Гибкие методологии разработки

Версия 1.2

Вольфсон Борис

1.Предисловие

В области производства программного обеспечения нет такой темы, вокруг которой ведется так много споров, как методологии для управления программными проектами. Все специалисты ищут серебряную пулю, хотя опыт подсказывает, что ее не существует. На смену одной методологии приходит другая, одну тему фокусирования всего софтверного сообщества сменяет другая – этот круг действительно бесконечен.

Матерые специалисты с седыми бородами настаивают на тяжеловесных методологиях с сотнями ролей, процессов, артефактов и толстенными описаниями. Молодые управленцы же предпочитают гибкие методологии или «Agile», как они говорят. Кто же прав в этом противостоянии отцов и детей?

В этой книге я расскажу о современных гибких методологиях, причем постараюсь осветить те аспекты, которые обычно не упоминаются либо раскрываются недостаточно глубоко. Кроме теории в книги содержится множество конкретных приемов и лучших методов, которые можно применять на практике.

Эта книга предназначена для широкого круга специалистов, работающих в области разработки программного обеспечения:

- Разработчики
- Ведущие разработчики и архитекторы
- Скрам-мастера
- Руководители проектов
- Владельцы продуктов
- Руководители отделов
- Аналитики
- Тестировщики
- Верстальщики
- Дизайнеры и специалисты по интерфейсу

Методы и инструменты, описанные в этой книге, позволяет организовать эффективную работу команд, состоящих из этих специалистов.

Об авторе

У меня есть достаточно обширный и разнообразный опыт в области разработки программного обеспечения и веб-разработки, чем я собственно и занимаюсь на коммерческой основе с 2003 года. За это время я успел поработать в разных должностях, начиная от верстальщика и разработчика, заканчивая руководителем крупного подразделения разработки с коллективом более чем в 100 человек.

С гибкими методологиями я познакомился в середине 2000-ных годов, а Scrum практикую с 2009 года. Моё видение гибких методологий (и Scrum в частности) прошло путь от набора лучших практик до философии производства программного обеспечения.

Можно сказать, что я отношусь к современному поколению управленцев, которые сочетают неплохое знание тяжеловесных подходов, но уверены, что настоящее и будущее за гибкими методологиями.

Эта книга не претендует на звание «единого и непогрешимого» источника знаний по гибким методологиям, потому что они постоянно развиваются: появляются новые подходы, дорабатываются старые и расширяются сферы применения.

Также на этих страницах отражено исключительно моё мнение и понимание гибких методологий, не связанное с моими нынешними или будущими работодателями или сообществами, в которых я состою. Тем не менее, не отрицаю их сильного влияния на меня.

Связаться с автором можно следующими способами:

- http://www.facebook.com/borisvolfson
- http://twitter.com/borisvolfson
- borisvolfson@gmail.com

Благодарности

Спасибо всем, кто помогал в работе над данными материалами, в том числе по плану внедрению Agile. Жирным шрифтом выделены авторы наиболее обширных и ценных комментариев, предложений и замечаний:

- **Евграшин Тимофей**
- Гармаш Максим
- **Ковязин Егор**
- **Козлов Илья**
- **Колосова Ксения**
- **Лукьянчикова Наталья**
- Паньшин Дмитрий
- Подурец Михаил
- Рогачев Сергей
- Свердлов Андрей
- Сорокин Евгений
- Сурикова Ирина
- Тарасенко Анна
- Уразбаев Асхат
- **Шабакаева Лия**

Особую благодарность также выражаю многочисленным рок и хард-рок группам, без которых создание этой книги было бы невозможно.

Благодарности компаниям и организациям

2.Agile-методологии

Семейство гибких методологий буквально ворвалось на софтверную сцену и перевернуло всё с ног наголову:

- Мы стали сосредотачиваться **на людях и улучшении коммуникаций между ними**, вместо выстраивания сверхжестких процессов;
- Мы стали концентрироваться **на продукте**, вместо того чтобы писать изощренную проектную документацию, которую никто не читает;
- Мы больше не заставляем расписываться заказчика кровью, ограничивая его жесткими и неудобными условиями договоров, мы строим действительно **партнерские отношения** и выясняем, что хочет заказчик и что ему нужно;
- Мы всегда **готовы к изменениям**, потому что понимаем, что мир вокруг нас меняется и то, что месяц назад казалось абсолютно необходимым в нашем проекте, сейчас уже не нужно вообще.

В более строгом варианте эти тезисы были сформулированы отцами-основателями гибких методологий в документе, который получил название Agile Manifesto:

- **Люди и их взаимодействие** важнее процессов и инструментов;
- **Готовый продукт** важнее документации по нему;
- **Сотрудничество с заказчиком** важнее жестких контрактных ограничений;
- **Реакция на изменения** важнее следования плану.

Рисунок 1. Визуализация ценностей манифеста гибкой разработки

Полный текст манифеста (и его переводы) доступны на сайте http://agilemanifesto.org. Каждый, кто хочет работать по гибкой методологии должен ориентироваться на эти четыре «взвешивания»: как только начинает перевешивать не та «чаша весов», надо задуматься: «А на верном ли я пути?». Таким образом, манифест станет вашим компасом, по которому можно определять направление движения.

Принципы Agile

1. Наш высший приоритет – это удовлетворение заказчика с помощью частых и непрерывных поставок продукта, ценного для него.
2. Мы принимаем изменения в требования, даже на поздних этапах реализации проекта.
3. Гибкие процессы приветствуют изменения, что является конкурентным преимуществом для заказчика.
4. Поставлять полностью рабочее программное обеспечение каждые несколько недель, в крайнем случае, каждые несколько месяцев. Чем чаще, тем лучше.
5. Представители бизнеса и команда разработки должны работать вместе над проектом.
6. Успешные проекты строятся мотивированными людьми. Дайте им подходящую окружающую среду, снабдите всем необходимым и доверьте сделать свою работу.
7. Самый эффективный метод взаимодействия и обмена информацией – это личная беседа.
8. Рабочее программное обеспечение – главная мера прогресса проекта
9. Гибкие процессы способствуют непрерывному развитию. Все участники проекта должны уметь выдерживать такой постоянный темп.
10. Постоянное внимание к техническому совершенству и качественной архитектуре способствуют гибкости.
11. Простота необходима, как искусство максимизации работы, которую не следует делать.
12. Лучшая архитектура, требования, дизайн создается в самоорганизующихся командах.
13. Команда постоянно ищет способы стать более эффективной, путем настройки и адаптации своих процессов.

Авторы манифеста

В феврале 2001 года 17 специалистов (консультантов и практиков) в местечке под названием Snowbird в штате Юта собрались, чтобы обсудить легковесные методики разработки. В результате родился документ «Манифест гибких методологий разработки» (Agile Manifesto). Позволю себе привести список авторов манифеста и краткую информацию о них.

Кент Бек – создатель разработки через тестирование и экстремального программирования. Автор нескольких книг на эти темы и соавтор JUnit.

Майк Бидл – основатель и генеральный директор e-Architects Inc., консалтинговой компании, которая специализируется на разработке распределенного ПО. Он также является соавтором книги **«**Scrum, Agile Software Development», написанной совместно с Кеном Швабером.

Эйри ван Беннекум участвовал в разработке методологии DSDM с 1997 года. А до этого активно работал над быстрой разработкой (Rapid Application Development).

Алистер Кокбёрн известен исследованиями проектных команд и активным участием в разработке семейства методологий Crystal.

Уорд Каннингем – основатель Cunningham & Cunningham, Inc. Он также широко известен за громадный вклад в развитие объектно-ориентированного программирования, экстремального программирования и концепцию вики.

Джеймс Гренninг – тренер и консультант по гибким методологиям. Является специалистом в области разработки и тестирования встроенного программного обеспечения и автором книги «Test Driven Development for Embedded C» (Grenning).

Стивен Меллор – специалист в области разработки программного обеспечение, соавтор метода ООАП Шлаера-Меллора, работал над UML в составе Object Management Group.

Мартин Фаулер – главный исследователь в компании Thoughtworks. Автор многих работ и книг по паттернам анализа, UML, рефакторингу и экстремальному программированию.

Джим Хайсмит – ведущий разработчик методологии «Adaptive Software Development» и автор соответствующей книги.

Эндрю Хант – соавтор книги «Pragmatic Programmer: From Journeyman to Master» и других работ, связанных с разработкой ПО.

Рон Джеффрис – владелец сайта XProgramming.com, консультант в компании Object Mentor и соавтор книги «Extreme Programming Installed».

Джон Кёрн участвовал во многих проектах по исследованиям и разработки в области авиастроения. Также он является евангелистом объектно-ориентированного программирования с начала 90-х годов.

Брайан Мэрик – программист и консультант по тестированию программного обеспечения. Основной вклад Брайана заключается в исследование процессов тестирования ПО в гибких методологиях разработки.

Роберт Мартин находится в отрасли разработки ПО с 1970 года. Он является президентом и основателем фирмы Object Mentor Inc., которая специализируется на консалтинге в области экстремального программирования, гибких методологиях разработки, консалтинге по архитектуре ПО и так далее. Он также является соавтором ряда книг по программированию и разработке программного обеспечения.

Кен Швабер – президент компании Advanced Development Methods (ADM), которая занимается улучшением методов разработки ПО. Он является опытным разработчиком, менеджером продуктов и консультантом. В начале 90-х годов он работал с Джеффом Сазерлендом над первыми версиями Scrum. Он также является соавтором книги «Scrum, Agile Software Development».

Джефф Сазерленд – технический директор (CTO) компании PatientKeeper, которая занимается создание ПО для медицинских учреждений. За свою карьеру был техническим директором (или вице-президентом по технологиям) в девяти компаниях. Свою известность приобрел, как изобретатель методологии Скрам.

Дейв Томас верит в то, что сердцем проекта по разработке является не методология, а люди. По этой причине он является соавтором книги «The Pragmatic Programmer».

Scrum в двух словах

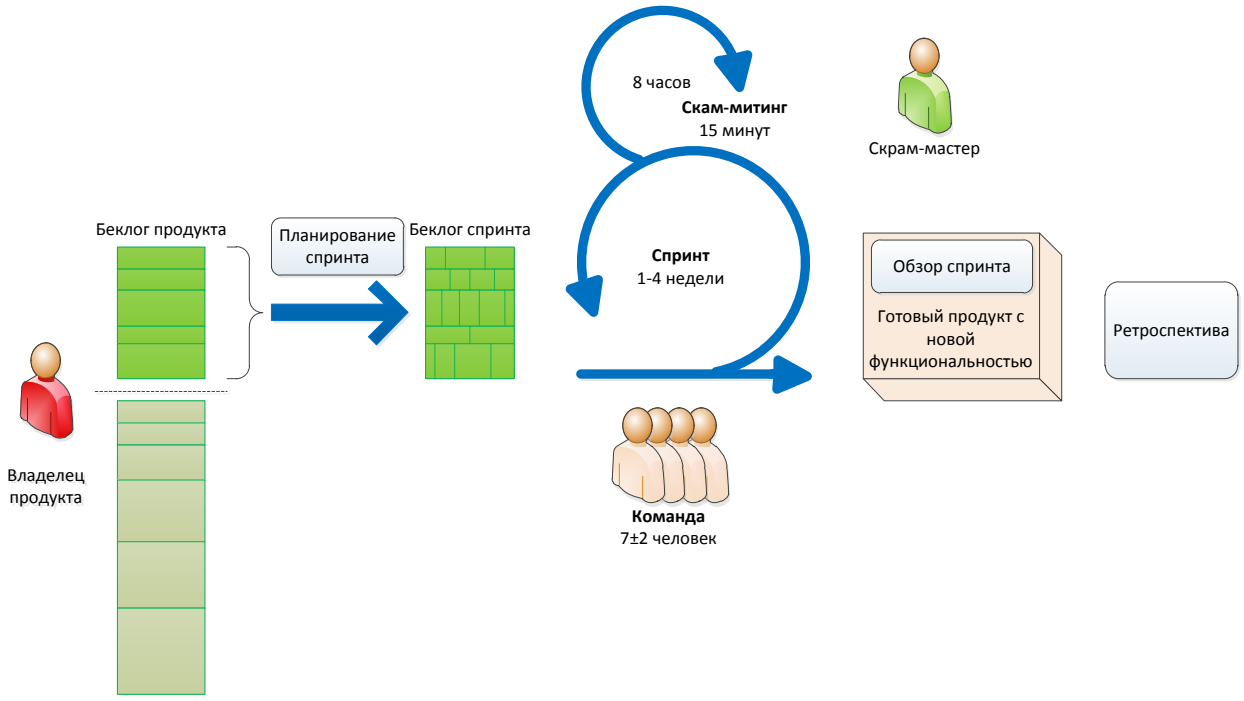


Рисунок 2. Общая схема Scrum

Организуйте работу в вашей организации в небольших кроссфункциональных командах, которые содержат всех необходимых специалистов. Выделите человека - скрам-мастера, который будет отвечать за соблюдение процессов в команде и конструктивную атмосферу.

Требования разбейте на небольшие, ориентированные на пользователей, функциональные части, которые максимально независимы друг от друга, в результате чего получите беклог продукта. Затем упорядочите элементы беклога по их важности и произведите относительную оценку объемов каждой истории. Выделите отдельного человека – владельца продукта, который будет отвечать за требования и их приоритеты, замыкая на себя всех заинтересованных лиц.

Всю работу ведите короткими (от 1 до 4 недель) фиксированными итерациями – спринтами, поставляя в конце каждого из них законченный функционал, который можно при необходимости вывести на рынок – инкремент продукта. Команда согласно своей скорости набирает задачи на неизменяемую по времени итерацию – спринт. Каждый день проводится скрам-митинг, на котором команда синхронизирует свою работу и обсуждает проблемы. В процессе работы члены команды берут в работу элементы беклога согласно приоритету.

В конце каждого спринта проводите обзор спринта, чтобы получить обратную связь от владельца продукта, и ретроспективу спринта, чтобы оптимизировать ваши процессы. После этого владелец продукта может изменить требования и их приоритеты и запустить новый спринт.

Не Scrum'ом единым

Scrum не является единственной гибкой методологией, хотя на данный момент он самый популярный среди собратьев. Но знание и понимание других методологий важно, чтобы понимать их преимущества и недостатки. Кроме того, на поздних этапах адаптации Scrum можно позаимствовать различные практики из этих методологий.

Также хотелось бы поделиться результатами исследования Agile Survey о популярности гибких методологий:

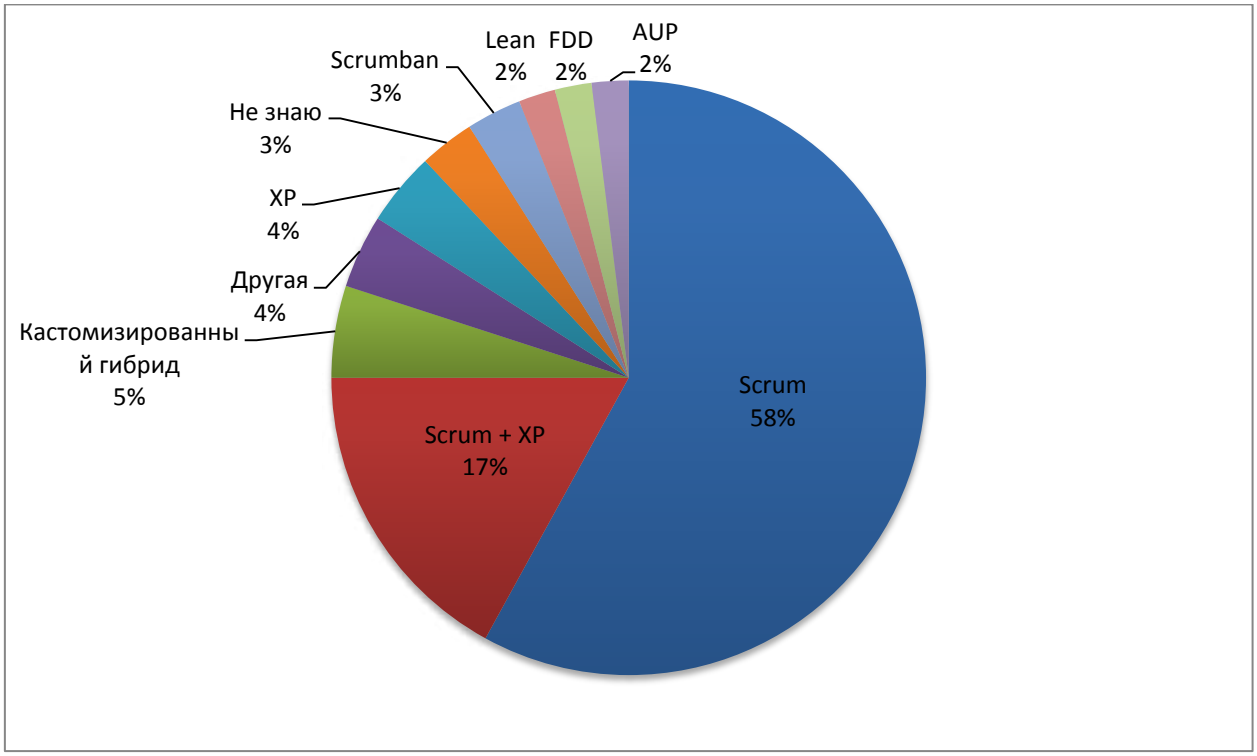


Рисунок 3. Популярность гибких методологий

Экстремальное программирование

Практики экстремального программирования можно разделить условно на управленческие и инженерные:

Рисунок 4. Практики экстремального программирования

Разделение весьма условное, но оно показывает, что экстремальное программирование имеет много общего со Scrum, но имеются и определенные отличия. Инженерные практики будут рассмотрены подробнее в соответствующем разделе, а управленческие практики применяются обычно из Scrum.

Crystal Clear

Crystal Clear – это легковесная гибкая методология, созданная Алистером Коуберном (Cockburn, 2004). Она предназначена для небольших команд в 6-8 человек для разработки некритичных бизнес-приложений. Как и все гибкие методологии Crystal Clear больше опирается на людей, чем на процессы и артефакты.

