Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Уральский государственный педагогический университет»
Институт математики, информатики и информационных технологий
Кафедра информационно-коммуникационных технологий в образовании

ТЕХНОЛОГИЯ ПРИМЕНЕНИЯ ГИБКОЙ МЕТОДОЛОГИИ «SCRUM»

Выпускная квалификационная работа
бакалавра по направлению подготовки
09.03.02 – Информационные системы и технологии

Исполнитель: студент группы БС-41
Института математики, информатики и ИТ
Догадин К.В.

Руководитель: к.п.н., доцент кафедры ИКТО
Стариченко Е.Б.

Работа допущена к защите
«____»_____ 2017 г.
Зав. кафедрой _____

Екатеринбург – 2017

Реферат

Догадин К.В. ПРИМЕНЕНИЕ ГИБКОЙ МЕТОДОЛОГИИ РАЗРАБОТКИ «SCRUM» В КОМАНДНОМ ПРОЕКТЕ, выпускная квалификационная работа: 46 стр., рис. 4, табл. 2, библ. 44 назв.

Ключевые слова: МЕТОДОЛОГИЯ РАЗРАБОТКИ, УПРАВЛЕНИЕ ПРОЕКТАМИ, IT-ПРОЕКТ, ВЕБ-РАЗРАБОТКА.

Объект разработки – методологии и подходы разработки программного обеспечения.

Цель работы – описать технологию применения «гибкой» методологии «скрам» в командном проекте разработки.

В ходе работы выявлена проблема организации и управления командой разработки в условия меняющихся требований заказчика, проанализированы различные способы управлениями командой и производством продукта. Разработано техническое задание для внедрения в проект «гибкой» методологии. Подход реализован при разработке программного продукта с помощью стека технологий, основанной на платформе Java: Java Se 8, Java EE 7, Sping, Postgres, а также онлайн сервисов для менеджмента отвечающих требованиям гибкой методологии.

Разработанный подход к внедрению может быть апробирован в командном проекте и может быть использован и применен в других.

Оглавление

Введение

Во многих компаниях занимающихся производством программного обеспечения, остро стоит вопрос оптимизации труда. Производство не эффективно по многим причинам, например в компаниях применяются «тяжеловесные» методологии, но иерархичные методы управления не дают нужного результата. Также проблемой перехода является отсутствие информированности о других, не иерархичных подходах, например «гибких», и самое главное, о способах внедрения «гибких» методологий в технологический процесс.

Существует множество различных методологий разработки программных продуктов, рассчитанных на крупные и небольшие проекты, с различным составом команды разработчиков. Следует точно определить, какая методология подходит в данном конкретном случае для данной компании.

Для получения максимальной пользы от гибкой методологии, необходимо правильно организовать и структурировать процесс разработки, для чего необходима система управления проектами, которая не только сможет помочь в этом, но и решит многие другие проблемы связанные со спецификой It-проектов. При ошибочном выборе можно получить неэффективное решение.

Цель: применение и внедрение в реальном проекте гибкой методологии разработки «скрам» и анализ её эффективности с точки зрения управления.

Задачи:

- анализ материалов предметной области;
- определение сущности и преимуществ «гибких» методологий разработки;
- выбор инструментов для применения «гибкой» разработки;
- разработка алгоритма внедрения;
- разработка документа руководства;
- внедрение методологии «скрам» в конкретный проект согласно разработанному алгоритму.

Глава 1. Анализ существующих методологий и подходов к их внедрению.

1.1 Гибкие методологии разработки и их виды

Прежде, чем внедрить подходящую для заказчика методологию разработки, необходимо проанализировать их, ведь внедрение напрямую зависит от особенностей методологии. Для этого нужно ясно понимать различия гибких методологий и жестких иерархичных, одна из которых с большой вероятностью будет использоваться во внедряемой компании.

Как правило, разработка программного обеспечения представляет собой довольно хаотическую деятельность, которую нередко можно охарактеризовать фразой "code and fix" ("пишем и правим"). Единого алгоритма не существует, а общий проект представляет собой просто смесь краткосрочных решений. Такой подход может сгодиться для небольшой системы, однако если система начинает расти, добавлять в нее новые свойства становится все более затруднительно [1]. Не стоит забывать и о том, что чем больше система, тем больше в ней появляется ошибок, которые все сложнее и сложнее исправлять. А это означает, что большинство крупных создаваемых систем имеют долгий тестовый период, хотя разработка всей функциональности давно закончена. Редко какие компании закладывают в план проекта этот отрезок времени, а потому очень часто происходит срыв сроков сдачи проекта. А точнее сказать, закладываться-то - он закладывается, но не настолько продолжительный, какой он получается в реальности, так как при подобном тестировании и исправлении ошибок адекватное планирование просто невозможно.

Однако у команд всегда была альтернатива - использовать методологию разработки, которая превращает создание программного продукта в упорядоченный процесс, с помощью которого можно сделать работу программиста более прогнозируемой и эффективной [1]. Для достижения этой цели создается детальное описание процесса, важное место в котором занимает планирование. Казалось бы - вот решение проблемы непредсказуемости разработки и выхода

за рамки установленных сроков. Однако в подобных методологиях есть один существенный недостаток, из-за которого большинство команд программистов от них отказываются. Чтобы сделать процесс разработки предсказуемым, необходимо наладить дисциплину внутри коллектива, а для этого нужно точно следовать предписаниям методологии и выполнять множество различных вспомогательных действий, что зачастую замедляет весь темп работ. И в итоге использование такого подхода часто становится бессмысленным. Именно поэтому такие методологии разработки программного обеспечения называют тяжеловесными, или, согласно термину Джима Хайсмита, - монументальными.

Несколько лет назад такое положение дел натолкнуло методологов на мысль, что нужно как-то менять процесс создания программных продуктов, и на свет появилась группа новых, облегченных методологий, которые коренным образом отличаются от монументальных. В настоящее время наиболее распространенный для них термин - гибкие (agile). Они являют собой нечто среднее между слишком перегруженным процессом разработки и полным его отсутствием. Иначе говоря, объема процесса в них как раз достаточно, чтобы получить разумную отдачу [2].

Одним из очевидных отличий гибких методологий от тяжеловесных является гораздо меньший объем артефактов и большая ориентация на исходный код как ключевую часть документации. Вовсе не обязательно создавать пачку никому не нужных документов только ради процесса. Но это не главное их различие, это скорее следствие. Более существенными различиями являются следующие:

"- Гибкие методологии адаптивны, а не предсказуемы. Для тяжеловесных методологий необходимо детальное планирование большого объема разработок, и такой подход работает - но до тех пор, пока не начнутся изменения. Следовательно, для этих методологий сопротивляться всяким изменениям совершенно естественно. Гибкие же методологии, напротив, изменения приветствуют. В отличие от тяжеловесных, они были задуманы как процессы, ко-

торые адаптируют изменения и только выигрывают от них, даже в том случае, когда изменения происходят в них самих.

Гибкие методологии ориентированы на человека, а не на процесс. В них ясно заявлено о необходимости учитывать в работе природные качества человеческой натуры, а не действовать им наперекор. Кроме этого в гибких методологиях особо подчеркивается, что работа по созданию программных продуктов должна приносить удовольствие. " [4].

Одними из главных критериев для сравнения методологий разработки программных продуктов являются итеративность и формализованность. В соответствии с этими критериями методологии бывают каскадные / итера-тивные и низкоформализованные / высокоформализованные.

Большая часть команд, разрабатывающих программное обеспечение, до сих пор используют в своих проектах каскадную разработку, при которой фазы выполняются в четкой последовательности: сначала требования, затем анализ и проектирование, потом реализация/интеграция и затем тестирование. Такой подход заставляет ключевых членов группы простаивать довольно продолжительное время, а тестирование откладывается на самый конец жизненного цикла проекта, когда становится дорого исправлять какие-то серьезные ошибки, и это ставит под угрозу сроки выхода конечного продукта.

Итеративный подход - это последовательность нарастающих шагов или итераций. Каждая итерация включает в себя некоторые или большую часть дисциплин разработки. У каждой итерации есть четко определенный набор целей, и она создает частично работающую реализацию конечной системы. Каждая последующая итерация строится на результатах предыдущих, развивает и усовершенствует систему до тех пор, пока не будет создан конечный продукт [2]. В процессе итеративной разработки рабочие версии системы, имеющие некий ограниченный набор требуемых свойств, производятся достаточно часто. Таким промежуточным версиям недостает функциональности, однако, во всем остальном они полностью соответствуют конечной системе. Промежуточные

варианты системы должны быть полностью интегрированы и так же хорошо оттестированы, как и конечная версия продукта. Все гибкие методологии используют именно итеративный подход, и это не случайно.

