких, как Дж. Сазерленд, Р. Коул, Л. Криспин, Дж. Грэгори, К. Швабер, Л. Адкинс, Ю. Аппело, Дж. Грин, Э. Стеллман, А. Ганноченко, и др.

Agile, или гибкая методология, является по своей сути философией, которая была придумана группой нескольких представителей IT-сферы, собравшихся на горнолыжном курорте Snowbird в 2001 году. Результатом их идеи стал Agile Manifesto – сбор принципов, который стал эволюционным прорывом в сфере разработки ПО. Данный свод принципов представлен на рисунке 1.2.

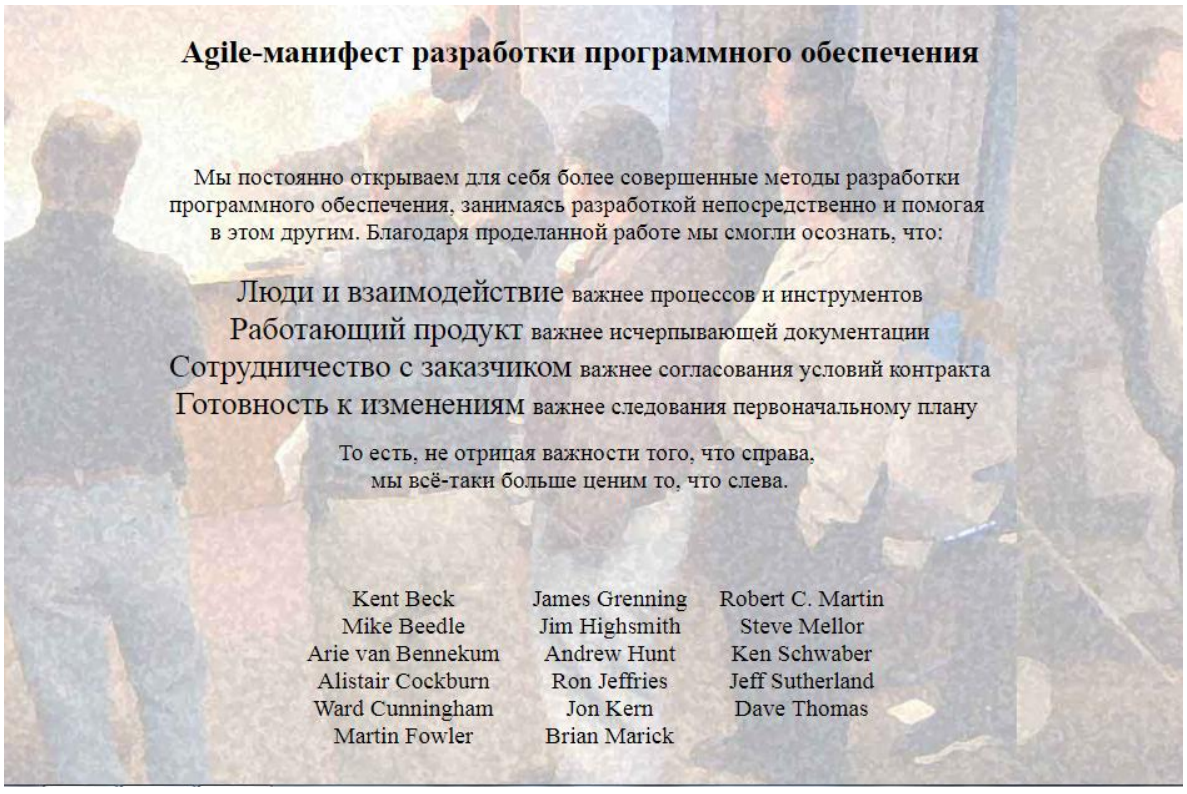

Рисунок 1.2 - Agile Manifesto [19]

Итак, в соответствии с данным манифестом, Agile можно описать, как совокупность подходов к разработке ПО, основанных на принципе итераций и ориентированных на активное и тесное сотрудничество с заказчиком, самоорганизацию и динамическое формирование требований. По факту, как считают авторы Agile и Agile принципов, Agile не является методикой управления работы над проектом, это скорее сбор неких правил и идей (или mindset), которые воплощаются в разнообразных Agile Frameworks (Kanban, Scrum, XP, Lean и т.д.).

Как пишут Р. Коул, Дж. Грин и Э. Стеллман, согласно гибким методологиям, работа над проектом разбивается на итерации, в конце каждой из которых заказчику поставляется готовый продукт. Как правило, сроки каждой итерации составляют от одной до четырёх недель, и количество таких итераций может быть достаточно большим. Самая первая итерация предполагает сбор начальных требований от заказчика и предоставление начальной версии продукта с минимально необходимым функционалом. На каждой следующей итерации команда разработчиков готовит и выдаёт в релиз более новую версию продукта с более развитым существующим или совершенно новым функционалом. Основные этапы каждой итерации похожи на этапы, описываемые каскадной методологией: этап сбора и анализа требований, этап написания документации, этап разработки, этап тестирования и этап внедрения и эксплуатации. Однако, есть очевидная разница в подходах двух этих методологий. Тогда как каскадная методология предполагает собой последовательный старт каждого нового этапа только после полного завершения предыдущего, гибкая методология подразумевает возвращение к предыдущим этапам в том случае, если требования заказчика изменяются. Зачастую работы бизнес аналитиков, разработчиков и тестировщиков могут вестись параллельно. То есть разработка продукта может начинаться тогда, когда есть требования, но дизайн находится на стадии написания, тестирование может начаться тогда, когда разработчик предоставил минимально готовый для тестирования продукт – допустим,

только sunny day (позитивный) сценарий работы функционала без error handling процессов и работающих валидаций [12, 25]. Схема этапов разработки по гибкой методологии представлена на рисунке 1.3.

Рисунок 1.3 - Итеративная модель [17]

Помимо манифеста команда энтузиастов разработала и описала непоколебимые принципы Agile:

- работать на благо заказчика, регулярно поставляя ему качественное ПО;
- быть готовым изменять продукт даже на поздних этапах разработки;
- как можно чаще выпускать готовый работающий продукт (не превышая срок в пару месяцев);
- на ежедневной основе работать с представителями заказчика, чтобы обеспечить полное взаимопонимание и построить и сохранить доверие между членами команды и клиентом;
- команда должна состоять из мотивированных профессионалов, чётко осознающих, для чего они работают и что они получат за свои старания;

- команда должна не бояться общаться друг с другом и с заказчиком. Таким образом, непосредственное общение преобладает над переписками и передачей\получением информации через третье лицо;
- прогресс должен измеряться только работающим продуктом (т.е. заказчику должно поставляться только правильно функционирующее ПО);
- команда, заказчики и пользователи должны иметь возможность работать в заданном отлаженном ритме на постоянной основе;
- команде необходимо уделять внимание качеству проектирования и техническим деталям;
- необходимо сделать процесс создания ПО максимально простым и понятным, также, как и само программное обеспечение не должно быть перегружено излишними деталями;
- команда должна быть самоорганизованной;
- команда должна анализировать риски и ошибки, улучшая эффективность работы с каждым спринтом [16, 20].

Дж. Грин и Э. Стеллман в своей работе «Постигая Agile», выделяют следующие основные преимущества методологии:

- возможность удовлетворять потребностям заказчика на любом этапе работ: что обеспечивает регулярную поставку качественного программного продукта и позволяет заказчику оставаться конкурентоспособным;
- быстрые и частые релизы: что также способствует конкурентоспособности продукта и позволяет быстро устранять ошибки, пропущенные в предыдущем релизе;
- снижение рисков: работая непосредственно с заказчиком и абсолютно прозрачно, команда способна выявить и исправить проблемы на ранних этапах разработки;

- более сплочённая команда в сравнении с командой, работающей по принципам Waterfall: это обуславливается тем, что гибкая методология подразумевает тесное сотрудничество между заказчиками и исполнителями, а также и тесное взаимодействие между самими членами команды-исполнителя. Чаще всего Agile команда состоит из сравнительно небольшого количества человек, которые территориально находятся в одном месте и на ежедневной основе участвуют в большом количестве совместных совещаний и митингов;
- нет тотальной зависимости от менеджмента и большого количества отчетов: подразумевается, что Agile команда должна быть самостоятельной. Зачастую единственным менеджером в команде-исполнителе является непосредственно проектный менеджер, а такие роли, как QA\DEV Manager, QA\DEV lead и др, могут исчезнуть с проекта;
- цикличность и частота повторяемых этапов позволяет команде учиться на своих ошибках и отлаживать процесс разработки и поставки ПО [12].

Однако, Agile методологии присущи следующие минусы и препятствия, с которыми столкнулся Дж. Сазерленд в работе со многими американским компаниями, желающими перейти на гибкую методологию разработки ПО:

- команда должна состоять из самоорганизованных профессионалов, способных самостоятельно выбирать себе задачи (из предлагаемого списка) и самостоятельно справляться с ними: не каждый человек способен без контроля свыше принимать важные решения и самоорганизовываться;
- не ясен срок окончания разработки, также, как и неясен бюджет проекта: это обуславливается тем, что к старту проекта заказчик не предоставляет конкретных, четких, неизменяемых требований, а

наоборот, в процессе разработки он может предлагать всё новые и новые изменения;

- практики Agile не всегда легки в исполнении для команд, которые ранее не работали по гибким методологиям. Отладка процессов может занять достаточно длительное время;
- нехватка подробной документации: постоянные изменения требований могут привести к тому, что документация может не поспевать за ходом разработки, что может привести к тому, что документация будет отличаться от реальной имплементации. Также это может является препятствием к правильному пониманию работы функциональности человеком, который недавно стал членом команды;
- agile предполагает, что члены одной команды в идеале должны находиться в одной и той же локации, максимум в двух: не всегда такой подход является удобным, так как у команды нет возможности подключить редкого специалиста, работающего по принципу фриланс или подключить необходимого проекту человека, работающего, к примеру, в другом филиале компании [12,16, 23, 25].

Таким образом, гибкие методологии подходят для ведения крупных долгосрочных проектов, когда заказчикам необходимо иметь каждый новый релиз как можно чаще, а также, если они хотят иметь возможность следить за ходом работ и видеть «сырой» продукт, изменяя требования в том случае, если продукт не отвечает их ожиданиям или если условия среды или рынка изменились. Agile подходит проектам с неопределённым концом, например, проектам-стартапам, блогам, инновационным проектам, проектам, которым постоянно необходимо внедрение изменений, например, ПО для операторов телекоммуникационных услуг и др. Разработка подобных проектов всегда сопровождается некой долей неопределённости, требования заказчика, детали раскрываются уже в ходе разработки. То есть крайне тяжело

организовать реализацию такого проекта в соответствии с каскадной методологией, так как работу над таким проектом очень сложно спланировать, также как практически невозможно определиться со сроками сдачи такого проекта. Гибкая методология подходит для сравнительно небольших команд-профессионалов, готовых к тесному сотрудничеству с заказчиком, изменению требований на любом этапе работ и способных приступать к своей работе без полностью готовой документации (разработчики, тестировщики) и без полностью написанного и протестированного разработчиками кода (тестировщики).

1.2 Программный продукт для телекоммуникационной компании и специфика разработки программного обеспечения на телекоммуникационных проектах

Телекоммуникационная компания – это компания, строящая свои телекоммуникационные системы и оказывающая услуги связи с использованием передовых информационных технологий частным пользователям и крупным международным корпорациям. Как правило, такая компания предлагает своим клиентам полный перечень профильных услуг, таких как стационарная телефонная связь, высокоскоростной Интернет, структурирование кабельных систем и компьютерная помощь. В настоящее время телекоммуникационные операторы переключаются на использование облачных технологий.

Деятельность телекоммуникационной компании покрывает две области: OSS\BSS – Operation Support System\Business Support System. Так, система поддержки операций\система поддержки бизнеса – являются теми самыми программными продуктами, которые используются телекоммуникационными компаниями для организации, управления и наблюдения за процессами и операциями, позволяющими предоставлять пользователям телекоммуникационные услуги (интернет, связь и др.). OSS

отвечает за управление сетевой инфраструктурой и ресурсами, то есть сетью, коммутаторами, подсетями и др. BSS – это, другими словами, биллинговая система, которая отвечает за взаимодействие с абонентами, то есть контроль всех финансовых операций с клиентом. Также к подобным программным продуктам можно отнести систему ERP – enterprise resource planning – она используется для управление ресурсами предприятия, управления проектами, организационной структурой, ведения бухгалтерского учета и т.д. Система CRM – Customer Relationship Management - отвечает за отношения с клиентами и управления маркетинговыми процессами. Таким образом, для обеспечения работы вышеупомянутых бизнес-процессов необходим отдельный программный продукт или целая сеть интегрированных программных продуктов.

Как правило, телекоммуникационные компании используют решения разных поставщиков, иногда вкупе с самостоятельно разработанными продуктами, что означает интеграцию каждой платформы с одной или несколькими другими платформами. Это происходит в связи с тем, что рынок постоянно расширяется, предлагая все новые услуги и операции, и, чтобы оставаться востребованным у конечного пользователя, телекоммуникационной компании иногда проще купить разработку у нового вендора, чем воссоздавать похожие функции усилиями уже сотрудничающих с ней разработчиков ПО. Это, в свою очередь, означает, что поставщики программного обеспечения для того, чтобы не терять своих клиентов в лице телеком операторов и, наоборот, приобрести еще больше заказов, должны быть готовы к постоянным совершенствованиям, быстрым адаптациям под нужды рынка и заказчика. В связи с этим, проектам, работающим над предоставлением ПО для телекоммуникационных компаний идеально подходят гибкие методологии разработки ПО.

Так как телекоммуникационные проекты чаще всего бывают очень сложными, ресурсозатратными и долгосрочными, на начальных этапах их разработка чаще всего реализуется с применением каскадных методологий,

однако, когда количество работ уменьшается или проект переходит на стадию поддержки, заказчики и команда чаще всего стремятся перейти на разработку проекта с использованием гибких практик, что может быть крайне сложной задачей, так как в таком случае команде необходимо полностью поменять мировоззрение и определиться, какие из большого количества гибких практик подходят именно им.

К специфике телекоммуникационных проектов можно отнести:

- частая изменчивость требований к программному продукту, обусловленная следованию современным тенденциям рынка;
- вовлечённость заказчика в рабочие процессы и его желание быть частью команды;
- как можно более частая поставка рабочего функционала;
- сложность функционала, обусловленная тем, что, телекоммуникационные операторы имеют большое количество пользователей, а, следовательно, предоставляемых сервисов, сложную архитектуру сети и д.р.
- необходимость в обучении заказчика и конечных пользователей использованию программного продукта, а также в демонстрации всех нововведений, что обуславливается сложностью функционала;
- большой процент разного рода рисков;
- частые изменения приоритетов задач;
- частое появление внеплановых задач, обусловленное большим количеством пользователей программного продукта и, как следствие, большим количеством дефектов с окружения заказчика.

Таким образом, основной спецификой проектов, занимающихся поставкой ПО для телекоммуникационных операторов, является интеграция заказчика в рабочий процесс команды и изменчивость требований к продукту, что еще раз доказывает необходимость применения гибких практик к проектированию ПО для телекоммуникационных операторов [17].

1.3 Анализ литературы по проблеме перехода от каскадной модели проектирования программного обеспечения на гибкую

В последние годы компании по всему миру (и не только в сфере IT) не только с самого старта реализуют разработку проектами с помощью гибких методологий, но и организовывают переход на Agile для старых проектов, которые не оправдывают ожиданий по срокам выдачи, бюджету, качеству функционала и др. Как правило, такой переход проходит очень сложно и болезненно, как для команд, так и для менеджмента и не всегда бывает успешным.

Исследованию проблем, связанных с переходом с каскадной на гибкую модель разработки проекта посвящены труды многих зарубежных и отечественных авторов, таких, как Дж. Сазерленд, Р. Коул, Дж. Грин, Э. Стеллман, А. Ганноченко и др.

Роб Коул в своей работе «Блистательный Agile. Гибкое управление проектами с помощью Agile, Scrum и Kanban» выделяет следующие причины, препятствующие быстрому переходу на гибкую модель:

- команда привыкла работать в соответствии с каскадной методологией, и её членам сложно перестроить своё мышление;
- непонимание принципов Agile;
- слишком большое разнообразие методик и инструментария, слишком разрозненная информация;
- некоторые члены команды не хотят переходить на новую методологию, не стараются, пускают всё на самотёк;
- отсутствие должной практики у скрам-мастера и владельца продукта;
- игнорирование обратной связи от заказчика;
- неправильное планирование работ;

- нежелание прежнего руководителя проекта ослабить контроль и дать командам больше свободы;
- проект не подходит для разработки по принципам Agile;
- нет единого пошагового руководства, которое бы объясняло, как быстро правильно перейти на Agile без временных и денежных затрат и с получением быстрого видимого результата [25].

Последний пункт особенно важен и правдив, так как каждый проект индивидуален и требует своего подхода. Какие-то команды и проекты работают лучше по одному Agile Framework, другим подходит другой. Каждая команда самостоятельно должна сформировать свой собственный график и подход.

Как пишет Джефф Сазерленд (автор книги «Scrum. Революционный метод управления проектами», один из авторов scrum подхода, один из тех, кто первым внедрил гибкий подход в реальное производство и один из авторов Agile манифеста), а также Дж. Грин и Э. Стеллман (авторы книги «Постигая Agile. Ценности, принципы, методологии») многие команды или менеджмент считают, что Agile позволит моментально спасти старый неудачный вялотекущий и затратный проект. Они уточняют, что это возможно, но только в том случае, если команда готова к радикальным переменам, основательно подготовлена и заранее определила намеченный путь и подходящий инструментарий. В своей работе «Scrum. Революционный метод управления проектами» Дж. Сазерленд описывает многочисленные примеры из своей собственной жизни, где он помогает отчаявшимся командам\компаниям осознать, что каскадная методология, зачастую, губит проекты, так как современный мир развивается с огромной скоростью, а значит и программный продукт должен реагировать на все изменения мгновенно, чтобы оставаться конкурентоспособным. Однако, даже осознав, что необходимо изменить подход к разработке проекта, команды оказываются не в состоянии внедрить Agile или слишком быстро опускают руки, так и не разобравшись в том, как настроить процесс на новый лад. Как

пишет А. Ганноченко в своей исследовательской работе «Результатники и процессники. Результаты, создаваемые сотрудниками», чтобы команда работала на 100% и быстро и качественно изготавливала программный продукт, необходимо проводить обучение для сотрудников, а также правильно их мотивировать [6, 10, 16].

Роб Коул в своей работе «Блистательный Agile. Гибкое управление проектами с помощью Agile, Scrum и Kanban» определил КФУ – ключевые факторы успеха для разработки проекта по принципам Agile:

- необходимо выбрать подходящий проект. Иногда некоторыми проектами все-таки проще управлять с использованием каскадной модели;
- начинать нужно с малого. Если проекту уже несколько лет, то не стоит сразу пытаться изменить всё. Нужно начать с определённого куска функционала, разработать работающий сценарий и затем потихоньку переводить весь проект на гибкую разработку;
- необходимо выбирать подходящую команду. В каких-то случаях лучше расстаться с некоторыми членами старой команды, если они не готовы мыслить гибко и работать по новой системе;
- ожидания должны быть реалистичными. Не стоит рассчитывать на то, что результат будет виден моментально. Для быстрого эффекта нужно много работать и полностью поменять свое мышление;
- нужно обучиться. Чтобы начать применять Agile можно просто ознакомиться с необходимой литературой. Однако, для массового обучения больше подходят тренинги или Agile-коучинг [25].

Таким образом, можно сделать вывод, что исследования, посвященные переходу с каскадных на гибкие модели управления проектами, занимающимися разработкой программного обеспечения, ведутся и по сей день, чем и обуславливается актуальность данной работы.

Выводы к первой главе

Основными моделями проектирования программного обеспечения являются модели, основанные на каскадных и гибких методологиях разработки. Каскадная модель подходит для крупных проектов с жесткими требованиями, временными рамками и бюджетом. Гибкие модели подходят проектам, требующим быстрой реакции на изменения рынка и частой поставки продукта. Таким образом, практики гибких методологий вписываются в разработку продукта для телекоммуникационного оператора.

Исследования, посвященные переходу с каскадных на гибкие модели проектирования ПО, требуют дальнейшего анализа и продолжения.