Crystal Clear использует семь методов/практик, три из которых являются обязательными:

1. **Частая поставка продукта**
2. **Улучшения через рефлексию**
3. **Личные коммуникации**
4. Чувство безопасности
5. Фокусировка
6. Простой доступ к экспертам
7. Качественное техническое окружение

Как вы видите, все практики характерны для семейства Agile-методологий. В графическом виде практики Crystal Clear можно изобразить таким образом:

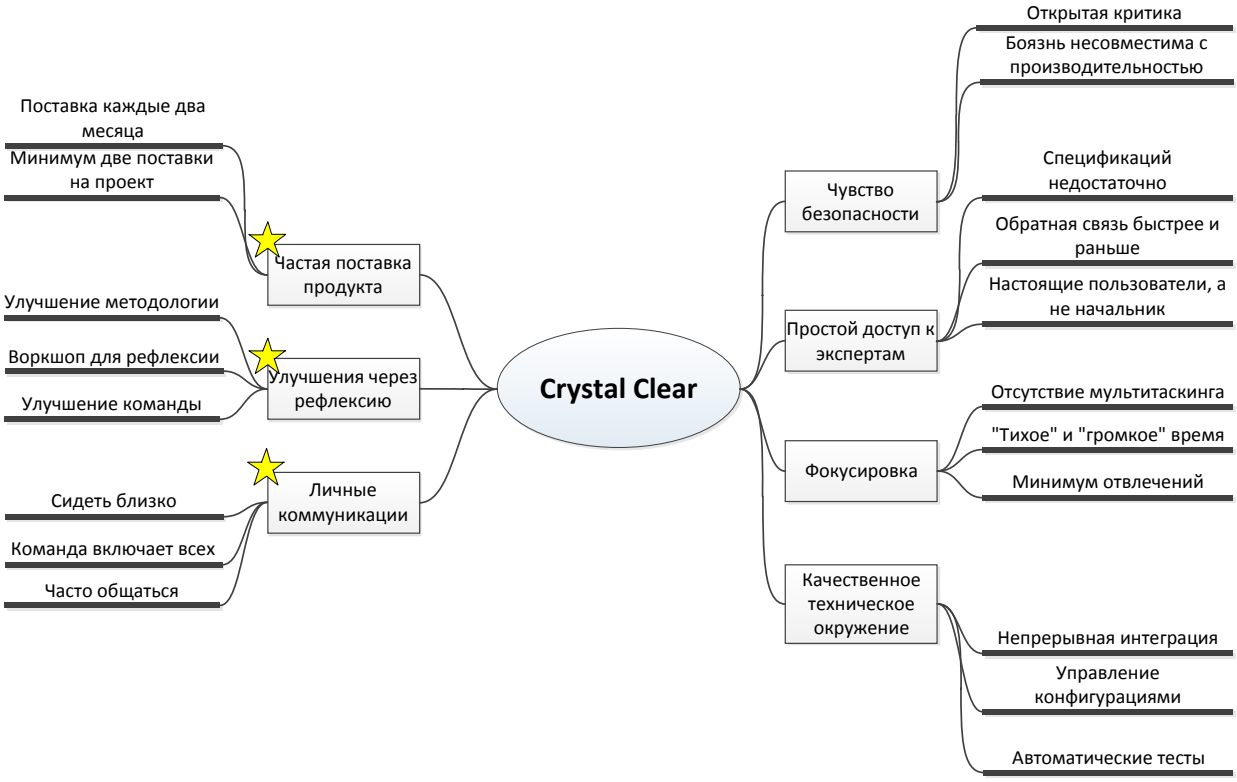


Рисунок 5. Методы и практики Crystal Clear

Dynamic Systems Development Method (DSDM)

Методология DSDM основана на подходе RAD (Rapid Application Development) и включает в себя три стадии:

1. Предпроектная стадия, на которой авторизуется реализация проекта, определяются финансовые параметры и команда.
2. Жизненный цикл проекта представляет собой реализации проекта и включает в себя пять этапов.
3. Постпроектная стадия обеспечивает качественную эксплуатацию системы.

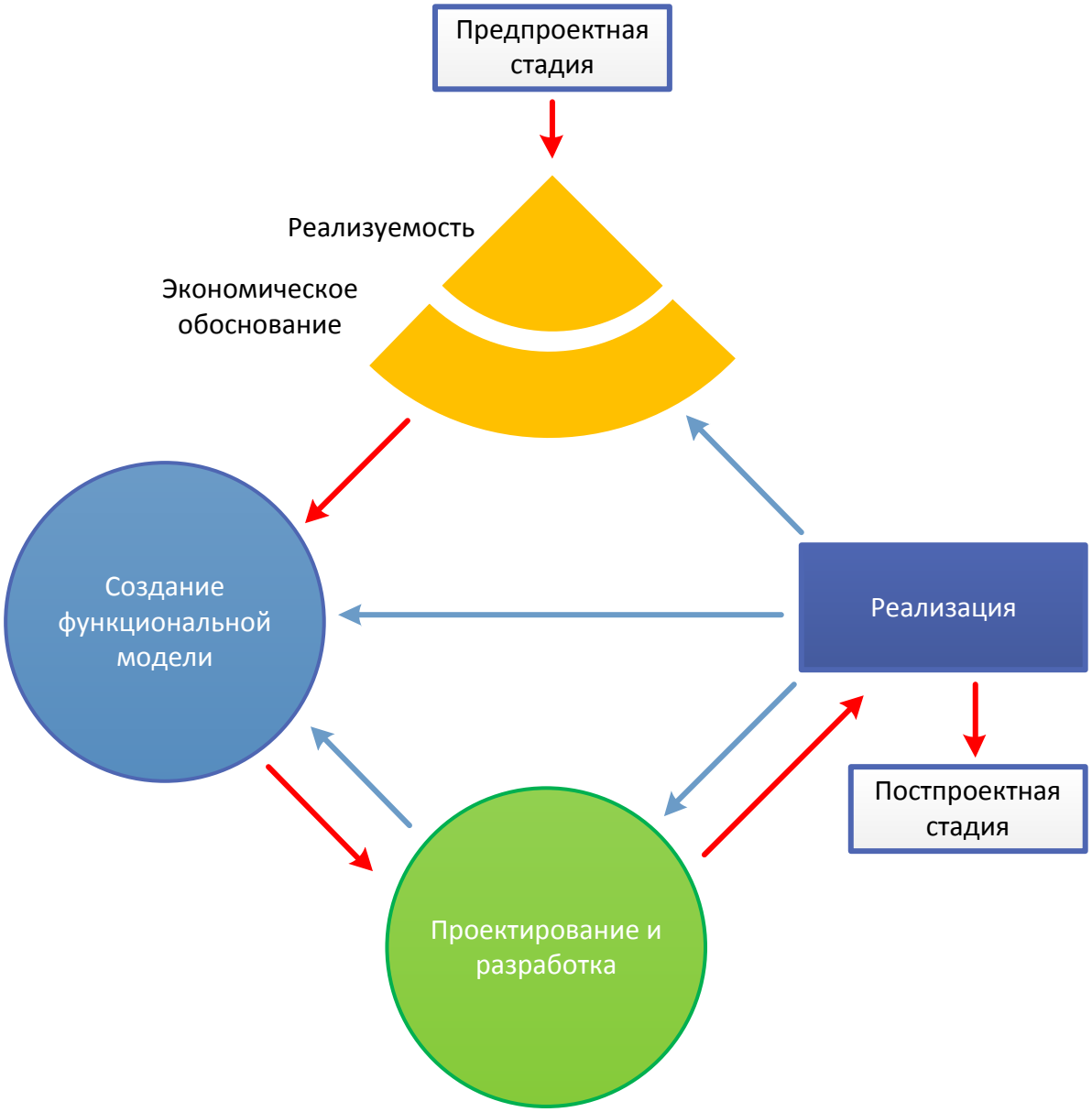


Рисунок 6. Общая схема DSDM

Жизненный цикл проекта включает в себя пять стадий (первые две фактически объединяются):

1. Определение реализуемости
2. Экономическое обоснование
3. Создание функциональной модели
4. Проектирование и разработка
5. Реализация

Agile Unified Process

Agile Unified Process (AUP) – упрощенная версия IBM Rational Unified Process, созданная Скоттом Амблером и состоящая из семи методов:

1. Моделирование используется для понимания бизнес-требования и предметной области.

2. Реализация – это преобразование модели в исполняемый код с модульными тестами.
3. Тестирование – способ поиска дефектов и верификации системы на предмет соответствия требованиям.
4. Размещение – доставка готовой системы пользователям.
5. Управление конфигурациями – управление доступом и версиями артефактов проекта.
6. Управление проектом – непосредственные активности, связанные с ходом проекта: управление и координация людей, управление рисками, управление финансами и так далее.
7. Среда – совокупность процессов, инструментов, стандартов и правил.

Feature-driven development

Feature-driven development – методология, созданная Джеффом Де Люка. Разработка ведется в пять этапов:

1. Построение модели
2. Создание списка функций
3. Планирование реализации функций
4. Создание архитектуры для функций
5. Реализация функций

Достоинством этой методологии стоит считать изначальную поддержку больших групп разработчиков, так как отдельные функции разрабатываются отдельными мини-командами во главе с ведущим разработчиком. Разделение и координация происходят на этапах 3-4.

ICONIX

ICONIX – это методология разработки программного обеспечения, сфокусированная на анализе требований и моделировании. В рамках ICONIX используется подмножество UML для анализа требований:

- Диаграмма вариантов использования;
- Диаграмма классов;
- Диаграмма робастности;
- Диаграмма последовательности.

Про использование методологии ICONIX смотрите главу «Анализ требований» раздел «Процесс ICONIX».

Канбан

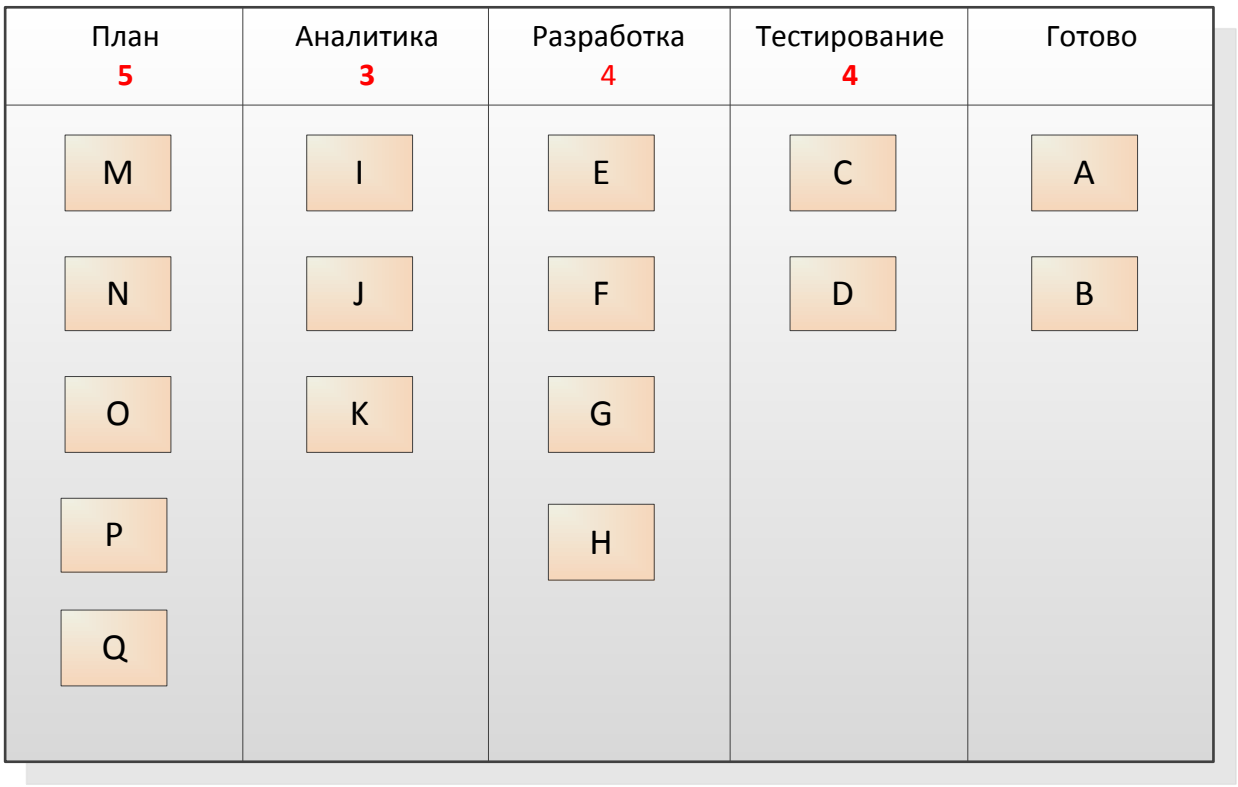


Рисунок 7. Доска задач в рамках канбана

Для того чтобы использовать канбан, достаточно следовать всего трём правилам. Таким образом, канбан является самой «не директивной методологией». Это может быть как плюсом, так и минусом, поэтому внедрять и использовать канбан хорошо после Scrum, а еще лучше не отказываться от полезных практик этого гибкого фреймворка.

Таким образом, канбан - это высоко адаптивный инструмент, который требует от команды, решившей использовать его, соответствующего уровня самоорганизации и дисциплины:

Среднее время реализации задачи (lead time) – метрика, показывающая, насколько быстро задача проходит весь поток.

1. **Визуализируйте производственный процесс**
   Для этого обычно используют доску, размеченную по этапам работы над задачей. Конечно, более продвинутым вариантом будет использовать плазму/проектор, и выводить туда состояние трекера.
1. **Ограничивайте количество незавершенной работы (WIP, Work In Progress)**
   У каждого столбца-состояния команда указывает максимальное количество задач, которые могут в нем находиться. Таким образом, минимизируется переключение между задачами и связанные с этим потери при производстве.
2. **Оптимизируйте процесс**
   Третье правило – основа оптимизации производства в рамках канбана. Необходимо отслеживать среднее время задачи и уменьшать его (например, с использованием Value Stream Mapping см. соответствующий раздел).

Совершенствование процесса

Ввиду малого количества директив, начать использовать канбан можно легко и просто, но для того, чтобы использовать данный инструмент эффективно, нужно не отклоняться от процесса непрерывного совершенствования.

3.Scrum – гибкий управленческий фреймворк

Сегодня самой популярной гибкой методологией разработки ПО является Scrum. Если вы спросите любого практика Agile, он обязательно подтвердит это, хотя каждое слово в предыдущей фразе неправда.

Начнем с конца – Scrum используют не только для разработки ПО, он отлично подходит для многих процессов по созданию продукта: от венчурных до маркетинговых продуктов. И соответственно подбираемся ко второму пункту – Scrum вовсе не методология, это гибкий управленческий фреймворк. Откуда следует и третий пункт – Scrum обычно дополняют инженерными практиками из других гибких методологий (например, практики разработки из экстремального программирования, или практики анализа и сбора требований из ICONIX). Так что в дальнейшем, если не оговорено иное, под Agile будем подразумевать семейство гибких методологий, а Scrum будем рассматривать в качестве управленческого фреймворка, дополненного практиками из других гибких методологий. Но довольно буквоедствовать, давайте начнем знакомиться со Scrum.

> Это был первый случай в моей жизни, когда я увидел, как методология работает "прямо из коробки". Просто подключи и работай. И при этом все счастливы: и разработчики, и тестеры, и менеджеры. Вопреки всем передрягам на рынке и сокращению штата сотрудников, Scrum помог нам выбраться из сложнейшей ситуации, позволил сконцентрироваться на наших целях и не потерять свой темп.
>
> Хенрик Книберг,
> Scrum и XP: Заметки с передовой

Классический Scrum состоит из следующих элементов:

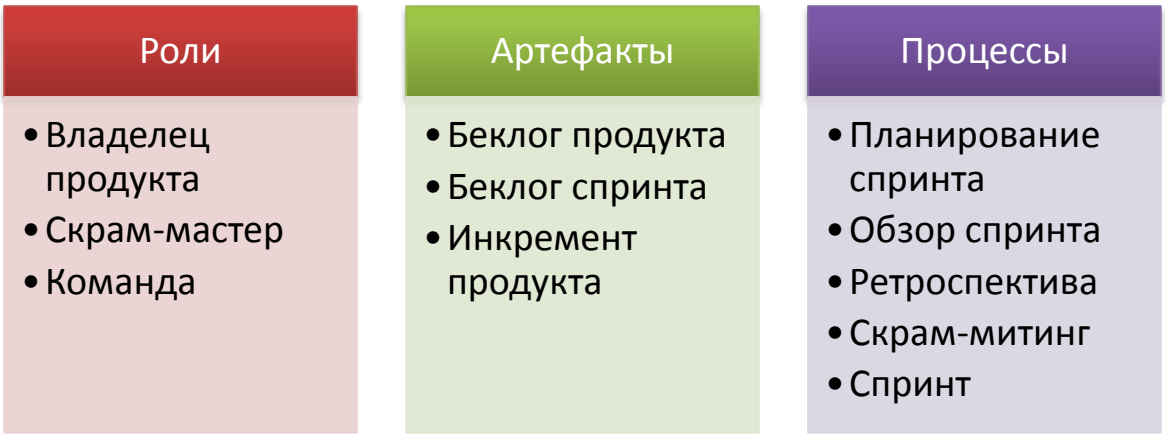


Рисунок 8. Элементы Scrum

Роли

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

- **Владелец продукта** (Продукт оунер, Product owner, Менеджер продукта) – это человек, ответственный приоритезацию требований и часто за их создание.
- **Скрам-мастер –** член команды, который дополнительно отвечает за процессы, координацию работы команды и поддержание социальной атмосферы в команде.
- **Команда –** 7±2 человек, которые реализуют требования владельца продукта.

Обязанности скрам-мастера

Скрам-мастер должен ежедневно следить за тем, чтобы скрам-митинг начинался и заканчивался вовремя. Рекомендуется выделять определенное время каждому участнику, чтобы общая протяженность скрама-митинга не превышала заранее оговоренного времени (например, 15 минут).