Итеративный подход оказывается эффективнее по нескольким показателям.

• Он приспособлен к меняющимся требованиям. Изменение требований и добавление новых свойств, определяемое заказчиком или нуждами технологии всегда было самым главным источником бед проектов. Оно приводило к опозданию с выпуском, нарушению графиков, неудовлетворению заказчиков и раздражению разработчиков [2]. Однако, это нормальная практика в разработке программного обеспечения, и было бы глупо этому удивляться. Конечно, можно считать, что часто меняющиеся требования – результат плохой проработки требований, однако дела обстоят далеко не всегда таким образом. Ведь сразу определить все возможные варианты требований непросто. Только получив какую-то версию системы, можно оценить, какие ее свойства важны, а какие абсолютно ненужные и неудобные. Выполнить проект без изменений можно, но это грозит выпуском продукта, который не будет иметь никакой коммерческой ценности. К тому же, не стоит забывать о таком важном факторе, как время. В мире бизнеса и информационных технологий не стоит на месте, а движется вперед стремительными темпами, поэтому то, что актуально сегодня, через полгода окажется устаревшим. А если проект рассчитан не на один год, то с самого начала проекта следует готовиться к меняющимся требованиям со стороны заказчика.

• Интеграция – это не один «большой взрыв» в конце проекта. Оставить интеграцию на самый конец означает получить переработки с большими затратами времени – иногда до 40% всего объема проекта [2]. Чтобы этого не допустить, итеративный подход предлагает разбить весь процесс разработки на небольшие итерации, в конце каждой из которых должна быть готова полноцен-

ная рабочая версия системы, полностью оттестированная и интегрированная. А значит, переработка минимизируется.

• Риски обычно обнаруживаются и устраняются на ранних итерациях. Итеративный подход минимизирует риски на ранних стадиях, когда тестируются все компоненты [2]. Он позволяет вовремя понять, является ли предполагаемый риск реальным, и выявить новые. В начале проекта это сделать гораздо легче и несильно дорого.

• Менеджеры имеют возможность внесения тактических изменений в продукт. Итеративная разработка быстро создает исполняемую архитектуру (хотя и с ограниченной функциональностью), которую можно использовать для быстрого выпуска продукта с некоторыми ограничениями [2].

• Облегчается повторное использование. Гораздо легче определить общие части тогда, когда они уже частично спроектированы или реализованы в итерациях, чем распознать их при планировании. Рецензирование проектных решений на ранних итерациях позволяет архитекторам указать на потенциальные возможности для повторного использования и затем разработать в последующих итерациях зрелый общий код для реализации таких возможностей [2].

• Дефекты можно найти и исправить за несколько итераций, что обеспечивает создание четкой архитектуры и высококачественного приложения. Узкие места обнаруживаются еще на ранних итерациях, а не в конце проекта при глобальном тестировании.

• Лучшее использование персонала в проекте. Многие организации совмещают использование каскадного подхода с организацией по типу конвейера: аналитики посылают готовые требования проектировщикам, которые отсылают свой продукт программистам, которые посылают компоненты специалистам по интеграции, которые отсылают систему тестировщикам. Такие многочисленные переходы являются источниками ошибок и недопонимания и заставляют людей меньше чувствовать ответственность за конечный продукт. Итеративный процесс расширяет области компетенции членов группы, разре-

шая им выполнять много ролей, позволяя менеджеру проекта лучше использовать имеющийся персонал, и одновременно убирает вредные передачи. [6]

• Члены команды учатся на ходу. Участники проекта на протяжении цикла разработки получают возможность учиться на своих ошибках от итерации к итерации.

• Улучшается процесс разработки сам по себе и становится по ходу работы более совершенным. В конце каждой итерации участники проекта могут оценить организацию процесса и внести в него необходимые корректировки, если что-то не будет подходить их команде.

1.2 Обзор гибких методологий

Существует много гибких методологий разработки, однако остановимся только на некоторых из них, наиболее известных и развитых:

- XP
- Scrum
- Crystal
- RUP

Здесь стоит сделать оговорку. RUP как таковой не является гибкой методикой, однако существует возможность настроить его таким образом, что он превратится в низкоформализованный, итеративный процесс.

Итак, эти методологии ориентированы на итеративную разработку программного обеспечения, а также или изначально созданы как низкофрмализованные, или существует возможность настроить их на минимальную формализацию процесса.[8]

Scrum в сравнении с остальными методологиями, и таким образом объясним, почему для разрабатываемой системы была выбрана именно она.

• Если рассмотреть такой критерий, как легкость внедрения методики в процесс разработки. XP содержит в себе достаточно жесткие правила ведения проекта, которые не все разработчики могут принять, или просто может понадобиться продолжительное время, чтобы получить от них хорошую отдачу. А

это рискованно, если проекты не очень большие и не рассчитаны хотя бы на год. Но в компании-заказчике дела обстоят именно таким образом. Если внедрение методологии идет тяжело, то велика вероятность не уложиться в сроки и потерять большую прибыль. Естественно, это недопустимо.

- XP – это больше набор методик, которые ничто не мешает внедрить и в другие методологии, в тот же Scrum или RUP. Они никак друг другу не противостоят. Scrum – это всего лишь каркас, а работу внутри самой команды можно наладить различными способами, включая, например, парное программирование.[9]
- Если рассматривать RUP в его тяжелом варианте, то он совершенно не подходит для компании-заказчика, потому как создание слишком большого объема артефактов при коротких проектах – совершенно не выгодно. Облегченный же RUP – более подходящий, однако для его внедрения необходимо, чтобы весь управляющий персонал, включая менеджеров, досконально его знали, чтобы процесс разработки пошел в нужном направлении. В этом случае тоже требуются достаточно большие усилия и некоторая перестройка того процесса, который естественным образом сложился в компании. Scrum же очень близок к существующему на данный момент процессу заказчика, поэтому его внедрение займет минимум усилий и затрат.[9]
- Из семейства методологий Crystal самой подходящей является Crystal Clear, но уж очень она абстрактная и общая. В ней не предусмотрены конкретные меры по организации работы в коллективе или планированию, а значит, в эффективности она проигрывает остальным методологиям, а по своей жесткости очень далеко отстоит от XP. Она не имеет смысла внедрения в работу компании-заказчика, так как по большому счету процесс, описанный Crystal, сам по себе сложился в данной организации.

Для того чтобы иметь полное представление о выбранной для компании-заказчика методологии разработки, приведем наиболее подробное ее описание.

Методология Scrum устанавливает правила управления процессом разработки и позволяет использовать уже существующие практики кодирования, корректируя требования или внося тактические изменения. Использование этой методологии дает возможность выявлять и устранять отклонения от желаемого результата на более ранних этапах разработки программного продукта.

Основа Scrum — итеративная разработка. Scrum определяет правила, по которым должен планироваться и управляться список требований к продукту для достижения максимальной прибыльности от реализованной функциональности; правила планирования итераций для максимальной заинтересованности команды в результате; основные правила взаимодействия участников команды для максимально быстрой реакции на существующую ситуацию; правила анализа и корректировки процесса разработки для совершенствования взаимодействия внутри команды.[11] Каждую итерацию можно описать так: планируем — фиксируем — реализуем — анализируем. За счет фиксирования требований на время одной итерации и изменения длины итерации можно управлять балансом между гибкостью и планируемостью разработки.[12]

Scrum — простой каркас, который можно использовать для организации команды и достижения результата более продуктивно и с более высоким качеством за счет анализа сделанной работы и корректировки направления развития между итерациями. Методология позволяет команде выбрать задачи, которые должны быть выполнены, учитывая бизнес-приоритеты и технические возможности, а также решить, как их эффективно реализовать. Это позволяет создать условия, при которых команда работает с удовольствием и максимально продуктивно. К примеру, возможность самостоятельного выбора объема и пути решения задач без внешнего давления позволяет всем участникам команды почувствовать себя активными игроками, вовлеченными в процесс, а не простыми исполнителями, от которых требуется лишь четкая реализация поручений[12].

Scrum фокусируется на постоянном определении приоритетных задач, основываясь на бизнес целях, что увеличивает полезность и доходность проекта

на его ранних стадиях. Так как при инициации проекта его доходность определить почти невозможно, Scrum предлагает концентрироваться на качестве разработки и к концу каждой итерации иметь промежуточный продукт, который можно использовать, пусть и с минимальными возможностями. Например, результатом итерации может быть каркас сайта, который можно показать на презентации.