Как показал анализ, необходимо разработать шаги для перехода с каскадной модели проектирования ПО для телекоммуникационного оператора на модель с использованием гибких практик, а также модель, позволяющую сочетать лучшие практики гибких методологий для быстрой и качественной поставки продукта телекоммуникационному оператору.

Глава 2 Анализ гибких моделей проектирования программного обеспечения и их основных инструментов

2.1 Анализ современных гибких моделей в разрезе телекоммуникационных проектов

Среди существующих гибких методологий преобладают следующие: Scrum, Kanban, Lean, Scrumban, XP и др. Зачастую методологии не используются в чистом виде, а комбинируются друг с другом. На рисунке 2.1 представлены результаты исследования о популярности гибких методологий, проводимого компанией Agile Survey.

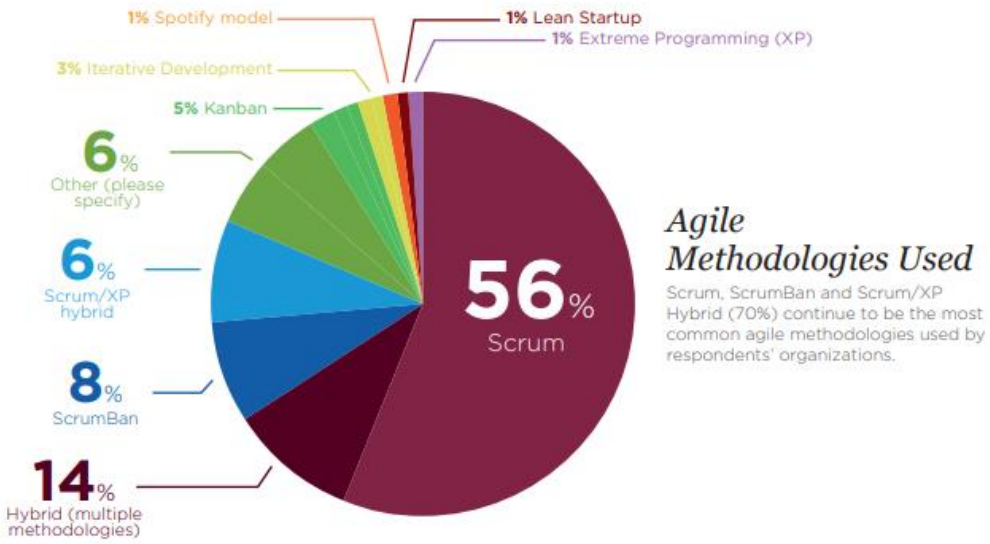

Рисунок 2.1 - Популярные гибкие методологии [29]

Самый популярный метод, базирующийся на идеях Agile – Scrum, основанный в 1986 году Дж. Сазерлендом. На данный момент он считается самым детальным и структурированным подходом и является сочетанием классической каскадной методологии и лучших практик гибких подходов. Scrum подразумевает разделение процесса практически на те же этапы, что используются проектами, которые разрабатываются на основе каскадной

методологии, однако данные этапы подразумевают инкрементальность, то есть повторение каждого этапа через определенный промежуток времени - спринт. Результатом спринта является, как правило, новый релиз продукта или часть продукта, выдаваемая на E2E тестирование. Данная черта Scrum является особенно притягательной для команды, создающей ПО для телекоммуникационных операторов, так как позволяет команде как можно более часто и быстро выдавать заказчику работающий функционал и решения для критичных дефектов, влекущих за собой потерю или перебои с мобильной связью, интернетом, личным кабинетом и т.д. что, в свою очередь, ведет к потере клиентов. Также именно для методологии Scrum характерно такое явление, как составление бэклога (списка задач на реализацию), его приоритезация и планирование. Кроме того, именно приверженцы Scrum используют в своей работе встречи, направленные на улучшение коммуникации между членами команды, слежение за статусами и улучшение процессов [16]. Основные роли в Scrum – владелец продукта, скрам мастер, команда разработчиков, заказчик. На рисунке 2.2 представлена модель разработки по Scrum.

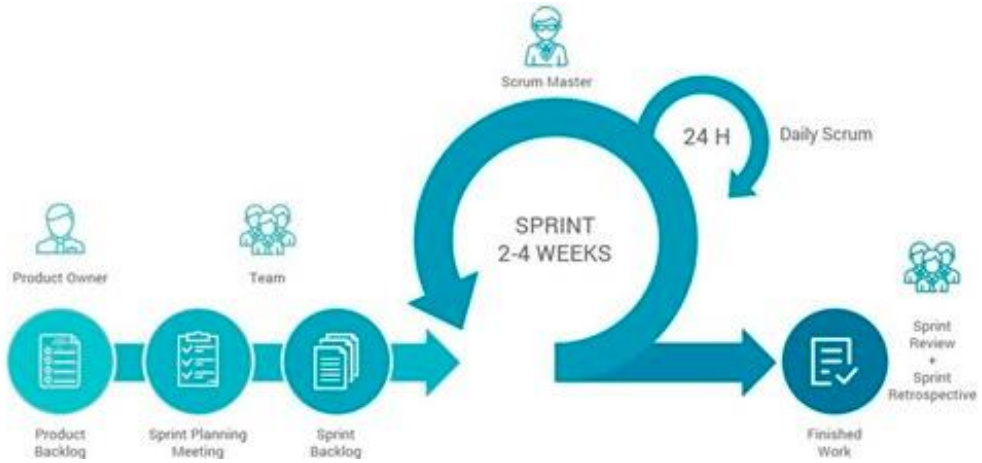

Рисунок 2.2 - Модель разработки по Scrum [28]

Следующей популярной гибкой моделью является Kanban. Принципом Kanban считается непрерывная разработка, а основным артефактом – доска задач со статусами, обеспечивающая полную прозрачность работ. Для

работы с Kanban необходимо определить этапы потока действий (отображаются как столбцы на доске статусов) и задачи – карточки в каждом из столбцов. Карточка перемещается из столбца в столбец, и, на выходе, получается готовый функционал или часть функционала.

Основные черты:

- карточки задач – на каждой карточке написана вся необходимая информация о задаче;
- ограничения на количество задач на каждом этапе – необходимо, чтобы следить за ходом работ и видеть и устранять bottlenecks;
- улучшение процесса – постоянный анализ и поиск способов улучшения производительности.

Главное отличие методологии Kanban от методологии Scrum заключается в том, что scrum команда должна состоять из широкопрофильных специалистов, тогда как Kanban предполагает балансировку узкопрофильных специалистов внутри команды. Кроме того, Kanban проект более прост в управлении и не подразумевает наличие каких-либо ролей, таким образом, за бэклог и планирование отвечает сама команда. Несомненным плюсом данной методологии для телекоммуникационного проекта является то, что заказчик непрерывно и постоянно получает работающий функционал и может давать команде новые задачи без ожидания окончания спринта или сеанса планирования [30].

Также, одной из популярных методик является гибрид Scrum и XP. Суть XP методологии – применение полезных традиционных практик и методов, но поднятие их до экстремального уровня. Таким образом, суть гибридной методологии Scrum + XP является использование полезных практик двух методологий.

Основные принципы:

- быстрая обратная связь;
- непрерывность процесса;
- понятность процесса, доступная всем;

- социальная защищенность команды.

Основные практики:

- парное программирование – это процесс code review, поднятый на новый уровень. ПП подразумевает, что один разработчик пишет код, другой тут же его читает, ищет ошибки, способы улучшения;
- test driven development – сначала пишется тест сценарий, по нему пишется код, который позволит пройти этот тест, затем делается рефакторинг кода на соответствие стандартам;
- непрерывная интеграция – это постоянное слияние рабочих копий кода, написанного разными разработчиками команды в единую ветвь разработки и выполнение частых автоматизированных сборок с целью выявления потенциальных дефектов.

Методология Scrumban подразумевает сочетание практик Scrum и Kanban. Как правило, от Scrum методологии берется традиционная итеративность, приоритезированный бэклог и роли, тогда как от методологии Kanban берется основной инструмент планирования и отслеживания статусов – доска задач. Суть данной методологии заключается в том, что при наличии приоритезированного бэклога отсутствует как таковое планирование и оценка задач. Все, что важно для команды, - это приоритет задач и сроки их выполнения. Так, члены команды выбирают из бэклога самые приоритетные задачи и работают над ними, непрерывно наблюдая за статусами задач на доске и включаясь в помощь другим членам команды в том случае, если какая-то задача не двигается в другой столбец доски слишком долго. Также в рамках такой разработки могут использоваться такие практики scrum, как ежедневные встречи и демо митинги [17].

В таблице 2.1 представлены преимущества и недостатки рассмотренных выше гибких моделей в рамках их применения для проектирования программного обеспечения на телекоммуникационных проектах, ранее работавших на основе каскадных методологий.

Таблица 2.1 – Преимущества и недостатки гибких моделей проектирования ПО для телекоммуникационного оператора

	Scrum	Kanban	Scrum + XP	Scrumban
ПЛЮСЫ	ежедневный контроль над ходом работ; понятные роли; прозрачность процессов; возможна интеграция заказчика в рабочий процесс; частые демонстрации продукта и обеспечение обратной связи; быстрая разработка и поставка решения. возможность вносить коррективы в требования на этапах разработки;	точный расчет нагрузки; концентрация на улучшении; правильная расстановка ограничений; достаточно просто в исполнении; команда может состоять из узкопрофильных специалистов; подходит для проектов, в которых задания формируются в процессе разработки.	все плюсы Scrum методологии; повышенное качество продукта; быстрое выявление дефектов; повышение командного духа.	ежедневный контроль над ходом работ; понятные роли; прозрачность процессов; возможна интеграция заказчика в рабочий процесс; частые демонстрации продукта и обеспечение обратной связи; поток задач непрерывен, не привязан к итерации; быстрая разработка и поставка решения. возможность вносить коррективы в требования на этапах разработки;
МИНУСЫ	scrum подразумевает командную работу на всех этапах разработки (планирование, анализ), что практически невозможно на телеком проекте, т.к. каждый член команды не обладает знанием	команда должна быть самоорганизованной; не подходит для проектов с жесткими дедлайнами; отсутствие конкретных практик; сложен контроль	минусы Scrum методологии; увеличение ресурсных затрат, а, следовательно, и денежных затрат.	команда должна быть самоорганизованной; подразумевается командная работа на всех этапах разработки (планирование, анализ), что практически

	Scrum	Kanban	Scrum + XP	Scrumban
	всего функционала проекта; команда должна быть кроссфункциональной (обладать >1 компетенцией); команда должна уметь самоорганизовываться;	времени окончания разработки и выдачи продукта; отсутствует элемент получения обратной связи и демо, что важно для телекоммуникационных проектов.		невозможно на телеком проекте, т.к. каждый член команды не обладает знанием всего функционала проекта; команда должна быть кроссфункциональной (обладать >1 компетенцией);

Таким образом, наилучшим решением для проектирования программного обеспечения представляется использование традиционной scrum модели с применением работающих практик других моделей. Однако, все традиционные scrum практики подходят именно для телекоммуникационного проекта в связи со спецификой его разработки. Основным негативным фактором является то, что scrum подразумевает командную работу на всех этапах разработки проекта, включая анализ и оценку. Однако, представляется сложным сделать оценку требованию, которое относится к функционалу, с которым сотрудник никогда не работал, тогда как телекоммуникационный проект предполагает, что определенный сотрудник обладает определенными компетенциями, например, занимается исключительно интеграциями, или исключительно сервис активацией, или менеджментом сетей и т.д. Это означает, что традиционная скрам модель требует доработки с учётом специфики телекоммуникационных проектов.

2.2 Основные артефакты, инструментарий и практики гибких моделей

Существует достаточно большое разнообразие гибких моделей разработки ПО, каждая из которых пропагандирует использование своих и

общих инструментов, практик и артефактов. Команда, выбравшая для себя разработку проекта по принципам agile, редко использует какую-то конкретную модель. Чаще всего команда начинает пользоваться стандартными инструментами, затем пробует использовать что-то новое, от чего-то отказывается, что-то изобретает. В этом состоит смысл гибких методологий: команда не обязана следовать конкретным правилам, она берет уже работающие инструменты и практики и адаптирует их под себя. Ниже представлен разбор основных практик и инструментариев, которые могут быть полезны для команды, работающей над разработкой ПО для телекоммуникационных операторов.

2.2.1 Проектные роли

В чем заключается основное отличие waterfall команд от agile команд? Во-первых, команда, работающая по методологии waterfall, большая, состоит из разных специалистов, каждый из которых выполняет работу исключительно в рамках своего департамента. Дизайнеры пишут и редактируют дизайн, разработчики пишут код, работают над дефектами, и выполняют исключительно те задания, которые даны им DEV лидом. Специалисты отдела тестирования проверяют дизайн и тестируют продукт, выданный разработчиками, и, в свою очередь, подчиняются QA лиду или QA менеджеру, и др. Все эти специалисты подчиняются проектному менеджеру, который уже непосредственно общается с заказчиками и выясняет все их нужды. В Agile, в большинстве своём, все должности и звания являются ярлыками, так как принцип Agile – создать небольшую самоорганизованную команду профессионалов, каждый член которой многофункционален и способен привнести ценность не только в рамках своей прямой обязанности. Однако, для удобства разработки некоторые методологии, такие как Scrum или гибриды Scrum с другими методологиями подразумевают наличие неких проектных ролей:

Product owner – это человек, который должен обладать видением продукта и работы команды, досконально знать бизнес и желания заказчика,

а также понимать все риски, слушать команду и всегда быть доступным для вопросов и обсуждений.

Основные задачи:

- создание и управление product backlog и sprint backlog;
- сбор требований и обеспечение прозрачности требований заказчика к продукту;
- общение с заказчиком;
- общение с командой;
- обеспечение коммуникации между командой и заказчиком;
- принятие всех важных решений относительно бэклога, формирование бэклога спринта;
- расстановка приоритетов задач;
- сбор обратной связи по продукту и донесение ее до команды;
- владелец продукта должен нести ответственность за полезность продукта;
- с точки зрения бизнеса – это прибыль. Деятельность владельца продукта оценивается из расчета, сколько прибыли было принесено.

Основные черты, присущие роли владелец продукта в рамках телекоммуникационного проекта:

- такой человек должен быть подкован в сфере телекоммуникационных технологий, чтобы понимать, что делает его команда, какой конечный продукт она поставляет и каковы тенденции ее развития относительно развития рынка телеком услуг;
- он должен быть компетентен в сфере рынка и бизнеса, так как он должен понимать, что и для чего хочет заказчик, какую прибыль принесет компании эта работа, как правильно разговаривать с заказчиками и т.д.;
- владелец продукта должен уметь принимать решения и обладать нужными для этого полномочиями. Владелец продукта – центральная фигура в команде, он может обсуждать работу с

командой, прислушиваться к её мнению, но в итоге окончательное решение остаётся за ним. Менеджмент также не должен вмешиваться в работу владельца продукта, менеджмент должен дать владельцу продукта полную свободу делать всё, что он захочет, чтобы добиться успеха;

- владелец продукта должен обладать таким качеством, как стрессоустойчивость, так как эта роль в команде является, пожалуй, самой сложной. Он испытывает давление со стороны заказчика, менеджмента, команды, он должен выслушивать все стороны и уметь принимать верные решения, а также брать на себя ответственность;
- он должен уметь наблюдать (видеть всю картину в целом), ориентироваться (видеть все возможные варианты развития событий), решать и действовать;
- действия владельца продукта всегда должны быть последовательны;
- владелец продукта должен быть всегда открыт для обсуждений и доступен для команды, чтобы уточнять требования, услышать пожелания, возможные риски и др.;
- владельцу продукта необходимо уметь отказывать заказчику и выяснять его приоритеты.

Исходя из всего вышеописанного, можно сделать вывод, что product owner – это человек, одинаково хорошо разбирающийся как в продукте, который он представляет, так и в бизнесе и рынке. Именно поэтому считается, что на эту роль лучше выбирать не генеральных директоров компании, технических менеджеров или программных менеджеров или других руководителей, а людей, связанных с бизнесом и имеющих возможность посвятить себя нуждам компании.

Скрам-мастер – главный помощник команды. Этот человек, следящий за ходом проекта, помогающий команде преодолевать трудности и дающий советы.

Основные задачи:

- обеспечивать проведение всех необходимых митингов: standup meeting, planning meeting, retrospective meeting и др.;
- выявлять и устранять трудности, с которым сталкивается команда;
- следить за атмосферой в команде, устранять конфликты;
- отслеживать прогресс команды и статусы задач;
- внедрять и следить за соблюдением Agile практик;
- обеспечивать донесения необходимой для принятия правильных решений информации до нужных людей;
- обеспечивать непрерывное совершенствование команды за счёт разбора проблем на ретроспективных митингах и проверки работы над ошибками.

Качества, присущие роли Scrum master:

- понимание принципов и ценностей Agile, умение использовать их в реальных процессах;
- стремление к постоянному обучению и совершенствованию;
- способность устранять конфликты и трудности, умение находить нужные контакты;
- терпение.

Хотя применение гибких методологий не всегда подразумевает наличие двух вышеописанных ролей, однако следующие характеристики присущи, пожалуй, большинству успешных гибких команд:

Команда – сплочённая команда, обладающая знаниями и навыками, необходимыми, чтобы воплотить задумку владельца продукта в жизнь. Несмотря на то, что у каждого члена команды есть своя приоритетная роль – написать дизайн, написать код или протестировать функционал, тем не менее, специалисты гибкой команды должны уметь и делать больше, они должны брать на себя любую задачу, которую позволяют им взять в работу их способности.

Черты:

- команда должна быть самоорганизованной, то есть каждый член команды должен уметь брать себе задачу из бэклога и брать на себя ответственность. В Agile мире нет места для менеджеров, DEV и QA лидов, т.е. тех людей, которые дают работу своим подчинённым в waterfall командах. В гибкой команде все стараются работать наравне;
- команда должна состоять из людей с гибким мышлением, готовых к изменениям и экспериментам, проявляющих инициативу и не боящихся высказать своё мнение;
- в команде должна преобладать атмосфера взаимодействия и взаимопомощи, а также взаимопонимания. Команда должна поддерживать атмосферу взаимного обмена знаниями;
- команда должна быть сравнительно небольшого размера. Желательно 5-10 человек. Такая необходимо обуславливается тем, что, когда в команде много человек – становится сложно следить за ходом работ, а на разбор и выяснения тратить больше времени, команда перестает укладываться во временные сроки stand-up и planning meetings;
- желательно, чтобы команда находилась в одной локации, так как это позволяет команде быстрее узнавать необходимую информацию, приходить друг другу на помощь и без усилий пользоваться знаниями, которыми обладают коллеги. То есть в таком случае начинают стираться грани между отделами. Например, тестировщики работают парно с дизайнерами над разработкой дизайна и тест кейсов, а затем могут работать парно с девелоперами непосредственно над продуктом по принципам test driven development, в то время как в классических командах каждый отдел выполняет свою работу только на определенном этапе разработки, т.е. тестировщики включаются в работу после написания дизайна и кода. В таком случае команда учится видеть продукт не только со

своего угла зрения и думать не только о своем кусочке продукта, но и видеть процесс и продукт с разных сторон;

- команда должна ставить перед собой все более высокие цели, чтобы постоянно совершенствоваться.

Как было упомянуто выше, гибкая команда, желательно, должна быть небольшого размера, т.е. 5-10 человек. Если каскадная команда насчитывает большее количество людей, то для удобства работы её можно разбить на несколько гибких команд, в каждой из которых может быть по 1-2 дизайнера, 2-4 разработчика, 1-2 тестировщика и 1 скрам мастер, владелец продукта может работать сразу на 2 команды [16, 23, 25].

Если же на разработку по принципам гибких методологий планируется перевести целое предприятие с более, чем 50 работниками, тогда практики гибкого фреймворка SAFE (Scaled Agile Framework) предлагают создание нескольких release trains, т.е. «релизных поездов», состоящих из нескольких команд, сконцентрированных на работе над одним проектом\программой. Все релизные поезда, в свою очередь, объединяются в единое портфолио, т.е. все проекты\программы, разрабатываемые организацией.