Скрам-мастер в начале спринта помогает команде проводить планирование спринта и запуск спринта.

В конце спринта скрам-мастер организует демонстрацию результатов спринта при участии всех заинтересованных лиц и проводит ретроспективу при участии всех членов проектной команды.

Также в обязанности скрам-мастера входит мониторинг социальных аспектов команды и поддержание командного духа.

Артефакты

- **Беклог продукта (Product Backlog) –** приоритезированный список требований с оценкой трудозатрат. Обычно он состоит из бизнес-требований, которые приносят конкретную бизнес-ценность и называются элементы беклога.
- **Беклог спринта (Sprint Backlog) –** часть беклога продукта, с самой высокой важностью и суммарной оценкой, не превышающей скорость команды, отобранная для спринта.
- **Инкремент продукта –** новая функциональность продукта, созданная во время спринта спринта.

Процессы

Большинство процессов Scrum носят характер встреч, так как данная методология основана на качественных коммуникациях.

Скрам-митинг

Скрам-митинг (Scrum meeting, скрам, ежедневный скрам, планерка) – собрание членов команды (с возможностью приглашения владельца продукта) для синхронизации деятельности команды и обозначения проблем. Каждый член команды отвечает на три вопроса:

1. Что было сделано с предыдущего скрам-митинга?

2. Какие есть проблемы?
3. Что будет сделано к следующему скрам митингу?

Если первый и третий пункт служат для синхронизации деятельности команд, то второй пункт очень важен для выработки решений проблем: если проблема действительно небольшая, ее можно решить или выработать решение прямо на скрам-митинге, если серьезная и требует обсуждения, ее решить после скрам-митинга.

Планирование спринта

Для планирования спринта необходимо иметь качественный беклог, что означает следующее:

- все элементы бэклога должны иметь уникальную числовую важность;
- самые важные элементы бэклога должны быть уточнены и понятны всей команде и владельцу продукта;
- владелец продукта должен четко представлять, что будет реализовано в рамках каждого элемента бэклога.

Основным результатом планирования спринта является беклог спринта – список задач, которые команда планирует реализовать в рамках спринта. Поскольку длина спринта в Scrum жестко фиксирована, то команда определяет количество элементов беклога (объем работ), которые она может реализовать. Можно данную ситуацию отобразить на классическом «треугольнике управления проектами»:

Обзор спринта

Рисунок 9.Проектный треугольник в Scrum

Обзор спринта (также часто используется термин «демонстрация» или сокращенно «демо») – показ владельцу продукта (и заинтересованным лицам) работающего функционала продукта, сделанного за спринт. Основная задача проведения обзора спринта заключается в получении обратной связи, а общий цикл ее получения выглядит следующим образом:

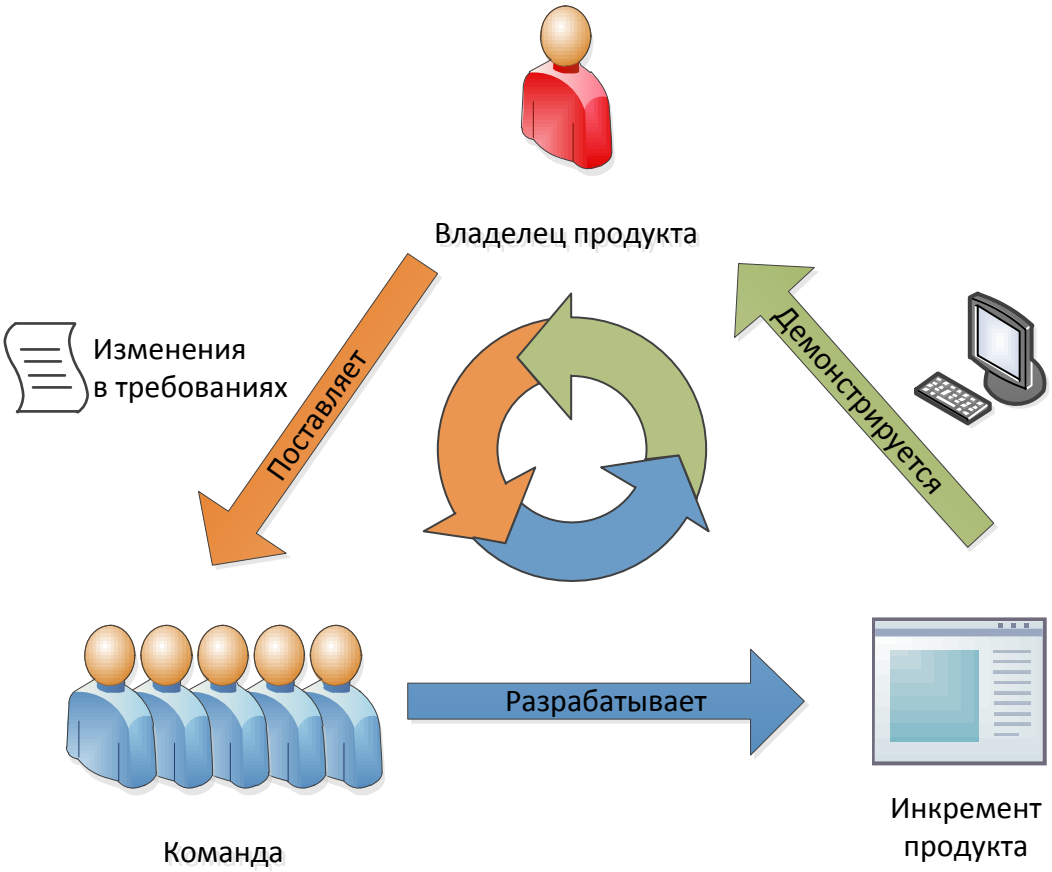


Рисунок 10. Получение обратной связи в рамках Scrum

Демонстрация результатов работы не только мотивирует команду, но и подталкивает реализовывать задачи полностью.

Если взглянуть еще раз на манифест Agile, то там есть пункт, который непосредственно касается демонстраций: «Готовый продукт важнее документации по нему». И действительно основной мерой прогресса является функционал нашего продукта, поэтому показывать на демонстрации надо именно программу. Если ваш заказчик находится не в одном помещении с вами, используйте специальные средства демонстрации. Гораздо хуже будет отправить заказчику презентацию или отчет по сделанному функционалу, ведь мы хотим получить качественную обратную связь.

В обзоре спринта обязательно должна принимать участие вся команда, при этом возможны разные стратегии показа. Антипаттерном можно назвать демонстрацию функционала одним человеком, например, скрам-мастером.

К паттернам можно отнести демонстрацию «чужих» реализованных элементов беклога и привлечение к демонстрации аналитиков, тестировщиков, верстальщиков, UI-специалистов и так далее. Такой подход позволяет выработать командную ответственность за результат.

Ретроспектива

В долгосрочном плане ретроспективы (или сокращенно «ретро») являются самой важной практикой Scrum: ведь именно они позволяют адаптировать и кастомизировать Scrum, делая из него по-настоящему гибкий фреймворк для управления проектами.

Ретроспективу традиционно проводят после обзора спринта спустя небольшое количество времени, чтобы оперативно получить фидбек. Скрам-мастер собирают всю команду для обсуждения результатов спринта. Рекомендуется на ретроспективу приглашать владельца продукта для получения дополнительной обратной связи.

Структура ретроспективы

Обычно ретроспектива занимает от 30 минут до 4 часов и ее продолжительность зависит от следующих факторов:

- Длина спринта: чем длиннее спринт, тем больше команда успевает сделать и тем больше материала для обсуждения;
- Размер команды: чем команда больше, тем больше надо времени, чтобы у каждого ее члена была возможность высказаться и тем больше функционала команда успевает сделать;
- Наличие проблем: со временем команда решает проблемы и ретроспективы сокращаются по времени.

В процентном соотношении принято выделять такую структуру:

Рисунок 11. Структура ретроспективы

Также традиционным является формат по сбору данных, который заключается в ответах каждого участника на три вопроса:

1. Что было сделано хорошо?
2. Что можно улучшить?

3. Какие улучшения будем делать?

Количество улучшений, которые команда берет в реализацию, не должно превышать 2-3, чтобы не снизить скорость реализации бизнес-функционала и не потерять фокус. Команда должна обязательно в том или ином виде составить план улучшений для контроля их исполнения.

Для максимальной открытости и прозрачности обсуждения необходимо использовать основное правило ретроспективы, которое можно озвучивать в начале:

«В независимости от того, что удастся выяснить в результате ретроспективы, каждый член команды сделал всё, чтобы добиться успеха»

Если у команды отсутствуют яркие проблемы, то желательно следующие темы обсудить на ретроспективе:

- скорость команды и ее изменение по сравнению с предыдущими спринтами;
- нереализованные истории пользователей и причины опоздания;
- дефекты и их причины;
- качество процессов (нарушения, отступления).

К паттернам можно отнести анализ сделанных улучшений за несколько прошлых спринтов. Такая «ретроспектива ретроспектив» может проводить раз в 4 спринта и позволяет контролировать уровень сделанных улучшений.

4.Управление продуктом

Для владельца продукта Scrum даёт уникальную возможность сосредоточиться на требованиях, вместо того, чтобы заниматься диспетчеризацией задач или иной оперативной деятельностью забывая о стратегии.

Для эффективного и непротиворечивого управления беклогом следует использовать два правила:

1. Добавлять истории пользователей в беклог может, кто угодно.
2. Определять порядок реализации, путем выставления важности, может только владелец продукта.

Построение бизнес модели

Наиболее подробно построение бизнес моделей описано в работе Александра Остервальда и Ива Пинье «Построение бизнес моделей. Настольная книга стратега и новатора». Бизнес-канвасы представляют собой способ визуализации бизнес модели и их можно адаптировать к проектам по разработки ПО (Филиппов, и др., 2011):

<table>
<tr><td rowspan="2">Проблемы
3 самые важные проблемы заказчиков</td><td>Решения
Функциональность продукта, которая решает проблемы</td><td rowspan="2">Уникальное предложение
Простое и понятное сообщение, почему заказчик должен выбрать именно вас</td><td>Преимущество
Что нельзя быстро скопировать или купить</td><td rowspan="2">Сегменты заказчиков
Заказчики или конечные пользователи вашего продукта</td></tr>
<tr><td>Метрики оценки
Как можно понять, что ваш продукт успешно решает проблемы?</td><td>Каналы продаж
Как ваш продукт достигнет ваших заказчиков?</td></tr>
<tr><td colspan="3">Структура затрат
На что вы будете тратить деньги при изготовлении продукта?</td><td colspan="2">Потоки прибыли
Как вы будете получать прибыль?</td></tr>
</table>

Такую визуализацию лучше всего проводить при помощи стикеров на доске с участием всей команды и заинтересованных лиц, например, маркетологов и продавцов.

Персоны

Практика анализа персон («Personas») пришла в управление продуктами из практик User Experience. Она заключается в описании пользователей создаваемого продукта как реального персонажа с конкретными ценностями и целями.

Сторимаппинг

После выявления персон необходимо перейти к выявлению функционала, который необходимо реализовать для персон. Для этого используется сторимаппинг («story mapping») – способ визуализации и планирования функционала.

Визуализация происходит на доске и начинается с высокоуровневых активностей выявленных персон. Активности разбиваются на задачи, которые в свою очередь декомпозируются на подзадачи.

Верхний слой подзадач представляет собой простейшую возможную реализацию функционала и обычно включается в первый релиз. Подзадачи, которые находятся ниже, представляют собой реализацию:

- дополнительных возможностей;
- безопасности;
- удобства использования;
- производительности.

Чем ниже мы опускаемся по подзадачам, тем меньше у них важность:

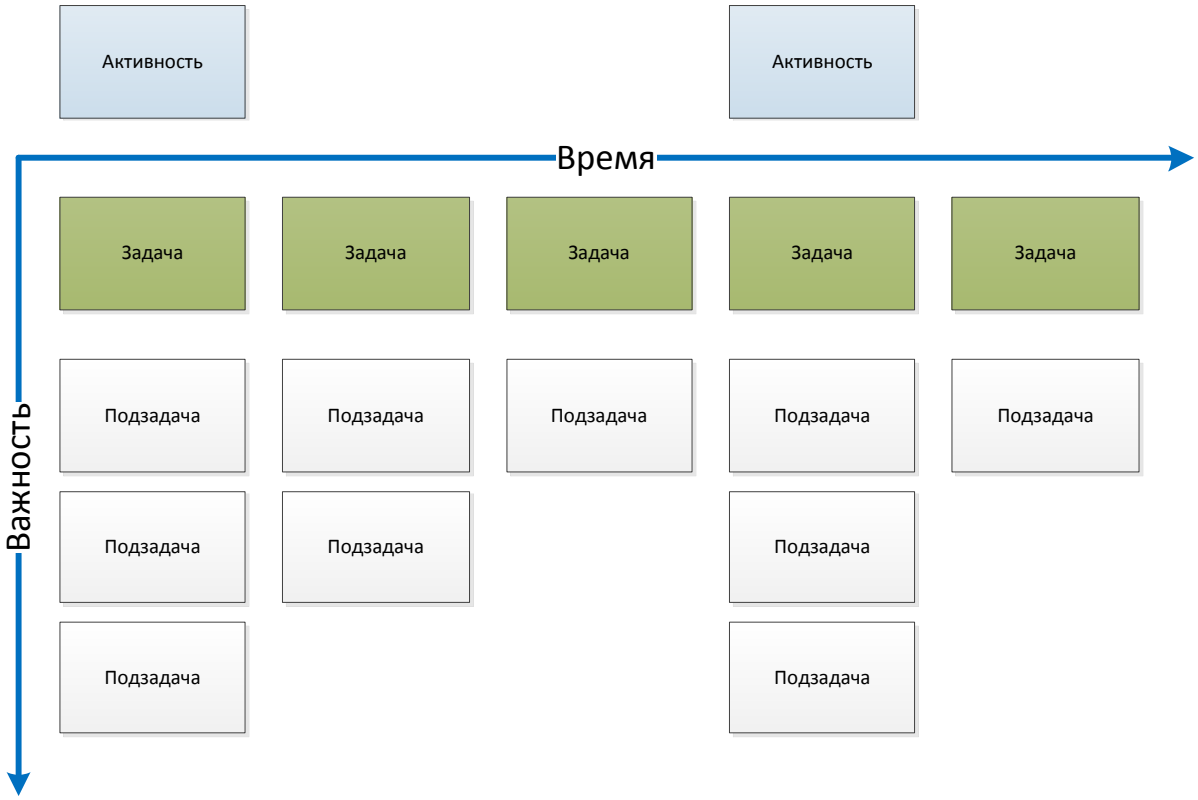


Рисунок 12. Визуализация беклога продукта при помощи сторимаппинга

Беклог продукта

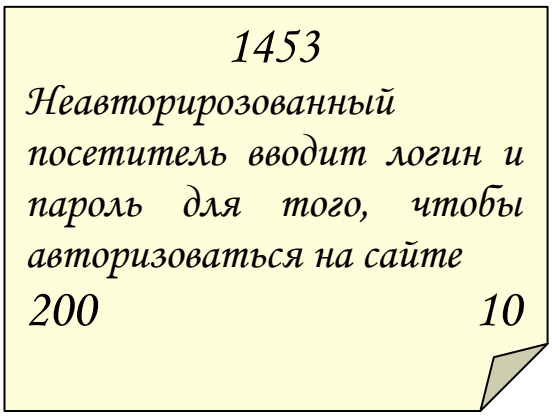
Как было сказано выше беклог продукта состоит из бизнес-требований, которые обычно оформляются в виде историй пользователей. Давайте взглянем более подробно, что представляет собой отдельная история пользователя:

- **Уникальный числовой идентификатор истории** – обычно совпадает с идентификатором истории пользователя из трекера, которым пользуется команда.

Этот идентификатор позволяет точно сказать, о какой истории пользователя в данный момент идет речь.

- **Название истории пользователя** – короткое (примерно до 10 слов) описание функционала с точки зрения пользователя, сформулированное в виде тройки «Роль», «Действие», «Цель». Например: «Пользователь вводит логин и пароль для того, чтобы авторизоваться на сайте».
- **Важность** – уникальный числовой приоритет истории пользователя, чем она выше, тем раньше данную историю необходимо сделать.
- **Оценка** – числовая относительная оценка истории пользователя по специальной шкале.

Указанные поля удобно размещать на стикере, который прикрепляется на доску. Например, историю пользователя для авторизации на сайте с оценкой в 10 сторипоинтов, важностью 200 и номером в трекере 1453, можно представить на стикере так:



Данные четыре поля являются фактически обязательными, но достаточно часто используются и дополнительные поля, которые, например, заносятся в трекер:

- **Подробное описание** – текстовое и графическое описание истории пользователя. Применяется, прежде всего, в распределенных командах для хранения знаний о функционале продукта.
- **Демонстрация** – достаточно подробный сценарий, позволяющий провести демонстрацию истории пользователя. Например, для вышеприведенной истории пользователя с авторизацией, можно использовать следующие краткие сценарии для демонстрации:
    1. Пользователь вводит логин «root» и пароль «pass», и переходит на страницу личного профиля на сайте.
    2. Пользователь вводит логин «root» и пароль «wrongpass», и получает сообщение «Введен неправильный логин или пароль».
- **Категория** – используется для повышения управляемости с помощью категоризации задач. В качестве категорий могут выступать как продуктовые категории («темы» и «эпики» в терминологии Scrum), так и категории типа «Оптимизация производительности», «Техническая история» и тому подобные.