Методология Scrum ориентирована на то, чтобы оперативно приспосабливаться к изменениям в требованиях, что позволяет команде быстро адаптировать продукт к нуждам заказчика. Такая адаптация достигается за счет получения обратной связи по результатам итерации: имея после каждой итерации продукт, который уже можно использовать, показывать и обсуждать, легче собирать информацию и делать правильные корректировки и изменять приоритеты требований[8]. Например, если каркас сайта показать потенциальным пользователям, то появится много вопросов, на основании которых можно скорректировать то, что уже написано или еще не реализовано, понять что более важно пользователю.[4]

Девиз Scrum — «анализируй и адаптируй»: анализируй то, что получил, адаптируй то, что есть, к реальной ситуации, а потом анализируй снова. Чем меньше формализма, тем более гибко и эффективно можно работать, — это основной принцип данной методологии. Но это не означает, что формальных процессов не должно быть совсем, их должно быть достаточно для организации эффективного взаимодействия и управления проектом. Формальная часть Scrum состоит из трех ролей, трех практик и трех основных документов.[4]

Владелец продукта (Product Owner) — человек, поставляющий требования программистам. От того, как четко написаны требования, зависит, насколько часто команде придется переключаться с задачи на задачу в связи с отсутствием нужной информации, как много нужно задавать вопросов, на которые уходит дополнительное время, как сильно придется изменять уже написанную функциональность от итерации к итерации и, соответственно, эффек-

тивность разработки в целом. Обычно владелец продукта является представителем или доверенным лицом заказчика, а для компаний, выпускающих коробочные продукты, он представляет рынок, на котором реализуется продукт. Владелец продукта должен составить бизнес план, показывающий ожидаемую доходность и план развития с требованиями, отсортированными по коэффициенту окупаемости инвестиций. Исходя из имеющейся информации, владелец продукта подготавливает список требований, отсортированный по значимости. Чем лучше владелец продукта описывает требования, управляет приоритетами и чем быстрее выдает информацию, тем больший финансовый эффект получит компания от методологии. В обязанности этого сотрудника входит своевременное предоставление требований к продукту, определение дат и содержания релизов, эффективное управление приоритетами и корректировка требований для достижения максимальной окупаемости инвестиций в продукт.

От человека, исполняющего роль *Scrum-мастера* (Scrum Master), во многом зависит самостоятельность, инициативность программистов, удовлетворенность сделанной работой, атмосфера в команде и результат всей работы. Этот человек должен быть одним из членов команды разработки и участвовать в проекте как разработчик. Он отвечает за своевременное решение текущих проблем, от ремонта сломанного стула до обеспечения необходимой информацией членов команды для продолжения их работы и загруженности, за поддержание нужных технических практик, используемых на проекте. В обязанности Scrum-мастера входит обеспечение максимальной работоспособности и продуктивности команды, четкого взаимодействия между всеми участниками проекта, своевременное решение всех проблем, тормозящих или останавливающих работу любого члена команды, ограждение команды от всех воздействий извне во время итерации и обеспечение следования процессу всех участников проекта.

Scrum-команда (Scrum Team) — группа, состоящая из пяти–девяти самостоятельных, инициативных программистов. Первая задача этой команды —

поставить реально достижимую, прогнозируемую, интересную и значимую цель для итерации. Вторая задача — сделать все для того, чтобы эта цель была достигнута в отведенные сроки и с заявленным качеством. Цель итерации считается достигнутой только в том случае, если все поставленные задачи реализованы, весь код написан по определенным проектом «стандартам кодирования» (coding guidelines), программа протестирована полностью, а все найденные дефекты устранены. Программисты этой команды должны уметь оценивать и планировать свою работу, работать в команде, постоянно анализировать и улучшать качество взаимодействия и работы. В обязанности всех членов Scrum-команды входит участие в выборе цели итерации и определение результата работы. Они должны делать все возможное и невозможное для достижения цели итерации в рамках, определенных проектом, эффективно взаимодействовать со всеми участниками команды, самостоятельно организовывать свою работу, предоставлять владельцу рабочий продукт в конце каждого цикла.

Подготовка к первой итерации, называемой *спринт* (Sprint), начинается после того, как владелец продукта разработал план проекта, определил требования и отсортировал их в количестве, достаточном для наполнения одной итерации. Такой список требований называется *журналом продукта* (Product Backlog). При планировании итерации происходит детальная разработка сессий планирования спринта (Sprint Planning Meeting), который начинается с того, что владелец продукта, Scrum-команда и Scrum-мастер проверяют план развития продукта, план релизов и список требований. Scrum-команда проверяет оценки требований, убеждается, что они достаточно точны, чтобы начать работать, решает, какой объем работы она может успешно выполнить за спринт, основываясь на размере команды, доступном времени и производительности. Важно, чтобы Scrum-команда выбирала первые по приоритету требования из журнала продукта. После того как Scrum-команда обязуется реализовать выбранные требования, Scrum-мастер начинает планирование спринта. Scrum-команда разбивает выбранные требования на задачи, необходимые для его реализации. Эта

активность в идеале не должна занимать больше четырех часов, и ее результатом служит список требований, разбитый на задачи, — *журнал спринта* (Sprint Backlog). Необходимо, чтобы все участники команды приняли на себя обязательство по реализации выбранной цели.

После окончания планирования начинается итерация. Каждый день Scrum-мастер проводит «скрам» (Daily Scrum Meeting) — пятнадцатиминутное совещание, цель которого — достичь понимания того, что произошло со времени предыдущего совещания, скорректировать рабочий план к реалиям сегодняшнего дня и обозначить пути решения существующих проблем. Каждый участник Scrum-команды отвечает на три вопроса: что я сделал со времени предыдущего скрама, что меня тормозит или останавливает в работе, что я буду делать до следующего скрама? В этом митинге может принимать участие любое заинтересованное лицо, но только участники Scrum-команды имеют право принимать решения. Правило обосновано тем, что они давали обязательство реализовать цель итерации, и только это дает уверенность в том, что она будет достигнута. На них лежит ответственность за их собственные слова, и, если кто-то со стороны вмешивается и принимает решения за них, тем самым он снимает ответственность за результат с участников команды[12].

В конце каждого спринта проводится *демонстрационный митинг* (Sprint Review Meeting) продолжительностью не более четырех часов. Сначала Scrum-команда демонстрирует владельцу продукта сделанную в течение спринта работу, а тот в свою очередь ведет эту часть митинга и может пригласить к участию всех заинтересованных заказчиков. Владелец продукта определяет, какие требования из журнала спринта были выполнены, и обсуждает с командой и заказчиками, как лучше расставить приоритеты в журнале продукта для следующей итерации. Во второй части митинга производится анализ прошедшего спринта, который ведет Scrum-мастер. Scrum-команда ищет использованные в последнем спринте положительные и отрицательные способы совместной работы, анализирует их, делает выводы и принимает важные для

дальнейшей работы решения. Scrum-команда также определяет программы, которые могут работать лучше, и ищет пути для увеличения эффективности дальнейшей работы. Затем цикл замыкается, и начинается планирование следующего спринта.

В начале проекта владелец продукта готовит журнал продукта — список требований, отсортированный по значимости, а Scrum-команда дополняет этот журнал оценками стоимости реализации требований. Список должен включать функциональные и технические требования, необходимые для реализации продукта. Самые приоритетные из них должны быть достаточно детально прописаны, чтобы их можно было оценить и протестировать. О своевременной детализации требований должен заботиться владелец продукта и предоставлять необходимый объем в нужное время. В этом смысле программисты являются заказчиками требований для владельца продукта. И отношение владельца продукта должно быть соответствующее — каковы требования, такова и их реализация. В дальнейшем остальные требования должны постепенно уточняться и детализироваться до такого же уровня. Главное, чтобы у команды всегда был достаточный объем подготовленных к реализации требований.

После того как команда во время сессии планирования выбрала и обязалась реализовать набор требований из журнала продукта, эти требования разбиваются на более мелкие задачи, составляющие детализированный список требований — журнал спринта. Разбивка на задачи должна быть сделана таким образом, чтобы выполнение одной задачи занимало не больше двух дней (считается, что менее детальная, например, полдня, разбивка приводит к более грубой оценке, чем приемлема в большинстве проектов, использующих методологию Scrum). Разбивка на задачи поможет так спланировать итерацию, чтобы в конце не осталось ни одной невыполненной задачи и, соответственно, достичь ее цели. После завершения детализации оценка журнала спринта сравнивается с первичной оценкой в журнале продукта. Если существует значительное расхождение, команда договаривается с владельцем продукта об объеме работ, ко-

торый должен быть выполнен в течение итерации, и о том, какой объем будет перенесен на следующую. Менее важные и мало влияющие на цель итерации задачи выносятся из журнала спринта[6].