Наличие командных скрам ролей представляется целесообразным в разрезе телекоммуникационных проектов, так как, во-первых, команде необходим человек, хорошо разбирающийся в телекоммуникационном рынке и способный расставлять приоритеты задачам, во-вторых, команде необходим человек, способный управлять внутренними командными процессами и обеспечивающий прозрачность рабочих процессов в тот момент, когда команда будет заниматься непосредственно разработкой. Кроме того, одной из основных тенденций рынка является привлечение заказчика к командной работе, что возможно сделать, предложив представителю заказчика взять на себя роль владельца продукта. Таким образом, заказчик сможет быть тем самым специалистом, который понимает специфику рынка, нужды своей компании и нужды конечных пользователей и способен действительно правильно распределить задачам приоритеты.

2.2.2 Пользовательская история

Одним из основных артефактов гибких методологий является user story, или пользовательская история, которая может использоваться Agile командами на этапах оценки, планирования, имплементации и приёмки. Пользовательская история – это краткая формулировка желания заказчика, описывающая функцию системы. В отличие от спецификаций, пользовательская история не является детальным описанием требований, цель пользовательской истории – отобразить намерение заказчика, показав, что и для чего необходимо разработать.

Пользовательская история должна сочетать в себе следующие немаловажные качества и должна быть оценена по критериям INVEST, представленным в Таблице 2.2.

Таблица 2.2 - Критерии INVEST

Качество	Описание
Independent.	Пользовательская история не должна зависеть от других пользовательских историй, так как в противном случае её будет сложнее сделать и запланировать.
Negotiable.	Она должна быть редактируемой и легко читабельной и понятной всей команде, а также пользователям и стейкхолдерам. То есть на этапе оценки и подготовки к спринту user story может и должна обсуждаться командой, чтобы каждый член команды мог внести в неё свои поправки, предложить улучшения, подметить и оценить риски.
Valuable.	Она должна обладать реальной ценностью и приносить прибыль. То есть пользовательская история – это часть ценной функциональности, которая должна быть реализована и выдана в рамках одного спринта.
Estimable.	Это означает, что пользовательская история должна быть удобного размера для того, что команда смогла оценить ее перед\на планировании спринта.
Small.	Она должна быть правильного компактного размера и должна быть конечной. Это означает, что пользовательская история должна быть осуществимой, должна приводить к целостному результату. Также она должна умещаться в спринт, так как основным принципом концепции пользовательский историй является полная завершенность истории к концу спринта. Если пользовательская история слишком расплывчата, то лучше разделить ее на более мелкие блоки.
Testable.	Она должна обладать свойством тестируемости. Это значит, что

Качество	Описание
	до деливери пользовательская история должна быть протестирована от и до относительно заранее продуманных критериев, которым она должна соответствовать.

Для правильного написания user story лучше всего – писать её командой. Причем в данном случае под командой подразумевается, как минимум, команда разработки и владелец продукта (если такая роль существует на проекте), а еще лучше – привлекать к написанию клиента и конечных пользователей. Текст должен объяснять роль юзера\системы, его\её действия и причину, побуждающую его совершать эти действия: будучи <…> я нуждаюсь в <…> по причине <…> [17]. Данные шаги представлены на рисунке 2.3.

Рисунок 2.3 - Схема работы над пользовательской историей

- есть одна роль: для начала нужно определить все роли в системе, затем написать пользовательские истории от имени всех ролей или групп ролей. Это поможет проанализировать работы, правильно расставить приоритеты и понять, истории для какой целевой группы наиболее важны для разработки и заказчика;

- есть одна активность\действие: действие - это основная мысль, цель истории. Главное, чтобы действие было одно. Важно – описать что нужно сделать, для пользовательской истории и не важно как. Способы решения определяются командой уже на этапе планирования\разработки;
- есть причина, которая ведёт к бизнес-ценности: зачем мы это делаем? Это самая сложная часть истории, так как легко указать, кому нужен функционал и какой функционал, но для чего? Необходимо определить, какую ценность принесёт эта пользовательская история – позволит ли она создать новый функционал, который привлечёт заказчику новых клиентов, позволит ли улучшить старый функционал, сделать работу системы быстрее и т.д. Причем user story должна формулировать не только бизнес-ценность, но также требования для команды разработчиков, т.е. что нужно разрабатывать и тестировать.

Наконец, к каждой пользовательской истории необходимо составлять критерии приёмки – критерии, которым user story должна отвечать на выходе. Они нужны для того, чтобы уже до начала разработки истории было понимание того, каким должен быть конечный результат и каким требованиям он должен отвечать.

Формулировка должна быть функциональной и очень конкретной, пользовательская история не должна звучать обобщенно, то есть, например, так: «будучи клиентом телеком оператора, я нуждаюсь в личном кабинете, чтобы совершать различные операции со своим аккаунтом». Более правильным будет разбить данную user story на более маленькие кусочки: «будучи активным пользователем мобильного телефона, я нуждаюсь в удобном личном кабинете, чтобы проверять свои расходы за установленный временной срок с возможностью скачивания и печати отчётов», «будучи активным пользователем мобильного телефона, я нуждаюсь в личном кабинете, чтобы подключать\отключать дополнительные платные и

бесплатные услуги», и т.д. Какие могут быть критерии приёмки, например, для второй истории:

- я могу видеть список всех подключенных услуг;
- я могу видеть список всех доступных для подключения услуг;
- я могу видеть список бесплатных и платных услуг;
- я могу подключить услугу;
- я могу отключить услугу.

Наличие бэклога, состоящего из пользовательских историй представляется рабочей практикой в разрезе телекоммуникационных проектов, так как подобная постановка задачи отображает не только технические характеристики к функционалу, который необходимо реализовать, но и отображает в своей формулировке бизнес смысл функционала, что позволяет команде лучше понять, зачем и для кого необходимо реализовать то или иное требование.

2.2.3 Инструменты управления гибкими проектами

Для быстроты и удобства управления гибкими (а также и каскадными) проектами команды используют современные онлайн платформы, наиболее популярными из которых являются: Jira, TFS, Version One, Rally. Сравнительные характеристики этих платформ представлены в Таблице 2.3.

Таблица 2.3 – Сравнение инструментов управления проектами

Параметр	Jira	TFS	Version One	Rally
Лицензия	Proprietary/Free community licenses for open source and academic projects	Proprietary, Commercial	Proprietary, hosted	Proprietary/Free trial
Платформа	Web-Based/Installed	Web-Based/Installed	Web-Based	Web-Based
Пользователи	фрилансеры крупные/средние/ мелкие компании	крупный/средние/ мелкие компании	фрилансеры крупные/средние /мелкие компании, некоммерческие	фрилансеры крупный/средние/мелкие компании

Параметр	Jira	TFS	Version One	Rally
			компании	
Управление бэклогом через drag-and-drop	да	да	да	да
Иерархия объектов бэклога	да	да	да	да
Диаграмма сгорания задач	да	да	да	да
Планирование и отслеживание релиза	да	да	да	да
Portfolio planning	нет	да	да	да
Поддержка мультипроектности	да	да	да	да
Отслеживание дефектов	да	да	да	да
Оповещательная система	email	email	email	email
Пользовательские роли	нет	нет	PO, SM, Team Member, Stakeholder, plus custom roles.	SM, PO, Team Member.
Просмотр доски задач	да	да	да	да
Минусы	Недостаточно информативный бэклог: Отсутствие информативного репортинга из коробки	Продукт подразумевает использование иных продуктов Microsoft	Неудобный User Interface; Невозможность работы с мобильного устройства; Сложность в использовании	Неудобный User Interface; Сложное управление иерархией объектов; Отсутствие информативного репортинга из коробки
Плюсы	Поддержка на	Система контроля	Бесплатная	Интегрированны

Параметр	Jira	TFS	Version One	Rally
	различных языках; Удобный интерфейс; Большое пользовательское сообщество; 600+ plugins and add-on; Интеграция с кодом; Доступность с мобильного устройства; Возможность кастомизации; Аутентификация через LDAP	версий; Кастомизация; Удобные инструменты отчетности	пробная версия для команды до 10 человек. Обеспечивает поддержку работы кросс-функциональных команд; проработанный интерфейс для планирования, позволяющий осуществлять трекинг эпиков, юзер-стори и проектов	й инструмент менеджмента дефектов; Поддерживает работу кросс-функциональных команд; ht

Данные системы позволяют просматривать доску задач, поддерживать иерархию задач, отображать статусы задач, отслеживать приоритеты задач и дефектов, создавать удобные графики, диаграммы (наподобие диаграмм сгорания задач), планировать и отслеживать спринт и другое [10, 17].

2.2.4 Командные встречи

Важным принципом Agile методологий является непрерывное общение членов команды друг с другом с целью узнавания статусов задач, прогресса, планирования, обратной связи и быстрого устранения проблем [16]. Для этого теоретики и практики Agile методов разработали несколько различных типов командных встреч, описания которых представлены в Таблице 2.4.

Таблица 2.4 – Командные встречи

Название встречи	Принцип	Участники
Standup meeting	Утренние stand-up встречи позволяют команде видеть прогресс и проблемы. Данные встречи должны проходить ежедневно в одно и то же время. Основная цель данного митинга – проследить за процессом работ и статусом задач, узнать, есть ли у команды препятствия и, если есть, скрам мастер должен предпринимать меры, чтобы разрешить их.	Команда, скрам мастер, владелец продукта

Название встречи	Принцип	Участники
	Черты: - Проходит ежедневно в одно и то же время по утрам; - Вся команда должна находиться в одной локации. В том случае, если это невозможно, команда может собраться на видео-конференции; - Длительность встречи – не более 15 минут; - Обсуждаются три вопроса: что было сделано каждым членом команды вчера, что будет сделано сегодня, есть ли препятствия, мешающие работе.	
Planning meeting	Planning meeting позволяет каждому члену команды понимать, какие задачи в данный момент разрабатываются командой, может ли он внести свой вклад и позволяет в целом осознавать, какой продукт команда поставляет заказчику.	Команда, скрам мастер, владелец продукта Опционально: заказчик
Retrospective meeting	После того, как команда показала сделанную работу и получила фидбэк от заказчиков, стейкхолдеров, конечных пользователей, необходимо провести ретроспективную встречу, на которой обсуждается работа в прошедшем спринте, определяются положительные моменты и моменты, над которыми команде необходимо поработать. Таким образом, данная встреча помогает определить, какие Agile практики подходят команде, а какие нужно исключить, над какими слабыми моментами стоит поработать. Чтобы собрание было действенным, нужно построить доверительные отношения между членами команды. Особенно важно, чтобы люди чувствовали себя одной командой и брали ответственность за процессы и результаты. Главное, о чем нужно помнить, что цель этого митинга не кого-то обличить, а улучшить рабочий процесс, определить, что пошло не так, и как можно избавиться от проблем и ускорить ход работ. Митинг начинается с того, что каждый член команды вкратце записывает свои впечатления о спринте в двух колонках: «что прошло успешно», «что пошло не так». Для этой цели может использоваться как обычная доска со стикерами, так и электронные таблицы, программы. Затем по кругу каждый член команды более развернуто описывает свои впечатления по написанному пункту, остальные члены команды также должны вступать в дискуссию, соглашаясь или не соглашаясь с высказыванием. Далее общим голосованием определяются несколько наиболее важных проблем (чаще всего 3-4), которые команде необходимо устранить в следующем спринте. Далее команда обсуждает, каким именно способом можно разрешить эти проблемы и выдвигает action points, за выполнением которых следит скрам мастер. Данная встреча – один из главных инструментов отладки процесса работы в команде и	Команда, скрам мастер, владелец продукта Опционально: заказчик

Название встречи	Принцип	Участники
	должна проводиться на регулярной основе и посещаться всеми членами команды.	
Demo meeting	Для того чтобы быть конкурентоспособным на рынке и иметь лояльного и счастливого заказчика, компании необходимо, чтобы её продукт как можно быстрее оказался в руках тех, кто будем им пользоваться. Для этого гибкие методологии предлагают демонстрации разрабатываемого продукта заказчику с целью получения обратной связи и узнавания, в правильном направлении ли двигается разработка. Так Agile практики советуют в конце каждого спринта собираться в составе: команда, заказчик, стейкхолдеры, ПО, скрам мастер, потенциальный потребитель на демонстрацию готовой части продукта и обсуждения. Как правило, дизайнер или специалист отдела качества готовят презентацию и показывают работу функционала в случае sunny day (успешный) и rainy day (работа системы в случае сбоя, ошибок) сценариев. После этого устраивается совещание, на котором все члены встречи озвучивают свое мнение и договариваются о возможных изменениях в том случае, если заказчик\конечный пользователь получил не совсем то, что хотел. Чтобы демо встреча прошла успешно, необходимо: − провести внутреннее командное демо для определения того, что необходимо показать на основном демо; − демонстрировать только важный и готовый функционал, опуская технические детали; − строгая агенда встречи с попунктным описанием порядка демонстраций и обсуждений; − создать повторяющиеся встречи, чтобы все люди, чье присутствие необходимо, всегда знали, где и когда будет проходить следующее демо. Это позволяет участникам заранее запланировать своё расписание; − проверять заранее готовность конференц-зала, оборудование, интернет подключение, качество связи − фиксировать соглашения и выводы из дискуссий для сохранения истории и чёткого понимания измененных требований заказчика; − ввести ведущего демо, который будет следить за временем, ходом встречи и фиксировать важные пункты дискуссий**.**	Команда, скрам мастер, владелец продукта, заказчик, стейкхолдеры, потенциальные потребители
Backlog refinement meeting	Данная встреча проводится в конце каждого спринта или перед планированием нового спринта. Ее цель – рассмотреть, что команда успела сделать за предыдущий спринт, какие задачи команда не успела покрыть, принимается решение о реприоритезации задач, часть задач убирается из бэклога.	Владелец продукта, команда, скрам мастер

Использование вышеуказанных встреч представляется целесообразным для телекоммуникационных проектов, так как они могут способствовать налаживанию процесса общения как внутри команды, так и с заказчиками, а также отслеживанию статусов задач в ходе спринта, скорости команды, сроков поставки, реприоритизации задач. Кроме того, демо встреча и ретроспектива позволяют командам давать и получать обратную связь относительно задач и хода работ, что способствует непрерывному устранению дефектов и улучшению качества программного продукта.

2.2.5 Бэклог и методы его приоритизации

Перед стартом проекта команде необходимо набрать список функциональных требований к продукту, которые нужно будет взять в работу. Список таких задач, а точнее пользовательских историй и требований является Product Backlog. За продуктовый бэклог отвечает product owner, который составляет максимально полный список требований заказчика к функционалу, выявляет срочность выполнения каждой задачи и её ценность для бизнеса и заказчика и расставляет задачам приоритеты.

После сбора требований должны быть написаны пользовательские истории. Желательно, чтобы в их написании участвовали не только владелец продукта и команда, но также и сам заказчик. В таком случае, у команды будет уверенность в том, что она делает продукт именно таким, каким его хочет видеть заказчик. Однако, не всегда заказчика возможно вовлечь в данную активность, в таком случае команде необходимо, чтобы требования заказчика были предельно четко, ясно и лаконично сформулированы владельцем продукта.

Для того, чтобы правильно расставить задачам приоритеты, необходимо определить следующее:

- какие требования являются наиболее важными для хода разработки, послужат основой для дальнейших работ и принесут наивысшую прибыль компании;

- какие требования являются наиболее важными для заказчика и конечного пользователя;
- какие требования проще всего осуществить.

Необходимо с самого начала иметь понимание, что не все задачи из бэклога будут реализованы. Во-первых, ситуация на рынке телеком операторов крайне изменчивая и часть функционала просто будет уже не нужна к тому моменту, как команда разработчиков сможет начать над ней работу. Во-вторых, бэклог постоянно пополняется новыми задачами, поэтому часть задач с низким приоритетом может быть никогда не выполнена. В связи с этим необходимо перед каждым спринтом делать реприоритезацию задач (то есть заново определять, насколько важна каждая конкретная задача для продукта и заказчика, снижать или повышать приоритеты старых задач), чтобы бэклог всегда был актуальным по отношению к нуждам заказчика и рынка.

Существует несколько техник сортировки задач для бэклога и выявления приоритетов. Данные техники рассмотрены в таблице 2.4.

Таблица 2.4 – Техники приоритизации бэклога

Техника	Описание
Приоритезация на основе бизнес-ценности	Данная техника предполагает оценку требований\пользовательских историй с точки зрения её важности для бизнеса, т.е. сколько прибыли она может принести компании, насколько она важна для заказчика, конечного пользователя, репутации компании. Чем выше ценность требования, тем раньше оно должно быть взято командой в работу, реализовано и представлено на рынке. Бизнес-ценность требования может определяться владельцем продукта самостоятельно, однако, для минимизации ошибок в расставлении приоритетов задач, необходимо привлекать заказчиков и стэйкхолдеров.
Приоритезация на основе рисков	Приоритет задач определяется на основе риска, связанного с их реализацией: сложная логика, интеграции с другими системами, использование новой технологии и т.д. Как правило, пользовательская история с наибольшим реализационным риском считается наиболее приоритетной и включается в разработку в самых первых релизах.

Техника	Описание
Модель Кано	Данная техника предполагает классификацию требований по 5 категориям предпочтений и ожиданий клиента: Must-be quality – характеристики, которые необходимы заказчику и продукту в первую очередь; One-dimensional Quality – одномерные характеристики, функциональность, которую желает видеть заказчик и которая спроецирует его недовольство в случае её отсутствия; Attractive quality – привлекательные для заказчика функциональности, которые не требуются заказчикам, но принесут им и, соответственно, компании пользу в случае их разработки; Indifferent Quality – неважная для заказчика функциональность, то есть, например, технические решения, скрытые от конечного пользователя; Reverse Quality – функциональность, которая дополняет другие функциональности, улучшая и усложняя продукт. В ряде случаев, такие функциональности считаются минусом, так как заказчик не всегда хочет получить сложный продукт.
Walking skeleton	Данная техника берет за основу приоритезацию на основе связности компонентов системы между собой, т.е. высший приоритет получают те требования, которые тесно связаны между собой и должны быть реализованы одновременно. То есть для начала выбираются те задачи, реализация которых позволит создать первую рабочую версию продукта. Остальные требования группируются по схожему принципу и получают один приоритет. Так, реализация приоритетов из одной группы позволяет присоединить к продукту новый end-to-end рабочий функционал.
Метод MoSCow	Данная техника базируется на основе 4 утверждений, одно из которых соответствует пользовательской истории: − Must have - данная функциональность обязательно должна быть реализована в первую очередь и поставлена заказчику в первых релизах; − Should have – данную функциональность необходимо реализовать, если это возможно в данном спринте. То есть такие требования также являются очень важными для заказчика, но не настолько важными как требования, относящиеся к первой категории, а значит могут быть реализованы в последующих релизах; − Could have – данную функциональность можно реализовать, если на неё есть время и если она не сломает уже существующий функционал; − Won’t have – данная функциональность также важна для проекта, но в связи с нехваткой времени, она будет реализована в будущем.
Theme scoring – оценка по темам	Данная техника предполагает сравнивание требований друг с другом по различным критериям, которые позволяют определить важность включения данного функционала в спринт. Выбираются критерии для сравнения (риск, прибыль, важность для заказчика и т.д.), затем выбирается тема – базовая задача, с

Техника	Описание
	которой будут сравниваться остальные задачи. Если сравниваемая задача важнее, чем основная по какому-либо критерию, тогда ей добавляется дополнительный балл к приоритету, если менее важная, то балл отнимается. После оценки требований по всем критериям ведется подсчет баллов. Чем больше сумма баллов, тем выше приоритет задачи.

Следует обратить внимание, что приоритезацией задач занимается владелец продукта. Однако, он может обращаться к опытным членам команды за советом, обсуждать с командой список требований с целью выяснения мнения команды о наиболее приоритетных для разработки требованиях, но всё же ключевое решение остается за ним. Также приоритет задач может и должен быть уточнён с заказчиком [16, 17, 24]. В разрезе телекоммуникационных проектов наиболее перспективными представляются техники приоритезации на основе бизнес-ценности и рисков, так как эти параметры являются самыми существенными для телекоммуникационного рынка.

2.2.6 Методы оценки бэклога

После того, как владелец продукта составил бэклог задач, а пользовательские истории написаны, команде необходимо заняться его оценкой, то есть определить, выполнима ли задача, понятны ли требования заказчика, достаточно ли заказчик дал информации для её выполнения.