Размер беклога и стратегическое планирование

Для сохранения управляемости необходимо поддерживать минимальный размер беклога, но для стратегического планирования, скажем, на несколько кварталов вперед, необходимо иметь достаточно длинный беклог. Используя нотацию «грозовых туч» Э. Голдратта, это противоречие можно изобразить следующим образом:

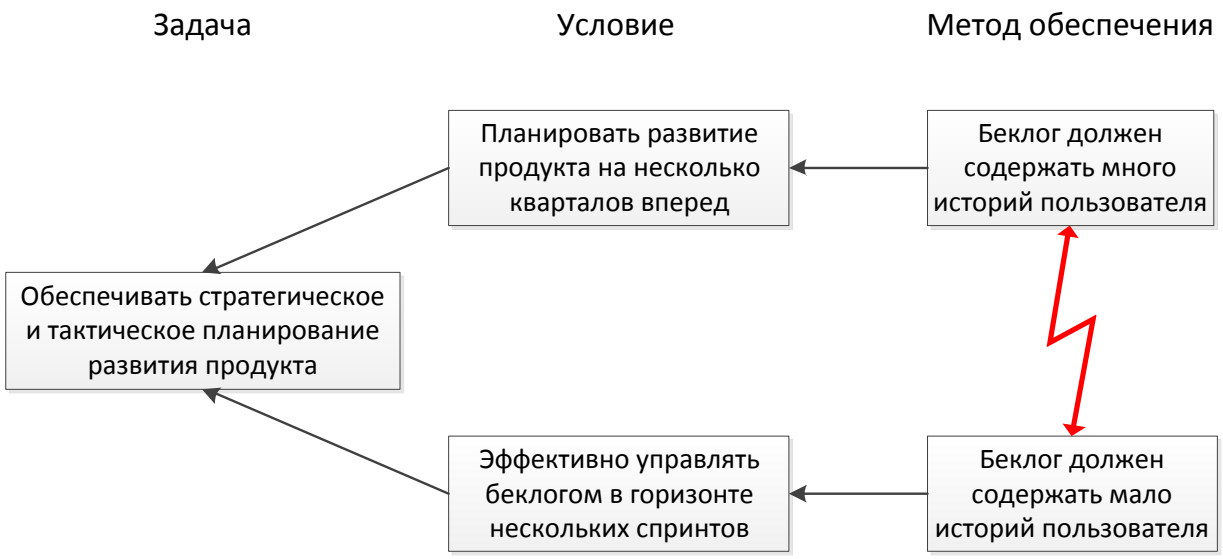


Рисунок 13. Противоречие между тактическим и стратегическим планированием

Какой бы размер беклога мы ни выбрали, у нас все равно не получится разрешить конфликт – нам нужно прорывное решение. Оно достаточно просто и лежит на поверхности: использовать метод «набегающий волны» («rolling wave planning»). В рамках Scrum такой подход означает, что мы разбиваем очень подробно истории пользователей на несколько ближайших спринтов, а остальные истории пользователей – храним в виде больших кусков функциональности, подробно не описывая их. Такие большие истории пользователей, которые в дальнейшем будут разбиты на маленькие, называются «эпиками» («epic»):

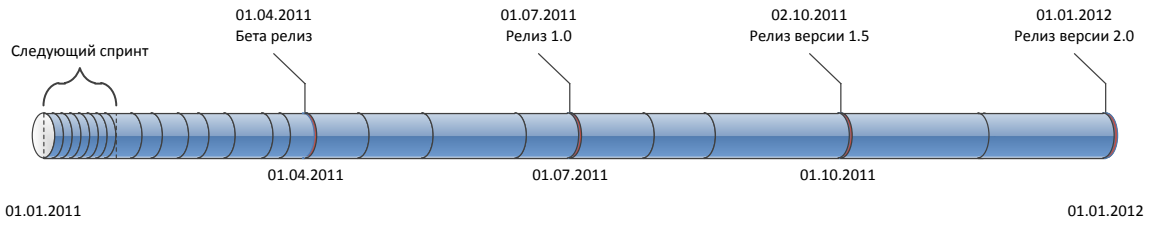


Рисунок 14. Более важные элементы беклога определены точнее

Как вы видите, чем позднее планируется реализация того или иного функционала, тем более крупные куски функционала берутся. Отмечу, что это также согласуется полностью с принципами KISS и YAGNI.

Технические истории

В любом проекте по разработке программного обеспечения есть задачи, напрямую не связанные с пользовательским функционалом. Такие задачи называются «техническими историями» или «техническими задачами». Они могут носить очень разный характер:

- Рефакторинг модуля
- Оптимизация производительности
- Исправления сложного дефекта
- Настройка инфраструктуры

Технические истории крайне желательно вносить в беклог, чтобы у владельца продукта была возможность определить их важности. Если у владельца продукта не хватает технической квалификации для этого, то необходимо ввести ограничение на количество незакрытых технических историй: как только количество задач превысит этот порог, технические истории с самым высоким приоритетом автоматически берутся в спринт.

Определение приоритетов историй пользователя

I can't get no satisfaction
I can't get no satisfaction
'Cause I try and I try and I try and I try
I can't get no, I can't get no

The Rolling Stones

Владелец продукта определяет порядок реализации функционала путем установки приоритетов, выбирая истории пользователей с наибольшей ценностью. Классическим представлением является мысль о том, что чем больше будет функционала в продукте, тем выше будет удовлетворенность конечного пользователя.

Давайте рассмотрим более точную модель – модель удовлетворения потребностей Кано. Японский профессор Нориаки Кано предложил в работе «Привлекательное качество и необходимое качество» еще в 1982 году.

Разделим всю функциональность продукта на три категории в соответствии с удовлетворенностью пользователя и полнотой функциональности продукта:

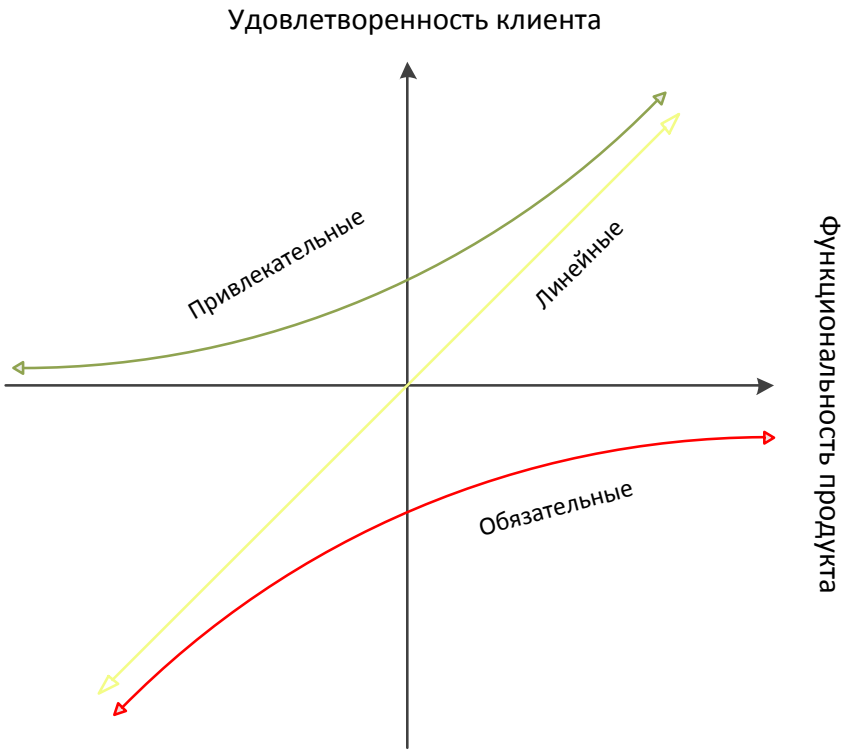


Рисунок 15. Типы функций продукта

Таким образом, можно выделить три типа функций продукта:

1. **Обязательные функции** – пользователь ждет этих функций от продукта, без них ему продукт не нужен. Например, для сотового телефона – эта возможность совершать звонки.
2. **Линейные функции** – чем больше и качественней они реализованы, тем больше доволен пользователь. Например, долгая работа сотового телефона без перезарядки.
3. **Привлекательные функции** – функции, которые придают вашему продукту «wow»-эффект. В качестве примера можно рассмотреть эргономику и юзабилити Apple IPhone.

Владелец продукта может либо воспользоваться своей интуицией и опытом для отнесения функционала к той или иной категории или собрать небольшую группу потенциальных (20-30 человек) и провести в ней опрос. Для оценки мы будем рассматривать отдельные истории пользователей либо целые эпики и задавать по каждой истории пользователю два вопроса:

1. Как вы отнесетесь к **наличию** данной функциональности в продукте?
2. Как вы отнесетесь к **отсутствию** данной функциональности в продукте?

Кроме функциональных требований (историй пользователей) можно проводить и анализ по нефункциональным требованиям и отображать их соответствующим образом в беклоге продукта.

В качестве ответов опрашиваемому пользователю предлагаются следующие варианты ответов:

- нравится
- ожидаю этого
- всё равно
- могу смириться с этим
- не нравится это

После такого опроса можно проводить анализ результатов при помощи следующей таблицы:

		Отсутствует				
		нравится	ожидаю этого	всё равно	могу смириться с этим	не нравится
Присутствует	нравится	Q	A	A	A	O
	ожидаю этого	R	I	I	I	M
	всё равно	R	I	I	I	M
	могу смириться с этим	R	I	I	I	M
	не нравится	R	R	R	R	Q

В результате функции продукта разобьются на шесть категорий:

- A (attractive) – привлекательный;
- O (one–dimensional) – линейные;
- M (must–be) – обязательные;
- R (reverse) – обратные (ненужные функции);
- Q (questionable result) – сомнительный/противоречивый результат;
- I (indifferent) – безразлично;

Первые три категории для нашего анализа и являются самыми интересными и дают нам более глубокое понимание требований по нашему продукту:

- Реализовывать требования, которые относятся к обязательным, необходимо до определенного предела, так как дальнейшие инвестиции в них не увеличат удовлетворенность пользователей;
- Вместо этого стоит сфокусироваться на линейных и привлекательных требованиях, так как они позволят значительно повысить удовлетворенность клиента.

После завершения опроса необходимо подвести итоги, просуммировав ответы пользователей:

Эпик	A	O	M	R	Q	I
Звонки в сетях стандарта GSM	1	2	**27**	0	0	0
Время работы без перезарядки	3	**23**	2	0	0	2
Эргономика и удобство использования	**16**	5	4	1	2	2
Прослушивание музыки	**10**	7	5	2	2	4
Получение и отправка MMS	1	4	2	3	8	**12**

Таким образом, можно сделать выводы к каким типам относятся те или иные функции исследуемого продукта.

Умные цели для спринта

Очень хорошей практикой является установка целей для конкретного спринта, чтобы команда могла сфокусироваться на этих целях и принимать решения исходя из них. Формулировка целей спринта обычно производиться владельцем продукта при активном участии команды, чтобы команда разделяла поставленную цель.

Владелец продукта должен ставить перед собой и командой четкие и понятные цели, для этого существует несколько критериев, которые собираются в английскую аббревиатуру SMART ("умный"):

Буква	Английский термин	Русский термин
S	Specific	Точные и конкретные
M	Measurable	Измеримые
A	Achievable	Достижимые
R	Relevant	Релевантные
T	Time bound/framed	Цели со сроком

Specific — точные и конкретные цели

Есть такой замечательный закон Мерфи, который можно в нашем случае сформулировать так: "Если есть несколько способов понять задачу, то кто-то обязательно поймет ее неправильно". Поэтому при постановке целей необходимо делать их максимально точными и конкретными, чтобы исключить различные интерпретации у постановщика и исполнителя. Хорошей практикой также будет запись целей письменно либо на бумагу, либо в электронном виде, благо сейчас имеется множество программ и веб-сервисов.

Неправильно	Правильно
Сделать верстку на сайте www.site.com кроссбраузерной	Сайт www.site.com должен одинаково отображаться в **браузерах IE6+, Opera 6+, Firefox 2+**
Сделать верстку на сайте www.site.com валидной	Сайт site.com должен полностью проходить проверки валидаторами **w3c.org (http://www.w3.org/QA/Tools)**

Точность и конкретность отнюдь не значит, что необходимо постоянно подробно расписывать каждый термин. Просто необходимо договорится об их значениях. Важно, чтобы каждый элемент беклога одинаково понимался и командой, и владельцем продукта.

Measurable — измеримые цели

Измеримость целей приносит сразу несколько бонусов. Во-первых, если имеется числовой показатель достижения цели, тогда вы точно знаете, на каком этапе работы вы находитесь и сколько вам осталось. Во-вторых, вы точно знаете, сколько вы прошли, иногда бывает полезно посмотреть назад, чтобы поднять свою мотивацию. И в-третьих, по завершении работ вы сможете посчитать, на сколько процентов ваша цель достигнута.

Неправильно	Правильно
Повысить посещаемость на сайте	Посещаемость на сайте должна составлять **2000 хостов в сутки**
Сделать так, что бы каждый посетитель покупал больше	Увеличить сумму среднего чека **на 10%**

Присутствие метрик важно не только в формулировки элементов беклога, но и в оценке размера самого элемента. Хотя последними тенденциями является отказ от числовых оценок, без них планирование не получается достаточно точным.

Achievable — достижимые цели

У каждого человека есть набор знаний и навыков, поэтому для различных людей необходимо подбирать соответствующие задания. В соответствии с навыками и знаниями человека, задачи можно категоризировать:

- **Недостижимые:** при постановке таких задач, заранее понятно, что исполнитель с ними заведомо не справится. Например, нельзя за день нарисовать 1000 качественных макетов для разных сайтов. Такие задачи перед подчиненными ставить нельзя.
- **Труднодостижимые:** для выполнения таких задач исполнителю будет необходимо использовать все свои знания и умения. Например, нарисовать 10 вариантов макета дизайна для сайта за день. Такие задачи команда должна решать, но не постоянно — иначе она просто перегорит. Поручать такие задачи следует, прежде всего, опытным и амбициозным сотрудникам, для которых они будут своеобразным вызовом. Чем труднее будет сделать поставленную задачу, чем значительнее будет чувство достигнутого.
- **Достижимые:** эти задачи точно соответствуют уровню знаний и навыков исполнителя. Например, нарисовать дизайн макета сайта по утвержденному наброску и брифу за один день. Такие задачи необходимы для передышки между более сложными задачами и для выработки уверенности в своих силах.
- **Легко достижимые:** эти задачи не соответствуют компетенции сотрудника, и ему будет не очень интересно их выполнять. Например, нарисовать кнопочку для формы в заданном стиле за день. Такие задачи, желательно поручать только новым сотрудникам для интеграции в команду.

Общий вывод можно сделать такой: необходимо чередовать достижимые и труднодостижимые задачи. При использовании гибких методологий, когда элементы беклога оценивает команда, такое чередование получается естественным образом: команда иногда оценивает в большую сторону, иногда в меньшую.

Relevant — релевантные цели

Релевантность целей нужно рассматривать с двух сторон: релевантность для исполнителя и для компании. Релевантность (значимость) для исполнителя тесно связана с его мотивацией. Например, сотруднику, который любит изучать новые технологии, можно и нужно поручить исследовательский проект, а не рутинную работу. Когда команда понимает, что цель важна, усилия, которые она направит на ее достижение, будут заведомо больше усилий, направленных на «неважную» цель. Отмечу также, что букву R (другие буквы этой аббревиатуры) расшифровывают по-разному.

Time-bound — цели со сроком

При обсуждении сроков для выполнения задач или достижения целей, нельзя не вспомнить еще один эмпирический закон — закон Паркинсона: "Любая работа увеличивается в объёме, чтобы заполнить всё отпущенное на неё время". По личному опыту скажу, что когда у задачи нет срока, она вытесняет срочными задачами и шанс того, что до нее когда-нибудь дойдут руки, падает. Поэтому при постановке любой задачи необходимо устанавливать срок исполнения.

Срочность целей достаточно тесно связана с достижимостью, чем меньше срок на задачу, тем она более труднодостижимая, что надо учитывать при постановке задач.

Неправильно	**Правильно**
Сделать сайт	Сделать сайт к 26.10.09
Сделать на сайте раздел "Контакты" для демонстрации клиенту до завтра	Сделать на сайте раздел "Контакты" для демонстрации клиенту до 21.10.09 к 12.00

Концепция ограничения по срокам встроена в Scrum: итерации всегда фиксированного размера и команда четко знает, когда будет проводиться демонстрация результатов спринта.

Scrum в заказной разработке

В этой главе я расскажу об особенностях Scrum в заказной разработке, которая отличается от «тепличных» условий внутренней разработки и можно ли применять Scrum в таких условиях и на что надо обратить внимание, прежде всего.

Как продать Scrum заказчику?

Первый вопрос, который обычно возникает «Как продать Scrum заказчику?», ведь его не очень волнует, каким иностранным словом вы называете свои внутренние процессы. На данном этапе очень важно объяснить заказчику (полагаем, что спринты у нас двухнедельные), что он сможет:

- каждые две недели получать новый функционал, который можно попробовать и выпустить на рынок для заработка денег;
- каждые две недели менять требования, чтобы изменения в бизнесе отражались и в ПО;
- регулировать сроки и стоимость работ по проектам за счет возможности остановить проект после любого спринта.

Второй вопрос, который появляется сразу после первого: «Как отразить процесс в контракте?». Самым лучшим вариантом здесь будет заключение контракта типа «Время и материалы» (Time and Material, T&M), при котором заказчик будет оплачивать вам стоимость команды плюс оговоренный процент. Преимуществом такого вида контракта является управление по срокам и стоимости со стороны заказчика, но надо заметить, что при таком подходе и риски перекладываются на него. Еще отмечу, что при таком подходе резко сокращается этап анализа проекта, так как нет необходимости давать точную оценку сроков и гарантировать ее.

Традиционным для нашей страны является подход по заключению контрактов с фиксированной стоимостью (Fixed price), который, казалось бы, переносит многие риски на исполнителя. В действительности исполнитель старается все эти риски заложить в стоимость контракта. Этот подход трудно назвать гибким, так как обе стороны пытаются победить друг друга, вместо того, чтобы искать решение Win-Win, основанное на взаимном доверии.

Нулевой спринт

О нулевом спринте обычно многие молчат или пишут совсем немного. Буквально год или два назад наконец-то начали активно обсуждать эту фазу проекта, причем на данный момент в основном обсуждается продуктовая часть.