График спринта (Burndown Chart) показывает ежедневное изменение общего объема работ, оставшегося до окончания итерации. Это график позволяет команде разработчиков делать анализ текущей ситуации и своевременно реагировать на отклонения. График спринта позволяет также владельцу продукта наблюдать за ходом итерации — если общий объем работ не уменьшается каждый день, значит, что-то идет не так. Во время сессии планирования команда находит и оценивает задачи, которые надо выполнить для успешного завершения итерации. Сумма оценок всех задач в журнале спринта является общим объемом работы, который надо выполнить за итерацию. После завершения каждой задачи Scrum-мастер пересчитывает объем оставшейся работы и отмечает это на графике спринта. Только в том случае, если объем работ по окончании итерации закончился (в журнале спринта не осталось незавершенных задач), итерация считается успешной. График спринта используется как вспомогательный инструмент, позволяющий корректировать работу для завершения итерации вовремя, с работающим кодом и требуемым качеством.

Время между итерациями — это время принятия основополагающих решений, влияющих на ход всего проекта. Во время итерации никакие изменения извне не могут быть сделаны. После того как команда дала обязательство реализовать журнал спринта, он фиксируется, и изменения в нем могут быть сделаны только по следующим причинам:

Scrum-команда в течение итерации получила лучшее представление о требованиях и нуждается в дополнительных задачах для успешного завершения итерации;

Найдены дефекты, которые нужно обязательно исправить для успешного завершения итерации;

Scrum-мастер и Scrum-команда могут решить, что небольшие изменения, не влияющие на общий объем работ, могут быть реализованы в связи с возникшей у владельца продукта необходимостью.

Исходя из того что журнал спринта не может быть изменен извне во время итерации, нужно выбирать ее длину, основываясь на стабильности требований. Если требования стабильны, меняются или дополняются редко, то можно выбрать шестинедельный цикл. В этом случае экономится время на переключение команды с активной разработки на планирование и демонстрационные митинги. Если требования часто меняются и дополняются, нужно отталкиваться от двухнедельного цикла, в любом случае длина итерации — это величина экспериментальная[5].

Основной упор методология Scrum делает на управление проектами и не задает никаких технических практик, что дает возможность использовать весь технический багаж, накопленный компанией. При внедрении Scrum чаще всего возникает две трудности. Первая — добиться активного участия от каждого разработчика и слаженной коллективной работы в команде. Похожую задачу решает тренер спортивной команды. Вторая — вовлечь поставщика требований в активное участие в проекте, заинтересовать его динамикой развития продукта, дать возможность быть активным болельщиком и спонсором команды. [4]

1.3 Внедрение

Сазерленд предлагает для внедрения следующий подход из десяти пунктов, одиннадцатым является повторение одного из пунктов:

1. Выберите “владельца продукта”. Он должен хорошо разбираться в сути продукта, знать рынок, быть всегда доступным для команды. “Владелец продукта” формулирует требования на текущий спринт и “несет ответственность за полезность продукта”.
2. Сформируйте команду. В группу должны войти специалисты, сумма навыков которых достаточна для выполнения работы. Команда сама

решает, как она будет работать; руководство отвечает лишь за постановку стратегических целей.

3. Выберите “скрам-мастера”. Его задача – проводить все короткие собрания, обеспечивать выполнение всех необходимых процедур и помнить, что сам процесс может стать препятствием к выполнению задач.
4. Создайте “бэклог” продукта. Как можно быстрее продумайте и запишите все функциональные требования. Упорядочьте список по степени важности задач.
5. Оцените и уточните “бэклог”. Объем усилий, которые требуются на выполнение каждой задачи, оценивается не в часах, а относительно. Один из способов такой оценки – присвоить задачам “баллы” на основе последовательности Фибоначчи: 1, 3, 5, 8, 13 и так далее.
6. Спланируйте первый “спринт”. Из верхней части “бэклога” команда выбирает задания для одного “спринта” и решает, какой будет “цель спринта”. Добавлять новые задачи до его завершения нельзя. В конце каждого “спринта” анализируется “динамика производительности”: для этого складываются “баллы” всех выполненных задач. Таким же образом оценивается сложность еще не сделанных работ. Через несколько спринтов уже можно приблизительно рассчитать необходимое для окончания проекта время.
7. Обеспечьте прозрачность. Завершить проект в кратчайшие сроки можно лишь при условии прозрачности процессов. Повесьте на видном месте скрам-доску. Также составьте “диаграмму выгорания задач”. Для этого по одной оси откладывайте число набранных за уже выполненные задачи баллов, а по другой – дни.
8. “Собрания на ходу”. Такие собрания проводятся “скрам-мастером” каждый день в одно и то же время и длятся не более 15 минут. Их цель – обсуждение хода работ.

10 .“Обзор спринта”. На этой встрече присутствуют владелец продукта, скрам-мастер, команда и “любые заинтересованные лица – заказчик, представители руководства, потенциальные потребители”. Показывается готовая часть продукта.

11. “Ретроспективные собрания”. После каждой демонстрации заказчику проводится обсуждение, несущее функцию проверки из цикла Деминга “Планировать, действовать, проверять, корректировать” (PDCA). Команда обсуждает, что удалось в завершенном “спринте”, а что можно было сделать лучше, какие улучшения можно внести в работу немедленно. После этого начинается новый “спринт”.[12]

Данные пункты можно объединить по группам:

С первый пункт по третий - Распределение ролей. Далее идет описание требований к системе и их разбиением на блоки - Формирование требований, их декомпозиция по задачам. Затем автор предлагает провести оценку, и спланировать спринт, назначив задачи - Оценка и распределение задач. Весь процесс должен сопровождаться практикой собраний, во время спринта ежедневные митинги, в конце спринта – «обзор спринта», и ретроспективные собрания - Проведение собраний.

После всех этапов, с учетом опыта полученного на предыдущем спринте стартует новый спринт.

Данный алгоритм не может быть внедрен, сразу же, ведь ещё не выбраны технологии, а команда не сформирована, а её будущие участники не имеют представления о данной методологии[15].

Данная инструкция является обобщенной и не содержит примеров и указаний своей реализации с использованием конкретных технологий.

Таким образом, необходим собственный алгоритм, включающий подготовительный этап.

1.4 Формализованное описание технического задания

на разработку плана внедрения гибкой методологии «Scrum» в командный проект разработки программного обеспечения.

1. Общие сведения.

1.1. Название организации-заказчика.

Доступ к системе будет предоставляться покупкой подписки.

1.2. Название продукта разработки (проектирования).

План внедрения гибкой методологии «Scrum» в командный проект разработки программного обеспечения.

1.3. Назначение продукта.

План предназначен для многоразового использования для оптимизации и повышения эффективности труда в компаниях, путем внедрение гибкой методологии.

1.4. Плановые сроки начала и окончания работ.

Начало работ: 01 сентября 2017 г.; окончание работ 15 мая 2017 г.

2. Характеристика области применения продукта.

2.1. Процессы и структуры, в которых предполагается использование продукта разработки.

- процесс разработки программного обеспечения;
- команды и компании, предоставляющие услугу производства программного обеспечения.

2.2. Характеристика персонала (количество, квалификация, степень готовности)

Продукт ориентирован на использование менеджерами, разработчиками, тестировщиками, аналитиками и может быть использован также людьми исполняющими другие роли, но причастные к производству программного обеспечения.

3. Требования к продукту разработки.

3.1. Требования к продукту в целом.

3.1.1. Структура системы

План должен содержать ясные требования к технологиям и людям, содержать поэтапные алгоритмы действий, для того, чтобы методология была внедрена.

3.1.2. Подготовительный этап.

3.1.3. Этап внедрения.

3.2. Указание программного обеспечения, используемого для реализации.

Trello, SmartGit, Slack.

3.3. Форматы входных и выходных данных.

Производственный процесс.

3.4. Меры защиты информации.

Используются стандартные меры защиты информации и средства разграничения прав представляемые Trello, SmartGit, Slack. Доступ по каналу с шифрованием. Хранилище привязывается к учетной записи на GitHub. В канал Slack можно вступить только по приглашению.

4. Требования к документированию.

4.1. Перечень сопроводительной документации.

Техническое задание – основной документ, сопровождающий разработку плана внедрения методологии. Также разработка полагается на основные методологические принципы описанные в манифесте гибкой методологии.

4.2. Требования к содержанию отдельных документов.

Техническое задание - это исходный документ для проектируемого объекта, содержащий технические требования, предъявляемые к объекту и исходные данные для его разработки; в документе указывается информация о назначении объекта, области его применения и сроках выполнения.

5. Порядок сдачи-приемки продукта.

Продукт считается принятым при его положительной оценке экспертами-преподавателями вуза.

Глава 2. Разработка поэтапного плана внедрения методологии

2.1 Разработка модели универсального плана внедрения

Подготовительный этап.

Подбор команды.

Для того чтобы начать внедрять систему существует необходимость сформировать команду из существующих команд, если нужно переопределить существующий состав или создать новую, если проект только стартует. Для этого необходимо отобрать сотрудников обладающими необходимыми компетенциями [12].