Основные характеристики процесса оценивания в Agile командах:

- оценивание должно происходить быстро: суть гибких методологий – скорость создания и внедрения ПО. Чтобы придерживаться высокой скорости и быстрой производительности, необходимо также уметь быстро давай оценку задачам. Причем, нужно понимать, что оценка не может быть предельно точной, и на основе этой оценки нельзя выделять для решения задачи фиксированный бюджет. Главное, для чего нужна оценка в Agile командах – это создание понимания того,

насколько сложно будет выполнить задачу, сколько человеческих ресурсов она может потребовать;

- оценивание должно быть командной работой: в Waterfall команде оценками, в основном, занимаются люди, находящиеся на лидирующих позициях, т.е. DEV и QA лиды, аналитики, иногда к оцениванию работ привлекают людей, которые хорошо разбираются в данном функционале. Agile подход подразумевает командную оценку – то есть вся команда собирается вместе, изучает пользовательские истории и оценивает по удобному ей принципу. Это может быть оценка в баллах, категориях small, medium, big, размерах одежды и так далее. Команда выбирает для себя удобный метод. Главное – не оценивать бэклог в часах, так как практика показывает, что подобные оценки редко бывают точными. Суть такой оценки заключается в том, что каждый член команды может высказывать свое мнение, озвучивать свои опасения и риски. Как итог – оценкой будет являться обобщенное мнение всей команды, то есть у отдельных ее членов не будет возможности и потребности в прибавлении лишнего времени на разработку той или иной пользовательской истории «на всякий случай»;
- оценка должна быть относительной: оценка в командах, работающих по принципам Waterfall, как правило, даётся в человекоднях (единица, равная одному рабочему дню одного человека) или часах. Agile команды же используют относительные единицы измерения, которые не несут бизнес ценности, а лишь позволяют сравнивать друг с другом задачи [24].

Таблица 2.5 – Распространённые методы оценки бэклога

Метод	Описание
Самый известный метод - покер	Команда использует специальные карты с баллами, пронумерованными чаще всего в соответствии с

Метод	Описание
планирования	последовательностью Фиббоначи (порядок чисел, при котором каждое последующее число является суммой двух предыдущих 0,1,1,2,3,5,8,13,21…), но возможны и другие варианты нумерации. У каждого участника встречи в руке колода карт, включающая в себя карты с баллами, карту со знаком вопроса (используется в случае возникновения вопроса) и карту с изображением, например, чашки кофе (используется, если человек хочет сделать перерыв). Владелец продукта анонсирует пользовательскую историю, затем каждый участник команды выбирает карту с оценкой и кладет ее на стол рубашкой вверх. После этого одновременно все карты переворачиваются. Если оценки отличаются не больше, чем на 1-2 карты, тогда оценкой пользовательской истории становится среднее арифметическое значение всех карт. Если оценки слишком разнятся, тогда участники, давшие самый низкий и самый высокий балл, приводят аргументы, после чего происходит повторный раунд оценки. Далее происходит оценка следующей пользовательской истории. Покер планирования – одна из самых удобных техник оценивания задач, но не является самым быстрым способом, поэтому подходит для тех команд, которые берут в бэклог спринта небольшое количество задач. Использование подобного метода позволяет не делать оценку слишком точной. Наиболее низкий балл показывает, что пользовательская история не требует слишком больших временных затрат у команды, а наиболее высокий, соответственно, оказывает, что подобная пользовательская история может занять очень много времени, вплоть до нескольких спринтов и, поэтому, подлежит расчленению.
T-shirt sizes	Единица измерения – размеры футболки\рубашки. Команды могут определять сложность и времязатратность задачи, проводя параллели с размерами одежды «S», «M», «L», «XL» и т.д. Для начала команда в ходе открытой дискуссии принимает решение о размере нескольких пользовательских истории (если единого мнения нет, можно применить голосование). После того, как команда совместно оценила несколько пользовательских историй, становится понятна степень декомпозиции задач. Команда задает самым маленьким пользовательским историям самый маленький размер, например, XS. После этого все остальные задачи оцениваются в сравнении с задачами XS – насколько они больше. Если пользовательская история имеет ярлык XXS или «очень большая», то, команде следует обратить на нее особое внимание, возможно, такая история требует дальнейшей декомпозиции. Данная техника может показаться не очень точной, однако, её преимущество – скорость.
Техника «bucket system» с ведёрком в качестве единицы меры.	Процесс оценивания выглядит следующим образом: все пользовательские истории, требующие оценки, описываются на карточках. На столе стоят несколько ведёрок или коробок разных размеров. Команда выбирает несколько произвольных карточек и в ходе дискуссии сравнивает их относительно друг друга и выносит окончательную оценку. Так оцененные

Метод	Описание
	пользовательские истории складываются в соответствующие ведерки\коробки и служат ориентиром для последующих оценок. Затем все оставшиеся карточки распределяются поровну между членами команды и оцениваются уже непосредственно одним человеком. Если член команды не может оценить пользовательскую историю, то он может передать её на оценку коллеге. Данный метод похож на покер планирования, однако, работает гораздо быстрее и позволяет оценить большее количество задач.
Метод big\uncertain\small	Данный метод похож на метод bucket system, только командой используется всего три «ведра» - маленькое, непонятного размера, большое, и на первом этапе оценивания, взяв в качестве основы 3-4 пользовательские истории, команда обсуждает масштаб каждой категории.
Dot Voting – голосование точками	Все пользовательские истории, которые необходимо оценить, выписываются на отдельные карточки и раскладываются на столе или развешиваются на доске. Каждому члену команды раздается определенное количество стикеров, наклеек, магнитов, маркеров (для рисования штрихов или точек) и др. Затем каждый человек распределяет между задачами свои «точки» по методу – чем больше точек, тем сложнее задача и тем больше нужно на нее потратить времени. После того, как все участники сделали свою оценку, происходит подсчет общего количества точек для каждой user story. Данный метод является достаточно простым и позволяет быстро оценить небольшое количество пользовательских историй.
Ordering rule	Данный метод похож на настольную игру, главная задача которой – выстроить все пользовательские истории на единой шкале измерения относительно друг друга. Все пользовательские истории описываются на карточках и произвольным образом раскладываются рядом со шкалой с двумя границами: маленький размер и большой размер. Каждый член команды по очереди может совершить одно действие – переместить пользовательскую историю на одно деление шкалы вниз или вверх (сделать оценку меньше или больше) или устроить обсуждение с командой. Таким образом, оценка пользовательских историй с каждым ходом корректируется относительно друг друга. Раунд оценивания заканчивается тогда, когда каждый член команды пропускает свой ход. Данный метод является действенным для оценки небольшого количества задач и подразумевает командную работу.

Самые часто используемые методы оценки в Agile командах представлены в Таблице 2.5. Существуют также и другие методы оценки пользовательских историй, некоторые из них не известны общественности, так как придуманы и используемы конкретной одной командой. Так, каждая

команда, перестраивающаяся с Waterfall на Agile может также придумать свой собственный метод, главное, придерживаться основного принципа оценивания пользовательских историй по Agile – не давать конкретную оценку в конкретных величинах. Планирование должно быть быстрым, эффективным и командным [16, 24].

Для телекоммуникационных проектов, несомненно, важно предоставление оценок задачам для понимания, сколько усилий может потратить команда на реализацию того или иного функционала, и сроков поставки. Желательно делать оценку относительной, так как оценка в человекоднях практически никогда не отвечает действительности. Однако, стоит обратить внимание, что действительно качественная командная оценка возможна лишь в том случае, когда каждый член команды имеет чёткое представление о том, что требуется для реализации той или иной пользовательской истории. В разрезе телекоммуникационных проектов это представляется практически невозможным, так как подобные проекты являются очень крупными и сложными и, как правило, члены команды занимаются чаще всего определенными задачами (например, кто-то занимается исключительно интеграциями, кто-то исключительно менеджментом сетей, активацией сервисов и т.д.) и не имеют широких знаний относительно всего функционала, работающего на проекте. Кроме того, разработка изменений или нового функционала на основе работающего кода ведет за собой риски поломки старого функционала, что подразумевает детальный анализ требований, старого кода, оценку возможных рисков и определение сценариев для регрессионного тестирования. Таким образом, командная оценка не представляется целесообразной для крупных телекоммуникационных проектов.

2.2.7 Инструменты для достижения прозрачности процесса

Прозрачность всех процессов – это основной принцип всех гибких методологий, она необходима для того, чтобы команда понимала на каком

этапе разработки находится та или иная задача, есть ли какие-то трудности у неё на пути, успевает ли команда закончить все задачи в срок.

Что требуется от команды для обеспечения прозрачности:

- каждый член команды должен быть готов делиться своими проблемами и сообщать о своих ошибках;
- каждый член команды должен относиться уважительно к своим коллегам и не должен никого обвинять, так как работа в Agile команде всегда является командной работой, а значит любая ошибка, любое невыполнение работы в срок является командной проблемой;
- скрам мастер и владелец продукта должны уметь мотивировать команду разделять свои опасения и проблемы, с которыми они сталкиваются во время выполнения работ;
- заказчик должен предоставлять свое мнение о выданном продукте и задача владельца продукта – доносить это мнение до команды;
- команда не должна пропускать традиционные встречи - stand-up, planning, demo, retrospective митинги;
- каждый член команды должен проставлять корректный статус задаче и подробно описывать в комментариях сделанную работу.

Таблица 2.6 – инструменты и практики, обеспечивающие прозрачность процессов разработки

Способ\артефакт	Описание
Внедрить разделение рабочего процесса на спринты	Разделение этапов разработки на инкрементальные равные по времени периоды, называется спринтом. Результатом спринта является, как правило, новый релиз продукта или часть продукта, выдаваемая на E2E окружение. Разделение на спринты позволяет планировать сроки выдачи функционала, отслеживать количество задач, сделанных командой в определенный срок и понять, успевает ли команда реализовать все требования к концу релиза.
Сделать визуальное изображение задач со статусами	Подобные доски позволяют добиться прозрачности работы и позволяют отслеживать статусы задач, количество сделанных и не сделанных задач в спринте. Доска может выглядеть как онлайн доска в jira или доска на картоне\доске\ватмане с колонками: «бэклог», «в работе»,

Способ\артефакт	Описание
	«сделано» или «бэклог», «в разработке», «тестируется», «сдано\принято». В каждой колонке находятся стикеры с описанием пользовательской истории, в процессе работы стикеры перемещаются из одной колонки в другую. Вместо доски со стикерами команда может использовать любой другой удобный инструмент, например, создать доску с задачами в jira и др. Главное, чтобы ход работ был наглядным. В том случае, если задача долгое время не передвигается из одной колонки в другую, команда понимает, что член команды, взявший задачу в работу, столкнулся с трудностями и нуждается в помощи.
Сделать видимыми скорость команды, состояние бэклога	Также еще одним способом сделать работу видимой является создание различных диаграмм, отражающих состояние бэклога, скорость команды и другие. Например, хорошим вариантом является составление диаграммы сгорания задач, показывающей количество уже сделанных и оставшихся задач. Сделать процесс более прозрачным помогает также подсчёт средней скорости работы команды. Это можно сделать следующим способом: Взять за основу уже выполненную стандартную задачу, посмотреть, сколько стори поинтов (или иных единиц, выбранных командой) на неё уходит и высчитать среднюю скорость.
Использовать jira инструменты для трекинга времени, потраченного на задачу, а также оставшегося времени.	Системы управления проектами, такие как jira, позволяют отмечать время, запланированное на задачу, и реальное время, потраченное на неё. Таким образом становится понятно, насколько правильно работает система оценивания задач и насколько быстро команда справляется с разного рода задачами.
Definitions of ready должны быть чётко определены	DoR – это чек лист, пункты которого должны быть выполнены до того, как задача из продуктового бэклога берется командой в разработку. В основном – этот чек лист является работой владельца продукта. Пример критериев приёмки: − владелец продукта донёс информацию о задаче до команды разработчиков; − пользовательская история имеет понятную ценность для бизнеса и проекта; − пользовательская история оценена командой; − пользовательская история достаточного размера для того, чтобы она могла быть разработана в рамках одного спринта; − критерии приёмки для пользовательской истории определены командой и заказчиком.
Definitions of done должны быть чётко определены	DoD – это критерии, которые позволяют понять, что задача полностью выполнена и никакая дальнейшая работа над ней не нужна. Чем больше таких критериев, тем больше вероятность того, что функционал будет работать в соответствие с ожиданиями бизнеса. Главное, чтобы критерии были чётко сформулированы и понятны команде и заказчику. Критерии приёмки могут быть следующими:

Способ\артефакт	Описание
	Для dev user story: − Код написан; − Девтест проведён, критический функционал работает; − Юнит тесты написаны; − Дефекты критического, высокого и среднего приоритета устранены; − Рефакторинг кода сделан; − Код выдан в master branch; − Ревизия, содержащая изменения, установлена на смоук сервер и куа сервер. Для qa user story: − Функционал протестирован; − Регрессионное тестирование завершено; − Автотесты написаны, выполнены, пройдены успешно (все зелёные); − Фиксы для дефектов критического, высокого и среднего приоритета проверены; − Нагрузочное тестирование завершено. Для design user story: − Дизайн написан; − Ревью с командой разработки проведено; − Поправки внесены в дизайн.

Применение основных артефактов, практик и инструментов, представленных в таблице 2.6 может принести высокие результаты на любом проекте, в том числе и на телекоммуникационном, так как они помогают командам добиться максимальной прозрачности всех процессов, связанных с разработкой [16, 20, 25, 27].

2.3 Шаги, необходимые для перехода с каскадных методов проектирования программного обеспечения на телекоммуникационном проекте на гибкие

Как было определено в пункте 2.1 данной исследовательской работы, для разработки программного обеспечения на телекоммуникационном проекте наиболее перспективным представляется взятие за основу базовых артефактов и инструментария скрам модели с применением работающих практик других гибких моделей. К подходящим особенностям

телекоммуникационного проекта практикам относится внедрение спринтов и традиционных скрам ролей, инструментов для обеспечения прозрачности процессов, создание приоритезированного бэклога из пользовательских историй, внедрение утренних статусных встреч, встречи-планирования, демо и ретроспективы.

В данной работе предлагаются следующие шаги, необходимые для перехода с каскадных методов проектирования программного обеспечения для телекоммуникационного оператора на гибкие:

- определить требования к новой модели, отвечающие запросам телекоммуникационного рынка, и создать её скелет,
- познакомить команду с определением и практиками Agile методологий и изменить её мышление,
- создать команды и распределить в них роли,
- включить заказчика в работу,
- создать приоритизированный бэклог продукта и оценить его,
- запланировать первый спринт,
- внедрить и укрепить культуру общения между командами и заказчиком,
- сделать работу прозрачной,
- совершенствовать процесс,
- мотивировать команду.

Подобные шаги позволят команде создать подходящую ей модель проектирования ПО на проекте и внедрить её в текущий процесс разработки.

Выводы ко второй главе

Современные гибкие модели разработки программного обеспечения, такие как Scrum, Kanban, Scrumban, Scrum + XP являются эффективными, однако, они обладают рядом минусов в разрезе использования данных

моделей на телекоммуникационных проектах, в связи с чем, они требуют значительной доработки.

В качестве модели проектирования программного обеспечения для телекоммуникационного оператора наиболее перспективной представляется гибридная модель, основанная на принципах Scrum и работающих практик других ныне популярных гибких моделей, и адаптированная под нужды, а также специфику телекоммуникационных проектов.

Использование основных артефактов, процессов, инструментов и практик современных гибких моделей позволяет добиться максимальной прозрачности процесса и ведет команду к непрерывному совершенствованию. Однако, традиционная командная оценка пользовательских историй, предлагаемая рядом гибких моделей, не представляется подходящей практикой для проектов, поставляющих программное обеспечение телекоммуникационным операторам.

Основными шагами для создания гибкой модели проектирования программного обеспечения для телекоммуникационного оператора являются шаги, направленные на внедрение заказчика в процесс работ, укрепление коммуникации между членами команд и заказчиком, итеративность и более частую поставку продукта на окружение заказчика, а также внедрение метрик, обеспечивающих прозрачность процесса и увеличения качества и скорости команды.

По итогам второй главы предложены шаги для перехода с каскадной модели проектирования программного обеспечения для телекоммуникационного оператора на модель с использованием гибких практик.

Глава 3 Создание гибкой модели проектирования программного обеспечения для телекоммуникационного оператора

3.1 Анализ существующей каскадной модели проектирования программного обеспечения для Европейского телекоммуникационного оператора

Основным продуктом, разрабатываемым в рамках телекоммуникационного OSS проекта, является программное обеспечение, обеспечивающее управление сетевой инфраструктурой и ресурсами оператора, то есть сетью, подсетями, коммутаторами и т.д., ордер менеджмент и д.р. Как правило, подобные проекты являются достаточно крупными, так как их конечный продукт используется целыми городами, странами, а иногда и рядом стран. Так, подобные проекты, как правило, тянутся несколько лет и насчитывает большое количество релизов, поставляемых заказчику 2-3 раза в год. Чаще всего начало разработки подобных проектов подразумевает использование традиционной каскадной модели проектирования программного обеспечения, так как изначально к модели работы над проектом, как правило, предъявляются следующие требования:

- фиксированный срок поставки первого релиза (год на разработку) и последующих релизов (выдача новой версии продукта раз в определенный срок: 3-4 месяца);
- жёсткий бюджет, выделяемый перед началом каждого релиза на основе выдвинутых командой оценок;
- чёткие требования, которые не могут быть изменены в ходе разработки (исключая форсмажорные обстоятельства);
- формальные этапы, следующие строго друг за другом: анализ -> дизайн -> реализация -> тестирование -> E2E тестирование –> внедрение -> сопровождение.

Традиционно проектная команда состоит из нескольких бизнес аналитиков, специалистов отдела разработки и тестирования, специалистов IT отдела, проектного менеджера и QA, BA, DEV лидов и аналитиков. В рамках рассматриваемого в данной работе проекта, команда состоит из 20 человек:

- команды бизнес аналитиков в составе 4 человек, занимающихся сбором и анализом требований, общением с заказчиками, а также написанием дизайна;
- дев команды в составе 10 человек, занимающихся написанием программ на основе требований от дизайнеров, во главе с 2 DEV лидами, занимающимися анализом требований со стороны девелопмента, предоставлением оценок заказчику, наблюдением за статусом задач, распределением задач между членами команды, отчетами;
- QA команды в составе 4 человек, занимающихся тестированием программного продукта, во главе с QA лидом, занимающимся распределением задач между членами команды, отчетами и анализом требований и предоставлением оценок заказчику;
- 2 support инженера, занимающихся анализом и устранением дефектов с окружения заказчика.

Также на проекте работают привлекаемые специалисты, такие, как:

- 1 RE инженер, занимающийся сборками программного кода;
- 1 инженер из deployment департамента, занимающийся поддержкой установок продукта на сервера заказчика, обслуживанием dev и qa серверов.

В таблице 3.1 представлена этапы проектирования ПО на данном проекте (на момент проведения исследований).