Самым главным документом в проекте, с которого обязательно стоит начать его разработку, является видение (vision) проекта. Этот краткий документ описывает, что представляет собой продукт, кто, когда, как и где будут его аудиторией, основные фазы проекта, бизнес-модель для монетизации и так далее. Словом, заменяет целый пакет документов, который принято использовать в более тяжеловесных методологиях. Для

создания видения можно использовать инновационные игры, разного рода мозговые штурмы и фреймворки.

Следующим этапом идет выявление персон и их активностей (вплоть до отдельных юзер-стори), которые фактически являются легковесным и более визуальным вариантом экторов и юзкейсов. Уже традиционно данную активность Scrum-команды реализуют в виде сторимаппинга, результатом которого становятся доски и целые стены с визуализацией активностей в проекте:

Рисунок 16. Сторимаппинг

Обязательно надо вытащить представителей заказчика на процессы по выработке видения и сторимаппинг (а на последнее собрание еще потенциальных пользователей).

Обратная сторона монеты – это выбор платформы для разработки и создания архитектуры. Конечно, я имею в виду не Big Upfront Design, а более гибкий подход, ведь десятки или сотни диаграмм, пылящихся в дальнем ящике стола – это не наша цель. Наша задача сделать продукт, минимизировав потери.

В самом простом случае можно ограничиться диаграммой предметной области с максимально простым синтаксисом, который будет понятен и заказчику. В более сложных случаях эту диаграмму стоит дополнить до диаграммы классов и набрать в корзину

архитектуры еще несколько UML-диаграмм, которые помогут описать динамическую часть вашей системы.

Можно взять ICONIX в качестве подмножества UML и процесса, но он будет скорее замещать сторимаппинг, чем дополнять его.

В нулевой спринт входит и подготовка к первому спринту. Для этого необходимо описать истории пользователей, спроектировать для них интерфейс и оценить. Работу лучше всего построить в таком порядке, так как он способствует более качественному пониманию требований и их оценке.

Практики Scrum или как посадить заказчика на итеративную иглу

Не буду подробно останавливаться на традиционных практиках Scrum’а, хочу отметить лишь, что в них надо по возможности вовлекать заказчика. Программа-минимум – это посещение заказчиком всех демонстраций: вам жизненно необходимо «подсадить» заказчика на свой ритм работы, чтобы он хотел получать очередной инкремент функционала, как наркотик. Это сильно поможет выстроить атмосферу доверия между вами.

Программа-максимум - это присутствие представителя заказчика и на других мероприятиях. Например, посещение покер-планирования, поможет заказчику лучше осознать размеры ваших задач и понимать, что команда работает с высокой скоростью. Участие заказчика в ретроспективах позволит глубже погрузиться в проблемы и риски команды, при этом он сможет принять активное участие в совершенствовании процессов.

5.Управление командой

> Если хочешь построить корабль, то не собирай людей, чтобы они принесли лес, и не распределяй задания, а лучше пробуди в них тоску по бескрайней дали моря.
>
> Антуан де Сент-Экзюпери

Гибкие методологии опираются на людей и взаимодействия между ними, поэтому грамотное управление людьми выходит на первый план. В этой главе мы посмотрим, как мотивировать команду, как оценивать и как развивать ее в сторону гибкости.

Что такое команда?

Определений «команды» существует несколько десятков. Будем придерживаться следующего определения, которое дал Майкл Армстронг:

Команда — это небольшая группа людей, взаимодополняющих и взаимозаменяющих друг друга, которые собраны для совместного решения задач производительности и в соответствии с подходами, посредством которых они поддерживают взаимную ответственность.

Этапы командообразования

Рисунок 17. Этапы командообразования

В своем развитии команды проходят несколько этапов, которые сменяют друг друга:

1. **Формирование**

На этапе формирования происходит создание команды и постановка целей, распределение и закрепление ролей (в том числе и социальных). Отдельные члены команды еще не очень понимают цель и задачи, которые перед ними поставлены:

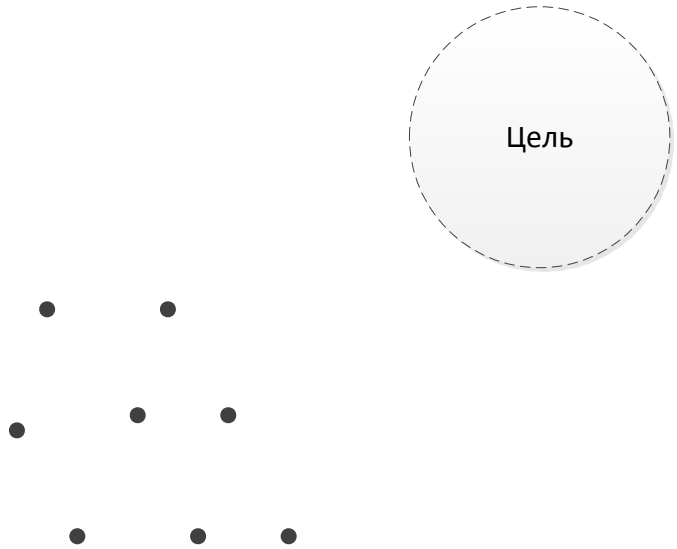


Рисунок 18. Отсутствие направленности к цели у членов команды

2. **Бурление**

На этапе «бурления» участники осознают свои цели и определяют вектор движения. Обратите внимание, что эти векторы разнонаправленные между собой и с направлением, которое необходимо для достижения цели:

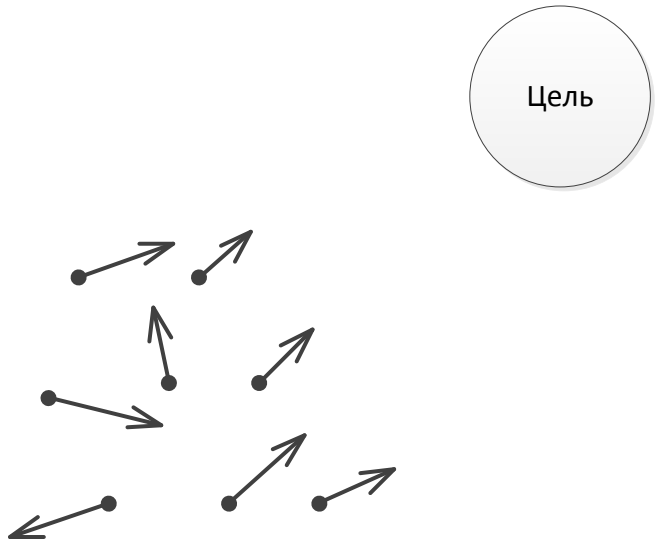


Рисунок 19. Разнонаправленность членов команды

Цель также стала более четкой и понимаемой для команды.

На данном этапе особенно часто возможны конфликты и противостояние между членами команды, поэтому особенно возрастает роль скрам-мастера как модератора.

3. Нормализация

Следующим этапом идет нормализация, когда члены команды притираются к друг другу и начинают двигаться сонаправлено:

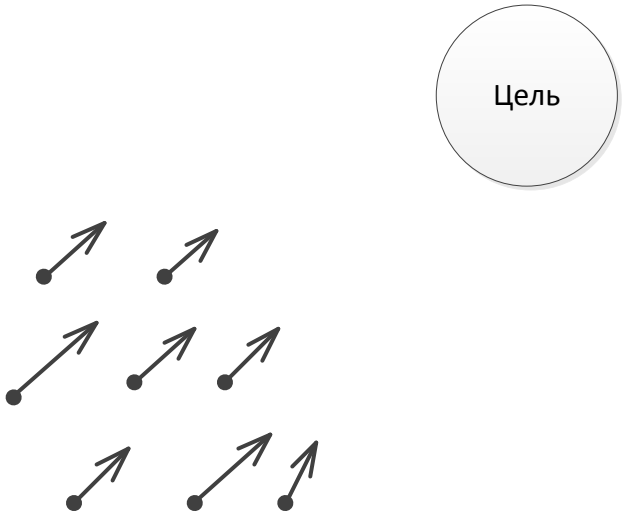


Рисунок 20. Сонаправленность членов команды

Основной задачей скрам-мастера является фасилитация для перехода на следующий этап.

4. Функционирование

На этапе функционирования команда становится самоуправляемой и способной оптимизировать свою производительность, поэтому векторы, направленные к цели, удлиняются:

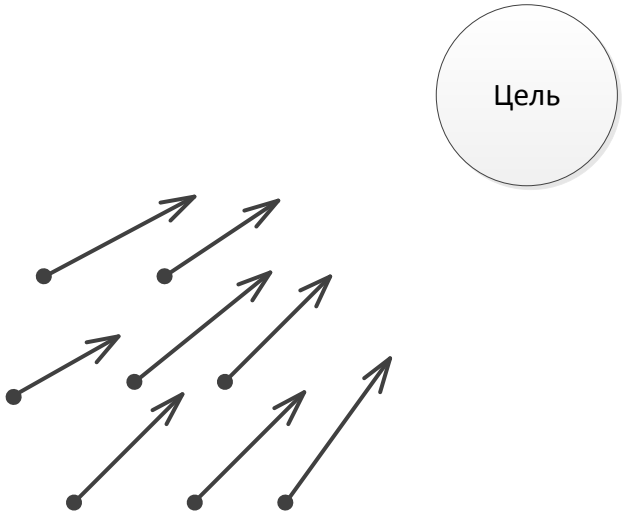


Рисунок 21. Сильная сонаправленность членов команды

5. Расформирование

Когда цели, поставленные перед командой, достигнуты, наступает этап расформирования, и направление «движения» участников снова рассинхронизируется:

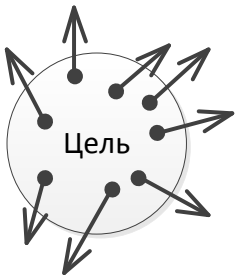


Рисунок 22. Рассинхронизация членов команды на этапе расформирования

Командообразование в Scrum

Как было описано выше, чтобы работать эффективно, команда должна находиться на этапе «Функционирование». Соответственно главной задачей команды (и в частности скрам-мастера) является максимально быстрый переход между этапами:

Этап	Быстрый переход	Средний переход	Долгий переход
Формирование	0-ой спринт	2-ый спринт	2-ой спринт
Бурление	1-ый спринт	4-ой спринт	6-ый спринт
Нормализация	1-ый спринт	6-ий спринт	10-ой спринт
Функционирование	2-ой спринт	8-ый спринт	16-ой спринт
Расформирование	Завершение проекта		

Спринты предполагаются двухнедельными. Самый лучший вариант в этой таблице не описан: можно просто взять сыгранную команду. Но даже при формировании новой команды можно добиться эффективной работы уже ко второму спринту. И наоборот, можно не достичь нужной производительности и до завершения проекта.

> Хотя команды являются самоуправляемыми, они не становятся бесконтрольными. Руководство устанавливаем контрольные точки, чтобы избежать нестабильности и хаоса. В то же время руководство избегает жесткого микроконтроля, который убивает креативность
>
> Такеучи Х. и Нонака И., «Разработка нового продукта. Новые правила игры»

Самоорганизация в командах

Давайте сначала четко определимся с тем, что не является самоорганизующейся командой и что не входит в ее полномочия.

Во-первых, команда не может сама себе ставить цели, они всегда приходят извне. Как правило, задачи спускаются «сверху» от руководства или от отдела продаж.

Во-вторых, команда не определяет свой состав сама, она опять же формируется сверху. Но могу сказать, что на определенном этапе взросления, когда команда может объективно оценивать вклад участников, даже формирование состава команды можно отдать в качестве самостоятельного решения.

А что же может команда, если цель ей ставится сверху? Она может выбрать путь, которым эта цель будет достигнута максимально эффективным способом. «Эффективность» в данном случае может быть измерена сроком, составом и ценой работ. При этом команде приходиться адаптироваться к тем ограничениям, которые заданы в проекте, и к условиям окружающей среды, потому что, как правило, механизмов влияния на них у команды нет.

Децентрализация и умение адаптироваться свойственно не только командам разработчиков. Посмотрите на колонию муравьев: как они строят муравейник, как они выполняют повседневную работу. Такая же параллельная работа ведется и в гибких командах: контроль и «власть» децентрализованы и распределены между членами команды. Соответственно решения, из которых складывается конечный результат, принимаются каждым членом команды, разделяя ответственность, но не размывая ответственность между всеми.

Модель CDE

Модель CDE (Container/Differences/Exchanges) представляет собой простой фреймворк для развития самоуправляемых команд. Данная модель подробно описана в презентации Mike Cohn «Leading a Self-Organizing Team» (Cohn, 2011)

Вы можете влиять на самоорганизующиеся команды, изменяя:

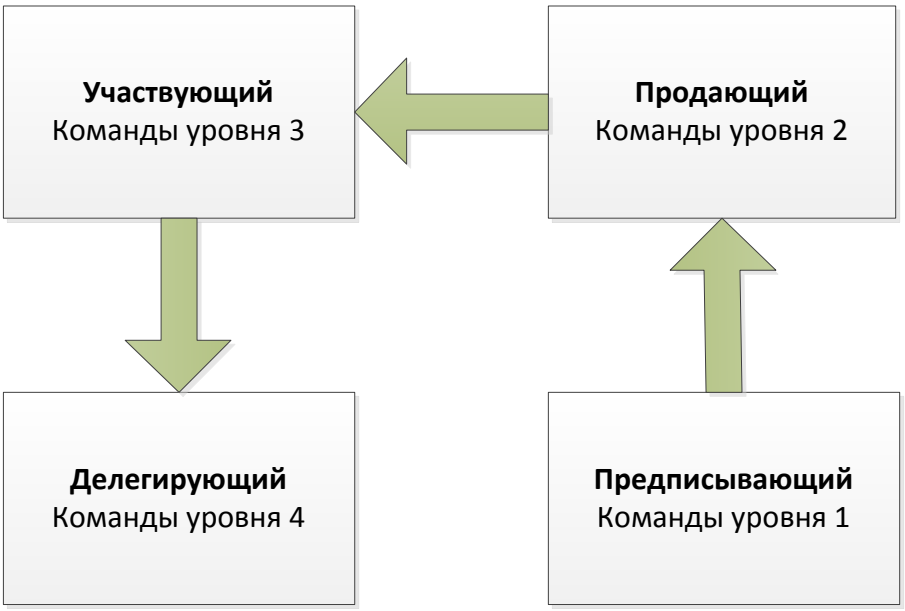
- **Контейнеры**: формальные команды, неформальные команды и их ожидания
- **Различия**: увеличивая или уменьшая различия внутри или между контейнерами
- **Обмен**: добавляя новых людей, инструменты и техники

Контейнеры	Различия	Обмен
• Увеличьте или сократите команду • Увеличьте или сократите ответственность • Измените границы команды • Создавайте новые команды или группы	• Не требуйте консенсуса: креативность появляется в различиях • Сильные споры не являются конструктивным инструментом • Задавайте сложные вопросы и требуйте от команд решений	• Поощряйте коммуникации между командами и группами • Добавляйте или удаляйте людей из коммуникаций • Поощряйте обучение

Вариация, выбор и закрепление

Эволюция в живой природе состоит из трех этапов: вариация, выбор и закрепление. Те же этапы можно выделить и в эволюции команды и ее процессов.

Возможность	Желание	Уровень готовности
Нет возможности	Нет желания	1
Нет возможности	Есть желание	2
Есть возможность	Нет желания	3
Есть возможность	Есть желание	4



Уровни команд соответствуют этапам командообразования из предыдущего раздела.

Команды уровня 1

Командам уровня 1 требуется предписывающий стиль лидерства. Лидер должен точно описывать, что команде необходимо сделать. Таким командам требуется выработать уверенность, и они не могут на данном этапе стать настоящими Agile-командами.
Такие команды еще не могут пробежать марафон, поэтому неплохо реализовывать небольшие и несложные проекты. Такой подход позволяет получать положительные результаты достаточно часто, что позволяет выработать уверенность в своих силах. Для этого необходимо выбрать итерации достаточно маленького размера (не более 2 недель) и помогать команде изо дня в день.
Командам уровня 1 требуется указывать, какие улучшения необходимо делать. Как правило, начинать стоит с базовых инженерных практик, например, непрерывной интеграции и модульного тестирования.

Команды уровня 2

Команды уровня 2 не могут стать гибкими, но хотят это сделать. Этим командам требуется «продающий» стиль лидерства. Команде необходимо ставить высокоуровневые цели и задавать направление движения. Основной целью является повышение командных качеств.
Такие команды могут стать гибкими, но им требуется сильный тренер или скрам-мастер, который научит команду вырабатывать и полагаться на свои решения. Конечно, часть решений команды будет ошибочными. В этом нет ничего страшного, ведь именно на ошибках и учатся. Особое внимание нужно уделить ретроспективам, как основному методу выработки улучшений в Scrum.

Команды уровня 3

Команды уровня 3 могут стать гибкими, но не хотят этого. Такой команде требуется «участвующий» стиль лидерства. Команде требуется меньше указывать цели, потому что команде необходимо опираться полностью на свои решения. Команда уже фактически является гибкой, до этого состояния всего один шаг.
Такие команды часто сосредоточены на тактических задачах, поэтому тренер или скрам-мастер должны максимально внимательно относиться к стратегическим задачам и долгосрочному планированию.

Команды уровня 4

Команды уровня 4 хотят и могут стать (и часто являются) гибкими, и им необходим «делегирующий» стиль лидерства.
Лидер не должен помогать принимать решения команде, но может помогать в выборе методов поиска решений. Скрам-мастер должен быть сосредоточен на максимизации производительности больше, чем на соблюдении сроков.

Лучшие практики управления командой в Scrum

Давайте посмотрим, какие лучшие практики и инструменты рекомендуется использовать команде и скрам-мастеру для наиболее эффективного создания продукта. Эти практики отлично интегрируется в Scrum и многими специалистами уже считаются его неотъемлемой частью.

Покер-планирование

Рисунок 23. Карты для покер-планирования от компании ScrumTrek

Покер-планирование (Planning poker) – консенсусная относительная оценка историй пользователей командой. Этот вид оценки не входит в классический Scrum, но является паттерном для оценки историй пользователей.

