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 кафедры)

09.04.03 Прикладная информатика
(код и наименование направления подготовки)

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

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему «Модели и инструменты оценки трудоемкости разработки программного обеспечения корпоративных информационных систем»

Студент А.С. Дуденко _____
(И.О. Фамилия) (личная подпись)

Научный руководитель Е.В. Панюкова _____
(И.О. Фамилия) (личная подпись)

Руководитель программы д.т.н., доцент С.В. Мкртычев _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»_____20_____г.

Допустить к защите

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»_____20_____г.

Тольятти 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Почти в любом бизнес-процессе, люди, отвечающие за него, сталкиваются с задачами его технико-экономического обоснования. Такие же задачи возникают и перед участниками разработки крупных программных систем. Одним из критериев успешного проведения и сдачи ПП заказчику невозможно без точного и быстрого способа оценки времени ресурсов, требуемых для его разработки. Для этого требуются некоторые инструменты, способные получать качественные и предсказуемые во времени результаты по оценке трудоемкости разработки ПП. На сегодняшний день не существует простых, точных и при этом универсальных методов оценки трудоемкости разработки ПП, которые бы смогли точно оценить размер ПП на этапах планирования работ. В связи с этим, на практике часто используются разнообразные периметрические подходы, содержащие в себе простые в использовании показатели, а также модели с более сложными показателями, которые отображают реальную картину.

Оценка стоимости ПП остается одной из важнейших и непростых задач. В связи с тем, что на текущий момент ПП выросли в своих размерах и сфера их применений также продолжает расти, необходимость в точной оценке разработки их стоимости также растет. Проблема оценки трудоемкости разработки ПО оставалась открытой порядка 10 лет, и только с начала 50х годов, разработчики программного обеспечения совместно со специалистами в смежных областях принялись за обсуждение и разработку первых методологий оценки трудозатрат разработки ПО. Первые модели оценки стоимости разработки ПО появились в технической литературе только спустя 30-40 лет. Тем не менее, проблема оценки стоимости программного обеспечения является не до конца решенной, оставаясь в зачаточном состоянии.

Последние 5-10 лет видна тенденция ухода от классической каскадной методологии разработки ПО к более гибким методологиям, которые предлагает «спиральная» модель разработки ПО. Существующие методы оценки трудозатрат на разработку ПО эффективно используются для разработки при классической методологии, однако появление новых подходов к разработке ПО

привело к тому, что существующие модели оценки либо вовсе неприменимы, либо приводят к неточным оценкам трудозатрат разработки ПО. Одними из популярнейших на текущий момент в разработке ПО являются методология «Scrum», которая основана на совершенно иных концепциях и подходах к процессу разработки ПО, факторы которых нужно учитывать при разработке новой методологии разработки ПО:

– дробление разработчиков на небольшие команды, что благоприятно сказывается на хорошем уровне коммуникации между ними;

– короткие итерации разработки ПО и быстрая поставка новых релизов заказчику;

– ориентированность на частые изменения требований к ПО.

Для использования классических методологий оценки трудоёмкости разработки ПО требуется знать четкие требования к будущему продукту, которые определены на длительный период разработки ПО. Данный факт не дает применить эти методологии для оценки трудоемкости при использовании методологии разработки «Scrum», т.к. изменение требований к ПО – являются важнейшей концепцией всех Scrum подходов к его разработке. В связи с этим оценка трудозатрат на разработку ПО усложняется, но остается одной из важнейших задач.

Таким образом, разработка новых научно обоснованных методик и инструментов оценки трудоемкости ПП, основываясь на тенденции изменении подходов в разработке ПО, является актуальной научно-технической проблемой, имеющей существенное значение как для бизнеса, так и для экономики всей страны в целом.

Цель и задачи исследования. Целью диссертационной работы является рассмотрение существующих моделей оценки трудозатрат на разработку ПО, оценка возможности их применения для оценивания трудозатрат при разработке реальных программных ПО с применением «Scrum»-методологии.

Объектом исследования являются модели оценки трудозатрат на разработку ПО.

Предметом исследования является метод и инструмент оценки трудоемкости разработки ПП, который разрабатывается с применением «Scrum»-методологии.

Гипотезой исследования является предположение о возможности получения адекватной оценки трудозатрат на разработку ПО, которое разрабатывается с использованием «Scrum» методологии, на основе разработанной автором данного исследования нового метода оценки трудоемкости разработки ПО.

Для достижения цели исследования необходимо решить следующие задачи:

– Исследование методов оценки трудоемкости разработки ПП, а также возможность их применения для формирования предварительных требований к разработке ПП по одной из «Scrum»-методологии.

– Разработка метода оценки трудоемкости разработки ПП при использовании «Scrum»-методологий.

– Проведение эксперимента для подтверждения возможности получения адекватной оценки трудозатрат на разработку ПО, которое разрабатывается с использованием «Scrum» методологии, на основе разработанной автором данного исследования нового метода оценки трудоемкости разработки ПО на реальных данных.

В работе применены аппарат логических операций, используются методы реляционной алгебры, а также присутствуют элементы математической логики.

Научная новизна исследования состоит в том, что в нем предложена модель оценки трудоемкости разработки ПО для модели разработки Scrum, которая набирает все большую популярность в последнее время.

В работе разработана и применена модель оценки общих трудозатрат на разработку КИС. В предложенной модели оценки трудоемкости разработки ПО используется новый подход в:

– В пересмотре общей технологии оценки трудозатрат на разработку КИС, которая на данный момент адаптируема к предметной области, постоянно

меняющимся требованиям заказчика и позволяет работать с любым размером исходных данных на старте проекта.

– В использовании методов, которые могут моделировать использовании программной системы в предметной области, и обеспечивающих возможность предоставить оперативную оценку будущего функционала разрабатываемой КИС.

– Использована формальная реляционная модель, которая позволяет воспроизвести процесс разработки КИС, а также оценить ее характеристики.

– Использованы характеристики и дефекты плана проекта, которые позволяют формализовать проведение анализа разработки будущей КИС.

Практическая значимость работы состоит в разработке и использовании новой модели оценки трудозатрат на работку ПО, при использовании «Scrum»-методологий, способных осуществить снижении/увеличение необходимых затрат на любом этапе разработки КИС, сократить риски срыва сроков проекта, подключать разработчиков меньшей квалификации на более ранних этапах разработки КИС. Таким образом, полученные в диссертации результаты могут иметь конкретную прикладную направленность, связанную с повышением качества планирования процесса производства программных систем, с сокращением сроков создания КИС, а также оценки трудозатрат, которые придется приложить на разработку КИС, а в следствии и оценить стоимость такой разработки.

В введении обосновывается актуальность темы исследования, формулируется цель и задачи диссертационной работы, ее теоретико-методологическая база и научная новизна.

В первой главе («Программный проект и оценка трудоемкости его разработки») даются определения программного проекта, а также проводится классификация проектов по структуре, составу и их типу, с учетом продолжительности, сложности и масштаба проекта, определены основные признаки программного проекта. Также, в первой главе приведены особенности различных методик оценки стоимости программных проектов, применяемых в

различных компаниях на сегодняшний день, занимающихся разработкой крупных ИС. В контексте этих особенностей рассматривается эффективность, и определяются наиболее эффективные методики оценивая.

Первая глава состоит из нескольких разделов, в которых освещены вопросы предметной области, определяются основные требования к методам оценивания и обосновывается необходимость применения новой модели, в которой учтен современный подход к разработке ПО.

Во второй главе даются теоретические основы диссертационного исследования, описывается различие метрик для оценки трудоемкости разработки ПО при использовании классических подходов к разработке и методологии Scrum, описывается предложенная модель для оценки трудоемкости при проектировании и разработки информационных систем.

В третьей, заключительной главе диссертационного исследования приведены результаты использования предлагаемой методики оценки трудоемкости разработки ПО в одном из программных проектов ИТ-предприятия ООО «НетКрэкер», а также производится сравнение прогнозируемых и реальных полученных данных.

Путем демонстрации условного применения разработанной методологии в четвертой главе, автором доказана эффективность использования разработанной модели и адекватность ее оценки.

В заключении исследования сформулированы основные теоретические и практические выводы, сделанные автором диссертационного исследования.

Работа изложена на 97 с. и включает 16 рисунков, 24 таблицы.

ГЛАВА 1 ПРОГРАММНЫЙ ПРОЕКТ И ОЦЕНКА ТРУДОЕМКОСТИ ЕГО РАЗРАБОТКИ

1.1 Основные понятия и данные, использующиеся в процессе оценки трудоемкости разработки программного обеспечения

Программный проект – процесс создания уникального ПО, разработка которого имеет ограничение по времени.

В состав программного проекта входят человеческие ресурсы (разработчики, тестировщики, бизнес-аналитики), а также и необходимые материальные ресурсы.

Время жизни конечного программного продукта может существенно превышать время жизни программного проекта.

К основным признакам программного проекта относят следующие характеристики:

– наличие четких целей, которое определяет техническое задание проекта;

– существует ограничение по времени на этап проектирования;

– ограниченность в ресурсах, финансах, людях, технике, оборудовании, материалах и др.;

– допустимость изменения предметной области, для которой ведется разработка КИС;

– допустимость изменения предметной области, для которой реализуется КИС;

– системность и разграничение.

Программные проекты можно классифицировать по их структуре, составу и типу, с учетом их сложности, длительности, а также масштаба проекта.

Схема, отображающая классификацию программных проектов, приводится на рисунке 1.1.

Человеческие ресурсы являются основным затратами при разработке КИС. Как считают эксперты по управлению проектами, стоимость разработки ПО – эквивалента трудозатратам на его разработку.

Рисунок 1.1 – Схема классификации проектов

Необходимо учитывать также то, что на реальную стоимость разработки программного продукта и его окончательной цены оказывают виляние различные экономические факторы, которые прямым образом не относятся к программной инженерии. Примером таких фактов могут являться: ставка конкретного разработчика, общая экономическая ситуация на рынке и т.д.

Факторы, влияющие на трудоемкость разработки КИС:

– возможности проектной команды, применяющего участие в разработке КИС;

– размер конечного программного продукта;

– среда разработки, в которой ведется разработка программного продукта, а также прочие инструменты автоматизации, позволяющие увеличить эффективность процесса разработки;

– требования, выдвигаемые к качеству будущего продукта, включающие в себя функциональные возможности продукта, а также основных характеристики производительности, надежности, адаптируемости и др.;

– общая методология разработки конечного программного продукта.

1.2 Планирование и оценка трудоемкости разработки программного продукта

Оценка и планирование критически важны для успеха проекта по разработке ПО любого размера и значимости. Планы определяют инвестиционные решения любой компании: компания может взяться за проект, на выполнение которого, по ее оценкам, потребуется полгода и 5 млн. рублей, и отказаться от этого же проекта, если на него потребуется два года и 10-15 млн рублей. Планы помогают понять, кого нужно привлечь к работам по проекту в течение определенного периода. Планы помогают понять, как продвигается создание функциональности, которая нужна пользователям и получения которой они ожидают. Отсутствие плана и оценки трудозатрат – способ вызывать большие проблемы в будящем. Процесс планирования, однако, сложен, а планы нередко получаются далекими от реальности. Как результат, программные проект зачастую впадает в одну из двух крайностей: участники либо полностью отказываются от планирования, либо тратят столько сил на составление планов, что начинают верить в их правильность. Программные проекты, которые отказываются от планирования, не могут ответить на такие фундаментальные вопросы, как «Когда это должно быть выполнено?» и «Можно ли ожидать выпуск продукта в июне?». Программные проекты, затратившая слишком много сил на планирование, обольщает себя уверенностью в том, что план в принципе может быть «правильным». План команды разработчиков может быть тщательно проработанным, но вовсе не обязательно отличаться высокой точностью или полезностью. Тот факт, что оценка и планирование – дело непростое, не является открытием. Это известно давным-давно. В 1981 г. Барри Боэм построил первую версию того, что Стив Макконнелл позднее назвал «конус неопределенности». На рисунке 1.1 показаны первоначальные диапазоны неопределенности Боэма в разных точках в процессе последовательного развития («каскадный процесс»). Конус неопределенности говорит о том, что на этапе оценки осуществимости проекта оценка обычно отклоняется от истины на 60–160 %. Иначе говоря, на проект, который, как ожидается, должен занять 20 недель, может потребоваться

от 12 до 32 недель. После формулирования требований в письменном виде оценка может отклоняться на ±15 % в любом направлении, т.е. плановый срок 20 недель может сократиться до 17 недель или вырасти до 23 недель.

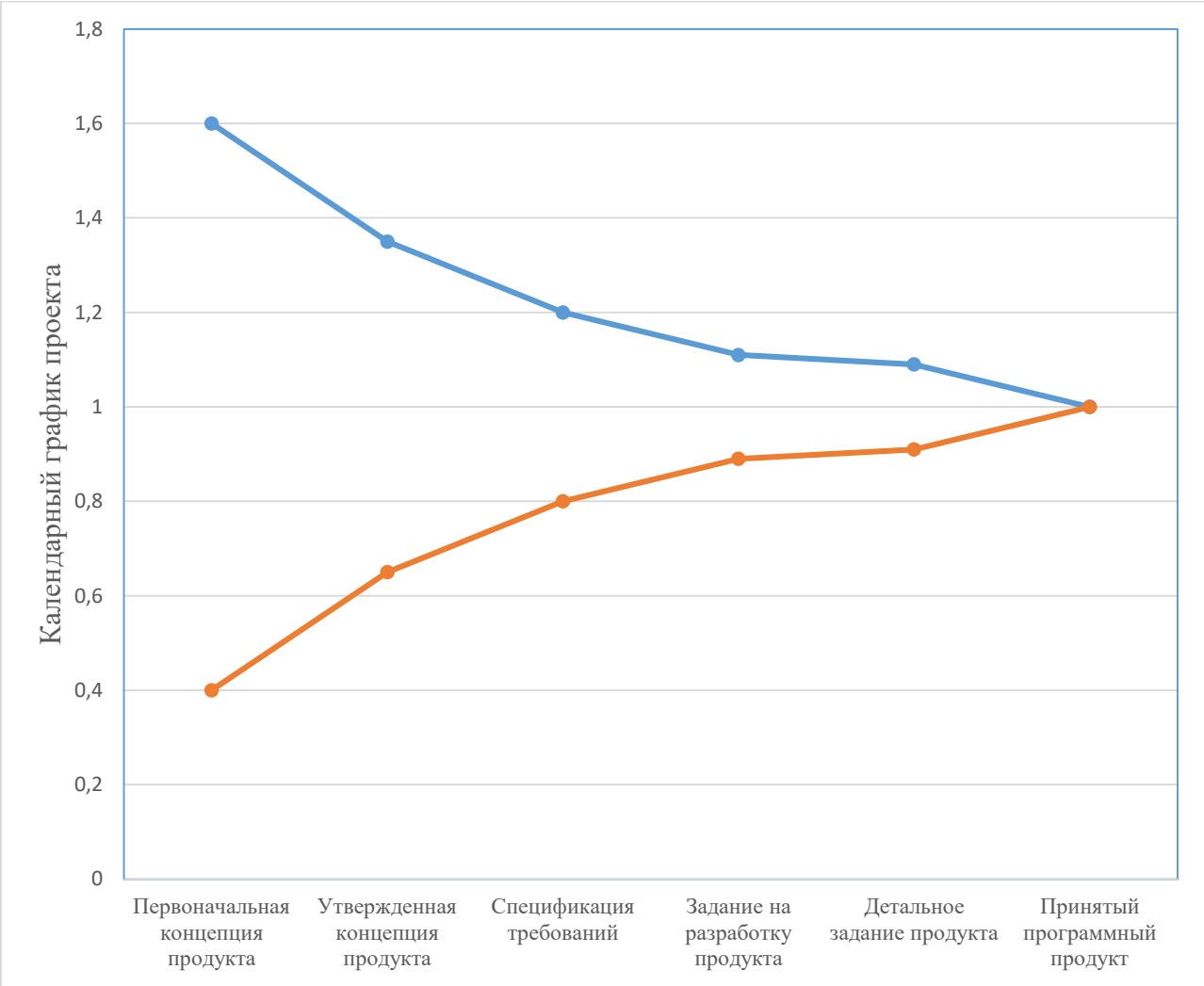


Рисунок 1.2 – Конус неопределенности программного проекта

Если оценка и планирование настолько трудны и если точную оценку невозможно получить вплоть до последней фазы выполнения проекта, то встает вопрос, в чем необходимость производить оценку трудоемкости? Конечно, очевидной причиной является то, что в организациях менеджмент нередко требует предоставления оценок. Планы и графики могут требоваться для таких вполне понятных целей, как планирование маркетинговых кампаний, планирование релизов продукта и обучение внутренних пользователей. Это очень важные потребности, и трудность оценки проекта не может служить

основанием для отказа от составления плана или графика, который организация может использовать для их удовлетворения. Вместе с тем помимо этих номинальных потребностей существует значительно более фундаментальная причина не жалеть сил на оценку и планирование. Оценка и планирование – это не просто определение сроков или календарных графиков. Планирование, особенно непрерывное планирование итераций, – это поиск стоимости. Планирование представляет собой попытку найти оптимальное решение всеобъемлющего вопроса разработки продукта: что должно получиться в итоге? Для ответа на этот вопрос команда анализирует функциональность, ресурсы и сроки. Ответ на данный вопрос нельзя найти одномоментно. Его ищут итерационно, шаг за шагом. В начале проекта, например, можем решить, что продукт должен иметь определенный набор функций, а его выпуск должен состояться 31 августа. Однако в июне оказывается, что лучше выпустить продукт немного позднее, но с более полным набором функций. Или может оказаться наоборот, что лучше сократить набор функций, но выпустить продукт чуть раньше. Хороший процесс планирования поддерживает такой подход, обеспечивая:

– сокращение риска;
– снижение неопределенности;
– создание условий для принятия более качественных решений;
– формирование доверия;
– распространение информации.

Планирование повышает вероятность успеха проекта, обеспечивая идентификацию проектных рисков. Одни проекты настолько рискованны, что лучше не браться за них. Другие могут содержать функциональности, риски которых, если к ним подойти должным образом с самого начала, поддаются ограничению. На обсуждениях, происходящих в процессе оценки, поднимаются вопросы, которые позволяют выявить подводные камни проекта. Допустим, перед программным проектом стоит задача оценить, сколько времени потребуется на интеграцию нового проекта со старой, построенной на основе

мейнфрейма системой, о которой участникам программного проекта ничего неизвестно. Это заставляет смотреть на функцию интеграции как на потенциальный риск. Проектная команда может устранить этот риск сразу, потратив определенное время на знакомство со старой системой. Альтернативно риск можно идентифицировать и учесть его в работе как отдельную величину или в виде диапазона и включить в общую неопределенность и риск.

В процессе реализации проекта команда создает новые функциональные возможности продукта, а также генерирует новые знания о продукте, используемых технологиях и своих собственных квалификациях. Крайне важно идентифицировать эти знания и учитывать их при итеративном планировании, которое должно помогать команде улучшать ее представления о продукте. Самым серьезным риском большинства проектов является риск создания несоответствующего продукта. Этот риск, однако, чаще всего полностью игнорируется. К счастью, на сегодняшний момент существует способ кардинально уменьшить такой риск. Один из способов является использование Scrum-подхода к планированию и ведению проекта, который получил большую популярность в последнее время. Нередко цитируемые исследования определяют успешный проект как такой, который выполнен в срок и в рамках бюджета и имеет все изначально предусмотренные функциональности [12]. Это опасное определение, поскольку оно не учитывает того, что функциональность, казавшаяся хорошей до начала проекта, может оказаться не стоящей вложений, когда проектная команда реально возьмется за дело. Одним из ключевых факторов, делающих компанию ООО «НетКрэкер» лидером в своей отрасли, это забота о заказчике, тем самым компания всегда готова подстраиваться под изменения требований заказчиков программных продуктов, тем самым подразумевается, что инвестиции, календарные графики и решения по функциональным возможностям периодически могут переоцениваться. Проект, имеющий все предусмотренные в первоначальном плане функциональности, не обязательно успешен. Пользователи продукта и его заказчики вряд ли будут довольны, если хорошие новые функциональности будут принесены в жертву

более простых просто потому, что те заложены в первоначальный план.

Наличие оценки и планов помогают принимать решения. Как организации определить, стоит ли браться за тот или иной проект, не имея оценки стоимости и затрат по проекту? Помимо поддержки решений относительно принятия проектов оценки позволяют гарантировать, что компания будет работать над самыми ценными проектами. Оценки необходимы организации для принятия и других решений, помимо решения о том, браться за проект или нет. В некоторых случаях штат исполнителей проекта более важен, чем календарный график. Так, реализация проекта может потерять смысл, если она требует участия ведущего разработчика организации, который полностью занят в другом проекте. Вместе с тем если удастся составить план, показывающий, как обойтись без участия ведущего разработчика, то за реализацию проекта, возможно, стоит взяться. Многие решения, принимаемые в процессе планирования проекта, являются компромиссными. Например, в любом проекте неизбежно приходится искать компромисс между временем разработки и затратами. Нередко наименее затратный путь разработки системы – это нанять одного хорошего программиста и позволить ему работать над созданием продукта 10 или 20 лет с возможность отвлекаться на освоение соответствующей профессиональной сферы, совершенствование в области администрирования баз данных и т.п. Очевидно, однако, что возможность ждать появления готового продукта 20 лет редко, когда возможно, поэтому разработка поручается команде программистов. Команде из 30 человек, возможно, потребуется год (30 человеко-лет) на разработку, с которой один программист мог бы справиться за 20 лет. Стоимость разработки при этом возрастает, однако стоимость, создаваемая при получении продукта на 19 лет раньше, покрывает увеличение затрат. Программным проектам постоянно приходится принимать компромиссные решения в отношении функциональности, трудозатрат, издержек и времени. Для принятия правильных компромиссных решений программным проекта необходимы оценки как затрат, так и выгод.