Таблица 3.1 – Этапы проектирования ПО на OSS проекте, использующем практики каскадной методологии

Этап	Действия	Роли
Анализ требований и планирование работ	Заказчик формирует change requests (реквесты на изменения) и выдвигает требования к данным изменениям. Команда анализирует данные изменения, дает им предварительную оценку в человекоднях. Заказчик, ПМ, аналитики и лиды команды решают, какие изменения включить в следующий релиз, определяют их стоимость и др.	Проектный менеджер, бизнес аналитики, DEV leads, QA leads, QA аналитик, заказчики
Дизайн	На данном этапе бизнес аналитики занимаются написанием детализированного дизайна, описывающего работу продукта. К написанию документации могут быть привлечены аналитики и лиды со стороны команды разработчиков и тестировщиков.	Бизнес аналитики, лиды
Реализация	Написание программного кода, проведение дев тестирования – тестирование части функционала самим разработчиком), выдача изменений на тестовые сервера	DEV команда
Тестирование	Тестирование продукта, написание автотестов (может быть начато на предыдущем этапе)	QA команда
E2E тестирование\приёмочное тестирование	Выдача релиз-кандидата продукта на E2E сервера заказчика, поддержка E2E (интеграционного) тестирования (проводится заказчиком), устранение дефектов, найденных на данном этапе, подготовка финальной версии продукта	QA и DEV команда, заказчик
Внедрение	Поддержка внедрения продукта на сервера заказчика	специалист отдела IT Deployment, команда поддержки в лице
Сопровождение	Troubleshooting дефектов, найденных на окружении заказчика, анализ, оценка, фикс и тестирование фикса для дефекта (фикс идет в следующий релиз или, если он срочный, на окружение может быть поставлен спот фикс или хот фикс)	Команда поддержки, BA, QA, DEV команда

Тогда как старт некоторых из вышеупомянутых этапов возможен только после завершения предыдущего (например, написание дизайна невозможно без анализа требований), некоторые этапы могут существовать параллельно. Например, после того, как работы запланированы, требования готовы и основная часть дизайна написана, разработчик начинает работу над

программным кодом (дизайн дописывается), как только разработчик написал основные сценарии, которые могут быть выданы на тест, он делает дев тест и включает изменения в основную ветку кода, далее тестировщики обновляют тестовое окружение и начинают тестирование (обычно к этому моменту дизайн уже написан). Особое внимание следует обратить на этап сопровождения, который, по факту, является бесконечным, так как на окружении заказчика постоянно возникают дефекты, которые необходимо анализировать и устранять. Кроме того, данные этапы подразумевают цикличность. Как правило, уже на этапах разработки\тестирования продукта, начинается этап анализа требований к следующему релизу, но окончательные требования к продукту определяются на стадии внедрения и сопровождения. Этапы разработки и тестирования заканчиваются за 3 недели до предполагаемой установки релиза на окружение заказчика и начинается этап E2E тестирования (длится 2 недели), затем происходит устранение найденных дефектов и подготовка финальной версии продукта. Так, на рисунках 3.1 и 3.2 представлена схема текущих этапов и процессов проектирования ПО.

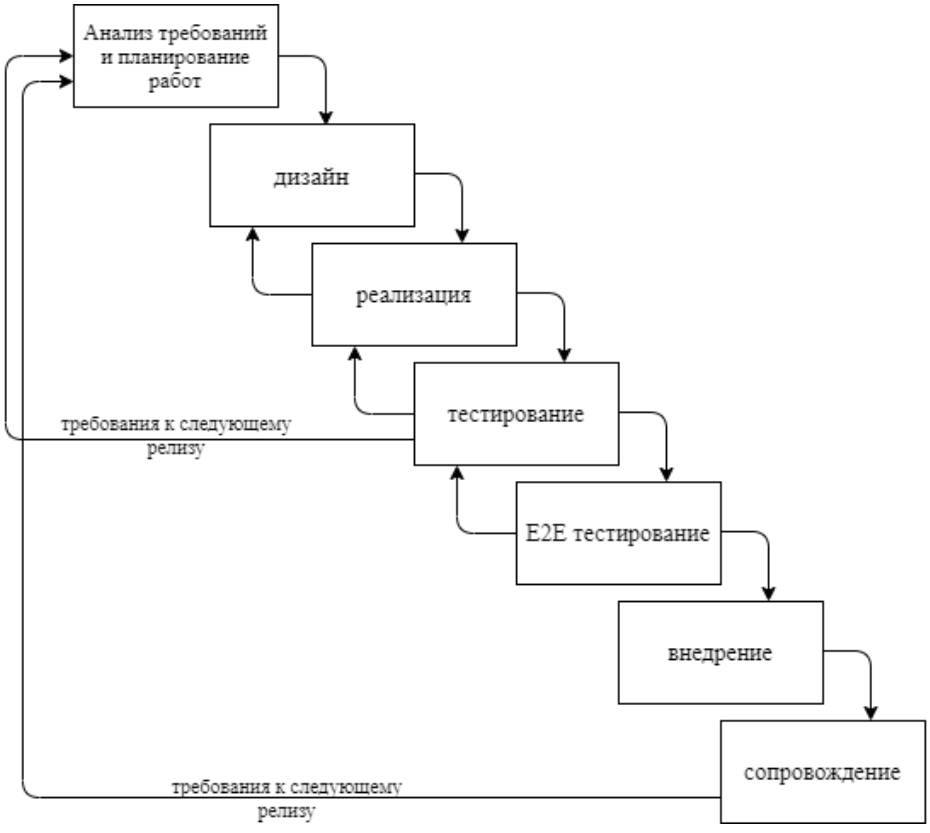

Рисунок 3.1 - Схема текущих этапов проектирования ПО

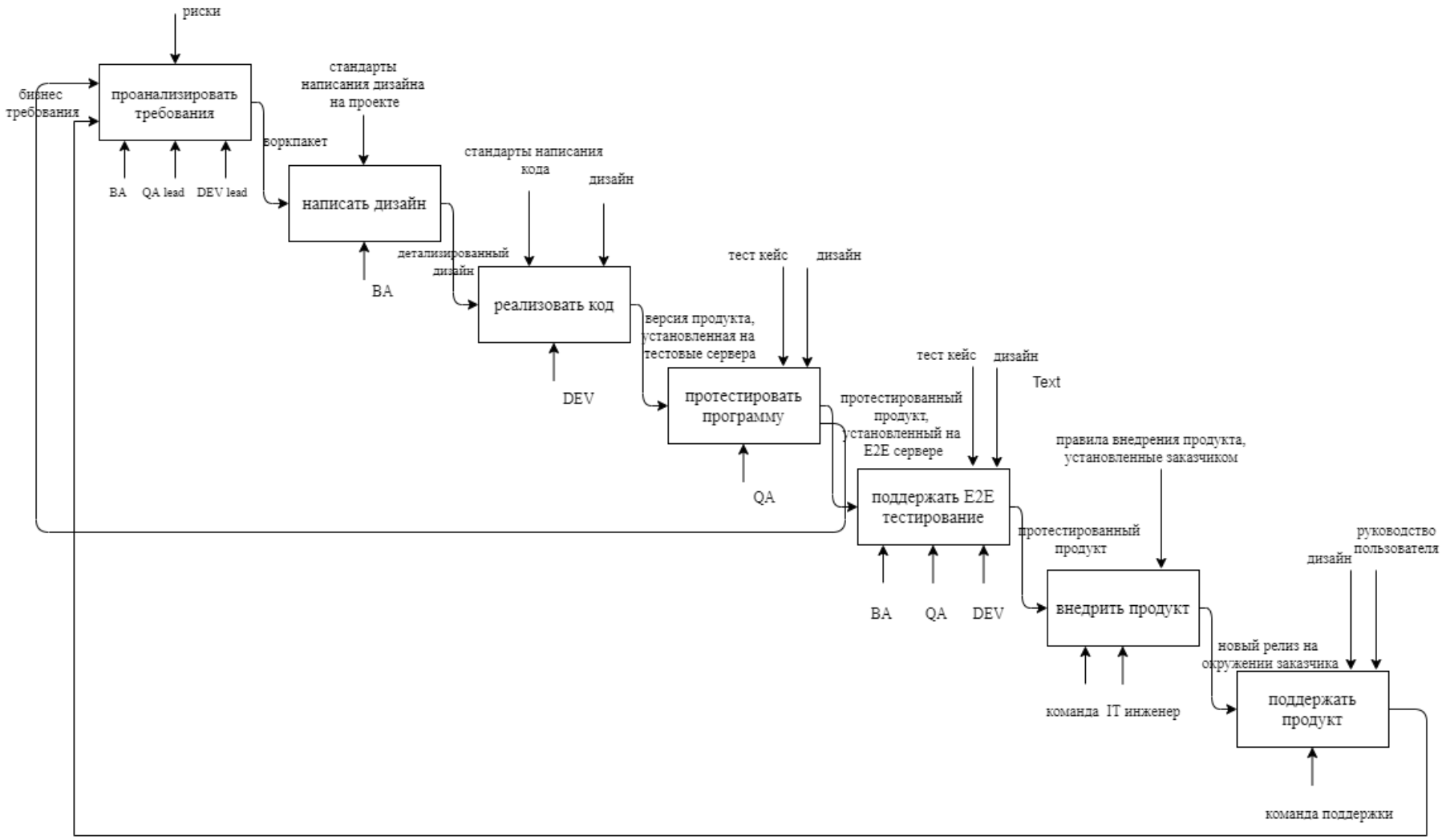

Рисунок 3.2 - Схема текущего процесса разработки ПО

Основным инструментом управления OSS проектом, рассматриваемым в данной работе, является JIRA. Для задач используется следующая иерархия, представленная на рисунке 3.3.

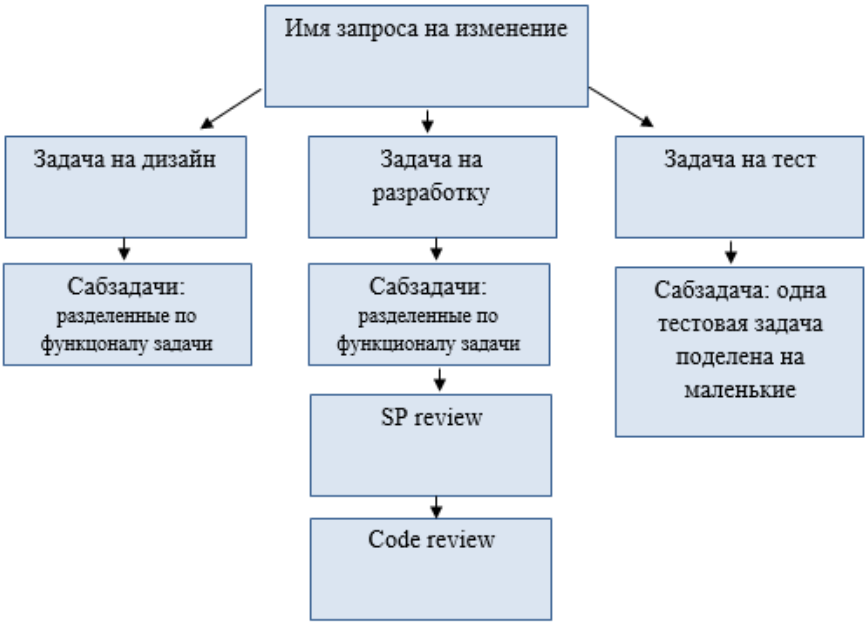

Рисунок 3.3 – Иерархия задач в Jira

Таким образом, перед началом написания дизайна бизнес аналитик создает из сохраненного в Jira шаблона указанную выше иерархию тикетов. В основной задаче содержится имя запроса на изменение и ссылка на требования. В задаче на дизайн прописана оценка, которая заложена на написание дизайна, там же дизайнер при помощи work log инструмента ежедневно отмечает время, потраченное на задачу. В сабзадачах дизайнер прилагает ссылки на написанный дизайн. По тому же принципу разработчик и QA инженер работают над своими задачами. В том случае, если тестировщик нашел дефект, он создает новый Jira ticket со статусом bug и при помощи функции link создает ссылку на данный дефект от основной задачи. Как только все задачи закрыты и все дефекты устранены, статус основной задачи переводится в closed. Кроме того, инструменты JIRA

используются DEV и QA лидами для создания репортов и dashboards (панель индикаторов), отображающих состояние задач, количество найденных дефектов, потраченное время и д.р.

Бэклог задач, как правило составляется на срок в 9 недель (до момента выдачи кода на E2E сервера). Он является неизменяемым. Оценка происходит в человекоднях и осуществляется BA, DEV и QA лидами на этапе планирования. Задачи распределяются внутри DEV и QA команды на этапе проектирования, DEV lead назначает задачи каждому разработчику его команды на основе его индивидуальных знаний и скорости работы. Тоже самое происходит в QA команде.

Особое внимание стоит обратить на команду поддержки, состоящую из двух product support инженеров, 1 постоянного разработчика и 1 периодически привлекаемого тестировщика. Инженеры разбирают и устраняют дефекты, возникшие на окружении заказчика. Если дефект является именно дефектом кода, инженеры создают bug тикет (с определенным приоритетом), который находится в бэклоге support команды до тех пор, пока его не возьмет на разработку dev инженер. Dev инженеры выбирают задачу на устранение проблемы с учётом её приоритета. Если данное поведение неучтенным поведением системы, то команда поддержки обращается к заказчикам с предложением создать запрос на изменение. Данная команда на текущий момент использует Kanban доску задач, представленную на рисунке 3.4 для отслеживания статусов всех дефектов, найденных на окружении заказчика.

Анализ и устранение дефекта	Необходим фикс	Фикс в работе	Фикс в тесте	Фикс готов
Дефект 7 Дефект 8	Дефект 6	Дефект 4 Дефект 5	Дефект 3	Дефект 1 Дефект 2

Рисунок 3.4 – Канбан доска Support команды

Таким образом, можно сделать вывод, что используемая командой модель имеет свои преимущества и недостатки, представленные в таблице 3.2.

Таблица 3.2 – Сильные и слабые стороны текущей модели разработки

Недостатки	Сильные стороны
в анализе требований участвует не вся команда, а это значит, что могут быть не учтены потенциальные риски, которые могут не заметить лиды, но заметить иной член команды	в процессе разработки уже присутствуют элементы инкрементальности, что способствует непрерывной поставке продукта заказчику
у всех членов команды нет полной картины того, что именно поставляется заказчику в релизе	члены команды являются хоть узконаправленными, но профессионалами
требования неизменяемы (подразумеваются крупные изменения, влекущие за собой большие изменения в уже написанном коде) и должны быть анализированы безошибочно, все риски должны быть учтены	на проекте внедрен процесс continuous integration (рабочие версии кода сливаются в одну ветку на постоянной основе, что позволяет быстро выявлять проблемы, сборка производится автоматически) – для этих целей используется Jenkins
оценка даётся в человекоднях, то есть не в относительной единице. Это приводит к тому, оценка редко оказывается реалистичной, т.к. невозможно учесть все факторы, которые могут повлиять на процесс работ	Jira, как инструмент управления проектом, показывает свою состоятельность и удовлетворение требованиям проекта
оцениванием работ занимается команда лидов, а дальнейшая разработка затем осуществляется другими людьми, что опять же приводит к тому, что оценка может не соответствовать реальной действительности	В этапах разработки присутствует элемент параллельности работы БА, QA и DEV
изменения дизайна подразумеваются только в случае небольших дефектов, это значит, что он должен быть высококачественным, что не всегда возможно	
команда не является кроссфункциональной. Задачи распределяются лидами на основе наличия тех или иных компетенций и знаний у человека	

Недостатки	Сильные стороны
недостаточно развита коммуникация между членами команды, так как основную работу по анализу, планированию выполняют определенные люди	
общение с заказчиками происходит исключительно через посредников, т.е. проектного менеджера, бизнес аналитиков и лидов, что приводит к потере информации	
нет пользовательский историй, описывающих зачем и для кого нужна функциональность. Команда видит лишь требования к продукту и не всегда понимает бизнес логику процесса	
отсутствует демонстрация продукта заказчику, в связи с этим команда тратит много времени на разбор дефектов с E2E, описания работ системы E2E тестировщикам и т.д	

Таким образом, анализ показал, что каскадная модель проектирования ПО для телекоммуникационного оператора не удовлетворяет текущим требованиям проекта и телекоммуникационного рынка и требует значительной доработки в рамках обеспечения более частой поставки продукта, прозрачности процессов и коммуникации между членами команды и заказчиком.

3.2 Разработка гибкой модели проектирования программного обеспечения для телекоммуникационного оператора

В данном пункте составляются требования к разрабатываемой модели, а также в соответствие с шагами, разработанными в пункте 2.3 данной работы, формируется ряд улучшений и нововведений, на основе которых создается новая модель.

Требования к новой модели строятся на основе анализа существующей каскадной модели проектирования ПО для телекоммуникационного оператора, а также современных тенденциях телекоммуникационного рынка,

заключающихся в желании заказчика интегрироваться в работу команды (например, взяв на себя роль скрам мастера и владельца продукта) и добиться более частой поставки продукта. Так, к новой модели были сформулированы следующие **требования**:

- максимальное оптимальное использование всех человеческих ресурсов;
- улучшение коммуникации между членами команды;
- внедрение новых проектных ролей и более тесной коллаборации с заказчиком;
- повышение скорости команды без потери качества;
- развитие специализации членов команды;
- внедрение средств достижения прозрачности процессов;
- срок поставки продукта – 1 раз в месяц;
- переход на новую модель должен быть быстрым;
- применимость модели к другим телекоммуникационным OSS проектам;
- способность применить модель из «коробки».

Как было определено в главе 2 данной диссертационной работы, представляется эффективным взять за основу для разрабатываемой модели традиционную модель скрам, внедрить гибкие практики и инструментарии, которые являются эффективными именно для телекоммуникационного проекта, а также предложить нововведения, которые отвечают тенденциям и особенностям современных телекоммуникационных проектов.

Для того, чтобы добиться максимально оптимального использования человеческих ресурсов, в данной работе предлагается сделать структурные изменения на проекте, основываясь на типах выдаваемых изменений:

- устранение дефектов, найденных на окружении заказчика;
- изменения для уже существующего функционала;
- совершенно новый функционал.

Так, существующая крупная команда может быть разделена на две команды, одна из которых будет заниматься исправлением существующего функционала и устранением дефектов, а также отвечать за выдачу сборки на окружение заказчика, тогда как вторая команда будет заниматься разработкой новых изменений. Также, так как представители заказчика изъявляют желание участвовать в процессе разработки, адаптируя требования своих заказчиков к продукту и участвуя в написании дизайна, а также, непосредственно, заниматься составлением бэклога для команды, в данной работе предлагаются следующие структурные изменения, представленные на рисунке 3.5.

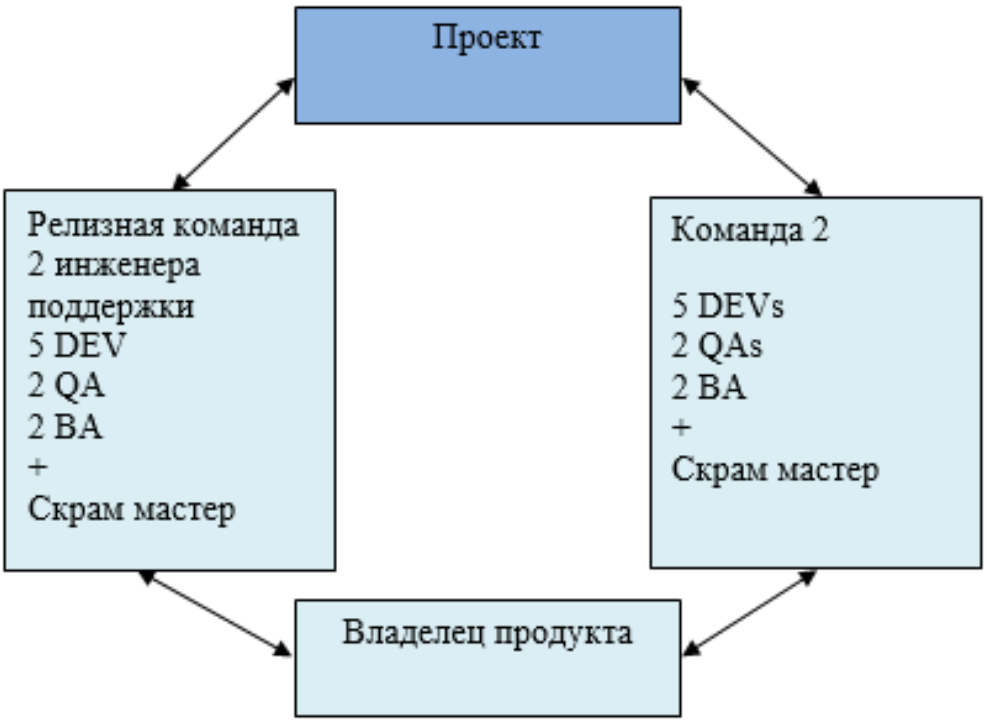

Рисунок 3.5 – структурные изменения в команде

В сферу ответственности Релизной команды входит:

- выдача продукта на окружение заказчика (бывший DEV лид продолжает заниматься подготовкой и выдачей продуктовых сборок, составлением документации к установке);

- анализ и устранение дефектов, найденных на окружении заказчика;
- разработка изменений для уже существующего на Prod окружении функционала.