Покер-планирование проводится следующим образом:

1. Каждому участнику раздается колода карт с числовыми весами для оценки требований.
2. Начинается обсуждение и оценка очередной истории пользователя: она зачитывается, команда задает вопросы владельцу продукта, выясняет детали, если это необходимо.
3. Каждый член команды делает свою оценку, кладя карту рубашкой вверх.
4. После того, как все члены команды сделали оценку – все карты переворачиваются и оценки сверяются.
5. Если оценки всех участников одинаковы – консенсусная оценка заносится в беклог, в противном случае начинается повторное обсуждение и проводится второй раунд.

Для подготовки к покер-планированию обычно разбивают истории пользователей на отдельные задачи, такой подход позволяет сделать оценку более точной. Но стоит понимать, что для относительных оценок путем сравнения историй пользователей между собой данная практика не является обязательной.

Очень важно понимать, что оценка должна быть именно консенсусной, а не оценкой большинства. Сюда же можно отнести и давление со стороны более авторитетных членов команды.

Выбор эталонной задачи

Для относительных оценок необходимо выбрать эталонную историю пользователей. Она должна быть:

- простая и понятная для команды;
- типичная для данного проекта;
- небольшого размера.

Эту историю пользователя команда принимает за 1 сторипоинт. Тогда история пользователя, которая будет в два раза меньше эталонной, будет иметь размер 1/2 сторипоинт, а история пользователя, которая в пять раз больше эталонной, будет иметь размер 5 сторипоинтов. Таким образом, сторипоинт – это относительная безразмерная единица измерения.

> Что оценка в «пунктах» (“попугаях”, story points) – это сравнительная оценка, которая показывает «размер» требований относительно друг друга. Т.е. важны относительные значения и не может быть эталона. «Пункты» не имеют физических единиц измерения.
>
> Евграшин Тимофей, тренер

Для оценки используется дискретная логарифмическая шкала:

0	0,5	1	2	3	5	8	13	20	40	100

Причем оценки в 40 и 100 сторипоинтов используются для эпиков и считаются не точными. Такая шкала относит к одному размеру истории пользователя в 20 и 21 сторипоинт, но к разным размерам – задачи в 2 и 3 сторипоинта.

В итоге все наши истории пользователей в беклоге будут «разложены» по следующим категориям:

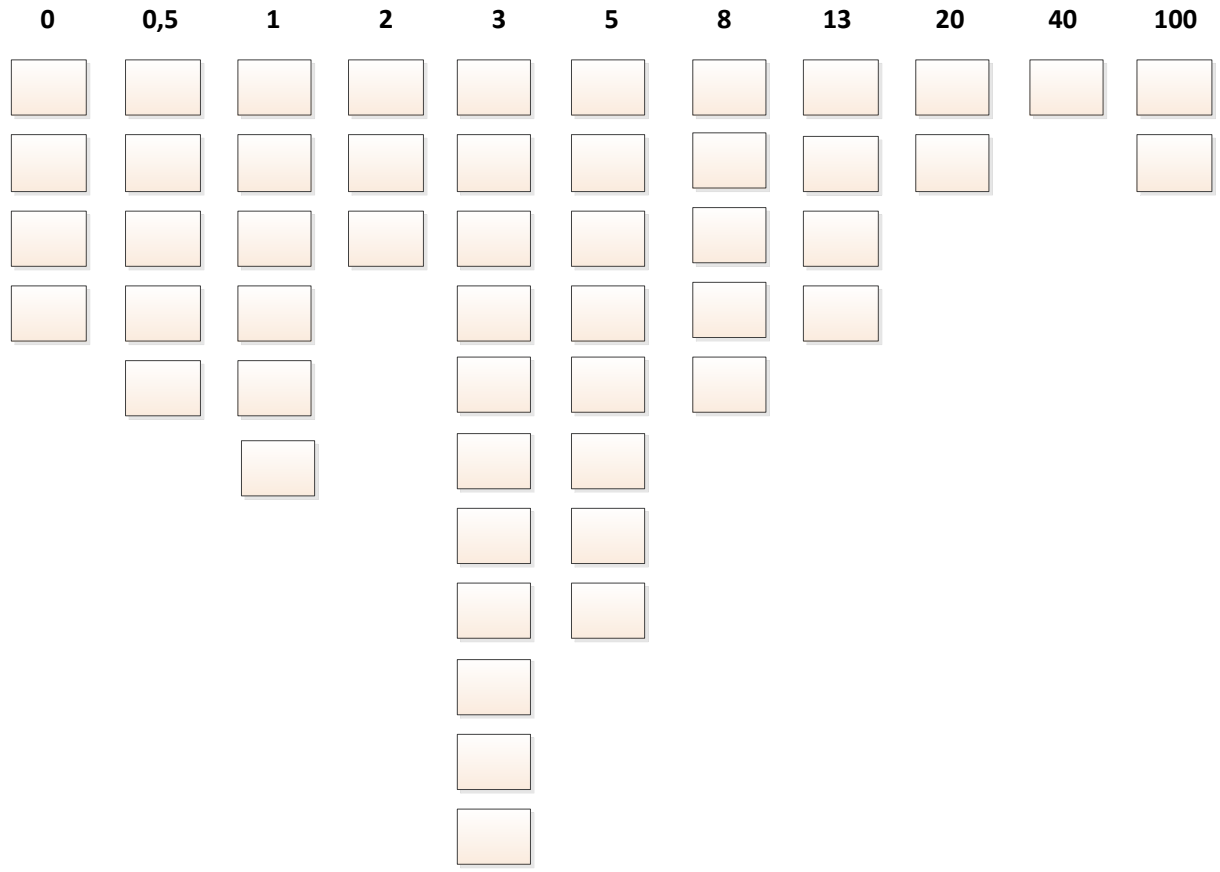
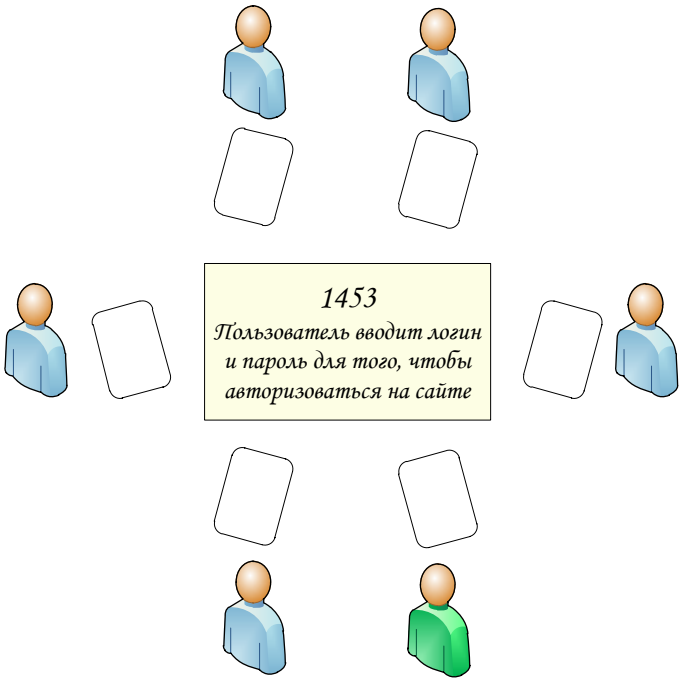


Рисунок 24. Шкала размеров историй пользователей

Получается, что с каждой новой оцененной историей пользователей у нас появляется новые «эталоны» для сравнения.

Ход покер-планирования

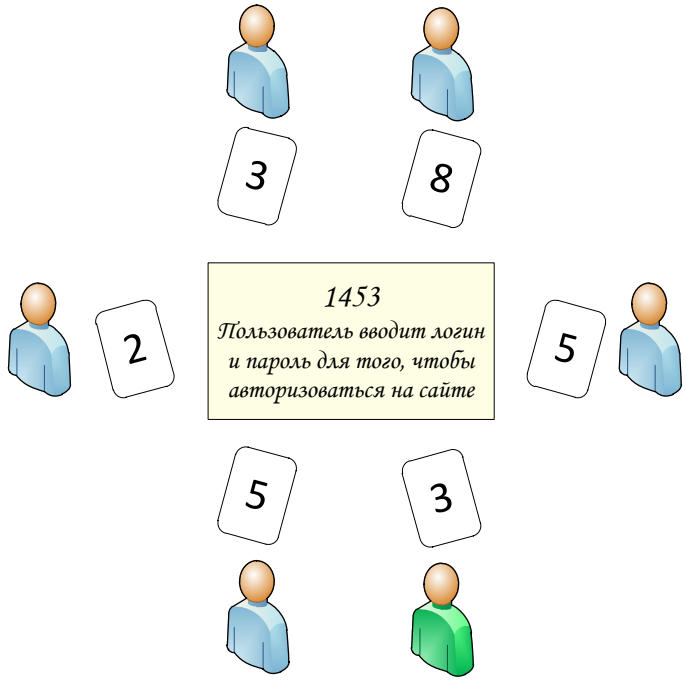
После обсуждения истории пользователей начинается первая раздача, при которой все участники команды выкладывают карты рубашкой вверх (на данной схеме скрам-мастер отмечен зеленым):



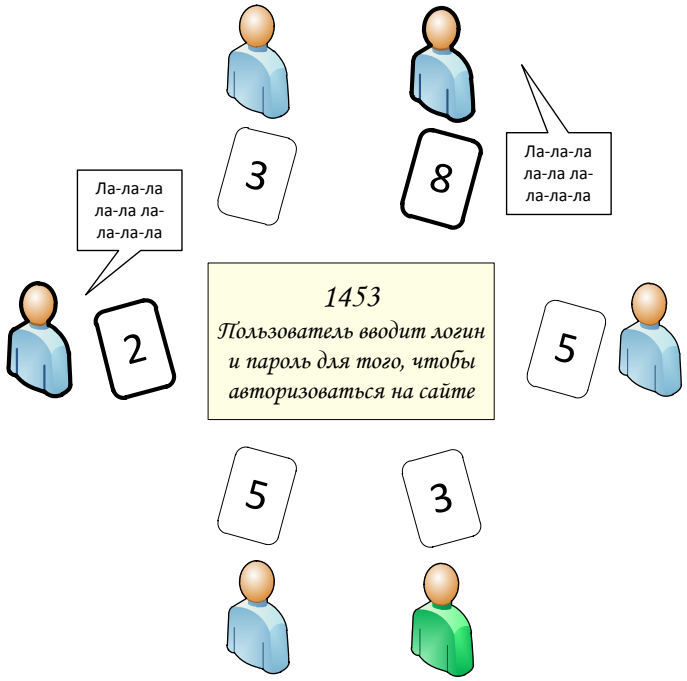
Важно, чтобы при обсуждении историй пользователя не было давления, которому все начинают невольно подчиняться и ставить оценки, уже не вдумываясь в суть.

Колосова Ксения, владелец продукта

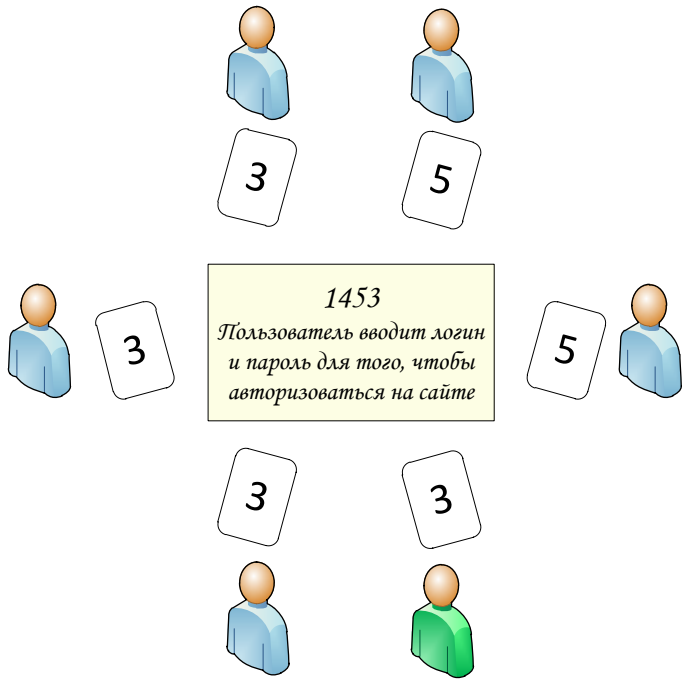
После этого все карты одновременно переворачиваются:



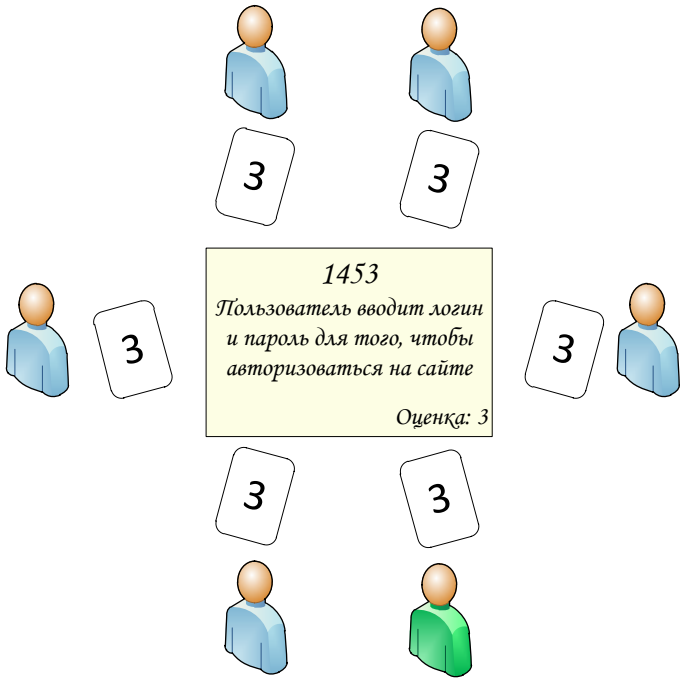
Затем скрам-мастер организует обсуждение: в качестве паттерна можно порекомендовать начинать с участников команды, которые выкинули максимальную и минимальную оценку:



Скрам-мастер должен следить за временем, которое длится обсуждение истории пользователей. По истечении времени происходит второй раунд и дальнейшее обсуждение:



Обычно разброс оценок снижается с каждым последующим раундом, и сходится к консенсусной оценке:



> В процессе обсуждения сглаживаются разногласия между владельцем продукта и командой, владелец продукта начинает понимать процесс работы над задачей, а команда точнее и детальнее понимает суть задачи.
>
> Колосова Ксения, владелец продукта

В конце оценка записывается на стикер с историей пользователей и при необходимости заносится в трекер.

Отбор задач на спринт

Команда отбирает задачи на спринт в соответствии со своей скоростью и приоритетами, установленными владельцем продукта. Скорость прогнозируется на основе эмпирических данных за прошлые спринты с учетом реальных обстоятельств, например, болезни или отпуска сотрудника:

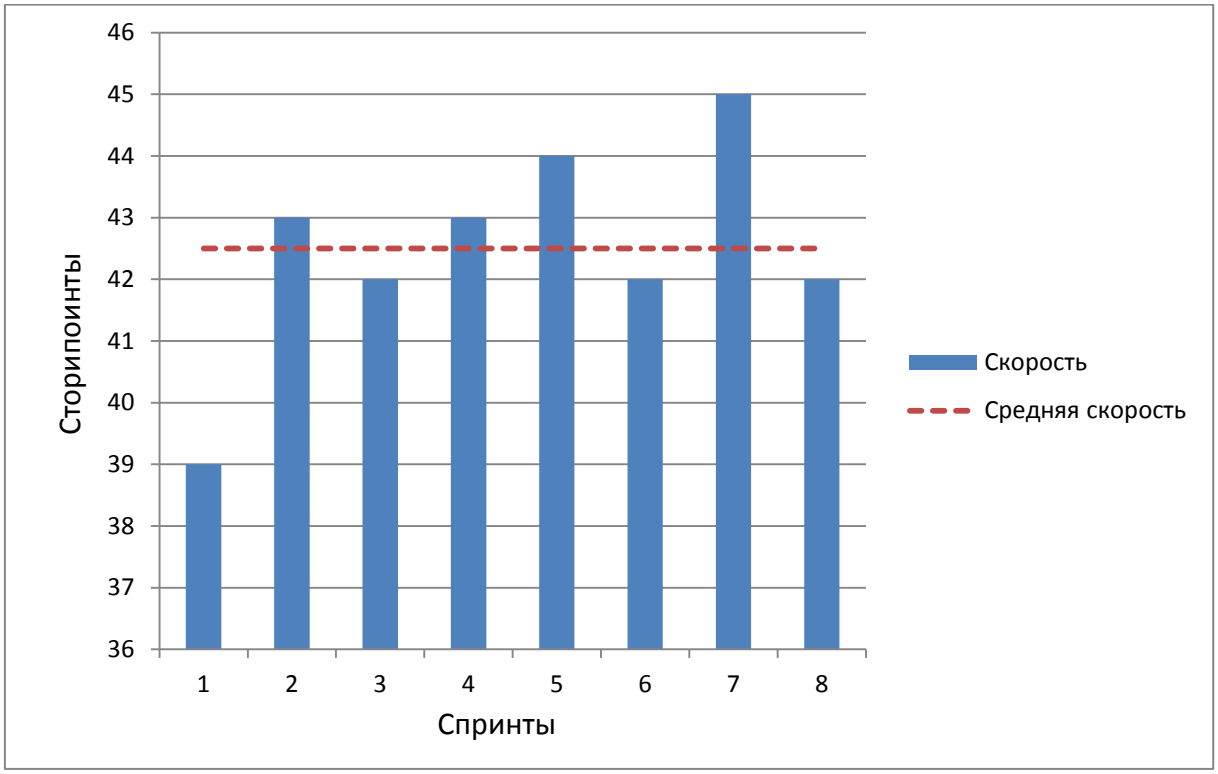


Рисунок 25. Скорость команды за 8 последних спринтов

На схеме изображенной ниже в спринт отбираются истории пользователей A, B, C, D, F.