Частая надежная поставка обещанной функциональности рождает доверие

между разработчиками и заказчиками продукта. Достоверные оценки обеспечивают надежность поставки. Оценки нужны заказчикам ПП для распределения приоритетов и принятия компромиссных решений. Оценки также помогают заказчикам ПП решить, сколько функций разрабатывать. Вместо того, чтобы потратить 20 дней и получить все, может быть, лучше ограничиться 10 днями и получить 80 % выгод. Заказчики ПП с неохотой идут на принятие подобных компромиссных решений на начальной стадии осуществления проекта, если оценки разработчиков не внушают доверия.

Достоверные оценки позволяют разработчикам двигаться в стабильном темпе. Результатом является высококачественный ПП и снижение количества ошибок. Это, в свою очередь, повышает достоверность оценок, поскольку на такую во многом непредсказуемую работу, как устранение ошибок, приходится тратить меньше времени.

План дает представление об ожиданиях и показывает, что может произойти в процессе выполнения проекта. План не гарантирует получения точного набора функций в точно определенную дату по заданной стоимости. Он содержит информацию и устанавливает набор базовых ожиданий. Планы, к сожалению, зачастую сводятся к определению конкретной даты, а все допущения и ожидания, которые привели к появлению на свет этой даты, забываются.

Хорошим считается такой план, который, по мнению заинтересованных сторон, является достаточно надежным для того, чтобы на его основе принимать решения. На начальном этапе осуществления проекта это может быть указание на то, что продукт будет выпущен скорее в третьем квартале, а не во втором и что он будет иметь примерно обрисованный набор функций. На более позднем этапе работ этот план, чтобы оставаться по-прежнему полезным для принятия решений, должен быть более точным.

Данное исследование посвящено оценке трудоемкости при использовании Scrum методологии разработки ПО, а не гибким планам. Планы –это документы и цифры, статическое описание представлений о развитии проекта в

неопределенном будущем. Планирование – это вид деятельности. Методология разработки ПО Scrum предполагает перенос акцента с планов на процесс планирования. Методология разработки ПО Scrum позволяет сбалансировать вкладываемые в планирование усилия и трудозатраты с учетом того, что план будет пересматриваться в процессе осуществления проекта. Никто не собирается менять план ради изменений, изменения вносятся потому, что информация актуализируется, изменяются требования, исправляются ошибки. В процессе разработки ПП, для реализации которого используется Scrum подходы, возможно получить информацию, например, о том, что пользователи хотят расширить конкретную функцию или, наоборот, урезать ее, или узнать, что простота использования ПП значительно важнее, чем казалось вначале, или обнаружить, что программирование на новом языке занимает больше времени, чем ожидалось. Финансовые последствия каждого такого изменения можно оценить и, если это целесообразно, изменить план и календарный график. Вновь выясненные обстоятельства влияют на планы программного проекта. Это означает, что программному проекту такие планы, которые можно легко изменять. Именно поэтому процесс планирования становится более важным, чем сам план. Знания и представления, которые становятся известными в процессе планирования, продолжают существовать и после того, как от старого плана отказываются и заменяют его новым. Таким образом, под гибким понимают такой план, который легко поддается изменениям. Изменение плана само по себе не означает изменение дат. Сохраняется возможность, как успеть сделать какую-либо функциональность, так и нет. Однако, если оказывается, что программный проект заблуждался в процессе оценки в отношении определенного аспекта целевого продукта и нужно устранить ошибку, в план необходимо внести изменения. Существуют разные способы корректировки плана без изменения даты. Можно отказаться от функции, можно сократить ее объем, можно увеличить численность работников, занятых в проекте, и т.д. Поскольку программный проект признает, что невозможно с абсолютной точностью определить все аспекты проекта с самого начала, ему не нужно пытаться

запланировать все и вся на начальном этапе. При использовании Scrum методологии разработки ПО планирование осуществляется более или менее равномерно на протяжении всего срока реализации проекта. Вслед за планированием релиза, закладывающим фундамент, выполняется серия раундов планирования каждой итерации разработки, и весь процесс многократно повторяется по мере осуществления проекта.

Итак, при определении особенностей Scrum методологии разработки ПО к процессу планированию установлено, что методология Scrum: фокусируется на планировании, а не на плане; поощряет и готов к изменениям; приводит к составлению планов, легко поддающихся изменению; распределяет процесс планирования по всему сроку осуществление проекта.

1.3 Модели разработки программного обеспечения

Модель разработки ПО – это упрощенное представление процесса разработки ПО, который представляет собой набор связанных действий, которые приводят к его производству. Модель разработки ПО включается в себя идеологические принципы, план, контроль над процессами, подход к сотрудникам. Каждая модель представляет процесс с определенной точки зрения.

На сегодняшний день можно отметить 5 основных моделей разработки ПО, которые получили широкое распространение и признание мирового сообщества:

- каскадная модель разработки или «водопад»;
- инкрементальная модель разработки;
- спиральная модель разработки;
- итеративная модель разработки;
- Agile-модель разработки.

Эти общие модели являются абстракциями процесса, которые можно использовать для объяснения различных подходов к разработке ПО. Они могут быть адаптированы и расширены для создания более специфических процессов.

Модель «Водопад» является одной из самых старых моделей разработки ПО. «Водопад» подразумевает последовательное прохождение фундаментальных стадий, каждая из которых предоставлена в виде отдельной фазы и должна завершиться полностью до начала следующей.

Фазы разработки каскадной модели (см. рисунок 1.3):

– определение требований к ПП (создание спецификации);
– анализ и проектирование;
– реализация (кодирование);
– тестирование;
– поддержка.

Рисунок 1.3 – Каскадная методология разработки ПО

Подразумевается, что результатом каждого этапа является один или несколько документов, которые должны быть утверждены, и следующий этап не следует начинать до тех пор, пока предыдущий этап не будет полностью завершен.

Однако на практике эти фазы пересекаются и передают информацию друг другу.

Например, во время проектирования могут быть выявлены проблемы с

требованиями, а во время реализации могут быть обнаружены некоторые проблемы проектирования и т. Д.

Следовательно, процесс разработки ПО, при использовании каскадной модели, не является простым линейным, а включает обратную связь от одной фазы к другой.

Таким образом, документы, подготовленные на каждом этапе, могут затем быть изменены, чтобы отразить внесенные изменения.

Подразумевается, что каскадная модель должна применяться только тогда, когда требования хорошо понятны и вряд ли радикально изменятся во время разработки, т.к. эта модель имеет относительно жесткую структуру, что делает относительно трудным приспособление к изменениям, когда процесс идет полным ходом.

Иначе, ПП, разработанные по данной модели без обоснованного ее выбора, могут иметь недочеты, о которых становится известно лишь в конце всего процесса разработки ПО, из-за строгой последовательности действий. Стоимость внесения изменений высока, так как для ее инициализации приходится ждать завершения всего проекта.

Инкрементальная модель разработки основана на идее разработки первоначальной реализации ПП, подвергая ее обратной связи с пользователями и развивая ПП путем создания новых версий ПП, пока не будет разработана приемлемая система, удовлетворяющая всем требованиям заказчика.

Каждое приращение системы отражает часть функциональности, которая необходима клиенту. Как правило, ранние приращения системы должны включать в себя наиболее важные или наиболее срочно необходимые функциональные возможности.

Схематичное изображение инкрементальной модели разработки ПО можно увидеть на рисунке 1.4.

В случае использования инкрементальной модели разработки, заказчик ПП может оценить систему на ранней стадии разработки, чтобы увидеть, обеспечивает ли она необходимым требованиям.

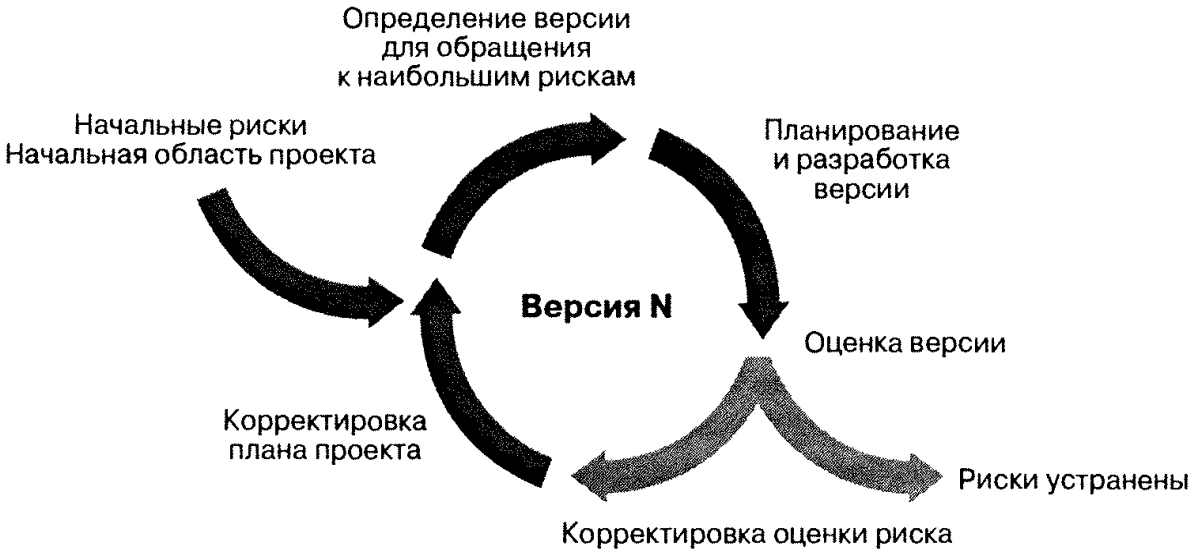


Рисунок 1.4 – Схематичное изображение инкрементальной модели разработки

Если система не обеспечивает требованиям, то необходимо изменить только текущей версии ПП и, возможно, новые функциональные возможности, определенные для последующих версий.

Инкрементная разработка программного зарекомендовала себя лучше, чем модель «Водопад», для самых различных ПП.

По сравнению с моделью «Водопад», инкрементальная модель имеет три важных преимущества:

– Стоимость размещения меняющихся требований клиентов снижается. Объем анализа и документации, который необходимо переделать, намного меньше, чем требуется для модели «Водопад».

– Намного проще получить отзывы клиентов о проделанной работе во время разработки, чем, когда система полностью разработана, протестирована и выдана заказчику.

– Возможна более быстрая выдача полезного для заказчика ПО, даже если не были включены все функциональные возможности. Заказчик может использовать и получать выгоду от ПО раньше, чем это возможно с моделью водопада.

– Благодаря постепенной разработке ПО дешевле и проще вносить изменения в ПП по мере его разработки.

Спиральная модель разработки была разработана с учетом того, чтобы включать в себя лучшие характеристики каскадной и инкрементальной моделей. Принципиальным отличием от каскадной и инкрементальной моделей является акцент на анализ рисков.

Спиральная модель хорошо работает для решения критически важных бизнес-задач, когда неудача несовместима с деятельностью компании, в условиях выпуска новых продуктовых линеек, при необходимости научных исследований и практической апробации.

Каждый цикл (от определения требований до поддержки, см. рисунок 1.5) в спирали представляет собой фазу.

Таким образом, первый цикл может быть связан с осуществимостью системы, следующий цикл может быть связан с определением требований, следующий цикл с проектированием системы и т.д.

Каждая петля в спирали разделена на четыре сектора:

1) Постановка цели: цели и риски на этом этапе проекта определены.

2) Оценка и снижение риска. Для каждого из выявленных рисков проекта проводится подробный анализ и предпринимаются шаги для снижения риска. Например, если есть риск, что требования неуместны, может быть разработан прототип ПП.

3) Разработка и проверка: после оценки риска выбирается модель процесса для системы. Поэтому, если в пользовательском интерфейсе ожидается риск, мы должны создать прототип пользовательского интерфейса. Если риск находится в самом процессе разработки, тогда следует использовать каскадную модель разработки ПО.

4) Планирование: проект рассматривается и принимается решение о том, продолжать ли его дальнейший цикл или нет.

Спиральная модель ориентирована на то, чтобы помочь людям подумать об итерации в процессе разработки ПО и внедрить в его разработку подход, основанный на оценке риска.

Однако, на практике, модель не получила широкого распространения.

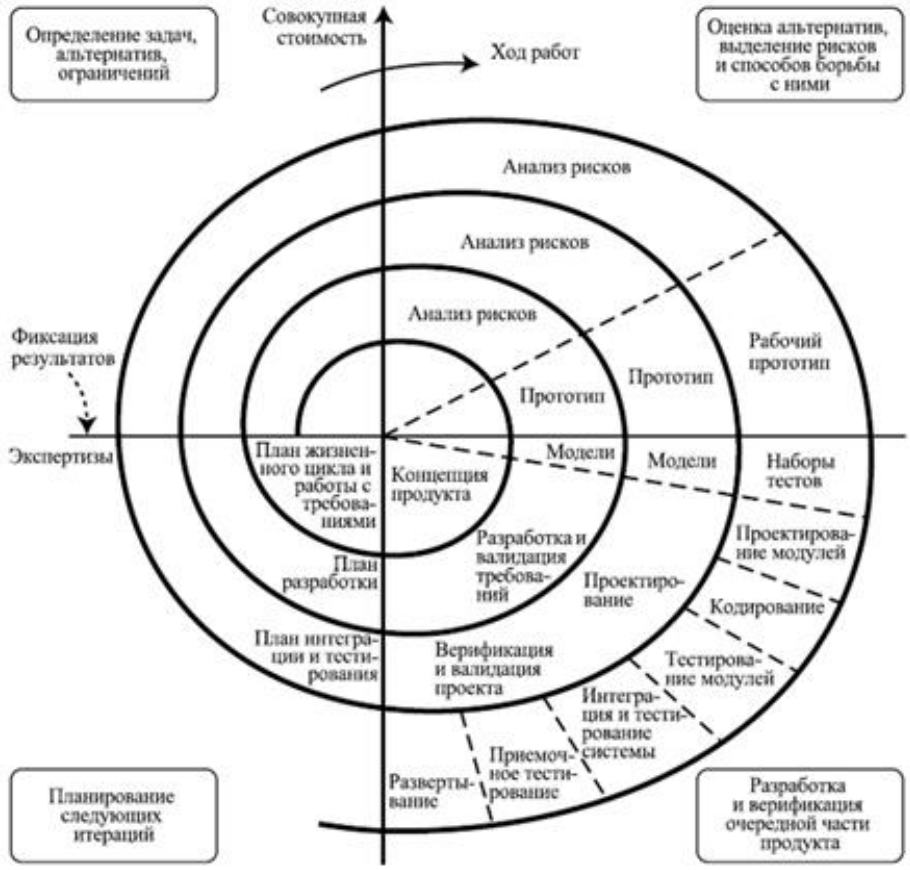


Рисунок 1.5 – Схематичное изображение спиральной модели разработки

Модель итеративной разработки направлена на разработку ПП путем построения небольших частей всех функциональных возможностей, одну за другой. Тем самым, итерационная модель жизненного цикла не требует для начала полной спецификации требований. Вместо этого, создание начинается с реализации части функционала, становящейся базой для определения дальнейших требований. Этот процесс повторяется. Промежуточная версия ПП не стремится быть идеально и сразу удовлетворять всем требования заказчика, главное, чтобы она работала. Основная концепция итеративной разработки состоит в понимании и стремлении к конечной цели таким образом, чтобы каждый итерация разработки была результативна, а каждая промежуточная версия ПП – работоспособна. Несмотря на схожесть на инкрементную модель разработки, использование итеративной модели не стремится создать полную функциональную возможность ПО, а подразумевает частичную разработку всех функциональных компонентов и получение обратной связи от клиентов системы

еще до окончательной разработки функциональной возможности.

Scrum-методологии стремятся быть максимально приспособленными к изменениям, адаптируемыми к новым требованиям.

Под Scrum-методами подразумевается группа методологий разработки ПО, которые, в первую очередь основаны на инкрементном подходе, при котором приращения незначительны, и, как правило, новые версии системы создаются и становятся доступными для клиентов каждые несколько недель. Требования и ключевые решения в процессе разработки основываются благодаря непрерывному и плотному сотрудничеству между участниками команд разработки и стороной заказчика ПП. «Scrum»-подход сочетает в себе инкрементный и итеративный подход к разработке ПО, создавая небольшую часть каждой функции, одну за другой, а затем постепенно добавляя функции и увеличивая их полноту. При этом, «Scrum»-подходы способствуют адаптивному планированию, эволюционному развитию, итеративному циклу разработки, тем самым способствует быстрому реагированию на различные изменения, которые могут быть вызваны в ПП, в том числе и на стадии его активной разработки. Схематичное изображение процесса разработки ПО при использовании Scrum методологий можно увидеть на рисунке 1.6.

В феврале 2001 года был выпущен основной документ «Scrum Manifesto», содержащий описание ценностей и принципов разработки ПО при использовании «Scrum»-подходов к разработке ПО:

– Люди и взаимодействие важнее процессов и инструментов разработки.

– Работающий продукт важнее исчерпывающей документации.

– Сотрудничество с заказчиком ПП важнее согласования условий контракта.

– Готовность к изменениям важнее следования первоначальному плану.

Часто, в процесс разработки привлекаются инженеры со стороны заказчика ПП, чтобы они могли предложить изменения требований. При этом, методология приветствует минимизацию документации, используя неформальные коммуникации, а не официальные встречи с письменными

документами.

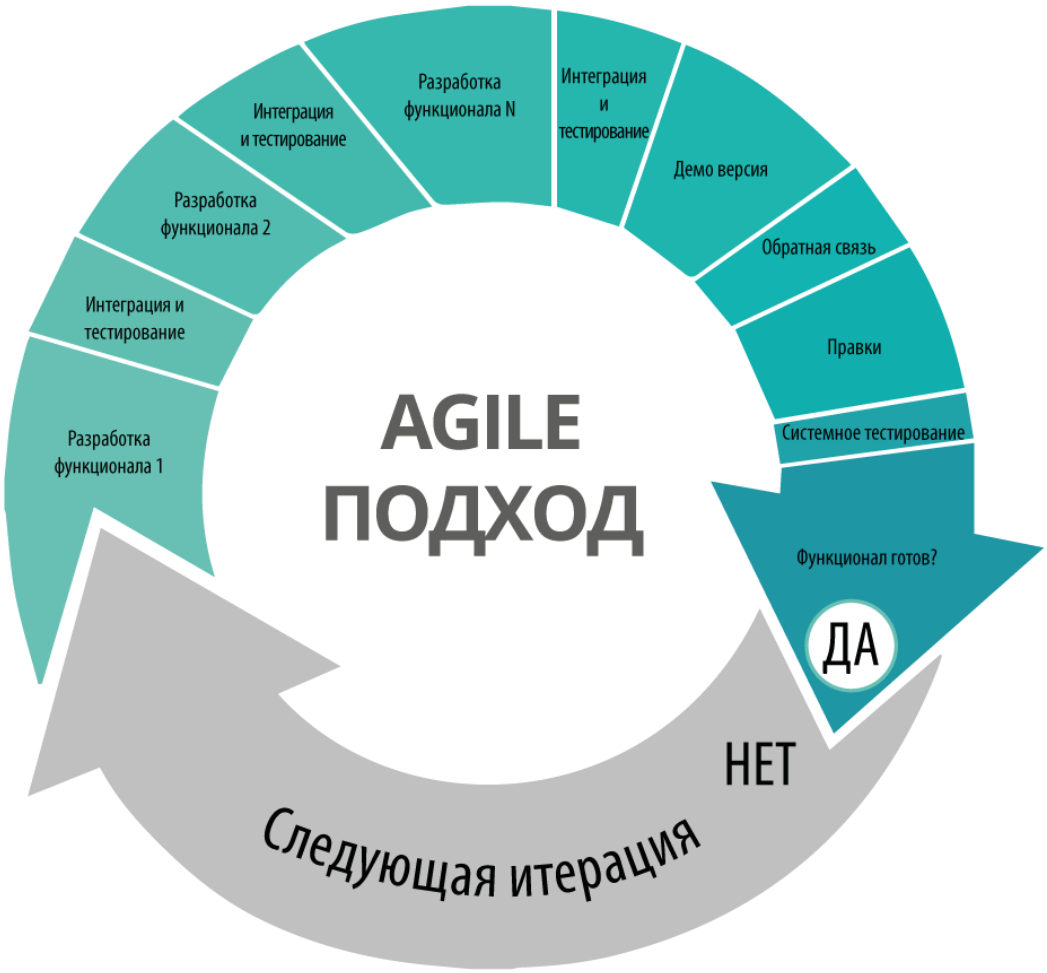


Рисунок 1.6 – Схематичное изображение Scrum модели разработки

«Scrum»-методологии на сегодняшний день получили широкое распространение, как по всему миру, так и российском бизнесе, т.к. общая идеология и используемые методологии подходят к современным реалиям ведения бизнеса, в том числе интернационального. В частности, в компании ООО «НетКрэкер» для большинства новых проектов выбирается именно эта методология разработки.