Команда состоит из 2 support инженеров (их деятельность остается без изменений: анализ и применение work arounds для устранения дефектов на окружении заказчика), 5 разработчиков, 2 тестировщиков, 2 бизнес аналитиков. Скрам мастером является представитель заказчика. Владельцем продукта, соответственно, также является представитель заказчика, который создает бэклог для двух команд.

В сферу ответственности Команды 2 входит: разработка и выдача нового функционала. Команда состоит из 5 разработчиков, 2 тестировщиков и 2 бизнес, скрам мастера со стороны заказчика и владельца продукта, работающего также на другую команду.

Для того, чтобы сделать поставку продукта более частой, а процесс разработки более прозрачным и понятным каждому члену команды, на проекте необходимо сделать следующие изменения:

- адаптировать существующие этапы под новые временные рамки и роли;
- привлечь всю команду к работе над анализом и планированием, чтобы каждый член команды был в курсе всех изменений, поставляемых заказчику;
- сделать процесс работы максимально прозрачным, внедрив standup meeting, planning meeting, backlog refinement meeting и др.;
- внедрить новые метрики скорости и качества работы;
- установить плотную коллаборацию внутри команды для получения обратной связи и обеспечения совершенствования процесса;
- минимизировать позднее обнаружение пробелов в дизайне и позднее нахождение дефектов.

В первую очередь в данной работе предлагаются изменения в текущих этапах проектирования ПО. На рисунке 3.6 изображена новая модель этапов.

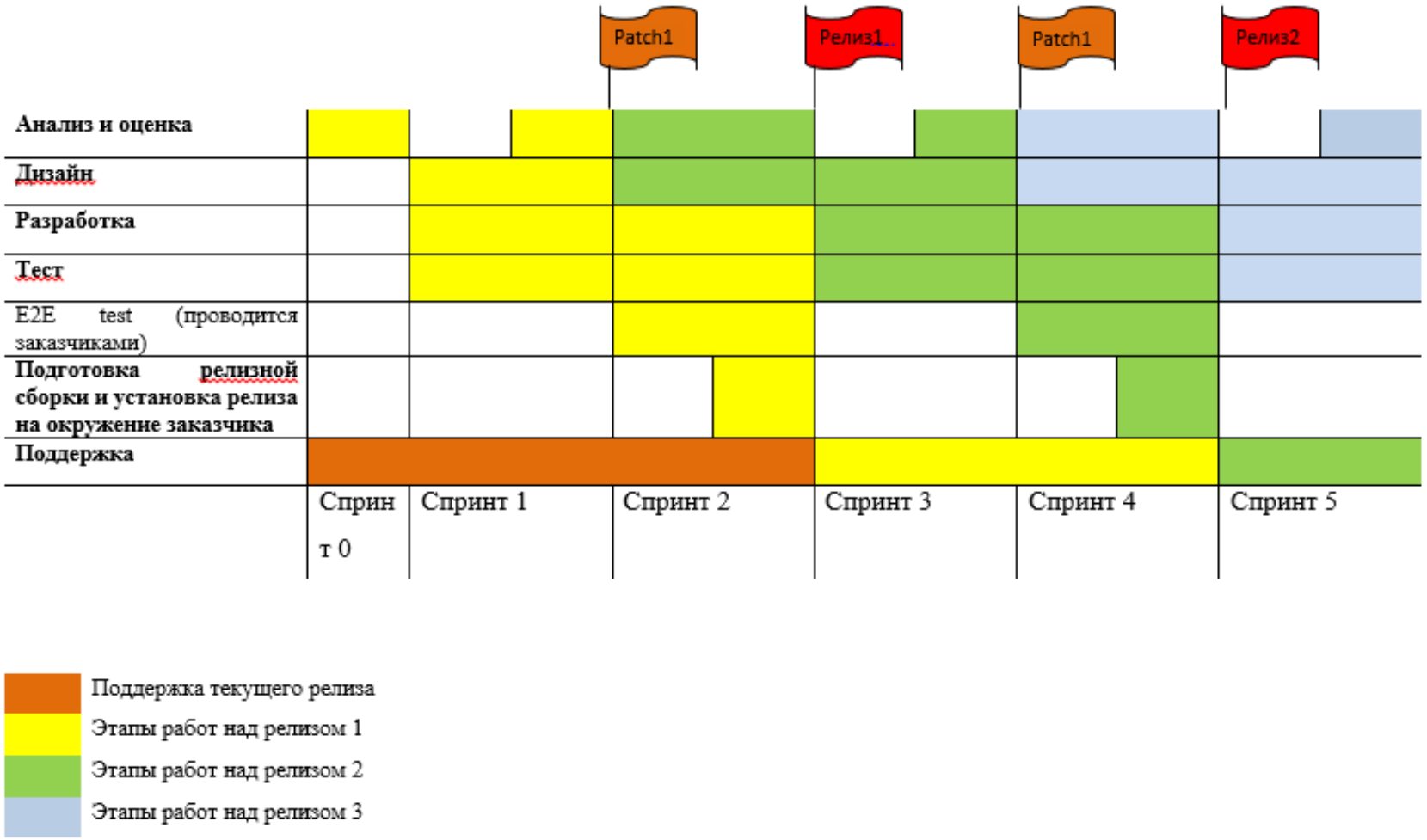

Рисунок 3.6 – Новая модель этапов проектирования ПО

Подобные этапы позволяют сделать разработку инкрементальной, а работа над релизом в итоге ведется 6 недель, тогда как сам релиз поставляется заказчику раз в 4 недели. В первый спринт (2 недели) проводятся тщательный анализ и оценка требований к пользовательским историям, бизнес аналитик может уже приступить к написанию дизайна, а QA инженер к написанию тест кейсов, в ходе 2 и 3 спринтов происходит непосредственно разработка и тестирование пользовательских историй, а также поддержка E2E тестирования и установка релиза на окружение заказчика. Ниже представлено подробное описание процессов на этапах разработки:

Спринт 0 (длится неделю) является отправной точкой для старта разработки продукта по новой методологии и позволяет команде внедрить нововведения и привыкнуть к ним перед началом работ.

Также, как и ранее, работа начинается с этапа анализа и оценки. Первым принципиальным изменением является то, что требования к продукту поставляются команде владельцем продукта в виде приоритезированного бэклога, состоящего из предварительно написанных владельцем продукта эпиков\пользовательских историй. На backlog refinement встрече ПО вкратце рассказывает команде, какие продуктовые изменения необходимо будет сделать в новом релизе и, если команда сразу видит, что пользовательская история слишком крупная, непонятная, она может быть скорректирована. Так как данный телеком проект является старым и сложным, и каждый член команды является достаточно узкоспециализированным специалистом, в ходе исследования не представляется целесообразным вводить на данном этапе типичные для скрама методы командной оценки без предварительного глубокого анализа требований. Более эффективным будет оценивание и ассессмент, сделанные следующим образом: познакомившись с бэклогом, члены команды, основываясь на своих компетенциях, разбирают требования (далее топики) на подробный анализ и оценку. Таким образом над каждым топиком работает

один дизайнер, один разработчик и один тестировщик. Дизайнеру на данным этапе необходимо проанализировать требования, обратиться к владельцу продукта за подробностями, в случае возникновения вопросов, и определить список дизайн документов, которые необходимо поправить (в случае внедрения изменений в уже существующий функционал). Далее дизайнер организовывает review встречу с разработчиком и тестировщиком, на которой он проводит презентацию изменений. Это поможет им сделать более детальный анализ предстоящей работы и на раннем этапе вычленить и устранить возможные недочёты и проблемы со стороны дизайна. Далее, разработчик делает детальный анализ существующего кода и описывает предполагаемые изменения (это касается обеих команд, так как даже абсолютно новые требования скорее всего будут подразумевать имплементацию с использованием уже написанного кода). Специалист по тестированию, в свою очередь, составляет список тест кейсов, которые должны полностью покрыть функциональное и регрессионное тестирование. Далее специалисты проставляют на пользовательской истории свою оценку на задачах (желательно, чтобы она была относительной). Чтобы сделать оценку относительной, команда, например, может взять усредненную пользовательскую историю и оценить её в один стори поинт. Далее каждый член команды сможет делать оценку относительно данной пользовательской истории.

Спринт 1 начинается с Planning встречи, на которой, непосредственно, решается, какие именно задачи будут взяты в первый спринт, итогом которого является промежуточная версия продукта, выдаваемая на E2E тестовое окружение заказчику. Подразумевается, что члены команды разберут задачи, основываясь на том, кто именно занимался их оценкой. Оценка, данная группой, которая занималась анализом требований, может быть скорректирована командой в ходе обсуждений. Как только задачи разобраны, начинается этап параллельного написания дизайна, кода и детальных тест кейсов. Ранее написание тест кейсов позволяет решить две

проблемы: устранить возможные погрешности и недочёты дизайна, а также минимизировать количество дефектов (т.е. разработчик видит, какие сценарии должен покрыть его код). Кроме того, в рамках исследования предлагается также введение написания автотестов (так как создать полное тестовое покрытие зачастую невозможно, предлагается сделать покрытие основных бизнес сценариев, что, как правило, составляет около 30-40 процентов основных тестовых сценариев), что позволит улучшить качество кода, так как тестирование, по факту будет проводится, два раза: сначала специалистами отдела качества, затем автотестами. Также в данной работе предлагается введение элемента парного программирование, то есть проверка кода, покрывающего реализацию основных бизнес сценариев пользовательской истории коллегой-разработчиком.

В конце первого спринта команде также выделяется некоторое количество времени на анализ и оценку новых требований к первому релизу. Следует обратить внимание, что во втором спринте разработки первого релиза, команда работает над доработкой выданного в первом спринте функционала, фиксом дефектов с окружения заказчика и действительно срочными новыми изменениями. Это означает, что команда не должна брать во второй спринт новые времязатратные и сложные задачи, так как это является риском для будущего релиза.

В начале спринта 2 также проходит планирование задач на ближайшие 2 недели. Как правило, в эти две недели команда доводит функционал до совершенства, занимается поддержкой E2E тестирования и устранения дефектов, найденных на этом этапе, а также анализирует и оценивает бэклог будущих работ для следующего релиза. Кроме того, во вторую неделю данного спринта происходит выдача релизной сборки, а также её установка на окружение заказчика (выходные).

Спринт 3 также начинается с планирования. Затем команда повторяет действия, совершенные в 1 спринте. Далее команды продолжают работать по заданному принципу с постепенным увеличением скорости.

На рисунке 3.7 представлен возможный процесс разработки одной пользовательской истории.

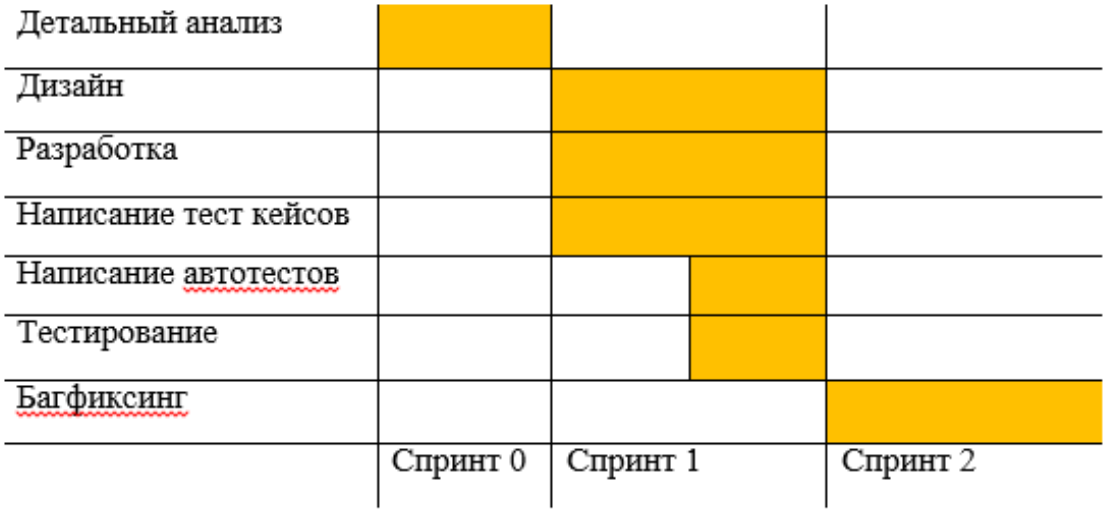

Рисунок 3.7 – Процесс разработки пользовательской истории

Также для решения проблем, связанных со слабой коммуникацией между членами команды и заказчиками, а также получения обратной связи, в данной работе предлагается ввести типичные скрам митинги. Во-первых, это Standup meeting, проходящий каждый день в одно и то же время и длящийся не больше 15 минут. Также рекомендуется ввести Ретроспективу, на которой будет обсуждаться прошедший спринт и проблемы, возникшие на нем, а также приниматься решения по устранению данных проблем в будущем. Рекомендованное время проведения данного митинга – первый день нового спринта (чаще всего команды, работающие с ретроспективами проводят данные встречи в последний день спринта, однако, в рамках данного проекта такое решение было бы нецелесообразным в связи с тем, что в последний день спринта происходит выдача продукта на E2E сервера или Prod окружение заказчика, а значит команда не сможет сконцентрироваться на ретроспективе). Кроме того, в работе предлагается введение обязательной демонстрационной встречи, на которой команда сможет показывать реализованные эпики заказчику. Рекомендуемое время проведения такой

встречи – предпоследний день спринта. Наконец, рекомендуется введение межкомандной встречи (назовем ее Common status meeting), на которой обе команды будут вкратце представлять реализованные эпики, делиться успешными метриками и процессами, а также обмениваться важными идеями. Подобные встречи позволят командам улучшать внутренние процессы, основываясь на своем и чужом опыте, а также позволит обеим командам иметь полное видение того, что выдается в каждом релизе. Кроме того, на данной встрече будет происходить предварительное демо, которое затем будет продемонстрировано заказчику и конечному пользователю. Таким образом, команды, во-первых, будут в курсе всех разработок релиза, во-вторых, смогут подготовиться к демо и увидеть недостатки реализации. Рекомендуемое время проведения – 2 день второй недели второго спринта релиза.

Последним из шагов, предложенных в пункте 2.3, предлагается непрерывная мотивация и обучение команды. С этой целью в данной работе предлагается освободить один день в релиз для членов команды с целью прохождения необходимых тренингов, доделывания важных задач, на которые не хватает времени.

На рисунке 3.8 представлена модель внедрения командных встреч в спринты, в рамках которой предлагается:

- сделать процесс работы максимально прозрачным, внедрив standup meeting, planning meeting, backlog refinement meeting и др.;
- укрепить культуру общения внутри команды для получения обратной связи и обеспечения совершенствования процесса, внедрив ретроспективу, стендап, демо встречу;
- организовать общение между командами, внедрив общекомандную встречу;
- мотивировать команду, внедрив один свободный день в спринт, когда каждый член команды может заниматься теми задачами, какими он хочет.

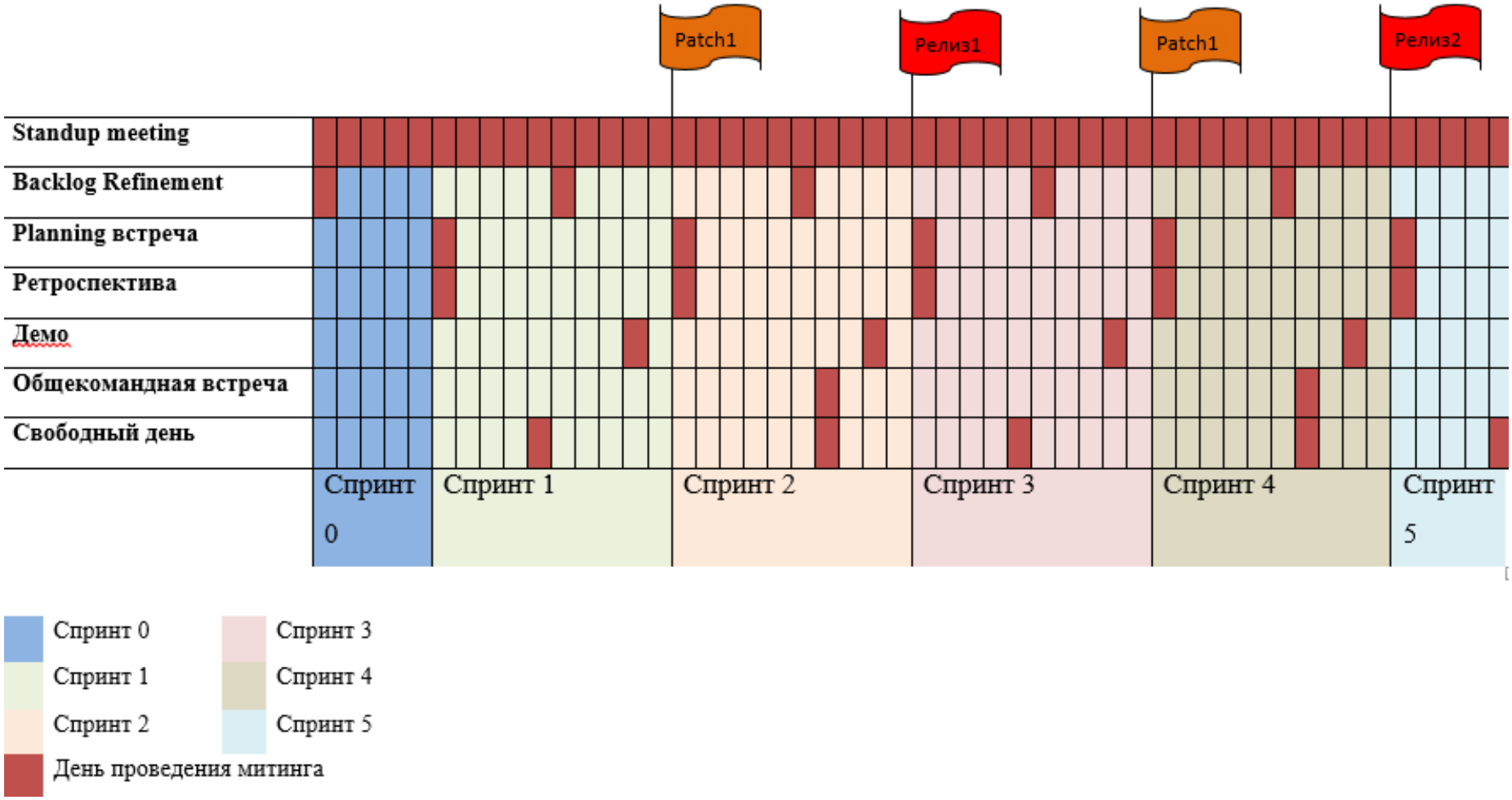

Рисунок 3.8 – Схема внедрения командных встреч в этапы разработки

Для того, чтобы отслеживать статусы задач, скорость команды и качество работ предлагается внедрение нескольких артефактов и метрик.

Во-первых, для отслеживания статуса работ командами может использоваться представленная на рисунке 3.9 модернизированная иерархия задач в уже используемой командами Jira.