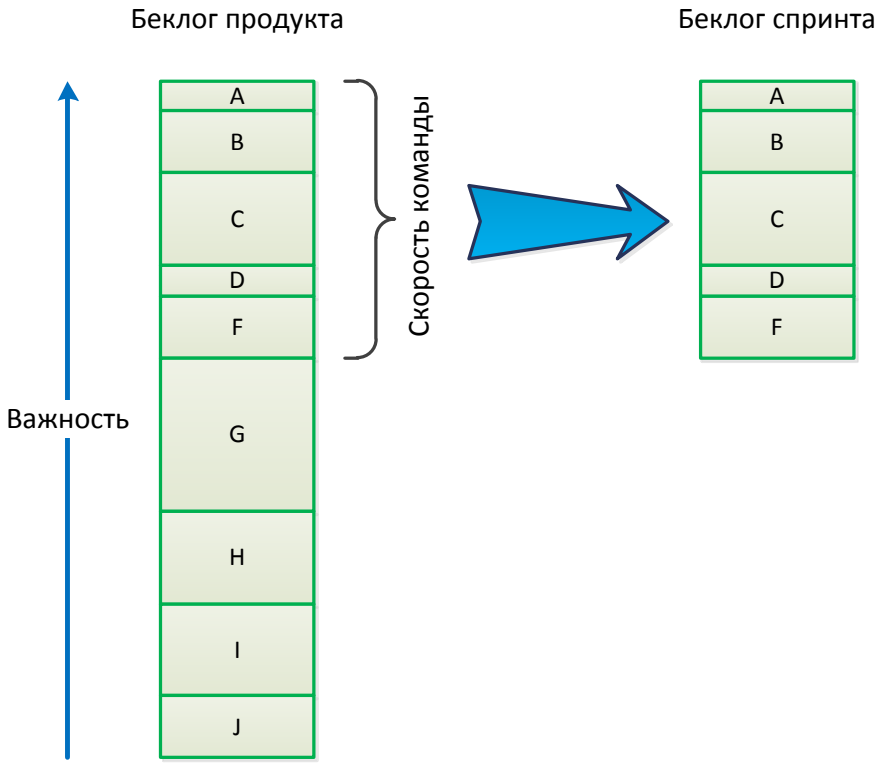


Рисунок 26. Отбор элементов беклога продукта в беклог спринта

Диаграмма сгорания

Для мониторинга прогресса в Scrum используется специальный график – диаграмма сгорания (burndown diagram). По горизонтальной оси на таком графике откладываются дни спринта, а по вертикальной количество оставшихся сторипоинтов и/или количество закрытых историй пользователей. Дополнительно строится идеальная диаграмма сгорания, которая показывает запланированный ход работ:

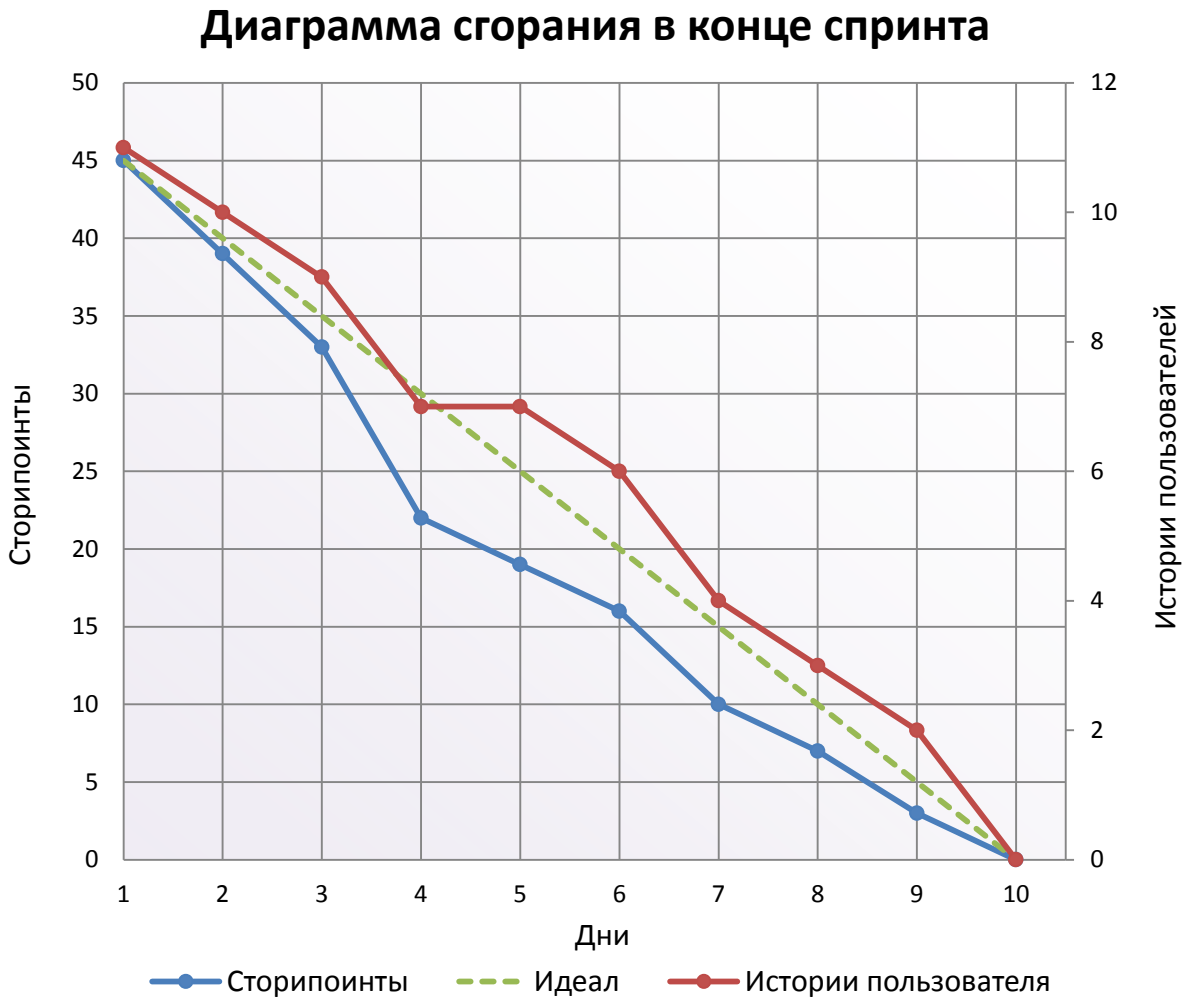


Рисунок 27. Диаграмма сгорания показывает, что спринт завершился в соответствии с планом

В дальнейшем анализ будем проводить по количеству оставшихся сторипоинтов, но все сказанное может распространяться и на истории пользователя.

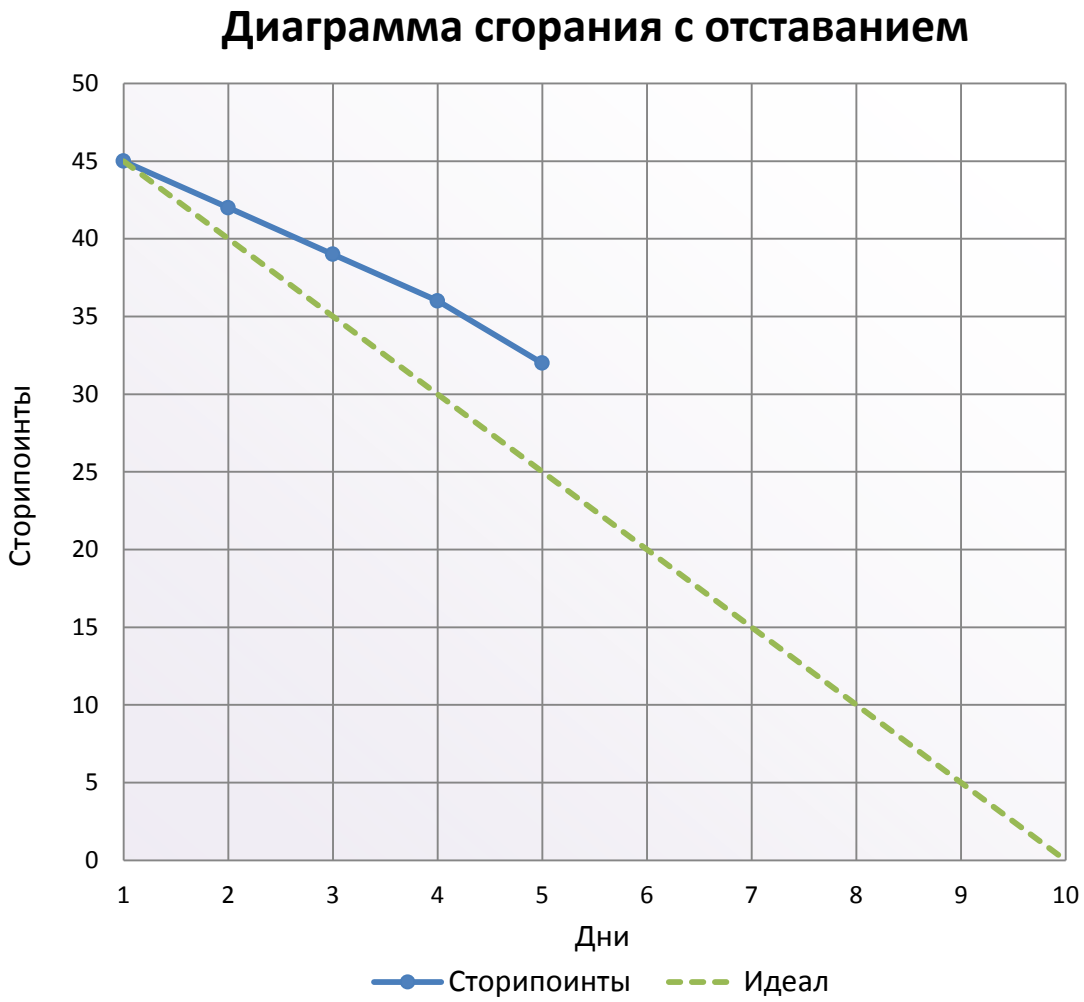
Анализ производится путем сравнения реального графика с идеальным:

- если реальный график выше идеального, значит, команда отстает от плана;
- если реальный график ниже идеального, значит, команда опережает план.

Анализ диаграммы сгорания во время спринта

Для единообразия анализ будем проводить на 5-ый день спринта, хотя на практике диаграмму сгорания можно использовать с 3-его дня для выработки корректирующих действий.

Давайте рассмотрим самую стандартную ситуацию отставанием от графика:

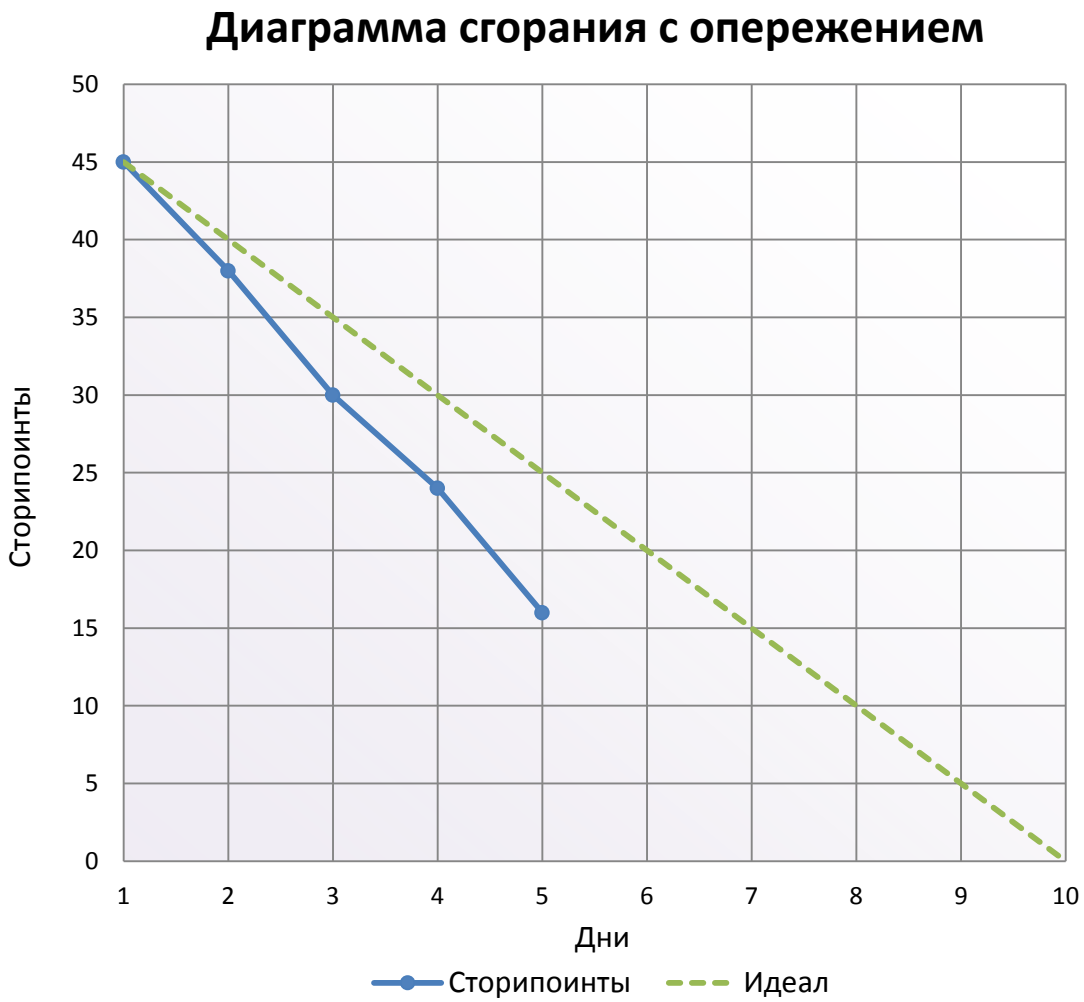


Команда должна на очередном скрам-митинге обсудить, почему сторипоинты «сгорают» так медленно и выработать ряд контрмер. Приведем самые распространенные причины:

- Сильная ошибка в планировании;
- Болезнь или иная причина отсутствия одного или нескольких членов команды;
- Недооценка и реализация рисков (обычно технологических).

Об отставании необходимо максимально оперативно сообщить владельцу продукта, если он не ведет ежедневный мониторинг диаграммы сгорания. Если команда сама не сможет выработать контрмеры, необходимо, чтобы владелец продукта убрал из спринта одну или несколько историй пользователя с минимальным приоритетом.

Обратной ситуацией является опережение плана:



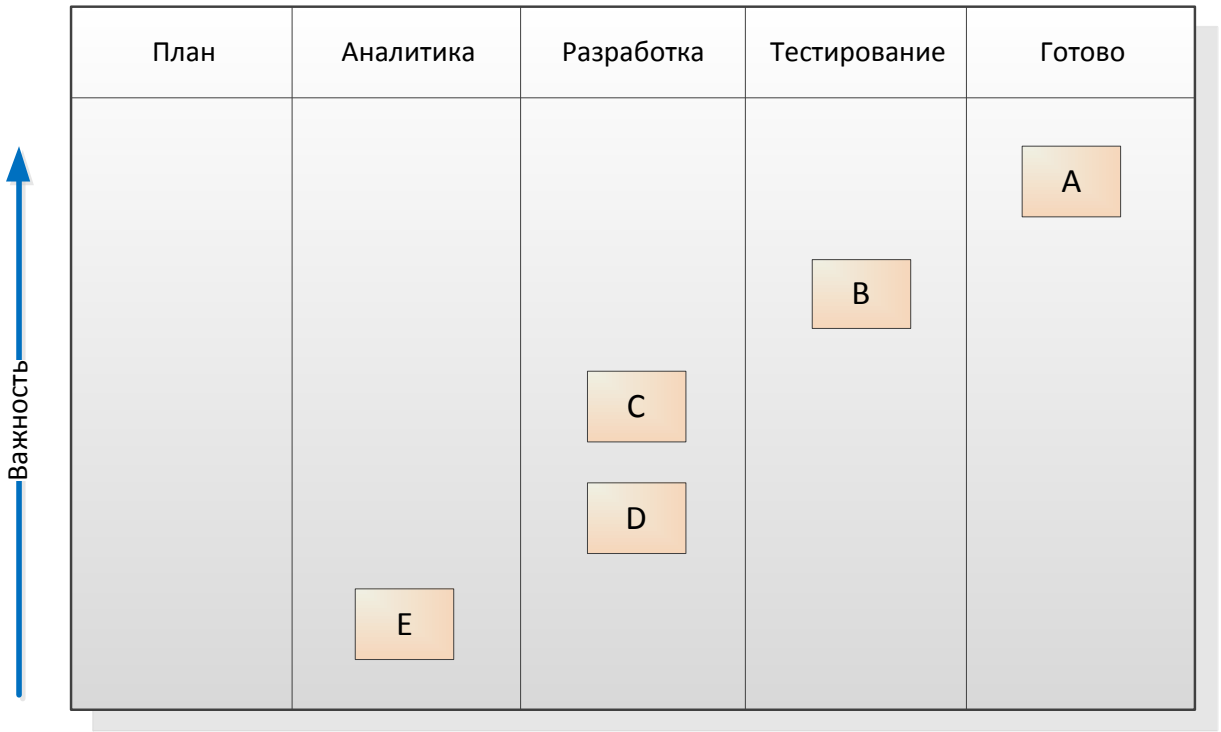
В таком случае команде необходимо в беклог спринта взять одну или несколько дополнительных историй пользователей с высшим приоритетом из тех, которые не вошли в спринт.

Доска задач

Самым наглядным инструментом мониторинга и управления внутри спринта является доска задач, которая по функционалу похожа на аналогичный инструмент из канбана. Но поскольку в Scrum имеются фиксированные итерации, доска по завершении спринта «обнуляется» - с нее убираются все стикеры.