Количественный состав команды должен варьироваться от двух до девяти человек максимум. Согласно исследованиям команды с большим количеством человек, чем в девять не увеличивают производительность, а теряют её [10]. Оптимальным является количество около пяти человек.

необходимо:

1. Обсуждение и выбор стека технологий для разработки.
2. Ознакомление команды с методологией и распределение ролей.
3. Выбор и внедрение программных продуктов для реализации методологии.
4. Регистрация всех пользователей в системах, предоставление им доступа.

Распределение ролей.

В Scrum принято выделять три основных роли: владелец продукта, скрам-мастер и команда.

Владелец продукта (Product owner) - человек, ответственный пиритизацию требований и часто за их создание, т.к. программа разрабатывается не для единого заказчика, а доступ к ней будет осуществляться при помощи системы подписок, то формируется список требований согласно рынку отрасли(исследованию), для которого разрабатывается приложение.

Скрам-мастер – член команды, который дополнительно отвечает за процессы, может быть выбран не менеджер, а опытный разработчик, для координацию работы команды и поддержание социальной атмосферы в команде.

Команда – разработчики количеством от двух до девяти человек, которые реализуют требования владельца продукта. Команды большим количественным составом, чем девять человек теряют свою эффективность.

Для разрабатываемых систем, доступ к которым будет осуществляться по подписке, т.е. они будет иметь множество клиентов, нельзя выделить одного владельца продукта, таким образом, к роле владельца продукта может быть привлечен специалист из данной области. Сам владелец продукта не обязан заниматься формализацией требований, обычно этим занимается аналитик, но если аналитика в команде нет, то формализацией требований может заниматься как команда, так и скрам-мастер.

1. Стек технологий разработки.

К приложению, имеющему множество клиентов, должна быть реализовано возможность доступа. Это может быть или цифровая копия каждого приложения, распространяемая через точки продаж или через онлайн-магазины цифровой дистрибьюции.

Для приложения, которое должно хранить в себе большую обновляемую базу знаний лучше подойдет реализация как веб-приложение.

Таким образом, должны выбираться языки, платформы и технологии для разработки именно веб-приложений.

Обоснованием к выбору может служить «популярность» технологии, например, полученная по версии индекса TIOBE. [44]

Jun 2017	Jun 2016	Change	Programming Language	Ratings
1	1		Java	14.493%

2	2		C	6.848%
3	3		C++	5.723%
4	4		Python	4.333%
5	5		C#	3.530%

Таблица 1. Таблица сравнения языков программирования[44]

«Популярность» и распространённость имеет большое значение для поддержки проекта, ведь от того насколько поддерживается платформа, зависят её функциональные особенности, частота обновления. Не обновляемые технологии устаревают, становятся небезопасны и не используются.[2]

Также гораздо проще найди разработчиков, владеющих распространённой технологией, чем экзотичной.

Программные продукты для реализации методологии.

Выбор системы контроля версий

В ходе разработки код проекта активно меняется. При этом нужно вести учет того, что уже было сделано, и согласовывать действия отдельных участников по одновременному изменению кода так, чтобы доработки участников проекта учитывали все ранее сделанные правки других участников. Система контроля версий позволяет автоматизировать этот процесс.

Система контроля версий — это система, которая регистрирует все изменения в файлах. В централизованных системах контроля версий (CVS, Subversion, Perforce) есть центральный сервер, на котором хранятся все данные, и ряд клиентов, получающих из него копии файлов. В таких системах всегда понятно, кто чем занимается в проекте, и их проще администрировать.

Распределенные системы контроля версий (Git, Mercurial, Bazaar) вместо традиционной клиент-серверной модели используют распределенную, то есть все изменения хранятся в локальном хранилище на компьютере и при необхо-

димости синхронизируются с другими. В этом случае устраняется зависимость от центрального сервера, можно вести работу даже без сетевого соединения с ним.

Факторы влияния на выбор системы контроля версий.

В первую очередь — соответствие возможностей системы контроля версий принятому в команде процессу разработки. Во вторую очередь, то с какой системой контроля версий привычнее работать участникам проекта.

Популярность системы контроля версий.

Владения ей разработчиками.

Рейтинг популярности систем контроля версий согласно исследованиям Тэглайн самого крупного русскоязычного аналитического агентства, исследующее рынок digital-продакшна, мобильной разработки, интерактивного маркетинга и сопутствующих услуг.[42]

Рейтинг систем контроля версий выпускается Тэглайном впервые и сформирован на основе анкетирования (проводилось с августа 2014 по апрель 2016 года)

Рейтинг систем контроля версий 2016 по убыванию:

2 02 1		GIT	2005	распределенная
3 0 2		SVN (Subversion)	2000	централизованная
ф 3		Mercurial	2005	распределенная
5 0 4		CVS (Concurrent Versions	1990	централизованная

		System)		
		Team Foundation Server	2005	централи-зованная

Таблица 2. Таблица сравнения систем контроля версий.

Выбор системы отслеживания задач.

Системы отслеживания задач также могут называться трекерами, а сам процесс отслеживания задач – трекингом задач.

Что должно быть в системе?

Перед тем как изучать системы, нужно четко определить критерии, по которым системы будут оценены:

Язык: в системе много пользователей, из них некоторые заказчики, и в любом случае далеко не все знают английский, поэтому очень желательно, чтобы система имела перевод.

Интеграция с Git: система должна давать возможность работать с Git или SVN.

Доска и диаграмма: должна быть качественная доска для Scrum.

Удобство использования: желательно, чтобы было сразу понятно, как пользоваться системой, и у заказчиков и разработчиков не возникало сотни вопросов.

Настраиваемые поля: поля должны настраиваться не хуже, чем в Redmine.

Фильтрация: поиск и фильтрация должны быть удобными.

Перевод: если русский язык есть, то он должен быть качественным.

Создание задач по email: задачи должны создаваться из писем, при этом нормально прикреплять файлы (Redmine прикрепляет файлы без имени).

Права доступа: любые настройки ролей для пользователей и групп.

Локальная установка: установка системы на свой сервер.

Экспорт: форматы, в которых можно экспортировать данные.

Цена: стоимость системы.

Данных систем существуют огромное множество, их рейтинг также можно посмотреть на

В данном случае первоочередную роль играет не популярность, а соответствие трекера методологии «скрам».

Выбор мессенджера для организации собраний.

Рейтинг популярности мессенджеров [42].

Средний размер штата компаний, принимающих участие в исследовании — 29 человек, средняя компания использует от 1 до 10 мессенджеров на разных проектах (или в разных командах) одновременно. Большинство мессенджеров — кроме специализированных, встроенных во внутрикорпоративные системы (таск-менеджеры, CRM и т. д) — используются параллельно для внерабочих целей.

- Набор требований для корпоративного мессенджера:
- каналы для группового общения по проектам,
- отправка различных материалов,
- возможность переключаться между разными командами.
- Специфичные требования, определяемые методологиями, компанией, командой, спецификой проекта, например, требования к безопасности.

Настройка систем. Предоставление доступа.

Не так уж много команд требуют сетевого подключения для своей работы, практически все команды оперируют с локальной копией проекта. Когда разработчики готовы поделиться своими наработками, то необходимо поделиться обновлениями с удалёнными репозиториями(хранилищами кода).

Для этого необходима настройка общего доступа, разграничение прав. Разграничениями назначением прав может заниматься как скрам-мастер, так и член команды разработки.

Для каждой системы есть свои особенности настройки прав общего доступа, поэтому нужно ещё более тщательно подходить к выбору системы контроля версия.

2. Этап внедрения.

1. Формирование требований к системе. Создание Бэклога.
2. Внедрение и выбор системы оценки сложности задач и прогнозирования. Ознакомление с ним команды.
3. Планирование спринта:
4. Оценка (если требуется) и распределение задач.
5. Внедрение практики постоянных собраний во время и демонстраций после окончания спринта.
6. Запуск нового спринта.

Формирование требований. Создание Бэклога.

Для создания любой системы должны быть сформулированы требования. Заказчик, почти всегда, не имеет технических компетенций и не может сформулировать свои требования формально.

Для получения требований возможно следующие варианты:

1. Общение с пользователем напрямую.

2. Создание документации с требованиями.

3. Использование Case-средств для формализации требований в виде схем.

Agile-команды предпочитают общение лицом к лицу подробной документации.

Далее нужно декомпозировать задачи из цели, т.е. разбить цель на подцели, которые нужно выполнить, чтобы её достигнуть.

Оценка.

Достаточно ли обозрима задача, чтобы ее можно было оценить?

Существуют различные способы оценки. Рекомендуется делать оценку относительной**,** а не в часах**.** Способ оценки задач:

«малый»;

«средний»;

«большой».