Существует ряд различных «Scrum»-методологий к разработке ПО, таких как:

– Scrum;

– Crystal Clear;
– Экстремальное программирование;
– Адаптивная разработка ПО;
– Feature Driven Development;
– Dynamic Systems Development Method (DSDM) (1995).

Основные характеристики «Scrum»-методологий разработки ПО:

1. Модульность: является одним из ключевых элементов «Scrum»-подходов, которая позволяет разбивать процесс разработки по компонентам, называющимися активностями.

2. Итеративность: «Scrum»-подходы сосредоточены на коротких циклах разработки (обычно от 2х недель до 1 месяца). Ожидается, что в течении каждого цикла разработки полностью завершается определенный набор задач (активностей), формируется релиз, который готов к выдаче заказчику.

3. Временная привязка: устанавливает временные ограничения для каждой итерации и обязывает существование расписания.

4. Адаптивность: «Scrum»-методологии разработки ПО адаптирует процесс разработки на готовность к неожиданным изменениям в ПП по различным причинам, появляющиеся на любой итерации. Иными словами, «Scrum»-методологии разработки ПО допускают как появление новых требований к программному продукту, несмотря на то, что о них не было известно ранее, так и модификацию уже существующих активностей.

5. Инкрементность: в случае использования «Scrum»-методологий не стоит задачи сразу создать полностью готовый ПП. Изначально поэтапно, создается нетривиальная система, и с каждой новой итерацией в нее добавляются новые функциональные возможности. Разработка новых функциональных возможности может производиться параллельно, в разное время и с разной скоростью.

6. Ориентированность на людей: Одной из ценностей, которая диктует «Scrum»-методологии разработки ПО, является то, что люди важнее самого процесса разработки и используемых технологий. Это означает, что в процессе

разработки люди эволюционируют, растут как специалисты. Тем самым, в процессе разработки растет производительность и качество выполняемой работой.

7. Совместная работа: «Scrum»-методы способствуют общению между членами команды. Коммуникация является важнейшей частью любого проекта разработки ПО. Как пример, между процессами быстрой интеграции большого проекта и в то же время процессами приращения программного продукта, которые происходят параллельно. Все это требует коммуникаций и сотрудничества между членами команды.

Также существуют различия в организации команды между «Scrum»-подходами к организации команды и традиционным подходом:

1. «Scrum»-команда – самодостаточная команда. Эта практика одной из «Scrum»-методологий под названием «экстремальное программирование», которая советуют проекту иметь достаточные навыки участникам команды, которые необходимы ей для выполнения задач проекта в полной мере. Следствием этого является то, что команда разработчиков обладает необходимыми навыками тестирования, опыт работы с базами данных, навыками проектирования пользовательского интерфейса и т.д. и ей не приходится полагаться на внешних экспертов или на узкоспециализированные команды, хорошо разбирающихся лишь только в такого рода вещах.

2. «Scrum»-команда формируется из специалистов широкого профиля. Специалисты широкого профиля обычно имеют одну или несколько технических специальностей (программисты, администраторы баз данных и т.д.) и способны внести ощутимый вклад в команду, обладают, по крайней мере общими знаниями в области разработки ПО и предметной области, в которой они работают, и, самое главное, активно стремятся к получению новых навыков, как в своих существующих специальностях, так и в других областях, включая как технические, так и смежные области. Очевидно, что начинающие ИТ-специалисты, чаще всего специализирующиеся только в одной области, должны будут работать над достижением этой цели. Специалисты широкого профиля –

это золотая середина между узкопрофильными специалистами и универсалами, которые понемногу знают о каждой предметной области и технологии.

3. «Scrum»-команды постоянны. Менеджер проекта, применяющий «Scrum»-методологию в своих проектах понимает, что изменение состава команды, вероятно, может негативно сказаться на успехе проекта и стремимся к тому, чтобы команды были как можно более стабильными.

При использовании каскадной модели разработки (например, «Водопад»), нагрузка на каждого члена команды определяется проектным менеджером, который самостоятельно оценивает, как долго будут выполняться та или иная задача, а затем назначает эту задачу на наиболее подходящего человека исходя из общего доступного времени, т.е. на практике чаще всего на того, кто на его взгляд, максимально быстро сможет выполнить эту задачу. Scrum-методологии используют значительно иной подход к определению возможностей каждого члена команды. Прежде всего, задача назначается всей команде, а не отдельному человеку, что делает упор на коллективные усилия. Во-вторых, менеджер отказывается количественно оценивать трудозатраты на работу с точки зрения времени, поскольку это подорвет самоорганизацию в команде, что также является одной из ключевых характеристик для успешного использования данной методологии разработки ПО. Эта особенность – большое различие с каскадной методологией разработки ПО: вместо того, чтобы менеджер оценивал время выполнения задач и назначал задачи сам, члены команды самостоятельно делают временную оценку.

Согласно Scrum-методологии, вся команда приходит на совещание по планированию будущего релиза для того, чтобы оценить свои усилия для общего списка задач. Владелец продукта требует эти оценки для того, чтобы иметь возможность эффективно распределить приоритеты по запланированным задачам, и, как результат, спрогнозировать релизы программного продукта на основе скорости команды. Это означает, что владелец продукта нуждается в честной оценке того, насколько сложной будет работа. Таким образом, владелец продукта не наблюдает за процессом оценки трудоёмкости, чтобы не оказывать

психологического воздействия на команду. Даже когда команда оценивает время выполнения между собой, необходимо принять меры, чтобы уменьшить влияние на оценку команды. Таким образом, всем членам команды рекомендуется раскрыть свои оценки одновременно.

Тем не менее возможна ситуация, когда команда трезво оценивает масштаб будущей работы, они могут оценить по-разному. Чтобы прийти к единой оценке трудозатрат, которая отражает сложность будущей разработки, иногда требуется не один этап оценки. Тем не менее, более опытные разработчики, знакомые с процессом, должны достичь консенсуса после нескольких этапов оценки трудоемкости. Обычно, оценка трудозатрат выполняется в начале каждой новой итерации, во время планирования будущего релиза.

Выбор обусловлен тенденцией выбора «Scrum»-методологий для управления всеми новыми проектами разработки ПП в компании ООО «НетКрэкер». Компания ООО «НетКрэкер» выбирает Scrum подходы к разработке ПП по следующим причинам:

– Адаптивность к изменениям – желания и потребности заказчиков имеют тенденцию часто меняться.

– Лучшее прогнозирование полного выполнения и сдачи работ.

– Эффективность, выраженная в скорости доставки версии программного продукта до заказчика.

– Часть заказчиков готовы и выражают желание принимать активное участие в процессе разработки ПО.

Данное исследование делает акцент на разработку модели оценки трудозатрат разработки ПО, которая, в свою очередь учитывает особенности «Scrum»-подходов в области его разработки, в частности методологии «Scrum».

1.4 Анализ существующих моделей оценки трудоемкости разработки программного обеспечения

Существующие традиционные модели оценки трудозатрат на разработку ПО классифицируются следующим образом:

– эмпирические параметрические модели оценки;
– эмпирические непараметрические модели оценки;
– экспертная оценка;
– модели аналоговой оценки;
– нисходящая оценка;
– восходящая оценка.

Эмпирическая параметрическая модели оценки основаны на опыте предыдущих программных проектов в том смысле, что они связывают размер и усилие с помощью явных математических формул, применяя метод регрессионного анализа. При этом наиболее широко используются линейная и экспоненциальная зависимость. Из положительных сторон эмпирических параметрических моделей можно отметить:

– объективность;
– формализм;
– эффективность.

Не мало важен и тот факт, что они основаны на опыте, полученном на реальном опыте и уже выполненных ПП. Недостатки этих моделей заключаются в следующем:

– необходимость калибровки перед применением в конкретном проекте;
– субъективность входных данных, которые использовались в прошлом проекте, а не в будущем.

Для эмпирических непараметрических моделей характерно, что они также используют данные о проектах, реализованных ранее. Однако оценка производится не с использованием математических формулы, а с помощью других подходов. В этих модели оптимизированы методики сокращения множеств (МСМ), магистральные линии принятия решений и нейронные сети. МСМ выбирает подмножество проектов, на основании которых оценивает продуктивность нового проекта. Трудоемкость на разработку определяется как усилие в человеко-месяцах, деленное на количество строк кода. Проекты, сгруппированные в оптимальном подмножестве, должны иметь такие же

факторы стоимости, как и новый проект. Первый метод использует алгоритм самообучения для получения дерева принятия решений. Другой метод основан на нейронных сетях. Модель нейронных сетей показывает меньшую среднюю ошибку, чем модель дерева принятия решений. Тем не менее, обучение нейронных сетей часто является весьма сложным, а в некоторых случаях, и долгим процессом. Точность оценки этой модели зависит от того, насколько правильно выбрано подмножество проектов, на основании которых выполняется оценка. Для того, чтобы можно было применить эти модели на практике, калибровка должна проводиться на очень большом количестве данных, поскольку эти модели имеют большое количество значений независимых переменных.

Экспертные модели основаны на консультациях одного или нескольких людей, которые считаются экспертами в разработке ПО. Согласование мнения среди оценщиков происходит во время обсуждения между участниками всех потенциальных проблем, которые могут встретиться при разработке ПП.

В аналоговых моделях оценки проводится сравнение между будущим проектом и аналогичными проектами, для которых известны данные о стоимости, времени выполнения и затраченных усилий. Эти модели требуют, как можно больше данных о реализованных проектах. Двумя наиболее известными аналоговыми моделями являются ESTOR и ANGEL.

ESTOR модель построена на рассуждениях и основана на конкретных случаях. Основанная на конкретных случаях форма рассуждения состоит из 4 основных этапов:

– определение специфики проекта;

– поиск наиболее подходящего проекта, который будет является аналогом будущего проекта, на основе которого можно оценить примерные трудозатраты;

– перенос оценки примерных трудозатрат с аналогичного проекта-источника на будущий проект;

– коррекция оценки трудозатрат на будущий проект на основе

найденных различий.

Модель ANGEL, в отличии от ESTOR основана на обобщении трудозатрат нескольких аналогичных проектов. В соответствии с подходом ANGEL, проекты представляются с помощью компонентов функциональных точек. Аналогичные проекты являются соседями будущего проекта, что достигается путем вычисления его векторного расстояния. Усилия, связанные с будущим проектом, оцениваются на основе среднего значения усилий в отношении соседних проектов. Как можно видеть, ESTOR и ANGEL имеют много общих черт в обоих случаях, проекты представлены посредством легко доступных метрик и аналогичных проектов, и трудозатраты в них оцениваются в обоих случаях путем расчета вектора расстояния между схожими проектами. ESTOR использует только один, наиболее общий аналогичный проект для определения трудозатрат, тогда как оценка ANGEL может основываться на нескольких аналогичных проектах. Преимущество аналоговых моделей оценки по сравнению с эмпирическими параметрическими моделями заключается в их успешном применении в тех случаях, когда невозможно определить действительную статистическую зависимость данных.

В моделях нисходящей оценки общие усилия на разработку производятся на основе глобальной характеристики ПП. Эта оценка обычно основывается на предыдущих проектах и берет в особенности берет во внимание усилия, затраченные на реализацию функциональности ПП. Полученные трудозатраты, затем распределяется по функциональным компонентам.

В моделях восходящей является противоположной модели нисходящей оценки, где сначала оцениваются трудозатраты на разработку в отношении каждого функционального компонента будущего проекта в отдельности, и общее усилие рассчитывается как сложение необходимых трудозатрат на каждый функциональный компонент. Зачастую такой подход не учитывает многие важные факторы и потенциальные проблемы, которые могут быть вызваны в процессе разработки, такие как проблемы с интеграцией, системным тестированием и различными факторами управления проектами.

1.5 Анализ существующих метрик, используемых в традиционных моделях оценки трудоемкости разработки программного обеспечения

Трудозатраты на разработку ПО могут быть оценены по следующим метрикам:

– метрики размера;
– функциональные точки;
– объектные точки;
– контрольные точки (метрики тестирования ПО);
– метрики использования ПО.

Метрики размера чаще всего используют SLOC-метрику. SLOC – это метрика ПО, используемая для измерения размера программы путем подсчета количества строк в тексте исходного кода программы. Этот показатель не учитывает пустые строки, строки комментариев и сторонних библиотек. Количественная характеристика SLOC зависит от конкретного языка программирования, который будут использоваться при написании будущего проекта.

Функциональная точка (FP), определенная Алланом Альбрехтом в IBM в 1979 году, является единицей измерения, выражающая количество функциональности ПО. Функция точечного анализа (FPA) – это метод измерения размера ПО. Преимущество данной метрики состоит в том, что она может избежать потенциальной ошибки в метрике SLOC, связанную с различной количественной характеристикой для каждого языка программирования, т.к. метрика функциональных точек не зависит от языка программирования, и поэтому идеально подходит для приложений, использующих наиболее популярные языки программирования. Метрика функциональных точек основана на данных, которые более вероятно будут известны на ранних этапах развития проекта. Метрика функциональных точек выражает произведения предварительно определенного коэффициента для каждого функционального компонента и коэффициент корректировки значения, указывающего на общие функциональные возможности, которые предоставляет приложение для

пользователя.

Метрика объектных точек являются альтернативной метрикой, связанной с измерением функциональных точек, когда для разработки используются 4-GL или аналогичные языки. В отличии от функциональных точек, базовыми характеристиками являются не функциональные компоненты, а 4-GL специфичные компоненты, такие как: языки БД, генераторы отчетов, создатели графических интерфейсов и т.д. Количество объектных точек и сложность каждой из них оценивается отдельно, а затем вычисляется взвешенное общее количество объектов-точек и используется для расчета трудозатрат, необходимых на разработку ПП.

Контрольная точка используется для оценки трудозатрат на тестирование системы и приемочных испытаний. Метрика охватывает тестирование черного ящика. Контрольная точка может быть в двух типов: динамические контрольные точки и статические контрольные точки. Динамические контрольные точки характеризует сумму всех контрольных точек, которые рассчитываются с использованием количества функциональных точек и функционально-зависимых факторов (ориентированность на пользователя, интенсивность использования ПО, сложность использования ПО), а также требований к качеству. Статические контрольные точки – это результат определения количества контрольных точек, необходимых для проверки статических измеримых характеристик.

Метрики использования ПО оценивается исходя из вариантов использования будущего ПП. Вариант использования – это поведение системы при различных условиях, основанное на запросе заказчика ПП. Метрика использования используется для оценивания вариантов использования для процесса тестирования ПО. Вариант использования служит входом для конкретного тестового сценария. Усилие, требуемое для прогона тестового сценария, рассчитывается исходя из варианта использования.

1.6 Анализ существующих методов оценки трудоемкости разработки программного обеспечения

В рамках данного исследования разберем две наиболее популярные модели оценки трудоемкости разработки ПО. Наиболее популярными на сегодняшний день моделями оценки трудоемкости разработки ПО являются:

– метод «функциональных точек»;

– методы, основанные на экспертных оценках.

Область применения метода функциональных точек – корпоративные информационные системы (КИС).

Достоинства функционально-ориентированных метрик:

– не зависят от языка программирования;

– легко вычисляются на любой стадии проекта.

Недостаток функционально-ориентированных точек: результаты основаны на субъективных данных, используются не прямые, а косвенные измерения.

Методика функциональных точек определяет трудоемкость на основе элементарных процессов, возникающих в пользовательском приложении. Элементарный процесс – это самая маленькая единица функционального требования заказчика ПП, которая:

– имеет смысл для заказчика ПП;

– составляет полную транзакцию;

– является автономной и при этом все приложение считается в согласованном состоянии.

Транзакция – это элементарный процесс, различаемый пользователем и перемещающий данные между внешней средой и программным приложением.

В методике функциональных точек для определения трудоемкости разработки используется 5 информационных характеристик на основе элементарных процессов [27]:

1. Внешний вход – элементарный процесс, характеризующий перемещение данных в приложение из-за его пределов, например, с

пользовательской формы или какой-либо другой внешней системы, с которой обеспечена интеграция. Данные могут использоваться для обновления внутренних логических файлов. Данные могут быть либо управляющей информацией, либо деловой информацией. Управляющие данные не модифицируют внутренние логические файлы.

2. Внешний выход – элементарный процесс, характеризующий перемещение данных из приложения во внешнюю среду. Кроме того, в этом процессе могут обновляться внутренние логические файлы. Выводы означают отчеты, экраны, распечатки, сообщения об ошибках или выходные файлы, посылаемые другим приложениям. Отчеты и файлы создаются на основе внутренних логических файлов и внешних интерфейсных файлов. Дополнительно этот процесс может использовать вводимые данные: критерии поиска либо параметры, не поддерживаемые внутренними логическими файлами. Вводимые данные носят временный характер.

3. Внешний запрос – элементарный процесс, работающий как с поступающими, так и с выводимыми данными. Результат процесса являются данные, возвращаемые из внутренних логических файлов и внешних интерфейсных файлов. Входная часть процесса не модифицирует внутренние логические файлы, а выходная часть не несет данных, вычисляемых приложением.

4. Внутренний логический файл – распознаваемая пользователем группа логически связанных данных, которая размещена внутри приложения и обслуживается через внешние вводы.

5. Внешний интерфейсный файл – распознаваемая пользователем группа логически связанных данных, которая размещена внутри другого приложения и поддерживается им. Внешний файл данного приложения является внутренним логическим файлом в другом приложении.

Вводы, выводы и запросы относят к категории транзакция.

Оценка числа функциональных точек (ФТ) для ПП выводится на основе данных, полученных на этапе анализа предметной области будущего

программного изделия и изучения особенностей его будущего функционирования.

Порядок расчета трудоемкости разработки ПО при использовании метода «Функциональные точки»:

– определение количества и сложности функциональных компонент будущего программного изделия;

– определение количества связанных с каждым функциональным компонентом элементарных данных, элементарных записей, а также файлов типа ссылок;

– определение общего коэффициента сложности на основе количества связей между функциональными компонентами;

– подсчет количества функциональных точек будущего программного изделия;

– калибровка количества функциональных точек с учетом общих характеристик системы, используя различные статистические данные.

Для транзакций ранжирование основано на количестве ссылок на файлы и количестве типов элементов данных.

Для файлов ранжирование основано на количестве типов элементов-записей и типов элементов данных, входящих в файл.

Тип элемента-записи – подгруппа элементов данных, распознаваемая пользователем в пределах файла.

Тип элемента данных – уникальное не рекурсивное (неповторяемое) поле, распознаваемое пользователем. В качестве примера рассмотрим таблицу 1.1.

В таблице 1.1 представлено 10 элементов данных:

– день;

– хиты;

– проценты от суммы хитов;

– сеансы пользователя;

– сумма хитов (по рабочим дням);

– сумма процентов от Суммы хитов (по рабочим дням);

– сумма сеансов пользователя (по рабочим дням);

– сумма хитов (по выходным дням);

– сумма процентов от Суммы хитов (по выходным дням);

– сумма сеансов пользователя (по выходным дням).

Таблица 1.1 – Отчет использования радиостанции

День	**Уровень активности дня недели, хиты**	**В процентах от Суммы хитов**	**Сеансы пользователя**
Понедельник	1887	16,41	201
Вторник	1547	13,45	177
Среда	1975	17,17	195
Четверг	1591	13,83	191
Пятница	2209	19,21	200
Суббота	1286	11,18	121
Воскресенье	1004	8,73	111
Сумма по рабочим дням	9209	80,08	964
Сумма по выходным дням	2290	19,91	232

Отметим, что поля «День», «Хиты», «В процентах от Суммы хитов», «Сеансы пользователя» имеют рекурсивные данные, которые не следует учитывать в расчете.

Примеры определения элементов данных для различных характеристик приведены в таблице 1.2.

Таблица 1.2 – примеры определения элементов данных для транзакций

Информационная характеристика	**Элементы данных**
Внешние Входы	Поля ввода данных, сообщения об ошибках, вычисляемые значения, кнопки
Внешние Выходы	Поля данных в отчетах, вычисляемые значения, сообщения об ошибках, заголовки столбцов, которые читаются из внутреннего файла Вводимые элементы: поле, используемое для поиска, щелчок мыши.
Внешние Запросы	Выводимые элементы – отображаемые на экране поля

В таблице 1.3 представлены правила учета элементов данных из графического интерфейса пользователя (GUI).

Например, GUI для обслуживания клиентов может иметь поля

– имя;

– адрес;
– город;
– страна;
– почтовый индекс;
– телефон;
– электронная почта.

Таблица 1.3 – Правила учета элементов данных из GUI

Элемент данных	Правило учета
Группа радиокнопок	Так как в группе пользователь выбирает только одну радио-кнопку, все радиокнопки группы считаются одним элементом данных
Группа флажков (переключателей)	Так как в группе пользователь может выбрать несколько флажков, каждый флажок считают элементом данных
Командные кнопки	Командная кнопка может определять действие добавления, запроса. Кнопка ОК может вызывать транзакции (различных типов). Кнопка Next может быть входным элементом запроса или вызывать другую транзакцию. Каждая кнопка считается от дельным элементом данных
Списки	Список может быть внешним запросом, но результат запроса может быть элементом данных внешнего ввода

Количество полей соответствует количеству элементу данных, которых в итоге получается семь. Восьмым элементом данных может быть какая-либо командная кнопка (добавить, изменить, удалить). В этом случае каждый из внешних вводов «Добавить», «Изменить», «Удалить» будет состоять из 8 элементов данных (7 полей плюс командная кнопка).

Обычно одному экрану графического интерфейса соответствует несколько транзакций. Типичный экран включает несколько внешних запросов, сопровождающих внешний ввод.