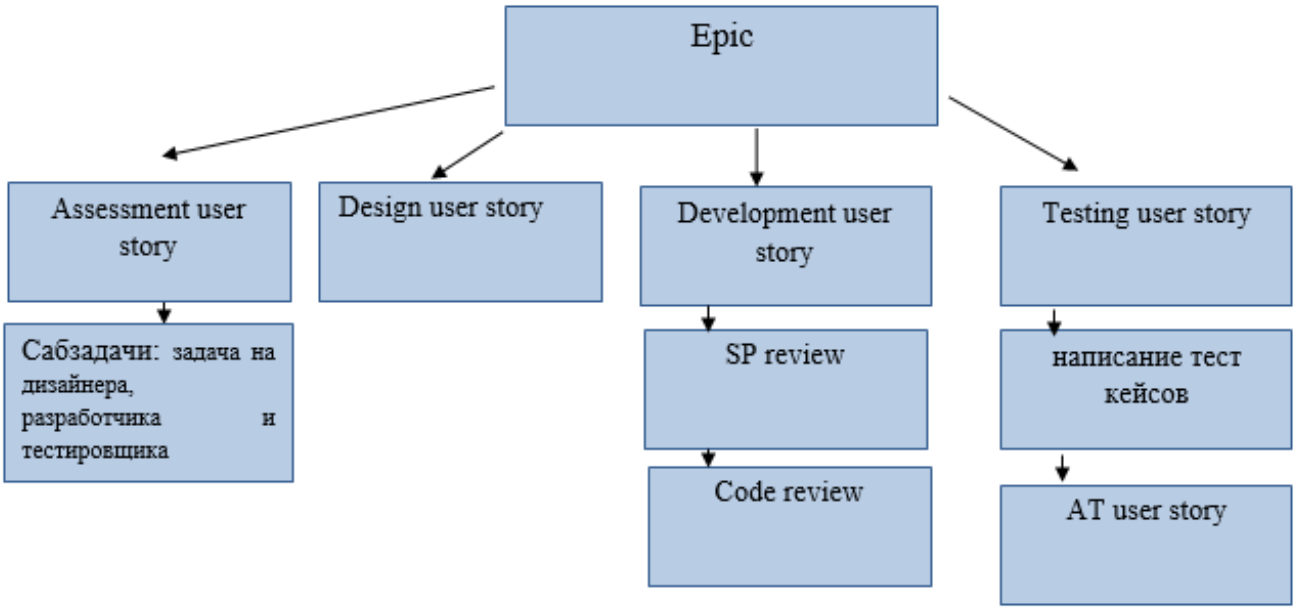

Рисунок 3.9 – Иерархия пользовательских историй в Jira

В предложенной выше схеме эпиком является часть функционала, которую необходимо выдать в релизе и которая обладает свойством завершенности, тогда как пользовательской историей является задача для отдельного члена команды, занимающегося работой над этим эпиком. Такая концепция несколько отличается от типичной концепции разделения бэклога на пользовательские истории, так как пользовательская история, как правило, представляет собой завершенную часть продукта, то есть то, что выше я называю эпиком. В тоже самое время, предлагаемые в данной работе пользовательские истории обычно являются подзадачами. В ходе проектирования модели предлагается именно такое разделение, во-первых, в связи с тем, что такое разделение позволит брать задачи на оценку и дизайн в

один спринт, а задачи на разработку и тест в другой спринт, так как не всегда возможно сделать все из вышеперечисленного в рамках одного спринта. А также для того, чтобы добиться наибольшей прозрачности процесса. Так как функционал может параллельно находится на стадии дизайна, разработки и тестирования, необходимо, чтобы у каждого члена команды была соответствующая задача на доске задач для того, чтобы команда понимала, на какой стадии развития сейчас находится тот или иной функционал и для того, чтобы скрам мастер понимал, чем именно занимается каждый член команды.

Также для обеспечения прозрачности процесса каждой из команд в полном составе необходимо каждое утро встречаться на утренней летучке для обсуждения статуса работ.

Статус	BA	DEV	QA
бэклог	Epic 5. Design US	Epic 6. DEV US Epic 7. DEV US	
Анализ\оценка	Epic 1. Assessment US. Design subtask	Epic 1. Assessment US. DEV subtask	Epic 1. Assessment US. QA subtask
Дизайн	Epic 2. Design US		
Имплементация		Epic 2. DEV US	
Dev test\code review		Epic 8. DEV US	
Ready for testing			Epic 3. QA US
Написание тест кейсов			Epic 2. QA US
тестирование			Epic 4. QA US
Завершено	Epic 4. Design US	Epic 4. DEV US	

Рисунок 3.10 – Статусная доска задач

Для наглядности, скрам мастеру необходимо создать при помощи JIRA инструментов спринт и бэклог спринта, а также доску со статусами задач, пример которой предложен на рисунке 3.10

Кроме того, для обеспечения прозрачности процесса командам необходимо внедрить практику definition of done, которая позволит им понимать, в какой момент пользовательская история может считаться завершенной.

Так, для assessment user story Definitions of Done могут выглядеть следующим образом:

- review проведено;
- риски учтены;
- DEV оценка выдана;
- DEV assessment сделан;
- QA оценка выдана;
- список автотестов и тест кейсов определен;
- BA оценка предоставлена;
- список дизайн документов на переработку готов.

Для QA user story:

- список тест кейсов с подробными шагами выполнен;
- список тест кейсов согласован с дизайнером и разработчиком;
- тест дата готова;
- тесты выполнены;
- критичных дефектов нет;
- автотесты написаны;
- автотесты проверены.

Для DEV user story:

- имплементация сделана;
- дев тест сделан;
- основные сценарии проверены коллегой-разработчиком;

- код выдан в основную ветку;
- SP review пройдено;
- код ревью пройдено.

Для BA user story:

- детализированный дизайн написан;
- дизайн проверен разработчиком и тестировщиком;
- открытых вопросов к дизайну не осталось.

Для того, чтобы определить, насколько быстро команда справилась с поставленными задачами и понять, сколько пользовательских историй команда может взять в следующем спринте, в работе предлагается следующий подход: в конце спринта во время Ретроспективы команда сопоставляет количество реального времени, потраченного на пользовательские истории и общую оценку, данную им на этапе планирования. В том случае, если оценка сильно отличается в большую или меньшую сторону, команде необходимо провести анализ и понять причину. Также суммарная оценка всех реализованных пользовательских историй и прод дефектов может являться объемом, который команда может запланировать на следующий спринт. Т.е. именно от этого числа команда может отталкиваться на следующем планировании. Однако, необходимо учитывать, что данное число необходимо умножать на capacity коэффициент, то есть на процент, показывающий, сколько дней суммарно команда сможет потратить на разработку (из общего количества дней вычитываются праздники, отпуска, болезни и т.д.).

Таким образом, можно вывести следующую формулу расчета объема работ на планируемый спринт:

N/K = реализованный объем работ с прошлого спринта/(((кол-во человек в команде) x (все дни спринта – дни, когда команда не работает)/ (кол-во человек в команде) x (все дни спринта))).

Данное нововведение позволяет рационально использовать ресурсы команды.

На рисунке 3.11 представлен процесс планирования capacity команды.

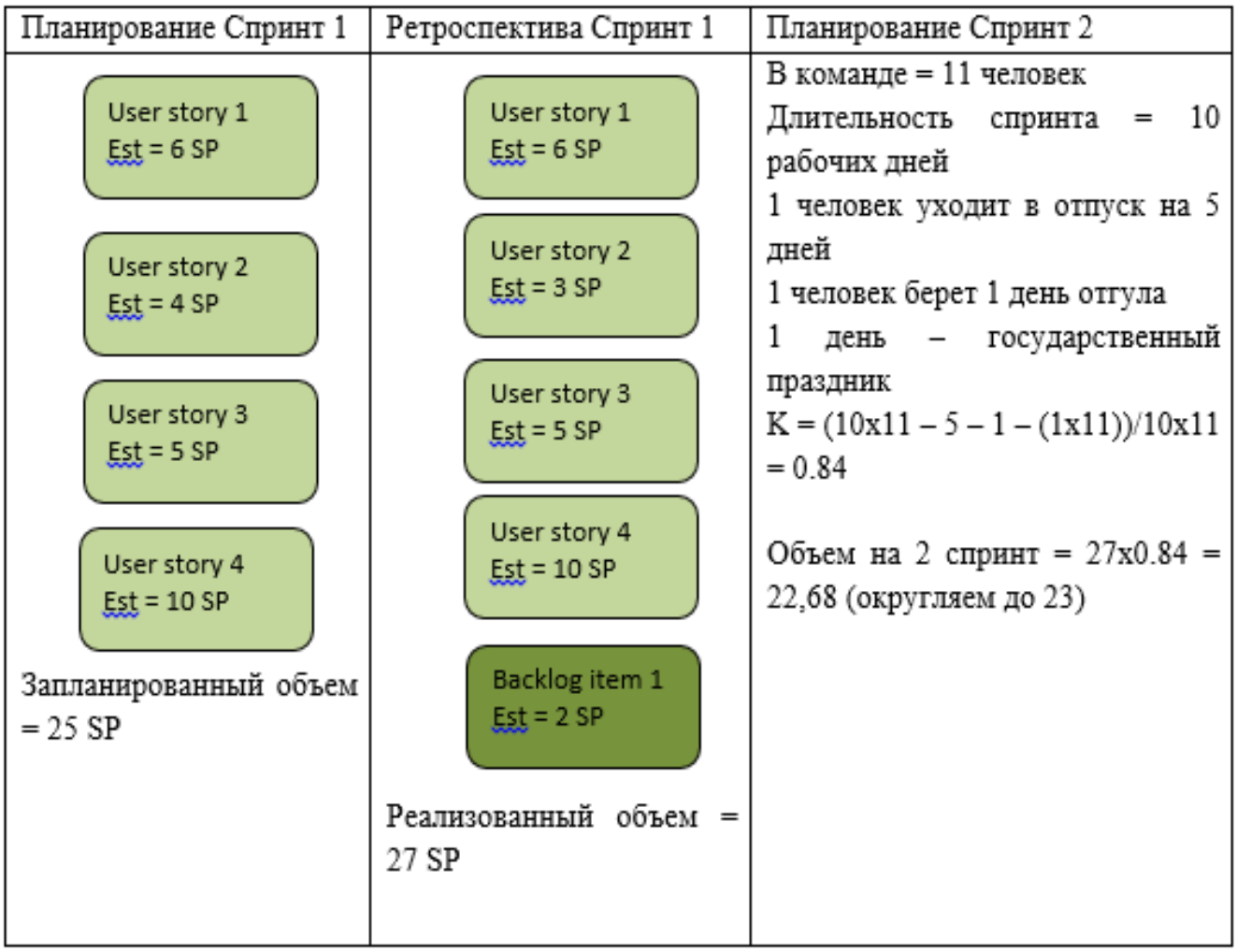

Рисунок 3.11 – планирование capacity команды

Также отличной метрикой отслеживания скорости выполнения задач является диаграмма сгорания задач, то есть метрика, которая помогает отслеживать ход реализации задач спринта и спрогнозировать, успеет ли команда выполнить все запланированные работы к концу сприта или нет. В том случае, если команда видит, что она набирает недостаточную для реализации всех запланированных задач скорость, она может попытаться понять причину проблем и постараться их устранить, или договориться с владельцем продукта об удалении некоторых задач из бэклога текущего

спринта. Если же скорость команды наоборот высока, команда может предложить владельцу продукта добавить задачи в спринт. Подобная метрика позволяет отследить, насколько верно команда планирует задачи и оценивает свою скорость.

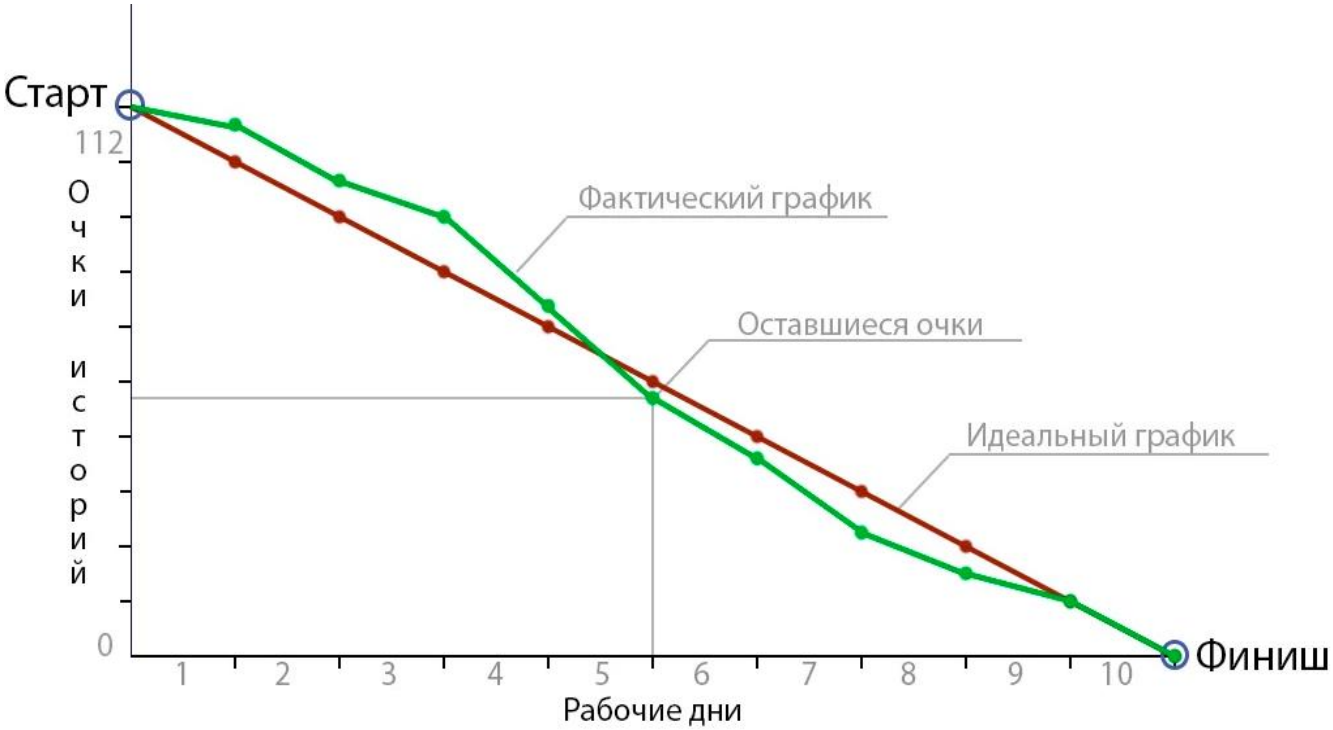

Рисунок 3.12 – Диаграмма сгорания задач [17]

Также данная диаграмма, представленная на рисунке 3.12 является отличным показателем точности планирования.

3.3 Описание предложенной модели

Разработанная на основе шагов, представленных в пункте 2.3, модель подходит для проектов, поставляющих ПО для телекоммуникационных операторов и отвечающих следующим условиям:

- есть четкие временные рамки выдачи релиза;
- заказчик готов интегрироваться в процесс;

- команда опытная и хорошо знает функционал проекта;
- члены команды обладают необходимыми soft skills, готовы к активному общению, обмену мнениями, изменениям в привычных процессах, ответственности;
- требования к задачам могут быть частично изменены во время хода разработки;
- проект долгосрочный;
- проектная команда составляет не менее 20 человек.

Разработанная модель может быть описана по трем критериям: роли, процессы и артефакты. В таблице 3.3 представлено описание предложенной модели.

Таблица 3.3 – Описание разработанной модели

Роли	Артефакты	Процессы
команда разделена на две более маленькие команды (релизная команда и команда, поставляющая новый функционал, 11 и 9 человек соответственно)	введено понятие двухнедельного спринта	оценка и анализ пользовательских историй производится не командно, а детально в тройках BA–DEV-QA
в каждой из команд появились новые роли – скрам мастер и владелец продукта, которыми являются представители заказчика	введено понятие бэклога спринта	новые этапы разработки разложены на работу в рамках спринтов
старые роли, такие как dev lead и qa lead упразднены	введено новое иерархическое разделение задач в Jira: основная задача – эпик, разделяется на подзадачи для оценки, написания дизайна, реализации и тестирования – пользовательские истории, которые в свою очередь разделяются на подзадачи	введена ретроспективная и common status встречи для обсуждения процессов в команде и на проекте и предложения идей для их улучшения
	предложено введение статусной доски задач со следующими статусами:	введена утренняя статусная встреча

Роли	Артефакты	Процессы
	бэклог, анализ\оценка, дизайн, имплементация, dev test\code review, ready for testing, написание тест кейсов, тестирование, сделано	
	введена формула расчета объема задач на спринт	введена Backlog Refinement встреча, на которой команда узнает все требования к релизу
	предложено введение диаграммы сгорания задач	введена Planning встреча, на которой команда планирует будущий спринт с использованием формулы расчета объема работ
	Предложены definitions of done для следующих пользовательских историй: оценка, дизайн, разработка, тест	предложено решение по улучшению качества продукта, благодаря раннему написанию тест кейсов, обеспечивающих разработчику картину ожидаемого результата для тестирования, а дизайнеру раннюю проверку дизайна
		предложено введение автоматизации тестирования – написание автотестов, покрывающих основные бизнес сценарии (30-40 процентов), тестируемые в рамках пользовательской истории
		введена демонстрационная встреча для получения обратной связи от заказчика и ознакомления всех членов команды с реализуемым функционалом
		предложены элементы парного программирования

На рисунке 3.13 представлена общая схема предлагаемой модели.

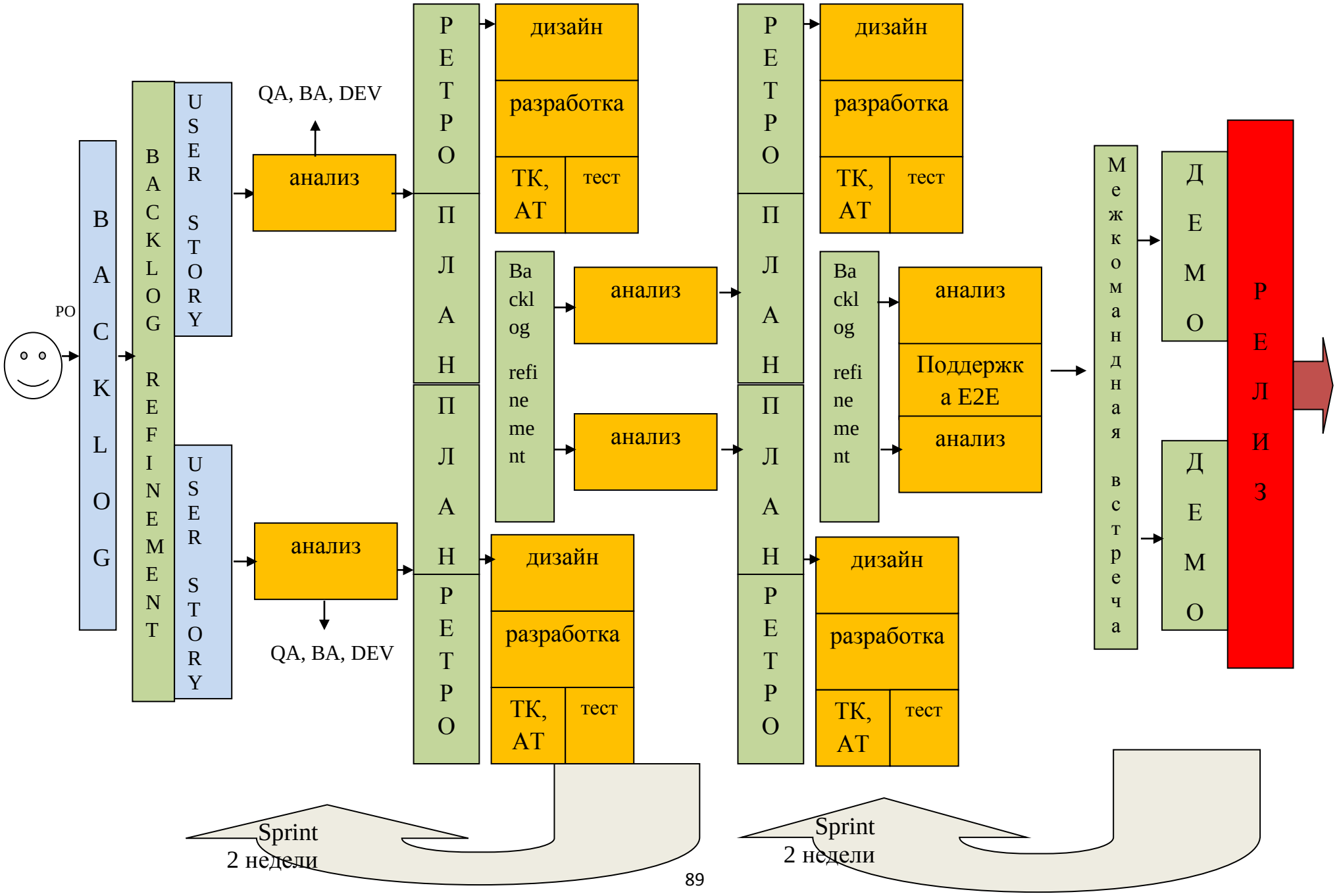
PO
BACKLOG
BACKLOG REFINEMENT
USER STORY
USER STORY
QA, BA, DEV
анализ
QA, BA, DEV
анализ
РЕТРО
ПЛАН
ПЛАН
РЕТРО
дизайн
разработка
ТК, АТ
тест
Backlog refinement
анализ
анализ
РЕТРО
ПЛАН
ПЛАН
РЕТРО
дизайн
разработка
ТК, АТ
тест
Backlog refinement
анализ
Поддержка E2E
анализ
Межкомандная встреча
ДЕМО
ДЕМО
РЕЛИЗ
Sprint 2 недели
Sprint 2 недели

Рисунок 3.13 – Гибкая модель проектирования ПО для телекоммуникационного оператора

Предложенная модель основана на традиционной модели Scrum и работающих практиках других гибких моделей, и имеет ряд особенностей, связанных со спецификой телекоммуникационных проектов и представленных в таблице 3.4.