Покер планирования (Scrum poker) — техника оценки, основанная на достижении договорённости, главным образом используемая для оценки сложности предстоящей работы или относительного объёма решаемых задач при разработке программного обеспечения.

Для проведения покера планирования необходимо подготовить список обсуждаемых функций и несколько колод пронумерованных карт. Список функций либо пользовательские истории описывают разрабатываемое программное обеспечение. Карты в колодах должны быть пронумерованы. Обычно колода содержит карты, содержащие числа Фибоначчи, включая ноль: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89; другие разновидности колод могут использовать аналогичные последовательности.

Колода карт для покера планирования

Аргумент в пользу использования последовательности Фибоначчи — отражение возрастающей неопределённости с ростом сложности оцениваемых функций или задач.

Одна из имеющихся в продаже колод содержит следующую последовательность: 0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100 и иногда знак вопроса («?»), означающий неуверенность, и чашку кофе, означающую требование перерыва. Некоторыми организациями используются обычные игровые карты, включающие туз, 2, 3, 5, 8 и короля. Король буквально означает: «Данный пункт слишком большой или его слишком сложно оценить». Выбрасывание короля завершает обсуждение пункта на текущем спринте.

По желанию, может использоваться таймер, чтобы устанавливать лимит времени одного круга.

Достоинства метода

Покер планирования — это средство оценки проектов по разработке программного обеспечения. Эта техника минимизирует эффект привязки путём опроса каждого из участников команды таким образом, что никто не знает чужого решения до одновременного оглашения выбора каждого из участников.

Существует способ, позволяющий увеличить точность прогнозирования можно при помощи использования метода оценки с использованием равновероятностного распределения.

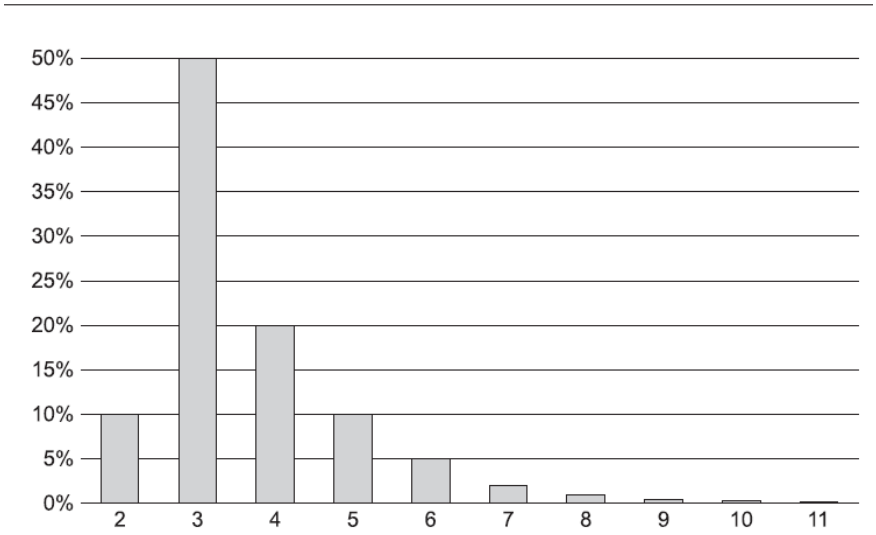

Рис. 1. Оценка с использованием равновероятностного распределения[8].

Программа PERT (Program Evaluation and Review Technique) была создана в 1957 году ВМС США для проектирования подводных лодок Polaris.

Одним из элементов PERT является способ вычисления оценок.

Схема PERT предоставляет очень простой, но исключительно эффективный способ преобразования оценок в вероятностные распределения. При оценке задачи предоставляются три числа (так называемый анализ по трем переменным):

О: оптимистическая оценка. Это значение выбирается предельно оптимистично. Задача может быть выполнена за это время только в том случае, если

все без исключения пройдет гладко. Более того, чтобы математическая теория сработала, вероятность такого исхода должна быть менее 1%1. Как видно на графике, это один день;

N: номинальная оценка (наиболее вероятная). На гистограмме она будет представлена самым высоким столбцом. На рис. 10.1 номинальная оценка составляет 3 дня; 1 Точное значение для нормального распределения равно 1:769, или 0,13. Вероятно, можно безопасно говорить о вероятности 1:1000.

Оценки

P: пессимистическая оценка. Эта оценка также должна быть крайне предельно пессимистической. В ней следует учесть все возможные неприятности, кроме ураганов, ядерной войны, блуждающих «черных дыр» и других катастроф. Математическая база также работает только в том случае, если вероятность этого исхода много меньше 1%. На графике представлена крайним правым столбцом (12 дней).

Планирование спринта.

Распределение задач производится не только менеджером. Участники должны быть сами причастны как к оценки, так и распределению задач, это важный принцип «Scrum». Скрам оперирует такими специфичными понятиями как «счастье». Самостоятельный выбор задачи, увеличивает его, что важно для методологии.

Продолжительность Спринта определяется множеством факторов, например, сроками проекта, количеством задач на одного участника и многими другими.

Собрания и демонстрации.

По своей задумке во внедряемой методологи выделяются три типа встреч, который также могут называться митинги:

«Ежедневный митинг-SCRUM». Ежедневные пятнадцатиминутные собрания

«Обзор спринта». Подведение итогов за одну итерацию (Спринт) демонстрация.

«Ретроспективное собрание». Совершенствование процесса.

Необходимость и частота таких собраний должна определяться согласно требованиям к проекту. Главное не забывать о том, что разработчики сами должны участвовать в постановке сроков, в том числе частоте иметь возможность устанавливать даты проведения собраний. Можно варьировать как количество встреч, так и их продолжительность.

Важно в конце каждого цикла проводить «демонстрацию» продукта для обсуждения его с заказчиком.

2.2 Разработка документа руководства

Согласно разработанной модели был создан документ «Руководство по внедрению гибкой методологии Scrum», в котором содержатся инструкции по внедрению методологии в проект.

Документ предназначен и может быть использован менеджерами, разработчиками, тестировщиками, аналитиками и может быть использован также людьми исполняющими другие роли, но причастные к производству программного обеспечения.

Документ разбит на содержание и содержит ссылки на необходимые российские и международные стандарты.

Документ разрабатывался, ориентируясь на два основных стандарта:

ГОСТ Р ИСО 21500—2014 Руководство по проектному менеджменту

РД 50-34.698-90 Автоматизированные системы. Требования к содержанию документов.

2.3 Внедрение в реальный проект

Выбор стека технологий разработки.

Данный разработанный подход был рассмотрен и одобрен к применению в реальном командном проекте.

Сначала, согласно разработанному плану, скрам-мастером совместно с участниками определил тип приложения и список технологий – веб приложение, в зависимости от специфики разработки – известно было, что приложение работать через браузер и быть доступным через систему подписок, т.е. может иметь множество заказчиков. Исходя из этого приложение было решено создавать на на стеке технологий Java:

- Java SE8
- Spring (MVC, Boot)

Введение команды в методологию и распределение ролей. Выбор системы оценки.

Для команды было проведено первое организационное собрание, на котором были разъяснены основополагающие принципы и приведены ссылки и документы для ознакомления с методологией «скрам». Было принято решения о записи текущего собрания и всех последующий для обращения участников.

Была достигнута договоренность о том, что задачи должны разбиваться минимальные блоки, с примерно одинаковой сложностью. Таким образом, т.к. сложность каждой задачи можно оценить константой, то количество и будет оценкой производительности труда.

Равновероятностное распределение по договоренности будет использоваться для оценки сроков выполнения сложных задач.

Т.к. приложение будет предоставляться через систему подписок, то к консультированию было привлечено несколько специалистов из области заказа, но единого заказчика приложение не имеет.

Внедрение программных продуктов и предоставление доступа.

Система контроля версий – Git. Десктопные клиенты каждый разработчик выбирает по своему усмотрению.

Пример интерфейса SmartGit

«Log версий проекта». Клиент Git «SmartGit».