Рассмотрим порядок учета сообщений. В приложении с GUI генерируются три типа сообщений: сообщения об ошибке, сообщения подтверждения и сообщения уведомления. Сообщения об ошибке (например, «Требуется пароль») и сообщения подтверждения (например, «Вы действительно хотите удалить клиента?») указывают, что произошла ошибка или что процесс может быть завершен. Эти сообщения не образуют самостоятельного процесса, они являются частью другого процесса, то есть считаются элементом данных

соответствующей транзакции. С другой стороны, уведомление является независимым элементарным процессом.

Например, при попытке получить из банкомата сумму денег, превышающую их количество на счете, генерируется сообщение «Не хватает средств для завершения транзакции». Оно является результатом чтения информации из файла счета и формирования заключения. Сообщение уведомления рассматривается как внешний выход.

Данные для определения ранга и оценки сложности транзакций и файлов приведены в таблице 1.4 – 1.8 (числовая оценка указана в круглых скобках). Например, внешнему вводу, который ссылается на 2 файла и имеет 7 элементов данных, по таблице 1.4 назначается средний ранг и оценка сложности 4.

Таблица 1.4 – Ранг и оценка сложности внешних вводов

Ссылки на файлы	**Элементы данных**		
	1-4	**5-15**	**>15**
0-1	Низкий (3)	Низкий (3)	Средний (4)
2	Низкий (3)	Средний (4)	Высокий (6)
> 2	Средний (4)	Высокий (6)	Высокий (6)

Таблица 1.5 – Ранг и оценка сложности внешних выводов

Ссылки на файлы	**Элементы данных**		
	1-4	**5-19**	**>19**
0-1	Низкий (4)	Низкий (4)	Средний (5)
2-3	Низкий (4)	Средний (5)	Высокий (7)
> 3	Средний (5)	Высокий (7)	Высокий (7)

Таблица 1.6 – Ранг и оценка сложности внешних запросов

Ссылки на файлы	**Элементы данных**		
	1-4	**5-19**	**>19**
0-1	Низкий (3)	Низкий (3)	Средний (4)
2-3	Низкий (3)	Средний (4)	Высокий (6)
> 3	Средний (4)	Высокий (6)	Высокий (6)

Таблица 1.7 – Ранг и оценка сложности внешних запросов

Типы элементов записей	**Элементы данных**		
	1-19	**20-50**	**>50**
1	Низкий (7)	Низкий (7)	Средний (10)
2-5	Низкий (7)	Средний (10)	Высокий (15)
> 5	Средний (10)	Высокий (15)	Высокий (15)

Таблица 1.8 – Ранг и оценка сложности внешних интерфейсных файлов

Типы элементов записей	**Элементы данных**		
	1-19	**20-50**	**>50**
1	Низкий (5)	Низкий (5)	Средний (7)
2-5	Низкий (5)	Средний (7)	Высокий (10)
> 5	Средний (7)	Высокий (10)	Высокий (10)

Следует отметить, если во внешнем запросе ссылка на файл используется как на этапе ввода, так и на этапе вывода, она учитывается только один раз. Такое же правило распространяется и на элемент данных (однократный учет).

После определения всех информационных характеристик программного продукта и их сложности приступают к расчету метрики – количества функциональных указателей FP (Function Points) [27]. Исходные данные для расчета сводятся в таблицу 1.9.

Таблица 1.9 – Исходные данные для расчета функциональных точек

Имя характеристики	**Ранг, сложность, количество**			
	Низкий	**Средний**	**Высокий**	**Итого**
Внешние вводы	x3 =___	x4 = ___	x6 =___	
Внешние выводы	x4 =___	x5 = ___	x7 =	
Внешние запросы	x3 =___	x4 = ___	x6 =___	
Внутренние логические файлы	x7 =___	x10=___	x15 =___	
Внешние интерфейсные файлы	x5 =___	x7 = ___	x10 =___	
Общее количество рангов				

В таблицу 1.9 заносится количественное значение характеристики каждого вида (по всем уровням сложности). Места подстановки значений отмечены прямоугольниками (прямоугольник играет роль метки-заполнителя). Количественные значения характеристик умножаются на числовые оценки сложности. Полученные в каждой строке значения суммируются, давая полное значение для данной характеристики. Эти полные значения затем суммируются по вертикали, формируя общее количество.

Количество функциональных указателей вычисляется по формуле 1.1:

$$FP = \text{Общее количество рангов} * \left(0{,}65 + 0{,}01 * \sum_{i=1}^{14} F_i\right), \qquad (1.1)$$

где F_i – коэффициенты регулировки сложности (таблица 1.10), принимающие

целые значения: 0 – 5 в зависимости от сложности реализации соответствующей характеристики ПП.

Из результатов получения коэффициента функциональных точек, результат переводится в LOC-оценку, показывающая коэффициент усилия с учетом конкретного языка программирования. Произведем количественную оценку функциональных точек в LOC-оценку, используя формулу 1.2:

$$V = К_{яз} * FP, \quad (1.2)$$

где коэффициент $К_{яз}$ зависит от языка программирования, используемого для реализации ПО, берется из таблицы 1.11.

Таблица 1.10 – Коэффициенты регулировки сложности

№	Коэффициент	Значение
1	Передачи данных	1
2	Распределенная обработка данных	2
3	Производительность обработки	1,0
4	Эксплуатационные ограничения	1,5
5	Частота транзакций	1,0
6	Оперативный ввод данных	1,0
7	Эффективность работы конечного пользователя	1,0
8	Оперативное обновление	1,0
9	Сложность разработки	1,0
10	Возможность переиспользования программного кода	1,0
11	Простота установки	0,5
12	Простота эксплуатации	0,5
13	Разнообразные условия размещения	2,0
14	Простота изменений	1,0

Таблица 1.11 – Соответствие сложности на одну ФТ для разных языков программирования

Язык программирования	Количество операторов на одну ФТ (Кяз)
Ассемблер	320
C	128
Паскаль	90
C++	64
Java	53
Visual C++	34
Smalltalk	22
Perl	21
HTML	15
Access	38

Язык программирования	Количество операторов на одну ФТ (Кяз)
ANSI SQL	13
C++	53
Delphi	18
Visual Basic 6	24
1C	10

Для пересчета объема программы из условных строк программного кода в трудозатраты обычно используется промежуточная модель COCOMO, в соответствии с которой номинальную трудоемкость (без учета коэффициентов затрат труда, стоимостных факторов и сложности) можно вычислить, используя формулу 1.3:

$$T = N1 \times KSLOC^{N2}, \quad (1.3)$$

где KSLOC (тыс. строк) = V/1000.

Значения N1 и N2 определяют распространенность ПО и берутся из таблицы. 1.12.

Таблица 1.12 – Коэффициенты распространенности ПО

Тип ПО	N1	N2
Распространенное	3,2	1,05
Полунезависимое	3,0	1,12
Встроенное	2,8	1,20

Распространенное ПО − ПО небольшого объема (не более 50 KSLOC), разрабатываемое относительно небольшой группой опытных специалистов в стабильных условиях.

Полунезависимое ПО − ПО среднего объема (не более 300 KSLOC), разрабатываемое неоднородной группой специалистов средней квалификации.

Встроенное ПО − ПО с жесткими ограничениями (система резервирования авиабилетов, система управления воздушным движением и т.п.).

Время разработки вычисляется при помощи формулы 1.4:

$$t_{разр} = 2{,}5 \times T^{N3} \quad (1.4)$$

Значение N3 определяет степень распространенности ПО и берется из

таблицы. 1.13.

Таблица 1.13 – Коэффициент, определяющий степень распространенности ПО

Тип ПО	N1	N2
Распространенное	3,2	1,05
Полунезависимое	3,0	1,12
Встроенное	2,8	1,20

Одно из рекомендуемых правил распределения затрат проекта «40−20−40» регламентирует следующее:

– на анализ и проектирование приходится 40% затрат (из них на планирование и системный анализ – 5%);

– на кодирование – 20%;

– на тестирование и отладку – 40%.

Подходы, основанные на экспертных оценках, применяются при отсутствии дискретных эмпирических данных. Они используют опыт и знания экспертов-практиков в различных областях. На основе экспертных оценок были разработан метод Дельфи для оценки трудоемкости разработки ПО.

Метод оценки трудоемкости разработки Дельфи используется для принятия решений по спорным вопросам. На первом этапе эксперты отвечают на вопросники в два или более раундов. После каждого раунда ведущий предоставляет анонимную сводку ответов экспертов из предыдущего раунда с обоснованием их рассуждений. Затем экспертам предлагается пересмотреть свои предыдущие ответы в свете ответов других членов группы.

Считается, что в ходе этого процесса диапазон ответов будет уменьшаться, и группа будет стремиться к «правильному» ответу. Наконец, процесс останавливается после того, как предопределенный критерий остановки (например, количество раундов, достижение консенсуса и стабильность результатов) или средние оценки финальных раундов определяют требуемый результат.

Выводы по главе 1

В данной главе даны определения основных понятий, использующиеся в процессе оценки трудоемкости разработки ПО, а проанализированы способы оценки трудоемкости. Описаны и проанализированы существующие модели и подходы к разработке ПО.

Были выделены особенности методологии Scrum, которые следует учитывать при разработке новой методики оценки трудоемкости разработки ПО.

Проведен анализ существующих моделей оценки трудоемкости разработки ПО, а также метрики, использующиеся для оценки трудоемкости разработки ПО в этих моделях.

Проведенный анализ позволил сформулировать критерии, выдвигаемые к разрабатываемой модели оценки трудоемкости разработки ПО.

ГЛАВА 2 МОДЕЛИ И ИНСТРУМЕНТЫ ОЦЕНКИ ТРУДОЕМКОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИ ИСПОЛЬЗОВАНИИ «SCRUM»-МЕТОДОЛОГИЙ

2.1 Scrum подход к планированию разработки будущего программного продукта

Оценка и планирование критически важны, однако сложны и подвержены ошибкам. Так или иначе, отказываться от них просто из-за трудности нельзя. Оценки, данные в начале проекта, значительно менее точны, чем оценки, полученные позднее. Графическое представление процесса постепенного повышения точности оценок называют конусом неопределенности. Цель планирования – получение оптимального ответа на глобальный вопрос разработки продукта – вопрос о том, что именно создавать. Ответ включает в себя описание функций, ресурсов, а также календарный график. Ответ на этот вопрос, подкрепленный снижающим риск процессом планирования, сокращает неопределенность, дает основу для объективного принятия решений, устанавливает доверие и обеспечивает распространение информации. Хорошим является такой план, который достаточно надежен для того, чтобы на его основе принимать решения относительно продукта и проекта. Scrum-подход к планированию сфокусирован больше на планировании, а не на создании плана, поощряет изменения, приводит к составлению планов, легко поддающихся изменению, и распределяет процесс планирования по всему сроку осуществления проекта.

Планирование в Scrum проектах фокусируется на разработке и поставке ценных для пользователей функций, а не на выполнении изолированных задач (которые в конечном итоге объединяются в ценную для пользователей функцию). Одним из лучших подходов к этому является работа с пользовательскими историями, которая представляет собой облегченную методологию представления требований к программному обеспечению.

Пользовательская история – способ описания требований к разрабатываемой системе, сформулированных как одно или более предложений на повседневном или деловом языке пользователя. Каждая пользовательская история ограничена в размере и сложности. История описывает функциональность системы с точки зрения пользователя с определенной ролью и целью этой системы. Для заказчиков ПП, пользовательские истории являются основным инструментом влияния на процесс разработки ПО.

Scrum-команды осуществляют планирование для трех четко определенных горизонтов – релиз, итерация и текущий день. Взаимосвязь между этими (и другими) горизонтами планирования показана в виде так называемой луковицы планирования на рисунке 2.1.

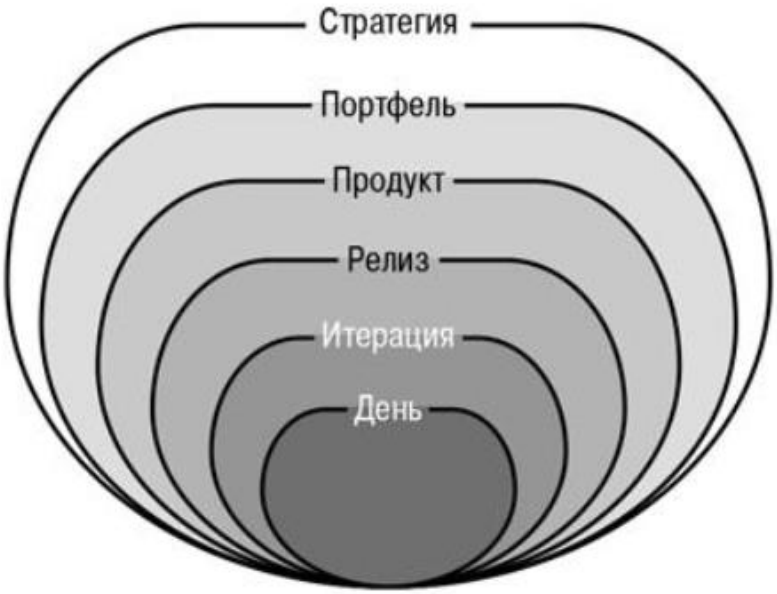


Рисунок 2.1 – Луковица планирования Scrum-методологии разработки ПО

Большинство Scrum-команд интересуют только три наиболее низких уровня луковицы планирования. При планировании релиза учитываются пользовательские истории или темы, которые создаются для нового релиза продукта или системы. Цель планирования релиза — это поиск приемлемых ответов на вопросы об объеме, календарном графике и ресурсах для проекта. Планирование релиза осуществляется в начале проекта, однако оно является первичным и осуществляется на одну или несколько будущих итераций

разработки. Хороший план релиза обновляется периодически на протяжении всего проекта (обычно в начале каждой итерации разработки), с тем чтобы он всегда отражал текущие ожидания относительно включаемой в релиз функциональности.

Следующий уровень – планирование итерации, осуществляемое в начале каждой из них. Опираясь на результаты только что завершенной итерации, владелец продукта идентифицирует высокоприоритетную работу, которой команда должна заниматься в новой итерации. Поскольку на этом уровне имеется более близкий горизонт, чем при планировании релиза, компоненты плана итерации могут быть более мелкими. В процессе планирования итерации говорится о задачах, реализация которых необходима для превращения запроса на функцию в работающую и протестированную программу. Наконец, существует еще дневное планирование. Большинство Scrum команд ежедневно проводят совещания для координирования работы и синхронизации действий. Хотя с формальной точки зрения такое планирование может показаться излишним, на этих совещаниях команды действительно составляют, оценивают и пересматривают свои планы, ставя себе достаточно короткое временное ограничение (15-30 минут). Во время ежедневных совещаний команда ограничивает горизонт планирования следующим днем, когда они вновь соберутся для обсуждения дел, поэтому участники фокусируются на планировании задач и на координировании отдельных видов деятельности, необходимых для выполнения этих задач. Осуществляя планирование на трех временных горизонтах – релиз, итерация и день, Scrum-команды концентрируют внимание на видимых и важных аспектах для создаваемого плана.

Планирование на уровне продукта и портфеля, а также стратегическое планирование находятся вне сферы интереса большинства Scrum-команд Планирование на уровне продукта требует от владельца продукта прогнозирования ситуации на более далеком горизонте, чем выпуск ближайшего релиза, и планирования эволюции выпущенного продукта или системы. Планирование на уровне портфеля предполагает выбор продуктов, которые

наилучшим образом соответствуют видению, сформированному в процессе стратегического планирования организации.

2.2 Метрики для оценки трудоемкости разработки программного обеспечения при использовании «SCRUM»-методологий

В «Scrum»-методологиях используется значительно иной подход к определению возможностей члена команды. Прежде всего, в Scrum-методологиях, менеджмент проекта назначает работу всей команде, а не отдельному человеку. Во-вторых, в методологиях Scrum отказываются количественно оценивать работу с точки зрения времени, потому что это подорвало бы самоорганизацию, лежащую в основе успеха методологии. Это не предписывает единый способ для команд оценить свою работу. Команда использует более абстрактную метрику для количественной оценки усилий. Обычно оценка усилий выполняется в начале новой итерации во время планирования следующей итерации разработки.

В Scrum-методологиях можно выделить 2 основных фактора, которые могут влиять на способность точно оценить результаты:

– размер «Пользовательской истории»;
– сложность «Пользовательской истории».

Из размера и сложность пользовательских историй выводится показатель, который определяет количество трудозатрат, необходимый для ее реализации. Если суммировать оценки в пунктах для всех пользовательских историй, то мы получим совокупный размер Scrum проекта.

Следующим ключевой метрикой для оценки трудоемкости разработки ПО, если используется Scrum подход, является «скорость». Скорость – это показатель темпа продвижения команды в пределах одной итерации разработки. Для его определения используют размер Scrum проекта, выраженный в необходимых трудозатратах для реализации всех пользовательских историй, а затем делят эту величину на количество дней в одной итерации разработки.

Если известна скорость работы команды, то можно разделить

коэффициент трудозатрат на эту скорость и получить расчетное количество в днях, необходимых на завершение разработку, которые также можно представить в количестве необходимых итераций. Этот показатель срока можно превратить в календарный график, отобразив итерации в определенном порядке в календаре, как показано на рисунке 2.2.

Рисунок 2.2. – Оценка срока выполнения проекта, используя метрики размера и сложности пользовательских историй.

Разные подходы к планированию между классическими методологиями разработки ПО и Scrum подходами приводит к невозможности использования методов и моделей, которые используется для оценки трудоемкости разработки ПО при использовании классических методологий разработки, в целях оценки трудоемкости разработки для Scrum проектов.

В таблице 2.1 представлено сравнении метрик, использующихся в Scrum-методологиях с метриками из традиционных методологий разработки.

Таблица 2.1 – Метрики для оценки трудоемкости разработки ПО

Традиционные метрики	**Scrum**
Показатель отставания оценить невозможно	Отставание при планировании известно (неизвестно только на первых итерациях разработки)
Функциональные точки	Метрики сложности
Общий комплекс работ	Спринт
Графики показателей эффективности предыдущих аналогичных проектов	Коэффициент затраченных усилий на одну итерацию разработки (для одной команды)
Метрика использования ПП	Не используется

Традиционные методы имеют много метрик, например, такие как «метрика

использования» или количество «функциональных точек», характеризирующая размер будущей разработки, на основе которых приходится использовать долгосрочное прогнозирование еще на начальном этапе разработки. Достоверные начальные оценки довольно сложно получить из-за отсутствия подробной информации на ранней стадии разработки. В «Scrum»-методологиях, в силу того, что планирование происходит на ближайшую, короткую по времени итерацию разработки, можно улучшить прогнозирование трудозатрат на нее. Также, можно утверждать и обратное, что для оценки эффективности использование метрик вида «функциональные точки», является сложной задачи, в силу того, что в Scrum-методологии готовы к изменяющимся требованиям со стороны заказчика будущего ПП. Для повышения точности усилий отставания можно комбинировать с UCP и точкой объекта.

Как можно видеть, трудозатраты на разработку будущего ПП оцениваются с помощью метрик. Для традиционной разработки ПО разработаны многочисленные модели и методы оценки трудоемкости разработки, в то время как для Scrum-методологий, на сегодняшний день, имеет очень ограниченное количество моделей оценки трудозатрат и в направлении оценки трудоемкости ПО существуют неразрешенный на данный момент вопросы. В связи с тем, что современная тенденция такова, что все для ведения разработки проектов все больше и больше используются Scrum-методологии, то на текущий момент, исследование данной проблематики является актуальной темой.

2.3 Параметры оценки трудоемкости разработки программного проекта

Существует множество факторов, которые могут влиять на способность точно оценивать трудозатраты. Для точной оценки требуется хорошее представление о всех этих факторах для получения точных и эффективных оценок. В первую очередь задача измерения трудоемкости ПО состоит в определении того, какие из факторов, влияющих на разработку, необходимо оценить и измерить. Влияющие факторы могут классифицироваться на внутренние и внешние, но при этом можно отбросить часть факторов, просто

сосредоточив внимание на тех, на которые имеется возможность повлиять, и наоборот обращаем меньше внимания на те факторы, влиять на которые не предоставляется возможным.

При использовании Scrum подходов существует 2 основных способа количественно измерить коэффициент усилия на разработку одной пользовательских историй:

– оценка трудоемкости пользовательской истории в пунктах;

– оценка трудоемкости пользовательской истории в идеальных днях.

При использовании оценки трудоемкости пользовательских историй в пунктах, используется следующие метрики:

– метрики размера пользовательской истории;

– метрики сложности пользовательской истории.

– коэффициенты трения, влияющие на разработку;

– переменные/динамические факторы, негативно влияющие на разработку.

Первые 2 параметра будут позволят оценить трудоемкость на разработку, а в следствие и ее темп, а последние 2 производить калибровку скорости разработку учитывая всевозможные риски и негативные факторы, которые могут возникать как до начала процесса разработки, так и в процессе самой разработки.

Размер пользовательской истории – это единица измерения общего масштаба работы с точки зрения реальных усилий по ее разработке. Величина исходных размеров не играет никакой роли. Что важно, так это относительные размеры. Пользовательская история, которой присвоено два пункта, должна быть в два раза больше, чем пользовательская история, которой присвоен один пункт. Также, эта пользовательская история должна также составлять по размеру две трети пользовательской истории, которую оценивают в три пункта и т.д. В таблице 2.2 показаны пять значений, присвоенных различным типам историй в зависимости от их размера. Формулировка рекомендаций может быть изменена в зависимости от специфики проекта или других особенностей, и даже остается возможно вовсе переопределить критерии размера пользовательской истории.