Таблица 3.4 – Отличия разработанной модели от скрам модели

Критерий для сравнения	Scrum	Предложенная модель
Длина цикла поставки продукта на окружение заказчика	1 спринт	3 спринта
Оценка и анализ пользовательских историй	Командная	В тройках BA-QA-DEV
Поставка	Рабочая часть функционала	Полностью готовый функционал = релиз
Роль заказчика	Участвует в демо митингах, может участвовать в расстановке приоритетов	Является полноправным членом команды, занимается приоритезацией бэклога, участвует во всех командных встречах
Качество	Ручное тестирование, автотестирование	Предлагается ручное тестирование, автотестировние, элементы парного программирования, написание тест кейсов производится на этапе анализа и оценки, согласуется с BA и DEV
Командные встречи	Standup встреча, ретроспектива, демо встреча, планирование	Помимо традиционных скрам встреч предлагается общекомандная встреча для обсуждения рабочих практик и идей Ретроспектива проводится в первый день нового спринта (т.к. в последний день спринта происходит выдача функционала)

Преимущества модели:

- хорошая применяемость для крупных и сложных проектов (телекоммуникационных проектов);
- подразумевает интеграцию заказчика в рабочий процесс;
- направлена на повышение качества продукта;
- направлена на повышение скорости реализации функционала;
- направлена на развитие компетенций членов команды;
- направлена на улучшение коммуникации внутри команды;
- позволяет часто поставлять готовый инкремент продукта;
- направлена на улучшение прозрачности процессов;
- достаточно проста в интеграции.

Недостатки модели:

- требует достаточно высокого уровня профессионализма членов команды;
- подходит для людей с хорошо развитыми коммуникативными навыками;
- применима для проектов, состоящих из достаточно небольших команд.

3.4 Анализ эффективности разработанной модели

Оценка эффективности модели проведена на основе статистического анализа данных об объёмах работ на проекте, занимающимся разработкой ПО для телекоммуникационного оператора, с использованием каскадной модели и модели, предложенной в данной работе. Также в ходе исследования и апробирования модели был проведен опрос у членов команды с целью выяснения насколько более прозрачным стал процесс разработки на проекте.

Анализ объема работ. Одной из лучших метрик эффективности моделей является сравнение количества выполненных работ, а также скорости команды. Все данные для таблицы 3.5 представлены в стори-

поинтах. Из статистических данных были взяты данные за последний релиз, разработанный с использованием каскадной модели и 1,5 релиза, разработанного с использованием новой модели.

Таблица 3.5 – Количество реализованных стори-поинтов

Этап	Планировалось	Выполнено	%
Релиз X	930	895	96%
Спринт 1	200	187	93%
Спринт 2	185	184	99%
Спринт 3	190	192	101%

Из таблицы 3.5 видно, что количество реализованного функционала растет с использованием предложенной модели, к 3 спринту количество реализованного функционала даже превысило количество запланированной работы на 2 стори-поинта. Также можно обратить внимание на скорость команды, представленную на рисунке 3.14. Если в рамках 12 недельного релиза X она составляла 895\12 = 74,5 стори поинта в неделю, то в рамках 1 спринта она составляла 93,5, в рамках 2 спринта 92,5, в рамках 3 спринта 96.

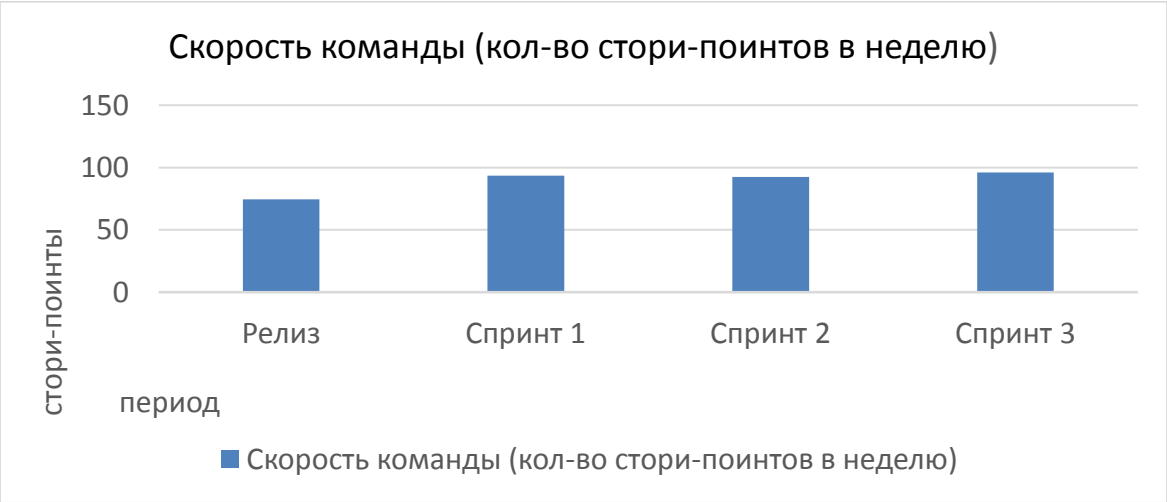

Рисунок 3.14 – Скорость команды

Таким образом, можно сделать вывод, что с предложенной моделью объем работ и скорость команды возрасли.

Прозрачность процессов. Одним из основных принципов предложенной в данной работе модели является улучшение прозрачности процессов разработки на проекте с использованием новой модели. С целью выяснения насколько команда удовлетворена прозрачностью процессов, был проведен опрос, состоящий из 2 вопросов:

1. Считаете ли Вы, что командные встречи способствуют улучшению коммуникации внутри команды и, непосредственно, с заказчиками?

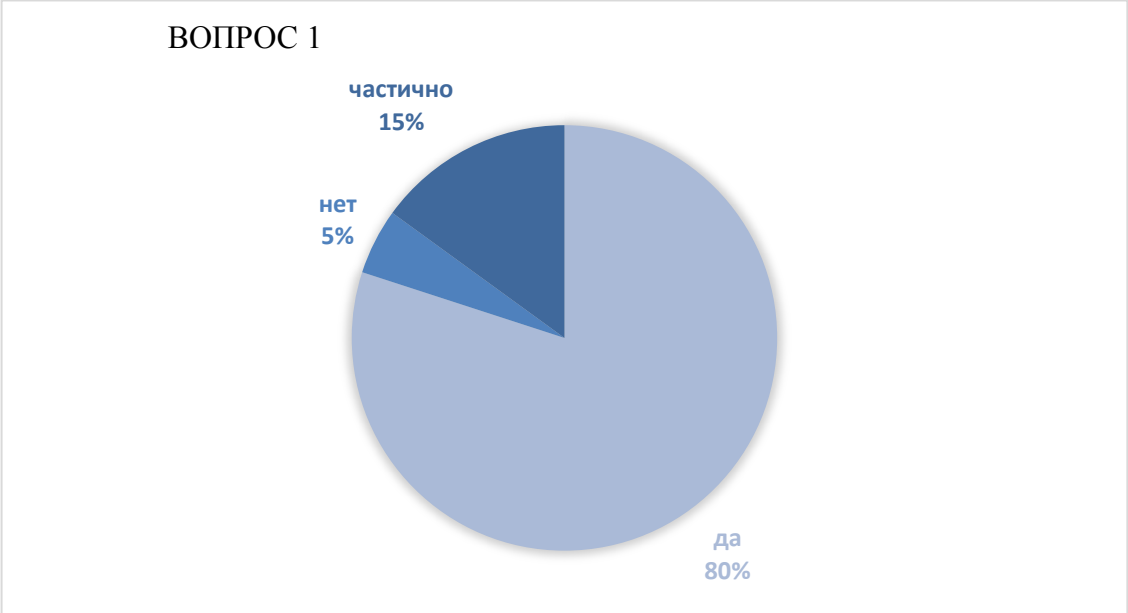

Рисунок 3.16 – Диаграмма, отображающая результаты опроса относительно улучшения коммуникации

2. Способствуют ли командные встречи и использование статусной доски улучшению понимания, какие задачи разрабатывает Ваша команда, чем занимаются члены Вашей команды и каков статус задач, взятых в спринт?

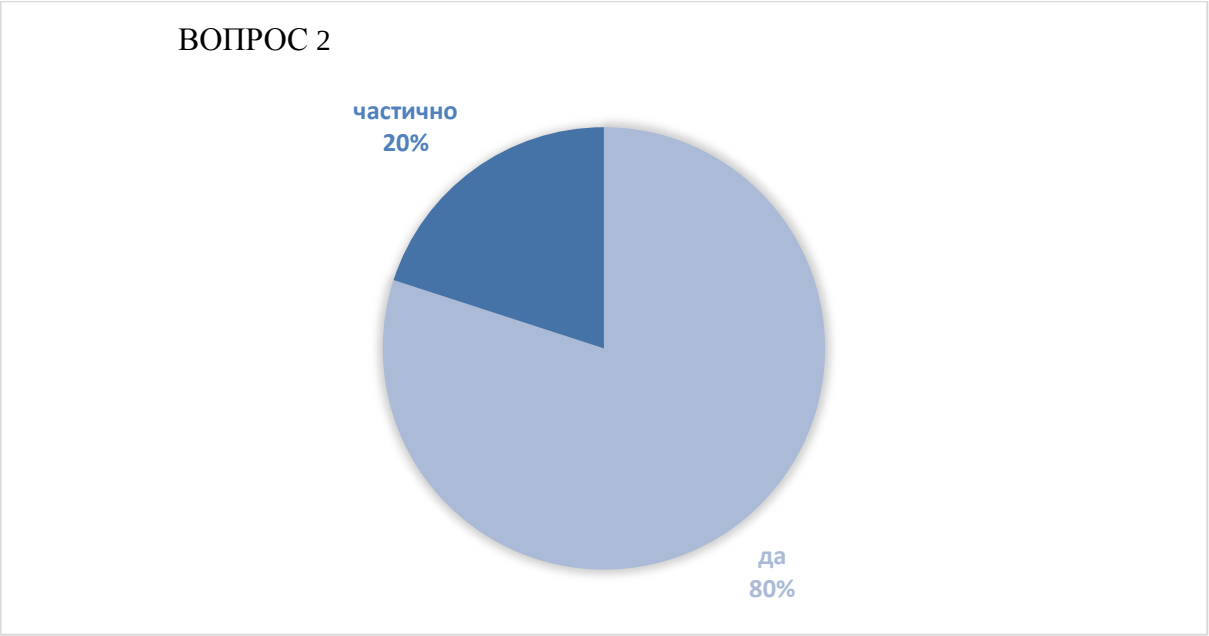

Рисунок 3.17 – Диаграмма, отображающая результаты опроса относительно прозрачности процессов

Опрос показал, что преобладающее большинство (около 80% членов команды) считают, что введение новой методологии позволило сделать процессы внутри проекта более прозрачными и улучшить коммуникацию как внутри команды, так и непосредственно с заказчиком.

Выводы к третьей главе

Каскадная модель проектирования ПО для телекоммуникационного оператора обладает рядом недостатков, связанных с недостаточной прозрачностью процессов, нехваткой коммуникации, нерациональным использованием ресурсов и др. Подобные проблемы ведут к тому, что проект становится неконкурентоспособным на рынке.

Предлагаемая модель основана на традиционной модели Scrum и работающих практиках других гибких моделей, и имеет ряд характерных особенностей, связанных со спецификой телекоммуникационных проектов, таких как: интеграция заказчика в работу команды, анализ задач

производится не в командах, а в тройках BA-QA-DEV, QA инженеры занимаются разработкой детальных тест кейсов, которые затем проверяются бизнес аналитиками и разработчиками, на этапе анализа.

Предлагаемая модель позволяет сделать поставку продукта более частой и стабильной, улучшить коммуникацию внутри команды, работать в тесном сотрудничестве с заказчиком, сделать процесс прозрачным, а также увеличивать компетенции членов команды. Это, в свою очередь, позволяет сделать процесс поставки продукта более быстрым и качественным.

Анализ разработанной модели в рамках объема выполненных работ, скорости команды и прозрачности процессов, показал, что она является более эффективной по сравнению с каскадной моделью.

Заключение

Выполненные в работе научные исследования представлены следующими результатами:

1. Произведён анализ литературы по предмету исследования, доказавший актуальность темы данной работы, посвященной проблеме перехода с каскадной модели проектирования ПО для телекоммуникационного оператора на гибкую.

2. Произведён анализ существующих гибких моделей проектирования ПО, определены преимущества и недостатки их использования на телекоммуникационных проектах.

3. Произведён анализ гибких практик и инструментов, определено, какие из них могут быть эффективно использованы на проектах, реализующих программное обеспечение для телекоммуникационного оператора.

4. Предложены шаги для перехода с каскадной на гибкую модель проектирования ПО для телекоммуникационного оператора.

5. Рассмотрена каскадная модель проектирования ПО на примере проекта, поставляющего ПО европейскому телекоммуникационному оператору, определены её слабые места и требования к новой модели, отвечающие текущему состоянию телекоммуникационного рынка.

6. На основе представленных шагов разработана гибкая модель проектирования ПО для телекоммуникационного оператора. Данная модель обеспечивает устранение проблем, связанных с недостаточно частой поставкой работающего продукта, недостаточной прозрачностью процессов и коммуникацией в команде, и предлагающая увеличение скорости команды, качества продукта и объемов выполняемых работ.

7. Произведён анализ эффективности разработанной модели, доказавший гипотезу исследования, что качество и скоростью работ

возрастают с применением предложенной модели при сравнении с каскадной моделью.

Список используемой литературы

1. Agile-менеджмент: Лидерство и управление командами / Юрген Аппело. – Москва : Альпина Паблишер, 2018. – 524с.

2. Брукс Ф. Мифический человекомесяц или как создаются Программные системы / Ф. Брукс. – Санкт-Петербург: Символ-Плюс, 1999. – 304с.

3. Бугакова К.В. Условия, влияющие на выбор методологии разработки программного обеспечения перед стартом проекта // Материалы научно-практической конференции (школы-семинара) молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук». – Тольятти: ТГУ, 2019. – С. 23-25.

4. Воробович, Н.П., Лопатеева О.Н. Анализ методологий разработки автоматизированных информационных систем [Электронный ресурс] // URL: https://cyberleninka.ru/article/v/analiz-metodologiy-razrabotki-avtomatizirovannyh-informatsionnyh-sistem (дата обращения 03.03.2019).

5. Ганноченко, А. Результатники и процессники. Результаты, создаваемые сотрудниками [Электронный ресурс] // URL: https://bookmate.com/books/dk7UCdCi (дата обращения 04.03.2019).

6. Грекул В.И., Денищенко Г.Н., Коровкина Н.Л. Проектирование информационных систем. — М.: Интернет-университет информационных технологий [Электронный ресурс] // URL: https://www.intuit.ru/studies/courses/646/502/info (дата обращения 04.03.2019).

7. Гугаев, К.В. Эволюция методологий разработки ПО [Электронный ресурс] // URL: https://cyberleninka.ru/article/v/evolyutsiya-metodologiy-razrabotki-po (дата обращения 04.03.2019).

8. Дж. Грин, Стеллман: Постигая Agile. Ценности, принципы, методологии [Электронный ресурс] // URL:

https://f.ua/statik/files/products/515942/postigaja-agile-cennosti-principiy-metodologii_8580.pdf (дата обращения 04.03.2019).

9. Избачков С.Ю Информационные системы // С.Ю. Избачков, Петров В.Н. – СПб.: Питер, 2008. – 655 с.

10. ИНТУИТ. Интернет университет информационных технологий [Электронный ресурс] // URL: http://intuit.ru (дата обращения 04.03.2019).

11. Каримов, Р.А., Качкынбеков Н.Р. Некоторые аспекты гибкой методологии разработки программного обеспечения [Электронный ресурс] // URL: https://cyberleninka.ru/article/v/nekotorye-aspekty-gibkoy-metodologii-razrabotki-programmnogo-obespecheniya (дата обращения 18.03.2019).

12. Ларман К. Итеративная и инкрементальная разработка: краткая история [Электронный ресурс] // URL: https://www.osp.ru/os/2003/09/183412/ (дата обращения 02.03.2019).

13. Макконнелл С. Совершенный код. Практическое руководство по разработке программного обеспечения / С. Макконнелл. – СПб.: Русская Редакция, Питер, 2005. – 896 с.

14. Орлик С. Введение в программную инженерию и управление жизненным циклом ПО [Электронный ресурс] // URL: http://software-testing.ru/files/se/4-software_lifecycle_models.pdf (дата обращения 02.03.2019).

15. Расмуссон Д. Гибкое управление IT-проектами. Руководство для настоящих самураев. – СПб.: Питер, 2012. – 340 с.

16. Сазерленд Д., Scrum. Революционный метод управления проектами / Джефф Сазерленд ; пер. с англ. М. Гескиной. — М.: Манн, Иванов и Фербер, 2019. – 288 с.

17. Хабр - ресурс для IT-специалистов [Электронный ресурс] // URL: https://habr.com/hub/api/ (дата обращения 02.03.2019).

18. Шопырин Д.Г. Управление проектами разработки ПО [Электронный ресурс] // URL: https://books.ifmo.ru/file/pdf/422.pdf (дата обращения 02.03.2019).

19. Agile Manifesto [Электронный ресурс] // URL: https://agilemanifesto.org/ (дата обращения 03.02.2019).

20. Agile Software Development with Scrum (Series in Agile Software Development) // Ken Schwaber, Mike Beedle. – 2001.

21. Brooks F. No Silver Bullet: Essence and Accidents of Software Engineering. Computer. –1987. vol. 4. pp. 10–19.

22. Ken Schwaber. Scrum Development Process // *D. Patel*, *C.* Casanave, G. Hollowell, J. Miller. Business Object Design and Implementation: OOPSLA'95 Workshop Proceedings / Eds. J. Sutherland, D. Patel. London: Springer. – 1997.

23. Lisa Crispin, Janet Gregory. Agile Testing: A Practical Guide for Testers and Agile Teams. Boston: Addison-Wesley Professional; 1 edition. – 2009. – 576 p.

24. Mike Cohn. Agile Estimation and Planning. Upper Saddle River, NJ: Prentice Hall. – 2005. - 360 p.

25. Rob Cole, Edward Scotcher. Brilliant Agile Project Management. A practical guide to using Agile, Scrum and Kanban. Lexray Limited and Agility in Mind Limited. NJ: FT Press; 1 edition. – 2015. – 200 p.

26. Royce, Winston Managing the Development of Large Software Systems [Электронный ресурс] // URL: http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf (дата обращения 03.02.2019).

27. Scott W. Ambler, Agile Modeling: Effective Practices for Extreme Programming. NJ: Wiley – 2002. – 404 p.

28. SkillsUp [Электронный ресурс] // URL: https://skillsup.ua/about/blog/2019/06/about-scrum/ (дата обращения 12.12.2019).

29. State of Agile [Электронный ресурс] // URL: (https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report) (дата обращения 02.02.2020).

30. Tripathi N. et al. Scaling Kanban for software development in a multisite organization: challenges and potential solutions //International Conference on Agile Software Development. – Springer, Cham, 2015. – P. 178-190.

31. D. Turk, R. France, B. Rumpe. Assumptions Underlying Agile Software Development Processes.. In: Journal of Database Management, Volume 16, No. 4, pp. 62-87, October-December 2005 Idea Group Inc., 2005.

32. D. Turk, R. France, B. Rumpe. Limitations of Agile Software Processes. In: Third International Conference on Extreme Programming and Flexible Processes in Software Engineering, XP2002, May 26-30, Alghero, Italy, pg. 43-46, 2002.

33. Vijayasarathy L. R., Butler C. W. Choice of software development methodologies: Do organizational, project, and team characteristics matter? //IEEE Software. – 2016. – Т. 33. – №. 5. – P. 86-94