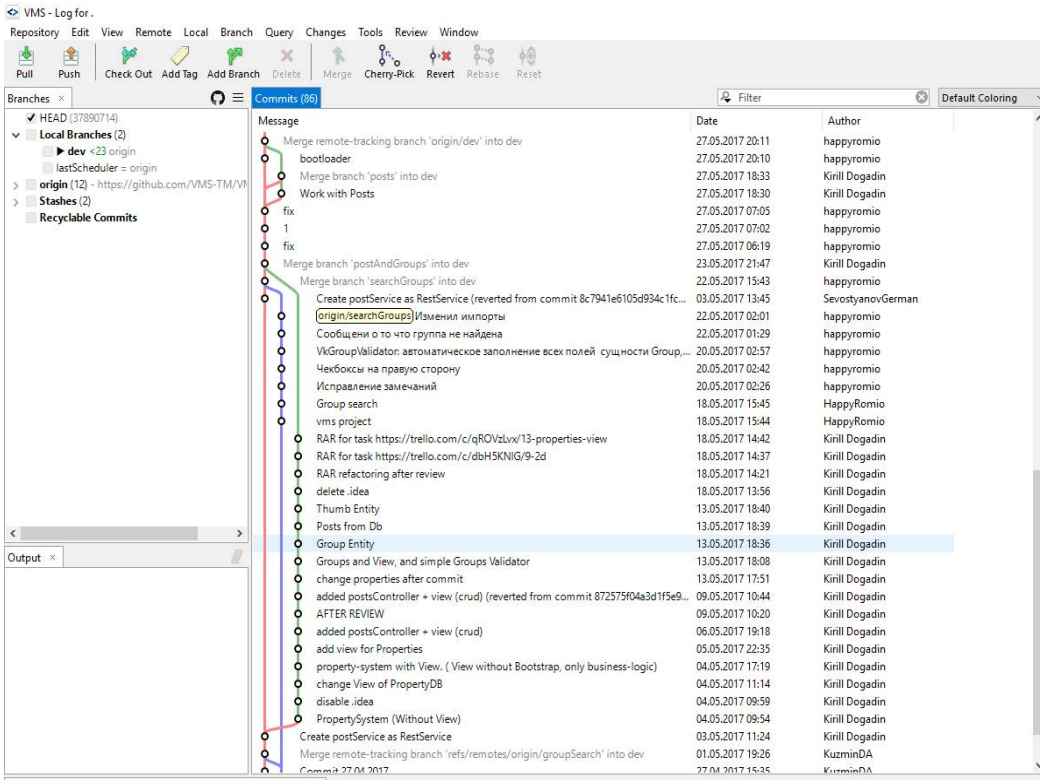

Рис. 2. Интерфейс программы SmartGit.

Интерфейс главного окна SmartGit/Hg похож на файловые менеджеры (такие, как Проводник Windows): слева в нем дерево каталогов, а справа — таблица файлов, содержащая все файлы репозитория и рабочего дерева каталогов. Он использует такие понятия файловых менеджеров, как диалоги, мастера Drag-n-drop для основных задач управления версиями, таких как сохранение, ветвление и сравнение. История открытого репозитория отображается в отдельном окне, также называемом Log window, а многие команды Git/Mercurial могут быть выполнены как через главное окно, так и через Log window, например, переключение между ветками, слияние веток, создание веток и тегов, и т. д.

Мессенджер – Slack.

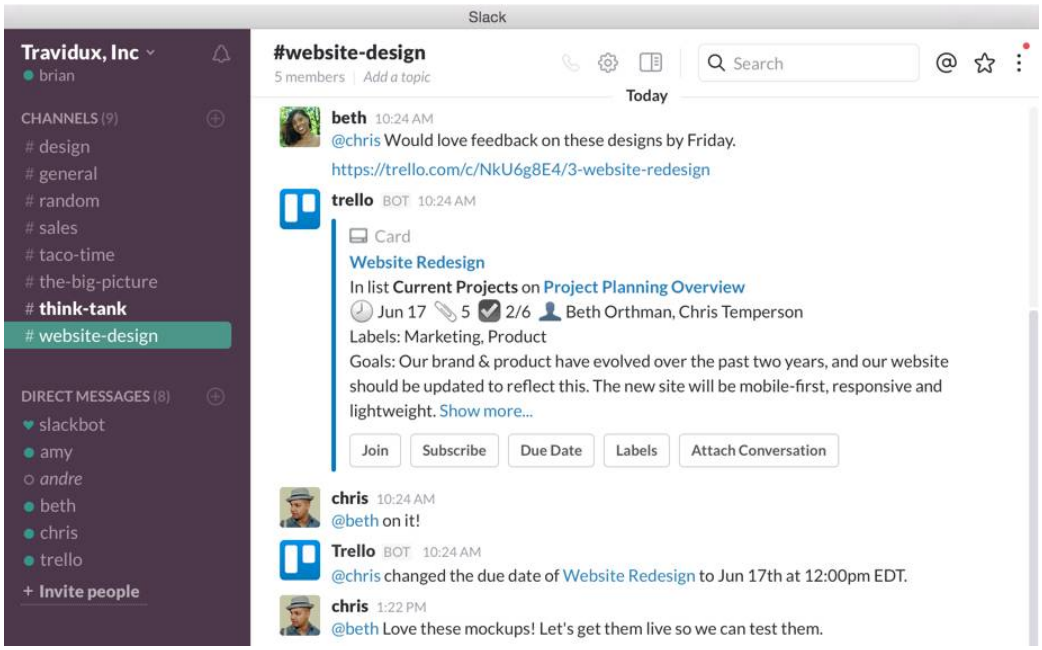

Рис. 3. Интерфейс программы Slack.

Slack собирает в одном окне обсуждения в общих темах (каналах), приватных группах и личных сообщениях; имеет собственный хостинг, режим превью для изображений и позволяет искать среди всех сообщений сразу. Кроме того, Slack поддерживает интеграцию с почти 100 сторонними сервисами, в их числе GitHub, Trello, что является основной причиной выбора.

В бесплатной версии Slack поддерживает неограниченное число пользователей, интеграцию с 10 внешними сервисами и поиск в архиве до 10 тысяч сообщений. Для проекта была использована бесплатная версия.

Отслеживание задач – trello.

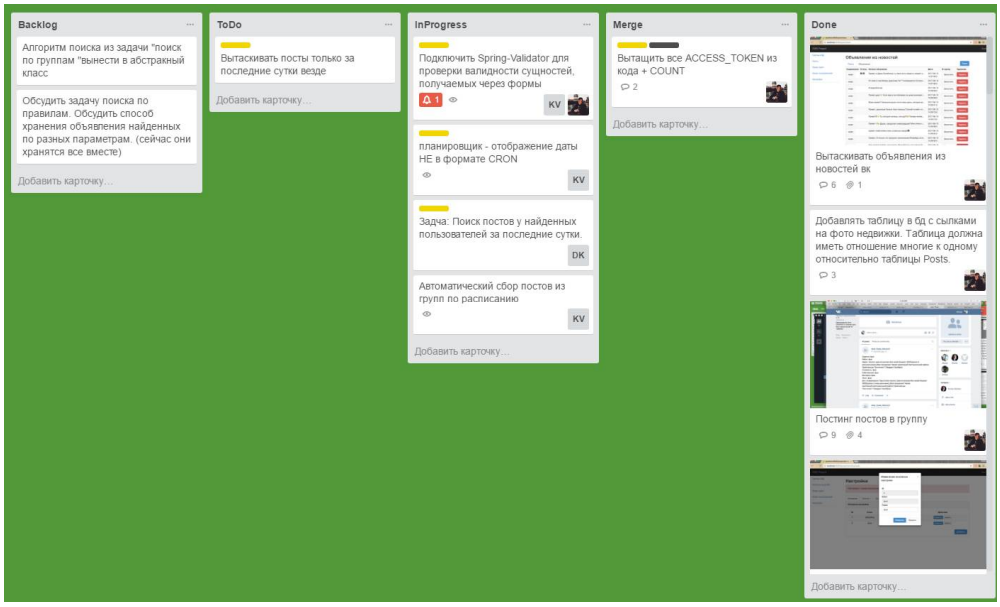

Рис. 4. Интерфейс программы trello.

Основным преимуществом Trello является возможность видеть несколько одновременно запущенных проектов и их состояние в текущий момент времени. При руководстве группой разработчиков или иных исполнителей, работающих над проектами с конечной датой выполнения или фиксированной целью, данная система может дать представление о ходе проектов в любой момент времени.

Trello — это пачки карточек. Каждая пачка показывает состояние любого проекта.

Trello доступно как на вебе, так и в виде приложения для iPhone. На iPad веб-сервис работает аналогично настольной реализации. Использование всех версий в настоящий момент бесплатное.

Данная система хорошо подходит для небольших проектов.

Формирование требований к системе. Создание Бэклога.

Т.к. приложение будет предоставляться через систему подписок, то к консультированию было привлечено несколько специалистов из области заказа, но единого заказчика приложение не имеет.

Бэклог был сформирован на основании требований и пожеланий консультантов, а также дополнен самими разработчиками.

Планирование первого спринта.

На первом собрании, посвященном планированию спринта, были назначены и распределены задачи в соответствии с пожеланиями разработчиков. Каждый имел возможность выбирать себе задачи, а также оценить их сложность.

Внедрение практики постоянных собраний.