Таблица 2.2. Классификация размеров пользовательских историй

Числовое значение	**Характеристики**
5	– размер истории очень большой; – очень сложно оценить трудозатраты; – историю следует разбить на более мелкие истории, возможно, можно выделить в отдельный проект разработки;
4	– большая история; – требует больших усилий для разработчиков на протяжении длительного времени, подразумевает разработку более, чем одной недели всей командой; – следует рассмотреть возможность разбить на более мелкие истории, если возможно;
3	– умеренно большая история; – разработка в течении 2-5 дней всей командой;
2	– небольшая история; – разработка в течении 1-2 дня всей командой;
1	– очень маленькая история, к разработке которой нужен небольшой уровень усилий; – разработка в течении 1 дня всей командой.

Сложность пользовательской истории вводит неопределённость в оценку трудоемкости разработки – наиболее высокая оценка сложности означает наибольшую неопределенность в оценке трудоёмкости.

Сложность пользовательской истории характеризует:

– ее зависимость от реализации других пользовательских историй;

– ее влияние на уже реализованные пользовательские истории;

– наличие соответствующего опыта и навыков в команде разработки;

– насколько легко ее формализовать;

– требование дополнительных исследований перед фазой разработки.

В таблице 2.3 показаны 5 значений, присеваемых пользовательским историям в соответствии с их характером сложности.

При планировании каждой новой итерации разработки составляются две таблицы, соотносящие в себе наименование/номер пользовательской истории, а также ее количественную оценку размера/сложности.

Пример такой таблицы можно увидеть на рисунке 2.3.

Таблица 2.3. Классификация сложности пользовательских историй

Числовое значение	**Характеристики**
5	– высочайшая сложность; – пользовательская история очень зависима от других историй, других систем; – требует для разработки некоторые навыки или опыт, без которого не обойтись, и при этом он отсутствует в команде; – историю трудно точно формализовать; – требуется значительные исправления кодовой базы существующего решение; – требуется дополнительные значительные трудозатраты на исследования перед активной фазой разработки; – пользовательская история может оказать существенное влияние на другие;
4	– очень сложная пользовательская история; – существует множество зависимостей от других историй; – требует для разработки некоторые навыки или опыт, без которого не обойтись, и он отсутствует в команде; – историю трудно точно формализовать; – требуется исправлять существенное решение (изменения в коде продукта); – требуется дополнительные исследования перед активной фазой разработки; – для разработки требуются большое количество старших разработчиков; – требует несколько технологически сложных решений; – пользовательская история имеет умеренное влияние на разработку других;
3	– умеренная сложность; – умеренное количество зависимостей от других историй, других систем; – представляет набор навыков или опыт, который достаточен в команде; – историю трудно точно формализовать; – может потребоваться некоторая переработка существующего решения; – требуется наличие специалистов со средними навыками программирования для успешного завершения; – требует незначительных исследований перед активной фазой разработки; – история имеет минимальное влияние на разработку других историй;
2	– легкий уровень сложности; – понятные технические и бизнес-требования; – требует минимальных исследований перед активной фазой разработки; – разработку истории можно поручить как средним, так и начинающим специалистам; – последствия разработки истории почти локализованы;
1	– очень простая; – технические и бизнес-требования очень четкие, без двусмысленности; – никаких исследований не требуется; – разработку можно поручить даже начинающим специалистам; – последствия разработки для других историй отсутствуют.

История	Оценка
Как тренер я могу приобретать программу SwimStats и использовать ее вместе со своей командой	5
Как пользователь я должен зарегистрироваться в системе и могу видеть только данные, к которым мне разрешен доступ	5
Как тренер я могу вводить имена и гендерно-возрастную информацию по всем пловцам моей команды	8
Как пловец я могу видеть все мои результаты в конкретных соревнованиях	5
Как пловец я могу корректировать свою гендерно-возрастную информацию	5
Как пловец я могу видеть мое лучшее время в каждых соревнованиях	5
Как пловец я могу получить отчет, который показывает рост моих результатов по всем соревнованиям с определенной прошлой даты	5
Как пловец я могу сравнивать свои результаты с национальными рекордами	8
Как тренер я могу определять расписание тренировок	8
Как тренер я могу рассылать электронные письма всем пловцам команды	3
Как тренер я могу определять по своему желанию поля для отслеживания по каждому пловцу	8
Как системный администратор я могу конфигурировать права доступа и пользовательские группы	3
Всего	**68**

Рисунок 2.3 – Количественная оценка сложности/размера пользовательской истории в пунктах

Как и таблица классификации размеров пользовательских историй, рекомендации по сложности пользовательских историй являются общими и могут быть скорректированы самой командой, исходя из собственных особенностей проекта.

Общим коэффициентом усилия при использовании метрик размера и сложности пользовательских историй, будет являться произведение эти величин.

Альтернативным способом количественно оценить необходимые трудозатраты разработки ПО при использовании Scrum подходов является способ «Идеального дня» [17]. В разработке большого и корпоративного ПП идеальное время отличается от общего затраченного времени из-за естественных непроизводительных издержек, которые происходят ежедневно. В любой отдельно взятый день в дополнение к запланированной работе над проектом, член команды может отвечать на электронные письма, консультировать

поставщика, проводить собеседование с кандидатом на замещение вакантной должности аналитика и присутствовать на паре совещаний. Дополнительные примеры причин, по которым идеальное время не совпадает с общим затраченным временем:

- поддержка текущего релиза;
- отсутствие из-за болезни;
- совещания;
- демонстрации продукта;
- работа с персоналом;
- телефонные переговоры;
- специальные проекты;
- повышение квалификации;
- электронная переписка;
- анализ сделанного и тестовые прогоны;
- собеседования с кандидатами;
- переключение на другие задачи;
- устранение ошибок в текущих релизах;
- управленческий анализ.

Кроме того, при выяснении причин, по которым идеальное время не соответствует общему затраченному, нужно учитывать, что руководители могут непрерывно работать от одного отвлекающего события до другого не более пяти минут. Даже если типичный разработчик отвлекается в три раза реже, время его непрерывной работы составляет всего 15 минут. Проблемы могут возникать, когда руководитель задает члену команды неизбежный вопрос: «Сколько времени на это потребуется?» Член команды отвечает «Пять дней», а руководитель отсчитывает пять дней в своем календаре и помечает срок окончания работы жирным красным знаком X. Член команды, однако, на самом деле хотел сказать: «Пять дней, если я буду заниматься только этим, но у меня полно других дел, поэтому примерно две недели».

В разработке большого и корпоративного ПП многозадачность еще

больше увеличивает разрыв между идеальным и общим затраченным временем. Разработчик программного обеспечения, которого заставляют работать в многозадачном режиме, в значительной мере теряет свою эффективность в результате переключения между двумя (или более) задачами.

Оценку трудоемкости пользовательских историй или другую работу можно оценить в идеальных днях. При оценке в идеальных днях следует исходить из следующего:

– оцениваемая пользовательская история – это единственная задача, над которой вы работаете;

– все, что необходимо, есть под рукой в момент начала работы;

– перерывы и отвлечения отсутствуют.

При оценке числа идеальных дней, необходимых для реализации той или иной пользовательской истории, не нужно учитывать влияние непроизводительных издержек, обусловленных средой, в которой работает команда, как это делается при оценке общей трудоемкости в пунктах.

Если на разработку конкретной экранной формы необходим один идеальный день, то разработчик будет заниматься ей один день, независимо от того, работает ли он в стартапе без непроизводительных издержек, в условиях, где разработчика отвлекают другие сотрудники, или в жесткой бюрократической системе. Возможно, разработчику удастся затратить на работу немногим больше идеального дня в стартапе с низкими непроизводительными издержками. Чем больше будет отвлекающих моментов, тем меньше времени у разработчика останется на разработку продукта для проекта и тем больше будет срок выполнения работы, рассчитанной на один идеальный день. Если оставить в стороне непроизводительные издержки организации, то идеальные дни можно использовать для оценки размера точно так же, как и пункты. Затем оценку размера в идеальных днях можно преобразовать в расчетный срок выполнения работы с помощью скорости аналогично тому, как это делается с оценкой размера в пунктах.

Как показатели размера пункты и идеальные дни имеют свои достоинства.

Ниже приводится перечень основных доводов в пользу выбора пунктов для оценки размера и сложности пользовательских историй:

– Пункты способствуют выработке кроссфункционального поведения.
– Оценки в пунктах не устаревают.
– Пункты – это чистый показатель размера и сложности.
– Оценка в пунктах обычно требует меньше времени.
– Разное понимание «идеальных дней» у разработчиков ПП.

Одна из причин успеха Scrum-команд заключается в том, что они являются кроссфункциональными. Иначе говоря, Scrum-команды состоят из представителей всех дисциплин, необходимых для создания продукта, включая программистов, тестировщиков, менеджеров по продукту, дизайнеров пользовательских интерфейсов, аналитиков и администраторов баз данных. Зачастую, когда кроссфункциональная команда только начинает формироваться, некоторые ее члены с трудом отказываются от своей цеховой принадлежности. Продукт выигрывает от того, что участники проекта смотрят друг на друга сначала как на членов команды, а уже потом как на специалистов, т.е. по принципу «я участник проекта Napa, и я тестировщик», а не «я тестировщик, прикрепленный к проекту Napa». Различие, может быть, и небольшое, но изменение психологической установки существенное.

Оценка в пунктах может помочь команде научиться работать кроссфункционально. Поскольку оценка в пунктах должна быть единой и представлять работу в целом для всей команды, она инициирует активное обсуждение всех затрагиваемых аспектов. Оценка в идеальных днях, в свою очередь, нередко приводит к тому, что специализированные группы прикидывают, сколько времени займет «их часть» истории, а потом суммируют полученные результаты. Например, программисты могут решить, что им нужно три идеальных дня, администратор баз данных – один день, а тестировщик – два дня. После этого истории присваивается оценка «шесть идеальных дней». Небольшое различие в том, как происходит первое обсуждение пользовательской истории, постоянно довлеет над процессом ее реализации.

Оценки, выраженные в пунктах, не устаревают значительно дольше, чем оценки в идеальных днях. Оценка в идеальных днях может меняться среди прочего в зависимости от опыта команды, области ее специализации и состава. Для более глубокого понимания причин этого предположим, что программиста, осваивающего новый язык, спрашивают, сколько времени потребуется на создание небольшого приложения. Если поинтересоваться у этого же программиста несколько месяцев спустя, сколько ему потребуется на разработку приложения такого же размера и сложности, то он вполне может сказать, что уложится в один день, поскольку приобрел опыт работы с новым языком. Таким образом, у возникает проблема, связанная с тем, что два приложения оцениваются по-разному, хотя имеют один и тот же размер. Все, что можно будет наблюдать, это стабильная скорость, хотя объем выполненной работы увеличивается. Предположим, что этот программист – единственный член команды и что он работает итерациями продолжительностью одна неделя. В первый раз, когда он разрабатывает это приложение, его оценка составляет пять идеальных дней и в его условиях календарный день равен идеальному дню. Он начинает работать над приложением в первый день итерации и заканчивает его на пятый. Через несколько месяцев, поскольку оценка аналогичного приложения уменьшается до одного идеального дня, программист разрабатывает пять приложений за одну итерацию. Его скорость опять равна пяти, несмотря на то что он делает в пять раз более объемную работу, чем прежде. В некоторых проектах, особенно в тех случаях, когда осваиваются новые технологии или команда не имеет опыта в данной сфере, этот фактор может быть очень значительным. Следует обращать внимание на то, что оценки и в пунктах, и в идеальных днях требуют пересмотра при изменении размера в процессе разработки. Вместе с тем, когда команда приобретает опыт работы в той или иной области, в пересмотре нуждаются только оценки в идеальных днях.

Первым шагом к оценке сроков реализации чего-либо является оценка размера объекта или объема предстоящей работы. Пункты представляют собой чистый показатель размера, а идеальные дни – нет. Идеальные дни можно

использовать в качестве показателя размера, но с определенными ограничениями. Как уже отмечалось в предыдущем разделе, оценка в идеальных днях меняется по мере изменения квалификации разработчика. Такого не происходит с пунктами, поскольку размер такой, какой есть, и он не меняется. Это очень желательное свойство для любого показателя размера. То, что пункты являются чистым показателем размера, дает два преимущества. Во-первых, это означает, что у нас есть возможность оценивать истории исключительно по аналогии. Существуют убедительные свидетельства того, что оценка намного лучше происходит по принципу «это похоже на то», чем определение абсолютного размера объектов [12]. При использовании метода оценки реализации пользовательских историй идеальных дней, также можно оценивать по аналогии, однако в этом случае оценка будет происходить в категориях календарного графика и продолжительности реализации пользовательской истории. Во-вторых, поскольку пункты являются чистым показателем размера и сложности, тем самым являясь абсолютно отвлеченной величиной, при их использовании нет соблазна сравнивать их с чем-то реальным. Команды, которые оперируют идеальными днями, практически неизбежно сравнивают их с фактическими днями. После этого они пытаются найти причины, по которым работа объемом «только» восемь идеальных дней выполняется за 10-дневнуюитерацию.

Также, как показывает опыт разработки многих ПП, Scrum команды, которые оценивают пользовательские истории в пунктах, решают эту задачу быстрее команд, использующих для оценки идеальные дни. Чтобы оценить большое количество пользовательских историй, сначала необходим этап обсуждение дизайна в общих чертах. Команды, использующие для оценки идеальные дни, склонны более глубоко прорабатывать данные вопросы, чем команды, проводящие оценку в пунктах. Разница, по всей видимости, связана с тем, что при оценке в идеальных днях высок соблазн заняться анализом индивидуальных задач, необходимых для реализации истории, в то время как необходимо думать о размере и сложности пользовательской истории

относительно других пользовательских историй.

Разница «идеальных дней» также накладывает ограничения и сложности для того, чтобы дать адекватную оценку усилий на разработку. Разработчики могут давать разные оценки исходя из своей квалификации и прочих условий. И вероятно, каждый из них будет прав. Тем не менее, разработчикам нужно придти к согласию. Можно выбрать в качестве ориентира наименьшую оценку, и отдать эту работу этому разработчику. Однако это может обернуться ошибкой, если к моменту фактического выполнения работы этот разработчик окажется слишком занят и задание придется выполнять другому разработчику, который давал более высокую оценку. Как следствие, разработчик выполнит эту задачу позже, поскольку она оценена в три дня, а этому разработчику нужно было пять дней. Большинство команд просто игнорируют эту проблему. Это вполне приемлемо, если все разработчики имеют примерно одну и ту же квалификацию или если программисты всегда работают парами, что помогает сгладить экстремальные расхождения в производительности.

Ниже приводится перечень основных доводов в пользу выбора идеальных дней для оценки коэффициента усилий на разработку пользовательских историй:

– идеальные дни легче объяснить за пределами команды;
– идеальные дни легче использовать для оценки в начальный момент.

Идеальные дни понятны на интуитивном уровне – «это количество времени, которое разработчику потребуется на эту работу, если он будет заниматься только ею». Поскольку это понятно на интуитивном уровне, оценки в идеальных днях легко объяснить другим людям за пределами проектной команды. Все понимают, что не все до единой минуты рабочего дня посвящаются программированию, тестированию, дизайну или иным образом используются для создания новых функций. Внешним наблюдателям (а сначала и команде) обычно приходится объяснять идею применения пунктов в качестве показателя размера. Вместе с тем необходимость объяснения смысла пунктов нередко можно использовать как удобный случай представить общий подход к оценке и планированию проекта.

Помимо простоты объяснения другим смысла оценки в идеальных днях, самой команде легче начинать оценку с использованием идеальных дней. Если команда выбирает в качестве показателя пункты, то ей довольно трудно дается оценка первых нескольких историй. Без ориентира, такого как рабочий день с девяти до пяти или оцененные ранее истории, команде, использующей пункты, необходимо найти какую-то базу, от которой можно отталкиваться. К счастью, большинство команд проходят через эту начальную стадию применения пунктов для оценки очень быстро. Обычно в течение часа они начинают воспринимать пункты так, словно пользовались ими годами. Так или иначе, первые несколько историй доставляют головную боль.

Преимущества, которые дает оценка пунктами коэффициента усилия на разработку, как чистый показатель размера и сложности, очень убедительны. То, что пункты способствуют выработке кроссфункционального поведения, очень положительно сказывается на работе команды. Мышление типа «моя часть займет три идеальных дня, на вашу часть потребуется два идеальных дня, так что в целом нам нужно пять идеальных дней» очень сильно отличается от такого мышления, как «в целом эта история имеет примерно такой же размер, как та история, поэтому давайте также присвоим ей пять пунктов». То, что пункты представляют для всех разработчиков одно и то же, в отличие от идеальных дней, является их большим достоинством. Это позволяет двум разработчикам с разной квалификацией или опытом легко договориться о размере объекта, хотя сроки его реализации будут у них разными. Недостатки пунктов по-настоящему незначительны. Конечно, легче начать оценку с использованием идеальных дней. Однако неудобство работы с туманными пунктами очень быстро исчезает. Суть оценки в идеальных днях определенно легче объяснить посторонним, но вряд ли стоит выбирать что-то из-за легкости объяснения. Более того, легкость понимания сути идеальных дней создает проблемы. В некоторых организациях стараются сблизить продолжительность фактического дня с продолжительностью идеального дня. Концентрироваться нужно не на этом, а на работе. Стремление организации приблизить фактический день к идеальному,

помимо прочего, заставляет давать оценку в фактическом времени, хотя его и называют идеальным. Иными словами, идеальный день определяют, как «день, в котором я шесть часов уделяю работе, а два часа занимаюсь прочими вопросами». Исходя из всех перечисленных достоинств и незначительности недостатков оценки трудозатрат на разработку пользовательской истории в «пунктах», модель оценки трудоемкости разработки ПП будет использовать именно эти метрики для общей оценки трудоемкости разработки ПП.

2.4 Модель оценки трудоемкости разработки

Используя метрики сложности и размера пользовательской истории, коэффициент усилия одной команды для конкретной пользовательской истории определяются с использованием следующей простой формулы:

ES = Сложность польз.истории * размер польз.истории (2.1)

Трудозатраты, приложенные командой на определенное количество итераций или вовсе весь проект можно выразить суммой трудозатрат на пользовательские истории в пределах итераций или всего проекта:

$$E = \sum_{i+1}^{n} ES_i \quad (2.2)$$

Одним из важных коэффициентов, характеризирующий темп продвижения команды, является оценка коэффициента скорости разработки в пределах одной итерации разработки. Расчет скорости разработки производится по следующей формуле.

$$V_i = \frac{\text{E}}{\text{Длительной одной итерации разработки в днях}} \quad (2.3)$$

После выполнения расчета коэффициента скорости разработки в пределах одной итерации разработки, обычно проводится калибровка этого коэффициента, и, следовательно, оптимизация трудозатрат.

В разработке ПО обычно существуют различные негативные факторы, которые оказывают негативное влияние на эффективность разработки. Для методологии оценки трудоемкости для Scrum-методологии разработки ПО возможно рассмотреть возможность оптимизации этих процессов. Данная

модель оценки трудоемкости ПО условно делит негативные факторы на 2 типа:

– Факторы «трения».

– Переменные/динамические негативные факторы, которые замедляют разработку проекта влияя на членов команды, тем самым вызывая неравномерность скорости разработки проекта.

Проведение работы по оптимизации коэффициента эффективности разработки команды в рамках одной пользовательских истории улучшает точность расчета. Разберем негативные факторы более подробно.

Факторы «трения»: первый закон Ньютона гласит, что «каждый объект будет оставаться в покое или в состоянии равномерного прямолинейного движения, если не будет вынужден изменить свое состояние действием внешней силы». Силы, которые не продвигают ваш проект, замедляют его. Минимизируя силы, замедляющие проект, происходит уменьшение «трения», что увеличивает эффективность разработки проекта.

В процессе разработки ПО существуют многочисленные факторы, которые могут повлиять на эффективность разработки у вашей команды. Руководитель команды, менеджер проекта или исполнительным директор, должны минимизировать внешние силы, которые отрицательно влияют на эффективность команды. Факторы «трения» более или менее постоянны. Невозможно устранить все из них, но предоставляется возможность уменьшить многие из них.

Факторы «трения» включают:

– Компетентность команды. Для снижения негативного воздействия этого фактора важно правильно подобрать разработчиков в команду с необходимыми навыками разработки для текущего проекта.

– Процессы сборки, релизов, тестирования. Для снижения негативного воздействия этого фактора важно увеличить скорость выполнения данных этапов, произвести частичную автоматизацию данных процессов.

– Факторы окружения: различные отвлекающие факторы: шум, плохая вентиляция, плохое освещение, неудобная стулья и столы, плохое оборудование,

нестабильно работающее ПО и т.д.

– Динамика команды: некоторые участники команды могут не быть настолько хороши по сравнению с другими.

Как можно видеть, большинство сил «трения» в значительной степени являются проблемами окружения. И их последствия выражаются в долгосрочной перспективе. Но при этом, на них, как правило, можно влиять. Несмотря на то, что каждая из сил «трения» обычно выражает не значительное воздействие на эффективность разработки, в совокупности они могут оказать значительное влияние. Получение оптимального коэффициента приложенных усилий команды на разработку одной пользовательской истории требует, чтобы силы трения были устранены или влияние их было уменьшено.

В таблице 2.4 показаны четыре коэффициента трения с их значениями. Эти значения были скорректированы в соответствии с их степенью риска.

Таблица 2.4 – Коэффициенты «трения», влияющие на разработку ПП

Силы трения	**Стабильное значение**	**Переменное значение**	**Высокая изменчивость значения**	**Очень высокая изменчивость значения**
Компетентность команды	1	0.98	0.95	0.91
Степень автоматизации процессов	1	0.98	0.94	0.89
Факторы окружения	1	0.99	0.98	0.96
Динамичность команды	1	0.98	0.91	0.85

Показатель воздействия фактора трения, оказывающих негативное на разработку, рассчитывается как произведение всех четырех коэффициентов трения:

$$FR = \prod_{i=1}^{4} FF_i \qquad (2.4)$$

Переменные или динамические силы часто неожидаемые и непредсказуемые.

Они замедляют общий процесс разработки проекта и вызывают потерю эффективности.

Их последствия иногда драматичны, но при этом, чаще их влияние часто

бывает краткосрочным.

Второй закон Ньютона гласит, что «ускорение объекта, создаваемого чистой силой, прямо пропорционально величине чистой силы в том же направлении, что и чистая сила, и обратно пропорционально массе объекта». Переводя этот закон на плоскость оценки трудоемкости, следует рассматривать его с точки зрения производительности отдельных участников в команде или всей команды в целом.

Если невозможно устранить силу, которая уменьшает эффективность команды и замедляет процесс разработки, необходимо сделать все возможное, чтобы снизить ее воздействие и при этом, более предсказуемо.

Чем меньшее влияние оказывает фактор, тем больше его влияние станет предсказуемым, и, следовательно, приветит к увеличению эффективности разработчиков.

Динамические и переменные факторы включают:

– Изменения в составе команды: добавление участника, потеря участника, изменение ролей и обязанностей разработчиков.

– Новые инструменты, используемые в разработке: внедрение новых инструментов разработки, технологий, баз данных, языков программирования и т.д. Все это требует дополнительных затрат на обучение и тем самым вызывает временное снижение скорости разработки до тех пор, пока новый инструмент или технология не будет изучены.

– Дефекты сторонних инструментов для разработки/тестирования: дефекты в сторонних инструментах и ПО.

– Обязанности членов команды за пределами проекта: члены команды могут брать на себя дополнительные обязанности вне проекта, например, работать на полставки. Переключение между разработкой двух разных проектов может оказать существенное влияние на производительность разработки каждого из них.

– Личные проблемы: болеющий ребенок дома, личное здоровье, семейная жизнь и т.д.

– Проблема качества коммуникаций со стороной заказчика: сторона заказчика в процессе разработки проекта могут долго реагировать или не реагировать вовсе на запросы информации от проектной команды, тем самым создавая задержки в процессе разработки.

– Неясные требования: Отсутствие ясности или деталей в требованиях может сбивать с толку всех участников команды: бизнес-аналитиков, разработчиков и тестировщиков, и как часто бывает, вызывать изменения в уже сделанной работе.

– Изменение требований: для новых спецификация проекта могут потребоваться навыки и опыт, либо отсутствующие в команде, либо имеющийся опыт будет являться недостаточным. Чаще всего этот фактор взывает необходимость привлекать нового участника в команду, обладающим необходимым навыками, и конечно же, это будет негативно влиять на производительность команды вдвойне, в связи с тем, что изменение состава существующей команды, как показывает практика, также негативно влияет на ее производительность.

– Релокация команды на новое местоположение нарушает биоритмы человека (и оказывает прочие воздействия на физиологию человека), что также может влиять на продуктивность работы команды.

Коэффициенты для разного уровня значений динамических факторов представлены в таблице 2.5.

Таблица 2.5 – динамические факторы, влияющие на разработку ПО

Переменный фактор	**Нормальное значение**	**Высокое значение**	**Очень высокое значение**	**Экстремально высокое значение**
Изменения в составе команды	1	0.98	0.95	0,91
Обучение новым инструментам и технологиям	1	0.99	0.97	0,96
Дефекты сторонних инструментов	1	0.98	0.94	0,90
Обязанности членов команды за пределами проекта	1	0.99	0.98	0,98
Личные проблемы	1	0.99	0.99	0,98

Переменный фактор	Нормальное значение	Высокое значение	Очень высокое значение	Экстремально высокое значение
Проблемы коммуникаций	1	0.99	0.98	0,96
Неясные требования	1	0.98	0.97	0,95
Изменение требований	1	0.99	0.98	0,97
Релокация	1	0.99	0.99	0,98

Коэффициент влияние динамической факторов (DF) рассчитывается как произведение всех девяти динамических факторов (VF):

$$DF = \prod_{i=1}^{9} (VF)_i \quad (2.5)$$

Следующий коэффициент, при помощи которого можно оценить общее негативное влияние на эффективность команды в рамках одного цикла называется коэффициентом замедления. Замедление является результатом произведения факторов «трения» и динамических факторов, влияющих на эффективность и вычисляется по следующей формуле:

$$D = FR * DF \quad (2.6)$$

Заключительным этапом калибровки является расчет влияния коэффициента замедления на коэффициент скорости разработки в пределах одной итерации разработки.

$$\mathrm{V} = Vi^D \quad (2.7)$$

Откалибровав скорость одной команды в пределах одной итерации разработки с учетом факторов, негативно влияющих на разработку, можно рассчитать продолжительность, необходимую для завершения проекта, который будет характеризировать необходимые усилия (в днях) для выполнения определенного числа задач (пользовательских историй). Для этого используется формулу:

$$T = \frac{E}{V} * \frac{1}{\text{Количество рабочих дней в месяц}} \text{ (мес.)} \quad (2.8)$$

Ключевым показателями данной модели оценки трудоемкости разработки

ПО, при помощи которого можно рассчитать необходимое время на выполнение определенного списка задач (календарного плана), является скорость одной команды в пределах одной итерации разработки, расчет которого можно произвести при помощи формулы (2.3). Для того, чтобы модель давала более точную и адекватную оценку, учитывается влияние факторов «трения» и динамических факторов, негативно влияющих на трудоемкость одной команды на разработку одной итерации ПП, для чего используется формула (2.7). Основным показателем, который характеризует необходимые усилия на разработку всего ПП (в текущем его виде), является продолжительность, необходимая для его завершения, расчет которого можно произвести при помощи формулы (2.8).

Выводы по главе 2

Были рассмотрены и проанализированы метрики оценки трудоёмкости разработки ПО, применяемы в Scrum подходах разработки ПО.

Были проанализированы существующие метрики оценки трудоемкости разработки ПО, применяемые в классических моделях разработки ПО.

Проведено сравнение традиционных метрик оценки трудоемкости разработки ПО с метриками, использующимися в «Scrum»-методологиях разработки ПО.

Проведено обоснование недопустимости использования классических метрик оценки трудоемкости разработки ПО, которые используется в классических методах оценки трудоемкости разработки ПО.

Представлена разработанная модель оценки трудозатрат на разработку ПО на основе двух основных метрик, использующихся в Scrum модели разработки ПО: «размер пользовательской истории» и «сложность пользовательской истории».

ГЛАВА 3 АПРОБАЦИЯ МЕТОДА ОЦЕНКИ ТРУДОЕМКОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПРИ ИСПОЛЬЗОВАНИИ «SCRUM»-МЕТОДОЛОГИЙ

3.1 Разработка интеграции продукта «Оркестратор сетевых сервисов» компании ООО «НетКрэкер» с внешней системой Versa Director

В рамках исследования апробация разработанного метода оценки трудоемкости разработки ПО для программного проекта была применена на программном проекте ИТ предприятия ООО «Неткрэкер». Задачей программного проекта является интеграция одного из программных продуктов компании ООО «НетКрэкер», имеющего название «Оркестратор сетевых сервисов» с внешней системой Versa Director. Данный программный проект и задача интеграции с внешней системой Versa Director сформирована на основе пожеланий одного из постоянных заказников компании ООО «НетКрэкер», в связи с использованием этой внешней системы в рамках своих инструментов автоматизации введения деятельности предприятия.

Функциональные возможности программного продукта «Оркестратор сетевых функций»:

– извлечение фактической конфигурации сетевого оборудования;

– создание новой конфигурации для оборудования;

– применение запланированной версии конфигурации к оборудованию;

– откат к определенной версии конфигурации;

– сохранение истории изменений конфигурации;

– отчет о несоответствии, показывающий различия между двумя выбранными версиями конфигурации;

– поддержка нескольких протоколов с поддержкой конфигурации и извлечения;

– проверка правильности конфигурации;

– предоставление сетевых услуг через SDN-контроллеры (SD-WAN, T-SDN, DC-SDN, LAN-SDN).

Задачей будущей разработки является интеграция с внешней системой

Versa Director, которая является SD-WAN-контроллером, который используют операторы связи и телекоммуникаций по всему миру.

Пример графического интерфейса SD-WAN контроллера Versa Director представлен на рисунке 3.1.

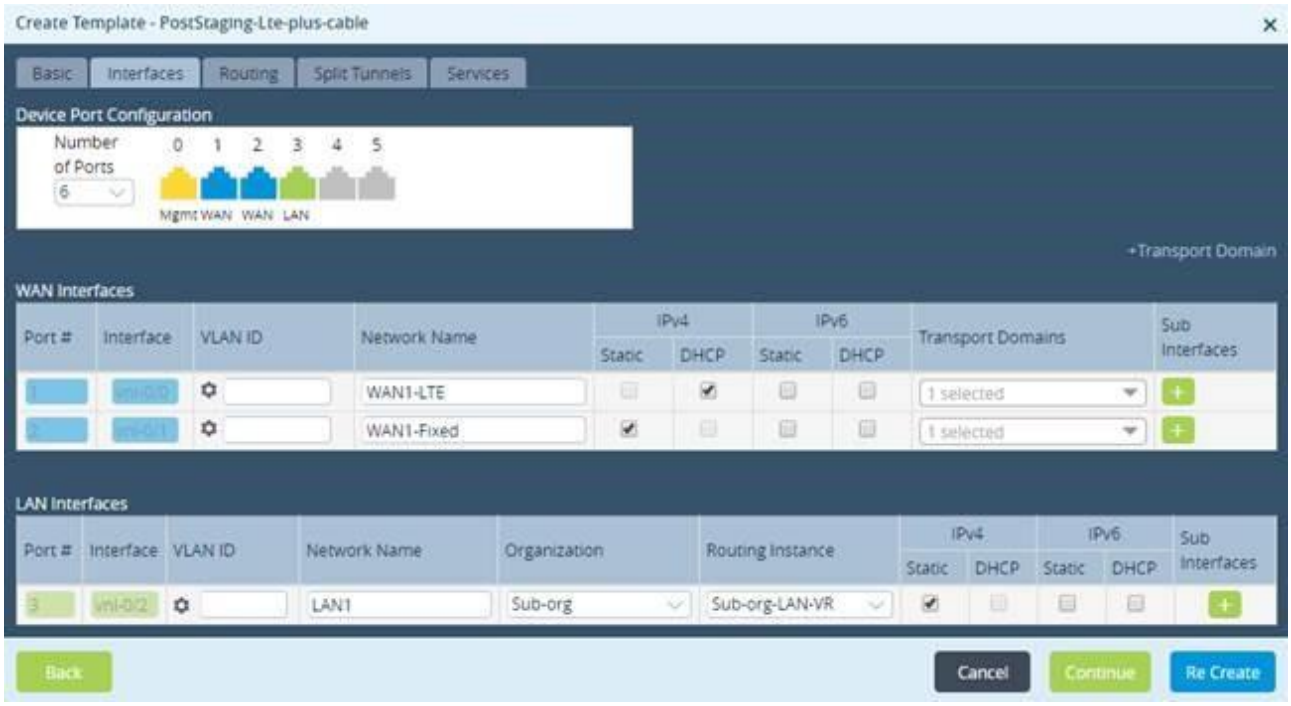


Рисунок 3.1 – Пользовательский интерфейс SD-WAN контроллера Versa Director

Для всех своих программных проектов компания ООО «НетКрэкер» использует Scrum подходы к управлению разработкой. Для текущей разработки была выбрана методология разработки Scrum.

В команду разработки входят функциональные специалисты, обладающие необходимыми навыками для работы. В структуре команды нет разделения на подразделения, выполняющие отдельные функции. Даже если отдельные члены команды владеют узкоспециализированными знаниями в различных областях, ответственность за разработку лежит на всей команде в целом.

Идентификация предварительных требований к новому ПП –это то, что позволяет определить характеристики нового ПП в соответствии с требованиями заказчика. Команда, которая занимается проектом, ставит определенные цели, соответствующие либо каким-то чертам продукта конкурента, либо требованиям

рынка, либо идентифицируют какие-то признаки, которые конечный пользователь хотел бы добавить к уже существующему ПП. Проблемы, которые приходится решать специалистам в процессе создания ПО, не всегда ясны. В частности, трудно чётко описать те действия, которые должна выполнять система. Описание функциональных возможностей и ограничений, накладываемых на систему, называется требованиями к этой системе. Требования к ПП должны быть установлены таким образом, что могло бы гарантировать их адекватность и верный «перевод» с языка пользователя. В первую очередь необходимо выявить все возможные требования, предъявляемые к будущей разработке. Выявить их можно с помощью анализа требований-процесса сбора требований к ПО, их систематизации, документирования, анализа, выявления противоречий, неполноты, разрешения конфликтов в процессе разработки ПО. В процессе сбора требований важно принимать во внимание возможные противоречия требований различных заинтересованных лиц, таких как заказчики разработки, сами разработчики, а также конечные пользователи. Анализ требований включает следующие шаги:

– сбор требований: общение с клиентами и пользователями, чтобы определить, каковы их требования;

– анализ требований: определение, являются ли собранные требования неясными, неполными, неоднозначными, или противоречащими, и затем решение этих проблем;

– документирование требований: требования могут быть задокументированы в различных формах, таких как простое описание, сценарии использования, пользовательские истории, или спецификации процессов.

Таким образом, первым делом была проведена работа с заказчиком системы. После этого был составлен список выявленных пользовательских требований (требования, формулируемые пользователями к конечному программной разработке), на которые будет полагаться разработка приложения:

– возможность создавать организацию (базовая сущность сетевой инфраструктуры);

– создавать сетевые сегменты с VPN и Интернетом;
– создавать добавлять/удалять сетевые сегменты из VPN;
– создавать привязку IP адресов;
– удалять привязку IP адресов;
– создавать проброс сетевых портов;
– удалить проброс сетевых портов;
– изменять существующие WAN/LAN интерфейсы;
– добавить и удалить антивирус в сетевом сегменте;
– добавить и удалить URL-фильтрацию в сетевом сегменте;
– создать настройки безопасности в сетевом сегменте;
– изменять настройки безопасности в сетевом сегменте;
– получать список пользовательских устройств;
– временно отключать сетевые сегменты.

3.2 Выделение входных параметров модели оценки трудоемкости и выбор их значений для реализации интеграции «Оркестратор сетевых сервисов» с внешней системой Versa Director

Применение предложенной модели для определения трудозатрат на разработку будущего ПП произведено в рамках будущего фронта работ (14 пользовательских историй) разработки интеграции внутренней продуктовой разработки компании ООО «НетКрэкер» под названием «Оркестратор сетевых сервисов» с внешней системой Versa Director. Разработка интеграции будет выполнена одной Scrum командой в составе 9 человек, что соответствует рекомендованному числу участников, а также делает возможность использования предложенной модели оценки трудоемкости в данном исследовании. Основными входными параметрами предложенной модели для определения трудозатрат являются:

– количество пользовательских историй;
– количество дней в одной итерации разработки.

Сотрудники компании ООО «НетКрэкер» имеют 40 часовую рабочую

неделю, что соответствует 5 рабочим дням в неделю. Учитывая вышесказанное, можно определить значения для основных входных параметров предложенной модели для оценки трудоемкости разработки ПП. Определение значений для основных входных параметров модели оценки трудоемкости разработки ПП представлены в таблице 3.1.

Таблица 3.1 – значения входных параметров модели оценки трудозатрат

Входной параметр модели	**Значение**
Количество пользовательских историй	14
Количество дней в одной итерации разработки	10

Также для определения трудозатрат на разработку, используя предложенную модель, необходимо указать значения метрикам размера и сложности пользовательских историй, а также значения метрикам факторов, негативно влияющих на разработку. Значения метрикам необходимо подбирать согласно рекомендациям, приведенной во второй главе. Значения для метрик размера каждой пользовательской истории приведены в таблице 3.2.

Таблица 3.2 – значения метрики размера пользовательских историй

Название пользовательской истории	**Размер пользовательской истории**
Я как Сетевой Инженер хочу создать организацию	3
Я как Сетевой Инженер хочу создать сетевые сегменты с VPN и Интернетом	3
Я как Сетевой Инженер хочу добавлять/удалять сетевые сегменты из VPN	2
Я как Сетевой Инженер хочу создавать привязку IP адресов	3
Я как Сетевой Инженер хочу удалить привязку IP адресов	3
Я как Сетевой Инженер хочу создавать проброс сетевых портов	2
Я как Сетевой Инженер хочу удалить проброс сетевых портов	2
Я как Сетевой Инженер хочу изменять WAN/LAN интерфейсы	2
Я как Сетевой Инженер хочу добавить/удалить Антивирус	4
Я как Сетевой Инженер хочу добавить/удалить URL-фильтрацию	4
Я как Сетевой Инженер хочу создать настройки безопасности	5
Я как Сетевой Инженер хочу изменять настройки безопасности	3
Я как Сетевой Инженер хочу получить список пользовательских девайсов	2
Я как Сетевой Инженер хочу временно отключать сетевые сегменты	3

Распределение всех пользовательских историй по их размеру в долевом

виде, по отношению всего плана работа, можно увидеть на рисунке 3.2.

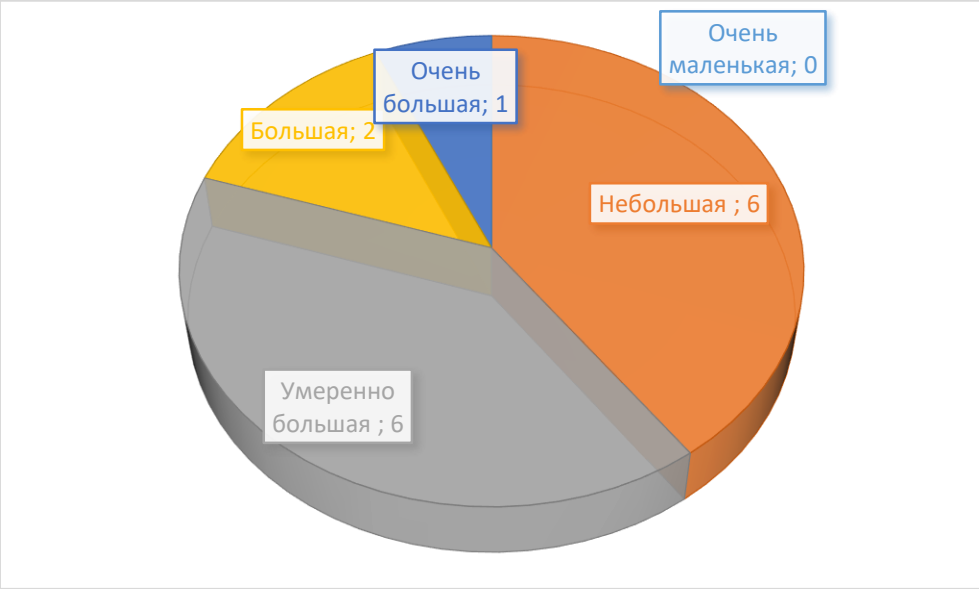


Рисунок 3.2 – Распределение пользовательских историй по коэффициенту размера

Как можно видеть, основное количество пользовательских историй, это умеренно большие и небольшие. В процессе сессии первого планирования команда разработки изучила необходимые требования, сформированные в пользовательских историях.

Также, в процессе планирования рассматривалась возможность разбить одну большую пользовательскую историю на более мелкие, связанную с настройкой безопасности в сетевом сегменте, но это не представилось возможным в связи, как сложностью самой разработки, так и ее спецификой.

В таблице 3.3 приведены значения для метрики сложности пользовательских историй.

Распределение всех пользовательских историй по их сложности в долевом виде, по отношению всего фронта работа, можно увидеть на рисунке 3.3.

В команду разработки входят функциональные специалисты, обладающие необходимыми навыками для работы. Не смотря на хороший общий показатель компетентности команды, существует «динамичность команды», которая выражается в том, что 2 из участников команды являются «младшими разработчиками», имеющие совсем небольшой опыт, а также в команде

разработки один старший разработчик с очень большим опытом.