График собраний было решено сделать плавающим и подстраиваться по возможности под каждого участника процесса разработки. Помимо отчётности на собраниях также происходило постоянное общение в мессенджере Slack. Запись всех собраний велась.

Было решено отказаться от ежедневных собраний, ведь в мессенджере всегда можно понятно статус выполнения и работы каждого участника.

Вместо трех собраний проводилось собрания, одно из которых являлось демонстрационным (концом спринта), а другое служило для решения возникших проблем и обсуждение того, как их можно избежать (ретроспективное). Иногда роли собраний совмещались.

Итоги внедрения

Первый месяц команда не полностью соблюдала принципы «Scrum», из-за чего отсутствовала «релизная» версия. Поэтому формально , подсчитанная, в баллах производительность ровнялась нулю.

Конец спринта не был определен конкретной датой, поэтому задачи брались из бэклога на выполнение, но не сдавались. Не был соблюден важный принцип – наличие демонстрационный версии программы в конце каждого спринта.

Большого прогресса в производительности удалось добиться, когда данный принцип стал соблюдаться.

«Пул энд Пуш», что означает получение и отправку последней актуальной версии, стал происходить часто.

Производительность выросла с 1-2 баллов на каждого члена команды в месяц, до 2-5 баллов в неделю.

Общее настроение, которое в методологии «скрам» является важным фактором, стало выше.

Также в проекте происходили непредвиденные изменения появление новых участников и уход старых, отсутствие скрам-мастера короткий период, что происходило незаметно, благодаря «гибкому» подходу, учитывающий человеческий фактор.

Проекту был поставлен плавающий срок три-четыре месяца. На данный момент проект создается ровно два месяца и выполнен примерно на девяносто процентов. Проект не только укладывается в срок, но и может быть закончен досрочно.

Заключение

Данная работа посвящена описанию технологии применения методологии «скрам». У данного методологии имеется ряд достоинств, которые оценены как самими и участниками проекта, так и формально. Применением данной методологии решается множество острых проблем связанных с управлением, как общих, так и присущих IT-области.

В процессе выполнения работы в рамках сформулированных задач было проделано следующее:

1. Рассмотрены различные методологии и способы их внедрения, их особенности сильные и слабые стороны, специфика применения.
2. В ходе рассмотрения способов внедрения, был разработан свой план-алгоритм внедрения гибкой методологии для внедрения в командный проект разработки программного обеспечения.
3. Создан электронный документ, содержащий поэтапный план внедрения методологии.
4. Методология «Скрам» была успешно внедрена в реальный проект согласно разработанному плану, план показал свою гибкость и жизнеспособность.

Список информационных источников

1. Богданов В. Управление проектами. Корпоративная система – шаг за шагом. – М.: Манн, Иванов и Фербер, 2012. – 248 с.
2. Брукс Фредерик. Мифический человеко-месяц, или как создаются программные системы. – М, Символ-плюс, 2010. – 304 с.
3. Вольфсон Б. Гибкие методологии разработки, версия 1.2 (электронная книга), 2012.
4. Грей, Клиффорд Ф. Управление проектами: учебник: пер. с англ. третьего, полн. перераб. изд./ Клиффорд Ф. Грей, Эрик У. Ларсон; [науч. Ред. перевода В. М. Дудников]. – М.: Издательство «Дело и Сервис», 2007. – 608 с. – Доп. тит. л. англ.
5. Грекул В. И., Коровкина Н. Л., Куприянов Ю. В. Методические основы управления ИТ-проектами. - Интернет-университет информационных технологий, 2011. – 392 с.
6. Гробовцов Г. Я. УПРАВЛЕНИЕ ПРОЕКТОМ: Учебно-методический комплекс. – М.: Изд. центр ЕАОИ, 2009. – 288 с.
7. Дитхелм Г. Управление проектами. В 2 т. Т. I. пер. с нем. – СПБ.: Издательский дом «Бизнес-пресса», 2004. – 400 с.
8. Дитхелм Г. Управление проектами. В 2 т. Т. II. пер. с нем. – СПБ.: Издательский дом «Бизнес-пресса», 2004. – 288 с.
9. Мазур И. И., Шапиро В. Д., Ольдерогге Н. Г. Управление проектами: Учебное пособие / Под общ. ред. И.И. Мазура. — 2-е изд. — М.: Омега-Л, 2004. — с. 664
10. Портни, Стэнли И. Управление проектами для «чайников».: Пер. с англ. – М.: Издательский дом «Вилямс», 2005. – 352 с.: ил. – Парал. тит. англ.
11. Разу М. А., Бронникова Т. М., Разу Б. М., Титов С. А., Якутин Ю. В. Управление проектом. Основы проектного управления: учебник / кол. авт,: под ред. проф. М. А. Разу. – М.: КНОРУС, 2006 – 768 с.

12. Сазерленд Д. SCRUM революционный метод управления проектами. 2016. Пер. с англ. - М.: Манн, Иванов и Фербер, 2016. – 142 с.
13. Товб А. С., Ципес Г. Л. Управление проектами: стандарты, методы, опыт. – М.: ЗАО «Олимп-Бизнес», 2003. – 240 с.: ил.
14. Черных Е. А. Agile Project Management и его история, М., «Менеджмент качества», 02(02)2008
15. Pekka Abrahamsson, Outi Salo, Jussi Ronkainen and Juhani Warsta. Agile Software Development Methods - Review and Analysis, by Technical Research Centre of Finland, 2002. – 107 p.
16. Gary Chin. Agile Project Management: How to Succeed in the Face of Changing Project Requirements, by AMACOM, 2004. – 229 p.
17. Charles G. Cobb. Making Sense of Agile Project Management: Balancing Control and Agility, by Wiley, 2011. – 265 p.
18. Michael Cohn. Succeeding with Agile: Software Development Using Scrum, by Addison-Wesley Professional, 2009. – 504 p.
19. Clifford F. Gray, Erik W. Larson. Project Management: The Managerial Process, 4th Edition, by McGraw-Hill/Irwin, 2008. - 608 p.
20. Bill Holtsnider, Tom Wheeler, George Stragand and Joseph Gee. Agile Development & Business Goals, by Elsevier Inc, 2010. – 256 p.
21. Mark C. Layton. Agile Project Management for Dummies, by John Wiley & Sons, 2012. – 360 p.
22. James P. Lewis. Fundamentals of Project Management (3rd Edition) by AMACOM Books, 2006. - 176 p.
23. Stanley E. Portny. Project Management For Dummies; 2nd Edition, by Willey Publishing, Inc., 2007. – 384 p.
24. Paul Roberts. Guide to Project Management, by Profile Books/The Economist, 2007. – 319 p.
25. Peter Schuh. Integrating Agile Development in the Real World, by Charles River Media, 2004. – 364 p.

26. Jason Westland, Project Management Life Cycle, by Kogan Page Ltd., 2006. – 255 p.
27. A Guide to the Project Management Body of Knowledge (PMBOK Guide), Fourth Edition, Project Management Institute, Inc., 2008.
28. H. Frank Cervone, (2011), "Understanding agile project management methods using Scrum", OCLC Systems & Services, Vol. 27 Iss: 1 pp. 18-22
29. Dr. Alistair Cockburn, "Using Both Incremental and Iterative Development", 2008, The Journal of Defense Software Engineering pp.27-30
30. Pete Deemer and Gabrielle Benefield. THE SCRUM PRIMER An Introduction to Agile Project Management with Scrum Version 1.04, goodagile from www.goodagile.com, 2007
31. Gabrielle Benefield. Rolling out Agile in a Large Enterprise, HICSS '08 Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences, 2008.
32. Pankaj Jalote, Aveejeet Palit, Priya Kurien, V.T. Peethamber, "Timeboxing: a process model for iterative software development", The Journal of System and Software (2004), pp. 117-127
33. Yu Beng Leau, Wooi Khong Loo, Wai Yip Tham and Soo Fun Tan. "Software Development Life Cycle AGILE vs Traditional Approaches", International Conference on Information and Network Technology (ICINT 2012)
34. Louise Ledbrook, "Waterfall Project Management: An Overview", 2012, http://projectcommunityonline.com/waterfall-project-management-an-overview.html
35. Xavier Amatriain Rubio, Gemma Hornos Cirera. Agile Methods in Research (presentation), by Telefonica, 2008. – 42 s.
36. Ken Schwaber, Jeff Sutherland, "The Scrum Guide", 2011. – 16 p.
37. Melonfire, "Understanding the pros and cons of the Waterfall Model of software development", 2006, http://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/6118423

38. www.agilelogic.com
39. http://agilemanifesto.org
40. www.koltsov.by
41. http://rsdn.ru/
42. http://tagline.ru/.
43. www.versionone.com
44. https://www.tiobe.com/tiobe-index/
45. http://www.wisegeek.com