Таблица 3.3 – значения метрики сложности пользовательских историй

Название пользовательской истории	**Сложность пользовательской истории**
Я как Сетевой Инженер хочу создать организацию	4
Я как Сетевой Инженер хочу создать сетевые сегменты с VPN и Интернетом	3
Я как Сетевой Инженер хочу добавлять/удалять сетевые сегменты из VPN	2
Я как Сетевой Инженер хочу создавать привязку IP адресов	4
Я как Сетевой Инженер хочу удалить привязку IP адресов	3
Я как Сетевой Инженер хочу создавать проброс сетевых портов	2
Я как Сетевой Инженер хочу удалить проброс сетевых портов	1
Я как Сетевой Инженер хочу изменять WAN/LAN интерфейсы	3
Я как Сетевой Инженер хочу добавить/удалить Антивирус	4
Я как Сетевой Инженер хочу добавить/удалить URL-фильтрацию	3
Я как Сетевой Инженер хочу создать настройки безопасности	3
Я как Сетевой Инженер хочу изменять настройки безопасности	5
Я как Сетевой Инженер хочу получить список пользовательских девайсов	4
Я как Сетевой Инженер хочу временно отключать сетевые сегменты	3

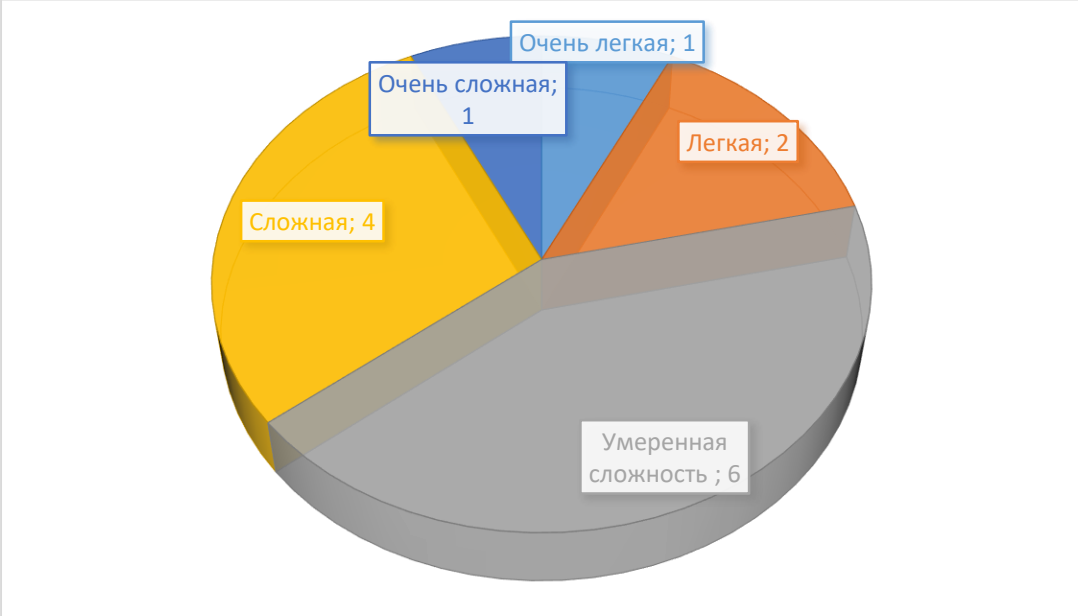


Рисунок 3.3 – Распределение пользовательских историй по коэффициенту сложности

Также, нужно отметить, сотрудники имеют хорошие условия труда, в частности, удобные кресла и столы, разработчики имеют по 2 монитора,

помещение для работы является хорошо проветриваемым. Все это является благоприятным фактором для разработки.

К сожалению, на текущем этапе разработки продукта часть процессов не является автоматизированными или плохо автоматизированными, в частности сборка исходного кода из репозиториев может происходить достаточно долго, также не текущий момент не автоматизирован процесс развертывания программного продукта.

Учитывая вышесказанное, заполним таблицу 3.4, указав в ней значения для метрик факторов «трения» программной разработки.

Таблица 3.4 – значения метрик факторов «трения» процесса разработки ПП

Фактор «трения» процесса разработки ПП	**Значение**
Компетентность команды	0,95
Процесс автоматизации	0,89
Факторы окружения	0,96
Динамичность команды	0,85

Команда разработки является достаточно постоянной. Учитывая, что требований к будущей разработки возникает немного, можно с уверенностью говорить, что изменений в составе команды разработки не планируется.

При этом, не смотря на высокую квалификацию команды разработки, их участникам на текущий момент не знаком SD-WAN контроллер Versa Director. Обязанностей за пределами текущей разработки команда разработки не имеет, кроме одного участника, который работает на полставки.

Требования к будущей программной разработке ясны, в связи с этим больших изменений требований со стороны заказчика программной разработки не ожидается Проанализировав динамические факторы разработки.

Учитывая выше сказанное, заполним таблицу 3.5, в которой будут приведены значения для метрик динамических факторов, возникающих в процессе разработки ПП.

Стоит указать, что размер команды не является входным параметров для предложенной модели оценки трудоемкости разработки ПП.

Таблица 3.5 – значения метрик динамических факторов процесса разработки ПП

Динамический фактор процесса разработки ПП	Значение
Изменения в составе команды	0,98
Обучение новым инструментам и технологиям	0,93
Дефекты сторонних инструментов	0,94
Обязанности членов команды за пределами проекта	0,98
Личные проблемы	0,96
Проблемы коммуникаций	0,96
Ясность требований	0,98
Изменение требований	0,97
Релокация	0,98

Это связано с тем, что модель расчета уже учитывает рекомендованное значение размера одной Scrum команды. Рекомендованное количество состава одной Scrum команды – 7-10 человек. Текущая внутренняя продуктовая разработка разработается силами 9 человек, из которых один инженер по тестированию ПО, два бизнес-аналитика и 6 разработчиков ПО.

3.3 Расчет трудоемкости разработки интеграции «Оркестратора сетевых функций» с внешней системой Versa Director

Используя метрики сложности и размера пользовательской истории, вычислим коэффициент усилия, необходимый одной команде для реализации конкретной пользовательской истории. Для вычисления коэффициента усилия на одну пользовательскую историю, необходимо использовать формулу 2.1 и данные из таблицы 3.2 и таблицы 3.3. Расчетные данные приведены в таблице 3.6.

Используя показатель Коэффициент усилия из таблицы 3.6 и формулу 2.2, вычислим общий коэффициент необходимых трудозатрат, который выражает общие усилия определенной команды на разработку, которые будут затрачены на протяжении всего проекта:

$$E = \sum_{i+1}^{n} ES_i = 133$$

Таблица 3.6 – расчет коэффициента усилия для пользовательской истории

Название пользовательской истории	**Сложность пользовательской истории**	**Размер пользовательской истории**	**Коэффициент усилия ES_i**
Я как Сетевой Инженер хочу создать организацию	4	3	12
Я как Сетевой Инженер хочу создать сетевые сегменты с VPN и Интернетом	3	3	9
Я как Сетевой Инженер хочу добавлять/удалять сетевые сегменты из VPN	2	2	4
Я как Сетевой Инженер хочу создавать привязку IP адресов	4	3	12
Я как Сетевой Инженер хочу удалить привязку IP адресов	3	3	9
Я как Сетевой Инженер хочу создавать проброс сетевых портов	2	2	4
Я как Сетевой Инженер хочу удалить проброс сетевых портов	1	2	2
Я как Сетевой Инженер хочу изменять WAN/LAN интерфейсы	3	2	6
Я как Сетевой Инженер хочу добавить/удалить Антивирус	4	4	16
Я как Сетевой Инженер хочу добавить/удалить URL-фильтрацию	3	4	12
Я как Сетевой Инженер хочу создать настройки безопасности	3	5	15
Я как Сетевой Инженер хочу изменять настройки безопасности	5	3	15
Я как Сетевой Инженер хочу получить список пользовательских девайсов	4	2	8
Я как Сетевой Инженер хочу временно отключать сетевые сегменты	3	3	9

При помощи формулы 2.3, используя общий коэффициент необходимых трудозатрат, а также входной параметра длительности одной итерации разработки, вычислим предполагаемую среднюю скорость разработки одной пользовательской истории для одной Scrum-команды на протяжении одной

итерации разработки, без учета влияния факторов, негативно влияющих на разработку:

$$Vi = \frac{E}{\text{Длительной одной итерации разработки в днях}} = \frac{133}{10} = 13{,}3$$

После того, как вычислен коэффициент скорости разработки конкретной пользовательской истории конкретной командой разработчиков, необходимо откалибровать этот показатель, учитывая негативные факторы, которые оказывают негативное влияние на эффективность разработки, а именно:

– факторы трения;
– переменные динамические факторы.

Для проведения работы по оптимизации коэффициента скорости разработки V_i, рассчитаем воздействие факторов трения и переменных и динамических факторов, негативно влияющих на разработку. Для расчета показателя влияния факторов трения (FR) воспользуемся формулой 2.4 и входными данными, которые приведены в таблице 3.4

$$FR = \prod_{i=1}^{4}(FF)_i = 0{,}95 * 0{,}89 * 0{,}96 * 0{,}85 = 0{,}69$$

Для расчета показателя негативного влияния динамических факторов разработки на процесс разработки ПО (DF), воспользуемся формулой 2.5 и входными данными, которые приведены в таблице 3.5.

$$DF = \prod_{i=1}^{9}(VF)_i = 0{,}98 * 0{,}93 * 0{,}94 * 0{,}98 * 0{,}96 * 0{,}96 * 0{,}98 * 0{,}97 * 0{,}98 = 0{,}72$$

Зная показания негативного влияния каждого из факторов, используя формулу 2.6, вычислим коэффициент замедления, который характеризует показатель общего негативного влияния на эффективность команды и выражается в произведении каждого из негативных факторов:

$$D = FR * DF = 0{,}69 * 0{,}72 = 0{,}49$$

Учитывая коэффициент замедления команды, используя формулу 2.7,

произведем калибровку коэффициента скорости разработки одной команды с учетом влияния негативных факторов:

$$\mathrm{V} = Vi^{D} = 13{,}3^{0{,}49} = 3{,}55$$

Имея откалиброванный коэффициент скорости разработки одной пользовательской истории, становится возможным рассчитать продолжительность, необходимую для завершения всех пользовательских историй. Рассчитаем продолжительность времени разработки, используя формулу 2.8.

$$T = \frac{E}{V} = \frac{133}{3{,}55} = 46 \text{ дней} \approx 37 \text{ дней}$$

Т.к. в текущей продуктовой разработке длинна одной итерации разработки равна 10 дней, то для реализации всего проекта потребуется 4 таких итерации, что будет являться оптимальным и достаточным для завершения проекта с учетом возможных рисков, которые могут возникнуть в процессе разработки.

Реальная разработка проекта продолжалась 39 дней, тем самым разница между рассчитанным и реальным временем выполнения составила 5,1 %, при этом предполагаемое количество итераций и реальное количество итераций является эквивалентным, благодаря чему можно говорить об адекватной оценке и возможности применения этой модели на других аналогичных проектах, использующих Scrum подходы для управления своих проектов.

В модели могут быть определенные недостатки, поэтому автор сохраняет надежду, что модель, выдвинутая в этой работе, сможет вызвать дальнейшие обсуждения и исследовании в направлении ее улучшении.

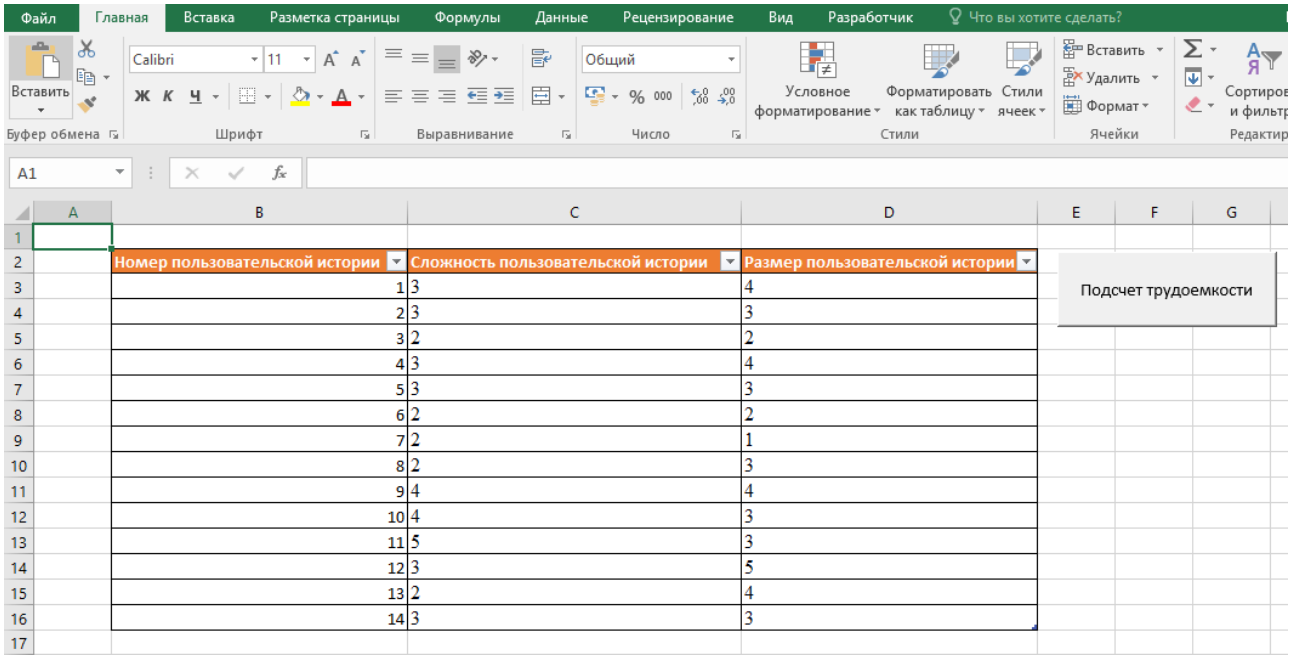
3.4 Инструмент оценки трудоемкости разработки ПО при использовании методологии «Scrum»

В рамках апробации разработанного метода оценки трудоемкости разработки ПО при использовании методологии «Scrum» был разработан инструмент автоматизации данного процесса. Данный инструмент автоматизации использует математическую модель из предложенного метода, а

также путем запроса входных параметрах математической модели, полностью автоматизирует весь метод оценки трудоемкости.

Инструмент автоматизации оценки трудоемкости разработки ПО при использовании методологии Scrum был разработан с использованием VBA-макроса в рабочей книге офисного продукта Microsoft Excel. VBA и офисный продукт Microsoft Excel был выбран по причине того, что при помощи VBA достаточно легкого и не несет больших трудозатрат написать программный код, который может построить различные графики и диаграммы. Все необходимые модули для построение различных график и диаграмм уже включены в офисные продукты Microsoft, в частности в Microsoft Excel.

Первым этапом использования инструмента автоматизации оценки трудоемкости является заполнение таблицы на листе «Main», которая характеризует сложность и размер каждой пользовательской истории (в пределах от 1 до 5), согласно рекомендациям, которые даны в разработанном методе оценки трудоемкости (см. рисунок 3.4).



Номер пользовательской истории	Сложность пользовательской истории	Размер пользовательской истории
1	3	4
2	3	3
3	2	2
4	3	4
5	3	3
6	2	2
7	2	1
8	2	3
9	4	4
10	4	3
11	5	3
12	3	5
13	2	4
14	3	3

Рисунок 3.4 – Пример заполнения таблицы с характеристиками сложности и размера каждой пользовательской истории

На основном листе «Main» рабочей книги Excel также имеется кнопка «Подсчет трудоемкости», которая является точкой входа в разработанный VBA-

макрос. При нажатии кнопки «Подсчет трудоемкости» вызывается процедура effortEstimationAgile() VBA-макроса, которая вызывает пользовательскую форму.

На пользовательской форме представлены основные поля ввода, данные которых необходимы для разработанной математической модели оценки трудоемкости разработки ПО для Scrum-методологии. Для удобства восприятия, 2 основных параметра модели «Количество пользовательских историй» и «Длинна одной итерации разработки» вынесены наверх и обособлены. Также, группа негативных факторов: «Факторы трения» и «Динамические факторы» обособлены в разные экранные фреймы на пользовательской форме. Пример заполненной данными пользовательской формы можно увидеть на рисунке 3.5.

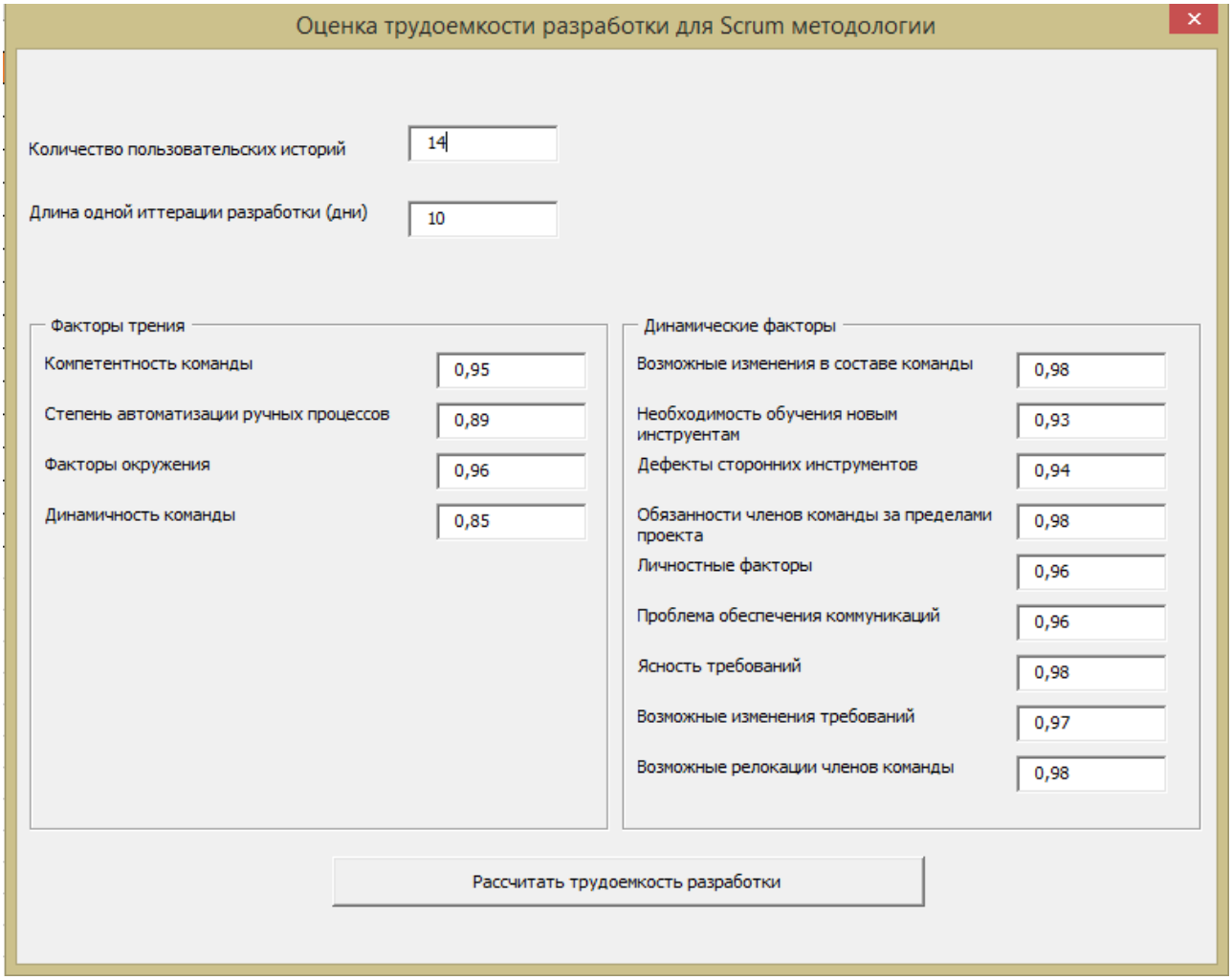


Рисунок 3.5 – Пользовательская форма инструмента оценки трудоемкости разработки ПО для Scrum методологии

После заполнения всех полей ввода, необходимых для использования разработанной математической модели, предполагается, что пользователь нажмет кнопку «Рассчитать трудоемкость разработки». При нажатии кнопки «Рассчитать трудоемкость разработки» вызывается процедура CalculateButton1_Click(), которая включает в себя следующие действия:

– расчет трудоемкости разработки для Scrum-методологии;

– процедура создания «технического» листа рабочей книги Microsoft Excel, а также листа с результатами, включающая в себя построение график прогнозируемой оценки на основе выполненных расчетов, с возможностью наложить на нее линию с реальными результатами, данные которых можно будет ввести после окончания каждой итерации разработки или всего программного проекта.

Результатом работы выполнения процедуры CalculateButton1_Click() будет являться:

– построенный график на основе выполненных расчетов в этой процедуре;

– пустая таблица, которая будет заполняться пользователем.

Построенный график, а также таблица для ручного заполнения пользователем будут созданы на результирующем листе «Result» рабочей книги Microsoft Excel.

Строки первой колонки построенной таблицы перечисляют номера Строки второй колонки построенной таблицы будут обозначать количество завершенных пользовательских историй для соответствующей итерации разработки.

Результат выполнения процедуры CalculateButton1_Click() разработанного VBA-макроса можно увидеть на рисунке 3.6.

После заполнения 2-ой колонки построенной таблицы, происходит наложение результирующий линии на прогнозирующую оценку, которая была построена при помощи VBA-макрос.

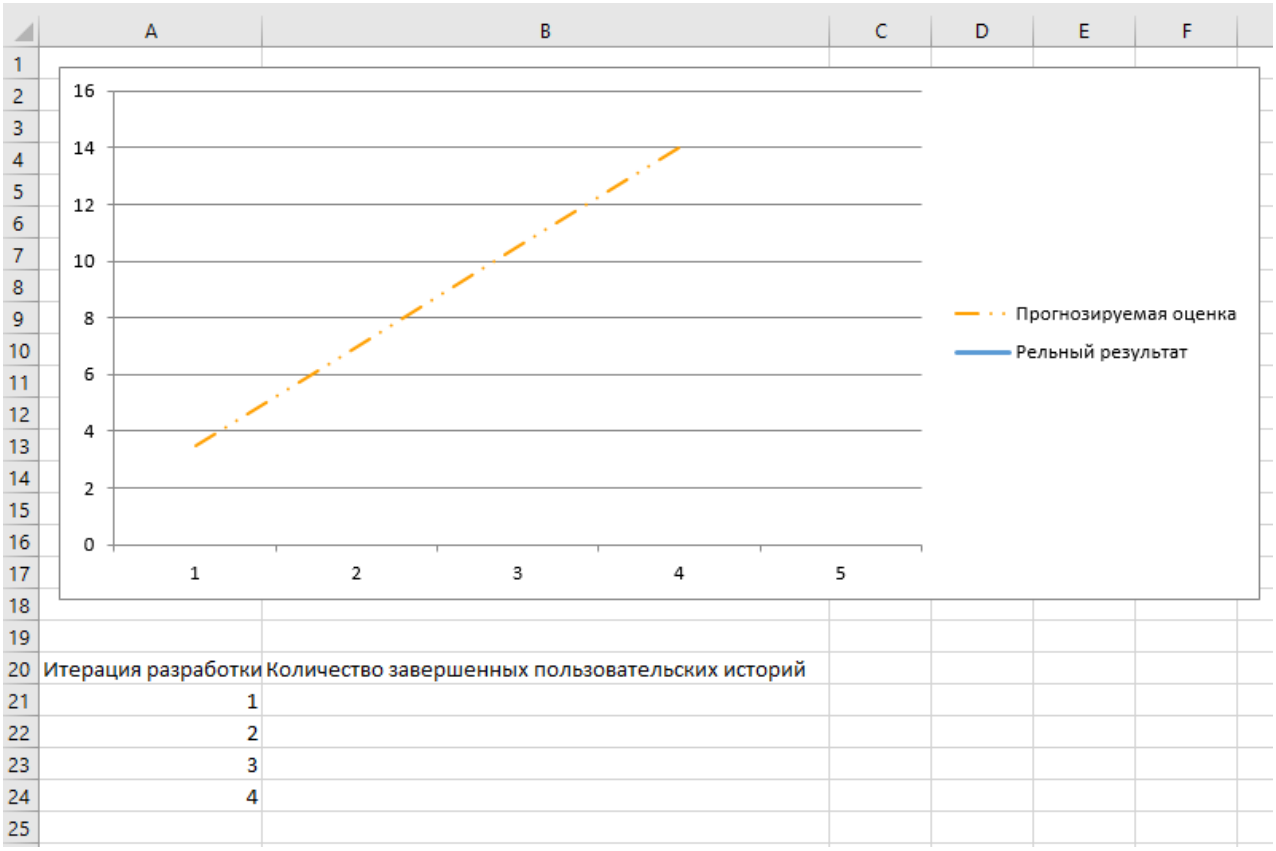
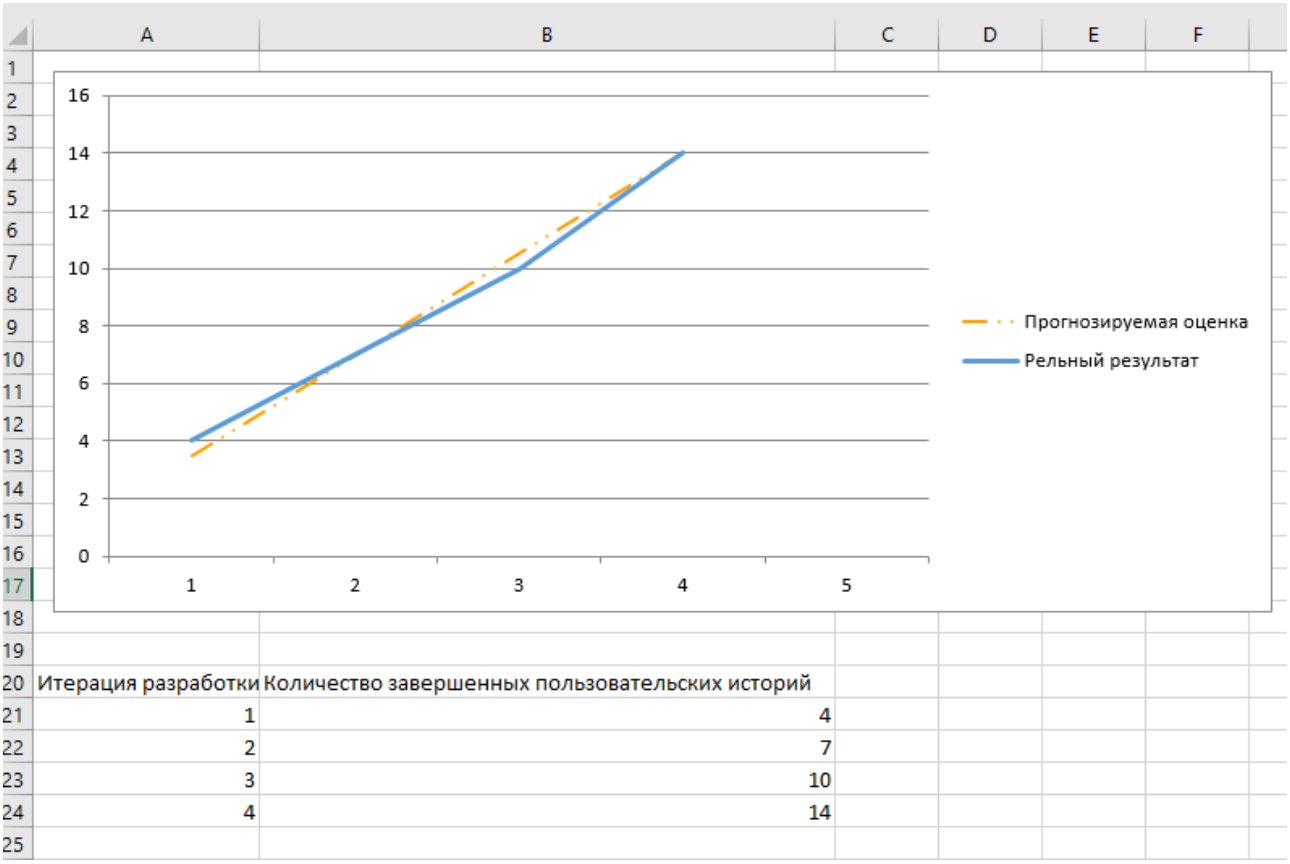


Рисунок 3.6 – График прогнозируемой оценки трудоемкости разработки ПО для методологии Scrum.

Наложенная линия будет показывать соответствие прогноза оценки трудоемкости разработки для каждой итерации разработки с реальными данными, которые введет пользователь во вторую колонку построенной таблицы. Тем самым, на графике можно будет увидеть отставание или же опережение календарного графика.

Пример заполнения таблицы реальными данными и наложения линии реальных данных на прогнозирующую линию (линию тренда) можно увидеть на рисунке 3.7.

Наложение линии реального результата на прогнозируемую оценку завершения программного проекта показывает, что все запланированное на разработку пользовательские истории были разработаны в срок, в пределах четырех итераций разработки.



Итерация разработки	Количество завершенных пользовательских историй
1	4
2	7
3	10
4	14

Рисунок 3.7 – Соответствие прогнозируемой оценки и реального результата на графике оценки трудоемкости разработки ПО для методологии Scrum

Листинг основной процедуры CalculateButton1_Click() разработанного VBA макроса представлен в Приложении А.

График, представленный на рисунке 3.7 доказывает эффективность разработанного метода оценки трудоемкости разработки ПО при использовании Scrum методологии и является подтверждением гипотезы данного исследования.

Выводы по главе 3

Произведена апробация разработанной модели в рамках продуктовой разработки компании ООО «НетКрэкер».

Были выделены основные входные метрики используемой модели для оценки трудоемкости разработки ПО для одной из продуктовых разработок компании ООО «НетКрэкер», разработка которой происходит при помощи

одной из Scrum методологии, а также определены и подобраны значения к этим метрикам.

Произведен расчет трудоемкости разработки ПО, используя входные значения модели, выраженный во времени и количестве необходимых итераций разработки, которое необходимо для завершения проекта.

Произведена автоматизация расчета трудоемкости разработки ПО с использованием VBA-макроса офисного пакета Microsoft Excel на примере рабочей книги Microsoft Excel. На основе построенных графиков инструментом автоматизации и последующего наложения линии реального результата доказана гипотеза данного исследования.

На основе построенных графиков инструментом автоматизации расчета трудоемкости разработки ПО для Scrum методологии произведено сравнение прогнозируемых данных с данными, которые были получены в процессе разработки ПП.

ЗАКЛЮЧЕНИЕ

В связи с популяризацией Agile-подходов для разработки ПО на данный момент существует небольшое количество моделей оценки трудозатрат на разработку сложной КИС, в частности для методологии разработки Scrum. Оценку, которую можно получить при использовании этих инструментов, является одним из основных аргументов при ТЭО программных модулей и систем для разработчиков (исполнителей) КИС. При этом, существующим моделям открыт потенциал для развития для повышения точности результата в текущих экономических реалиях мира. Автором работы была разработана новая модель оценки трудоемкости ПО при использовании Scrum-подходов при разработке КИС.

В результате работы автором достигнуты цели, поставленные во введении, а также решены следующие задачи:

– Исследование существующих моделей оценки трудоемкости разработки ПП, которые применялись для оценки трудоемкости при разработке по «водопадной» методологии и обоснована невозможность их применения для разработки по Scrum-методологиям.

– Определение набора характеристик, а также негативных факторов, которые позволяют формализовать методы анализа этих метрик, а также делают поправку на негативные факторы.

– Разработана модель, которая способна адекватно оценить трудоемкость разработки КИС для проектов, использующих Scrum-подходы, основываясь лишь на двух основных векторах, имеющихся на старте проекта или цикла разработки, и которую возможно применить для ТЭО проекта разработки КИС вне зависимости от специфики проекта и предметной области разработки.

– Проведение эксперимента для подтверждения возможности применения разработанной методики оценки трудоемкости на реальных данных.

В результате выполнения работы были рассмотрена, научно-обоснована и применена методология оценки общих трудозатрат разработки КИС на основе формализованной информации о историях пользователей, в которых

учитываются все специфичные требования заказчика. Расчеты и эксперимент показывают, что предложенная методика допустима и позволяет достаточно легко получить адекватную оценку трудозатрат на весь будущий проект или же для определенного временного интервала.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Научная и методическая литература

1. Белладжио, Д. Разработка программного обеспечения. Управление изменениями. [Текст] / Белладжио Д., Миллиган Т. – М.: ДМК Пресс, 2016. – 384с.

2. Благодатских, В.А. Экономико-правовые основы рынка программного обеспечения. [Текст] / Благодатских В.А., Середа С.А., Поскакалов К.Ф. – М.: Финансы и статистика, 2007. – 240с.

3. Васильев, Р.Б. Управление развитием информационных систем. [Текст] / Васильев Р.Б., Калянов Г.Н., Левочкина Г.А. – М.: Горячая Линия - Телеком, 2009. – 378с.

4. Вендров, А.М. Практикум по проектированию программного обеспечения экономических информационных систем. [Текст] – М.: Финансы и статистика, 2006. – 192с.

5. Верещагина, Е.А. Корпоративные информационные системы. Учебно-методический комплекс. [Текст] – М.: Проспект, 2015. – 104с.

6. Вигерс, К. Разработка требований к программному обеспечению. [Текст] / Вигерс Карл, Битти Джой – М.: Русская Редакция, БХВ-Петербург, 2016. – 736с.

7. Волкова, В.Н. Теория информационных процессов и систем. Учебник и практикум. [Текст] – М.: Юрайт, 2016. – 504с.

8. Гагарин, А.Г. Оценка динамической полезности программного обеспечения. [Текст] / Гагарин А.Г., Рогачев А.Ф. – М.: LAP Lambert Academic Publishing, 2011. – 204с.

9. Гвоздева, В.А. Основы построения автоматизированных информационных систем. [Текст] / Гвоздева В.А., Лаврентьева И.Ю. – М.: Форум, Инфра-М, 2009. – 320с.

10. Гусятников, В.Н. Стандартизация и разработка программных систем. – [Текст] / Гусятников В.Н., Безруков А.И. М.: Финансы и статистика, Инфра-М, 2010. – 288с.

11. Голицына, О.Л. Программное обеспечение. – [Текст] / Голицына О.Л., Партыка Т.Л., М.: Форум, 2010. – 448с.

12. Дамодаран, А. Инвестиционная оценка. Инструменты и методы оценки любых активов. – [Текст] М.: Альпина Паблишер, 2016. – 1316с.

13. Долженко, А. Нечеткие модели оценки качества информационных систем. – М.: LAP Lambert Academic Publishing, 2011. – 300с.

14. Еникеев, А. Специализированные модели для разработки информационных систем. – [Текст] / Еникеев А., Бендума Т. М.: LAP Lambert Academic Publishing, 2011. – 104с.

15. Жданов, С.А. Информационные системы. Учебник. [Текст] / Жданов С.А., Соболева М.Л., Алфимова А.С. – М.: Прометей, 2015. – 302с.

16. Затенко, С. Математические модели надежности программного обеспечения. [Текст] – М.: LAP Lambert Academic Publishing, 2011. – 152с.

17. Кон М, Agile: оценка и планирование проектов – [Текст] М.:Альпина Паблишер, 2018. – 245с.

18. Корнипаев, И. Требования для программного обеспечения. Рекомендации по сбору и документированию. [Текст] – М.: Книга по Требованию, 2013. – 118с.

19. Липаев, В.В. Технико-экономическое обоснование проектов сложных программных средств. [Текст] – М.: Синтег, 2004. – 284с.

20. Мартишин, С.А. Основы теории надежности информационных систем. Учебное пособие. [Текст] / Мартишин С.А., Симонов В.Л., Храпченко М.В. – М.: Форум, Инфра-М, 2013. – 256с.

21. Мишенин, А.И. Теория экономических информационных систем. – М.: Финансы и статистика [Текст], 2007. – 240с.

22. Моисеева, Т.В. Экономические и правовые основы рынка программного обеспечения. [Текст] / Моисеева Т.В., Полукаров Д.Ю. – М.: Солон-Пресс, 2008. – 224с.

23. Набатова, Д.С.. Математические и инструментальные методы поддержки принятия решений. Учебник и практикум. [Текст] – М.: Юрайт, 2015.

– 292с.

24. Олейник, П.П. Корпоративные информационные системы. [Текст] – СПб.: Питер, 2012. – 176с.

25. Олейник, П.П. Основные стандарты корпоративных информационных систем. [Текст] – М.: LAP Lambert Academic Publishing, 2011. – 88с.

26. Олейник, П.П. Методика построения трехзвенных объектно-ориентированных приложений. [Текст] – М.: LAP Lambert Academic Publishing, 2011. – 204с.

27. Орлов, С.А. Технологии разработки программного обеспечения. [Текст] – СПб.: Питер, 2012. – 608с.

28. Рубин, К.С. Основы Scrum. Практическое руководство по гибкой разработке ПО. [Текст] – М.: Вильямс, 2016. – 544с.

29. Семенов, С.С. Методы принятия решений в задачах оценки качества и технического уровня сложных технических систем. [Текст] / Семенов С.С., Воронов Е.М., Полтавский А.В., Крянев А.В. – М.: Ленанд, 2016. – 520с.

30. Рудаков, А.В. Технология разработки программных продуктов. Практикум. [Текст] / Рудаков А.В., Федорова Г.Н. – М.: Академия, 2010. – 192с.

31. Федорова, Г.Н. Информационные системы. Учебник. [Текст] – М.: Academia, 2013. – 208с.

32. Черников, Б.В. Оценка качества программного обеспечения. [Текст] / Черников Б.В., Поклонов Б.Е.. – М.: Инфра-М, Форум, 2013. – 400

Электронные ресурсы

33. Кент, Б. Agile-манифест разработки программного обеспечения, 2001 [Электронный ресурс]: https://agilemanifesto.org/iso/ru/manifesto.html

Литература на иностранном языке

34. A. Abran, Software Project Esstimation. Willey-IEEE Computer Society Press, 2015.

35. M. Ali, Z. Shaikh, and E. Ali, “Estimation of Project Size Using User Stories,” in The International Conference on Recent Advances in Computer Systems, 2015.

36. E. Coelho and A. Basu, “Effort Estimation in Agile Software Development using Story Points,” Int. J. Appl. Inf. Syst., vol. 3, no. 7, pp. 7–10, 2012.

37. M. Usman and R. Britto, “Effort Estimation in Co-located and Globally Distributed Agile Software Development: A Comparative Study,” in 2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2016, pp. 219–224.

38. M. Usman, E. Mendes, F. Weidt, and R. Britto, “Effort estimation in agile software development: A Systematic Literature Review,” in Proceedings of the 10th International Conference on Predictive Models in Software Engineering - PROMISE ’14, 2014, pp. 82–91.

39. S.Ziauddin, T.Kamal and Z.Shahrukh, “An Effort Estimation Model for Agile Software Development”, Adv. Comput. Sci. its Appl., vol. 2, no. 1, pp. 314–324, 2012.

ПРИЛОЖЕНИЕ А

Листинг. **Процедура CalculateButton1_Click() VBA-макроса**

```
Private Sub CalculateButton1_Click()
    Dim sumEffortLevel As Integer
    Dim rawVelocity As Double
    Dim velocity As Double

    Dim frictionNegFactor As Double: frictionNegFactor = 1
    Dim dynamicNegFactor As Double: dynamicNegFactor = 1
    Dim allNegFactor As Double

    Dim requiredDevelopDays As Integer
    Dim requiredDevelopTimeSprints As Integer

    Worksheets("Main").Activate

    '1 step
    For rowNum = 0 To UserStoriesCount.Value - 1
        userStoryDifficult = CInt(Cells(rowNum + 3, 3).Value)
        userStorySize = CInt(Cells(rowNum + 3, 4).Value)
        userStoryStoryPoint = userStoryDifficult * userStorySize
        sumEffortLevel = sumEffortLevel + userStoryStoryPoint
    Next rowNum

    '2 step
    rawVelocity = sumEffortLevel / CInt(SprintLength.Value)

    '3 step
    For Each frictionFactorFrameControl In FrictionFactorsFrame.Controls
        If TypeOf frictionFactorFrameControl Is MSForms.TextBox Then
```

```
            currFrictionFactor = frictionFactorFrameControl.Value
            frictionNegFactor = frictionNegFactor * currFrictionFactor
        End If
    Next frictionFactorFrameControl

    '4 step
    For Each dynamicFactorFrameControl In DynamicFactorsFrame.Controls
        If TypeOf dynamicFactorFrameControl Is MSForms.TextBox Then
            currDynamicFactor = dynamicFactorFrameControl.Value
            dynamicNegFactor = dynamicNegFactor * currDynamicFactor
        End If
    Next dynamicFactorFrameControl

    '5 step
    allNegFactor = frictionNegFactor * dynamicNegFactor

    '6 step
    velocity = rawVelocity ^ allNegFactor

    '7 step
    requiredDevelopDays = Round(sumEffortLevel / velocity)
    requiredDevelopTimeSprints = Round(requiredDevelopDays /
CInt(SprintLength.Value))


    ' remove technical and result worksheets if they exist
    For Each Sheet In ActiveWorkbook.Worksheets
        If Sheet.Name = "Technical" Or Sheet.Name = "Result" Then
          Sheet.Delete
        End If
```

```
    Next Sheet

    ' create technical worksheet
    Worksheets.Add(After:=Worksheets(Worksheets.Count)).Name = "Technical"
    Worksheets("Technical").Activate

    ' fill technical worksheet
    For rowNum = 1 To requiredDevelopTimeSprints
        Range("A" & rowNum).Value = rowNum
        Range("B" & rowNum).Value = (CInt(UserStoriesCount.Value) * rowNum) /
requiredDevelopTimeSprints
    Next rowNum

    'create result worksheet
    Worksheets.Add(After:=Worksheets(Worksheets.Count)).Name = "Result"
    Worksheets("Result").Activate

    ' prepare user table for real estimation results on result worksheet
    Range("A20").Value = "Итерация разработки"
    Range("B20").Value = "Количество завершенных пользовательских историй"
    Range("A20:B20").EntireColumn.AutoFit

    For rowNum = 1 To requiredDevelopTimeSprints
        Range("A" & 20 + rowNum).Value = rowNum
    Next rowNum

    ' draw graph on result worksheet
    Dim resultChart As Chart

    Set resultChart = Sheets("Result").ChartObjects.Add(10, 10, 700, 250).Chart
```

```
    With resultChart
        .SetSourceData (Worksheets("Technical").Range("A1:B" & requiredDevelopTimeSprints))
        .ChartType = xlLine
        .SeriesCollection(2).Border.Color = RGB(255, 160, 0)
        .SeriesCollection(2).Border.LineStyle = xlDashDotDot
        .SeriesCollection(2).Name = "Прогнозируемая оценка"
        .SeriesCollection(1).Delete
    End With

    ' add new series to graph for ability compare forecast and actual results
    Dim realEstimateChartSeries As Series

    Set realEstimateChartSeries = resultChart.SeriesCollection.NewSeries
    With realEstimateChartSeries
        .Values = Range("B21:B" & 21 + requiredDevelopTimeSprints)
        .Name = "Рельный результат"
        .Border.Weight = 3
    End With
End Sub
```