На стикерах указывается наименование истории пользователя, и они двигаются по соответствующим состояниям во время спринта. В начале спринта все истории пользователей находятся в первом столбце, отсортированные вертикально по важности:

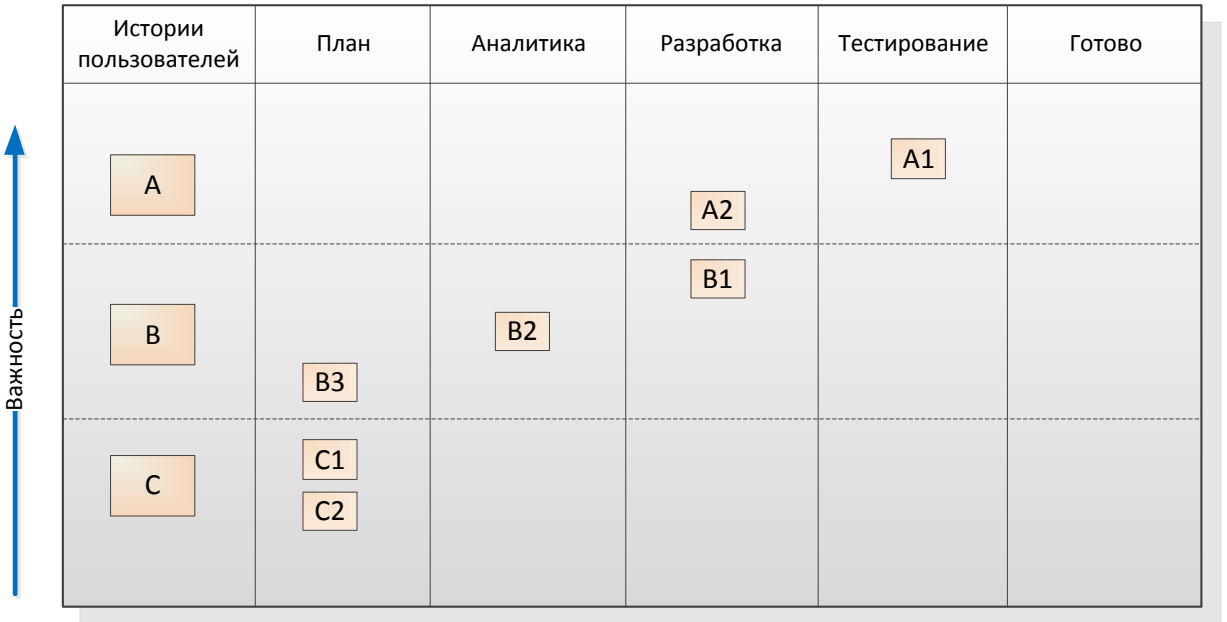
По мере того, как истории пользователей реализуются, члены команды меняют статусы у задач, и доска к середине спринта выглядит примерно так:



Команда делает задачи по важности, начиная с самых верхних, доводя их до статуса «Готово». Если все истории пользователей удалось реализовать, то в завершении спринта доска будет выглядеть так:

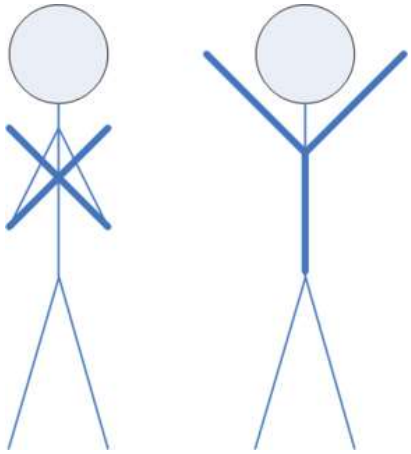
Еще одним способом использования доски является следующий подход:

- Истории пользователей берутся достаточно большие (3-7 на спринт) и располагаются в отдельном столбце;
- Истории пользователей разбиваются на небольшие продуктовые задачи, которые и передвигаются по состояниям по соответствующей дорожке.



Теории X и Y

Существуют две, фактически, противоположные друг другу теории в мотивации людей: теория X и теория Y.



Теория X

Согласно теории X люди работают только для того, чтобы получать зарплату. Думаю, все сталкивались с такими сотрудниками: пришел с утра, посидел до обеда, потом потерпел до вечера и домой. Они не работают, а отрабатывают. Для них работа измеряется часами, а не достигнутыми результатами.

Что делать руководителю?

Можно было бы дать совет: «Не брать таких людей на работу». Но, во-первых, их действительно много и среди них есть очень талантливые люди, а, во-вторых, такое поведение и низкая мотивация не является врожденной и неизлечимой. Да, таких людей зажечь и увлечь работой будет не просто, да для них нужно будет осуществлять тщательное планирование и контроль результатов, да, такие люди, безусловно, вызов и проверка для руководителя. Часто в результате у руководителя вырабатывается авторитарный стиль руководства — подчиненные отвечают ему тем же, в итоге получается замкнутый круг. При таком подходе не будет поступать «снизу» никаких рацпредложений по инновациям, даже если первоначально они были. В данном случае действовать методом кнута следует не всегда, так как это только усилит отторжение коллективом руководителя. Команда, настроенная против лидера, не проявляющая инициатив, вряд ли сможет приносить прибыль своей компании, таким образом, цель не будет достигнута.

Теория Y

Теория Y гласит, что люди в принципе высокомотивированы на достижение результатов и сама работа приносит им удовольствие. Опять же всем нам знаком типаж людей, с большой самомотивацией. У них и вокруг них всегда кипит работа (передаю «привет» моей команде), они успевают за день сделать не одно дело и их производительность труда намного выше среднего.

Что делать руководителю?

Необходимо внимательно следить за мотивацией своей команды, четко понимать чего ожидает команда, поддерживать каждый успех и внимательно следить за инициативами, чтобы они не пропадали, а реализовывать. Управлять лучше по отклонениям, не сильно

наседая. При осуществлении контроля в таком стиле необходимо ставить четкие цели и добиваться того, чтобы команда их понимала и разделяла.

X+Y

Обычно руководитель работает на стыке обеих теорий, используя тот или иной подход в зависимости от ситуации. В общем случае надо стараться, работать по принципу теории Y, зажигая людей. В крайнем случае, прибегать к методам теории X, четко объясняя, в чем именно проблема.

Для работы в рамках Scrum команда должна состоять в основном из людей, которые больше соответствуют теории Y. Иначе команда не будет самоорганизованной и эффективной.

«Эффект наблюдателя»

> Скажи, как ты будешь измерять мою эффективность, и я скажу, как я буду себя вести.
> Э. Гольдратт
>
> Самые важные вещи нельзя измерить.
> У. Деминг

Чтобы узнать, посолен ли борщ, достаточно опустить в него два электрода, а затем пустить по ним ток. Если появится запах хлора — значит, борщ уже посолен.

В физике есть такое понятие как «эффект наблюдателя»: если над системой ведется наблюдение, то оно вносит изменение в ее поведение. Очень интересно этот эффект проявляется в организации работы команд разработчиков (да и в любых производственных процессах). Как только мы начинаем считать любые метрики, мы вносим изменения в поведения команды и отдельных ее членов.

«Не навреди»

С точки зрения управления, мерить в численном виде (вводить метрику), идея, на первый взгляд, очень здравая. Мы получаем точные данные и на их основе производим необходимые действия. Но, к сожалению, не все так просто: как только мы вводим метрику, поведение команды начинает меняться. Команда и каждый ее член начинает подстраиваться под введенную нами метрику.

Далеко за примерами ходить не надо. Все знают понятие «индусский код»: как только разработчикам начинают платить за количество написанных строчек кода, они и начинают писать больше кода, зачастую бессмысленного, забывают про рефакторинг, его делать становиться невыгодно и так далее. Таким образом, наше начинание по замеру производительности оборачивается против нас.

Можно попробовать сократить количество дефектов в нашем продукте, для чего считать, сколько багов сделал каждый разработчик. Лучших разработчиков будем награждать бонусами, а к худшим разработчикам - применять санкции. Все просто и прозрачно. Но на деле получится по-другому: разработчики будут спорить по каждому багу с тестировщиками, появится боязнь и желание избегать рисков. В результате вы получите отсутствие инициативы и нежелание делать поставленные задачи («Меньше задач, меньше багов»).

Давайте подойдем с другой стороны и будем считать, сколько багов находят тестировщики. В этой ситуации также появятся косвенные эффекты, ведь каждый тестировщик будет расценивать малейшее отклонение как дефект.

Вывод из приведенных выше примеров простой: при введении метрик необходимо руководствоваться, прежде всего, принципом «Не навреди».

Что же делать?

Проанализируйте, как измерения могут повлиять на поведение команды и отдельных ее членов. Причем анализ надо проводить, в первую очередь, по поводу не очевидных аспектов, которые могут проявиться не сразу. Подумайте, какие метрики могут послужить основой для обсуждений, например, на ретроспективах, если вы используете Scrum. Может быть, ваши измерения в действительности никому не нужны и вы меряете, просто, потому что можете мерить?

Посчитайте, сколько времени у вас уходит на сбор информации по метрикам. Очень вероятно, что этот процесс можно автоматизировать, в особенности, если мы говорим о метриках кода, метриках качества и тому подобных.

При внедрении метрик, как и любых других инноваций, очень неплохо использовать цикл Деминга: Plan-Do-Check-Act, чтобы у вас была возможность получить обратную связь от команды и внести изменения в случае необходимости.

А между тем, это как ни парадоксально, это очень точная метрика. Два программиста в команде правят примерно одинаковое количество строчек за один и тот же промежуток времени. Конечно, необходимо много условностей (схожие задачи, наличие стандартов кодирования, например, максимальная длина метода, отсутствие дублирования, рефакторинг и так далее), но все же метрика достаточно **точная**… если не использовать ее как оценку производительности.

6.Управление контрактами

Сроки и долгосрочное планирование в Agile

Существует мнение, что в гибких методологиях планирование либо отсутствует вообще, либо носит краткосрочный характер.

Задача классического управления проектами: «Сделать проект в срок, в рамках бюджета, реализовав полностью функционал и соблюдая установленные критерии качества». Давайте посмотрим на первую составляющую – на сроки.

Главной причиной появления этого ограничения проекта является недоверие. Это может быть недоверие между заказчиком и исполнителем, между менеджментом и командой и так далее. Наличие сроков создает ощущение управляемости (при поставленных процессах и обеспечивает управляемость), но приводит к недопониманию и еще большему недоверию, образуя замкнутый круг, компоненты которого усиливают друг друга:

Ограничение по срокам возникает как институт, который призван снизить риски заказчика. Но на практике получается наоборот, жесткие давящие сроки приводят к деструктивной позиции обеих сторон: исполнитель запрещает заказчику менять требования, даже если для этого имеется реальная бизнес-необходимость, а заказчик жестко требует исполнения сроков порой в ущерб качеству, ведь масштаб проекта также фиксируется.

Необходимо отметить, что недоверие преодолеть очень непросто.

Оценка сроков методом PERT

Давай рассмотрим стандартный подход к оценке сроков завершения проекта – метод PERT или метод оценки по трем точкам. Его преимуществом является простота и определенный учет рисков. К недостаткам относят маленькую точность и необходимость иметь полную информацию о масштабе работ, что не очень подходит для Scrum.

Метод заключается в получении трех оценок:

- Оптимистичной (Мин)
- Вероятной (Сред)
- Пессимистичной (Макс)

Тогда вычислить наиболее вероятную дату завершения можно по формуле:

(Мин + 4*Сред + Макс) / 6

Отклонение можно вычислить по формуле:

(Макс – Мин) / 6.

Оценка сроков релиза в Scrum проекте

Для оценки сроков релиза в Scrum проектах необходимы исторические данные либо из нескольких первых спринтов, либо из предыдущих проектов данной команды. После этого можно построить диаграмму сгорания для релиза целиком:

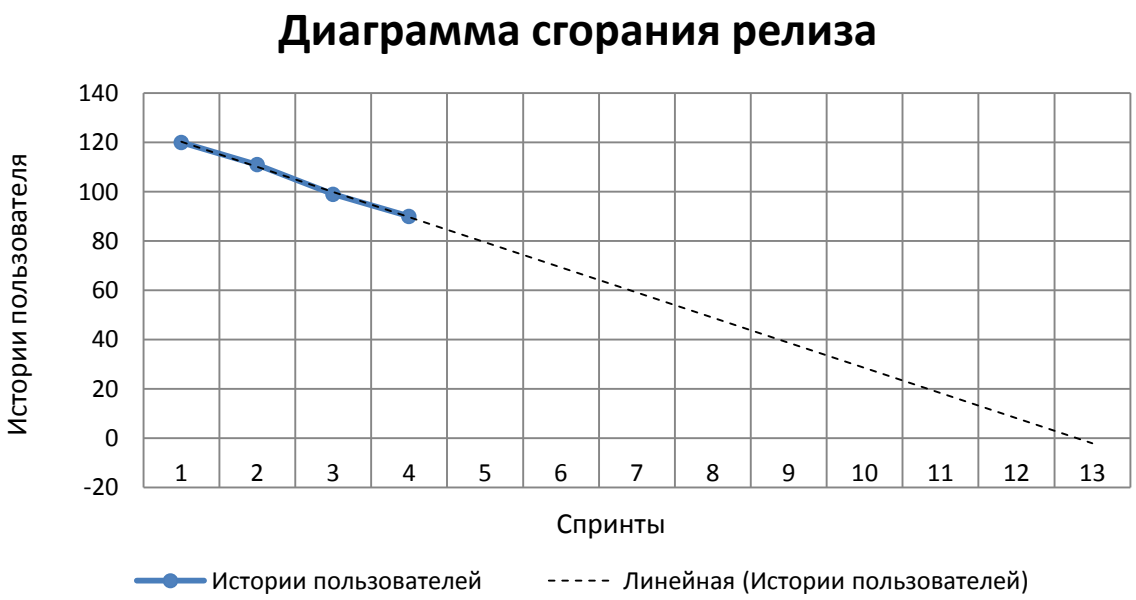


Рисунок 28. Диаграмма сгорания релиза показывает расчётную дату завершения проекта

Обратите внимание, что пунктиром построена линейная аппроксимация, с помощью которой можно вычислить, что проект завершится на 13-ом спринте. Если необходима большая точность, то истории пользователя можно оценить и вести подсчеты уже в сторипоинтах, потому что истории пользователи неодинаковы по размеру.

Отмечу, что в количество оставшихся историй пользователя нужно добавлять и истории пользователя, которые владелец продукта добавил в беклог продукта после очередного спринта.

Можно использовать и более изощренный подход: строить график добавления историй пользователя отдельно «вниз» в отрицательные квадранты и просчитывать вероятность завершения релиза в данном спринте (подробнее у (Дорофеев, 2010)):

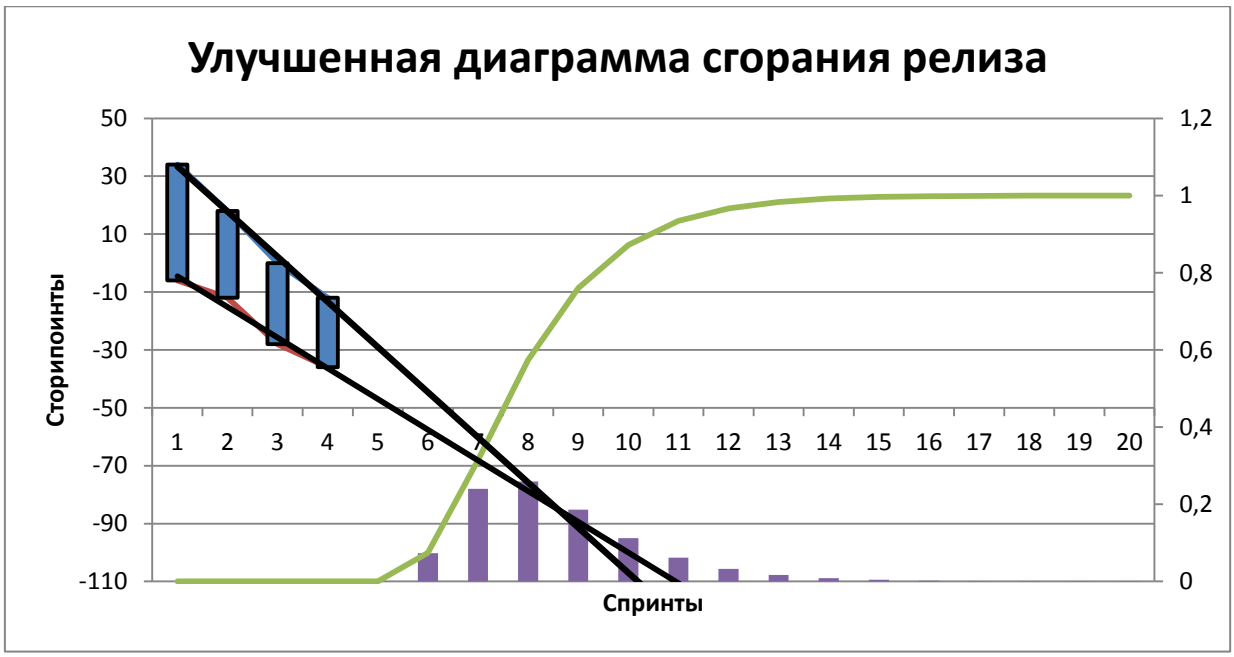


Рисунок 29. Данная диаграмма сгорания дополнительно показывает дифференциальную и кумулятивную вероятность даты релиза

Преимуществом такого подхода является то, что мы можем выбрать величину риска, с которой мы готовы работать, что очень важно в условиях высокой конкуренции:

- Если штрафные санкции по проекту велики, а вознаграждение не очень, мы можем ограничиться кумулятивной вероятностью в **96,6%** и возьмем обязательства по завершению проекта за **12** спринтов;
- При более мягких штрафных санкциях и большем вознаграждении можно использовать более мягкие кумулятивные вероятности завершения в **93,4%** за **11** спринтов и в **87,2%** за **10** спринтов;
- При отсутствии штрафных санкций (например, для внутренних проектов) можно поставить в качестве срока завершения **9** спринтов с кумулятивной вероятностью **75,9%**.

Подчеркну, что у самого вероятного исхода (завершение на **8** спринт) кумулятивная вероятность составляет всего **57,3%.**

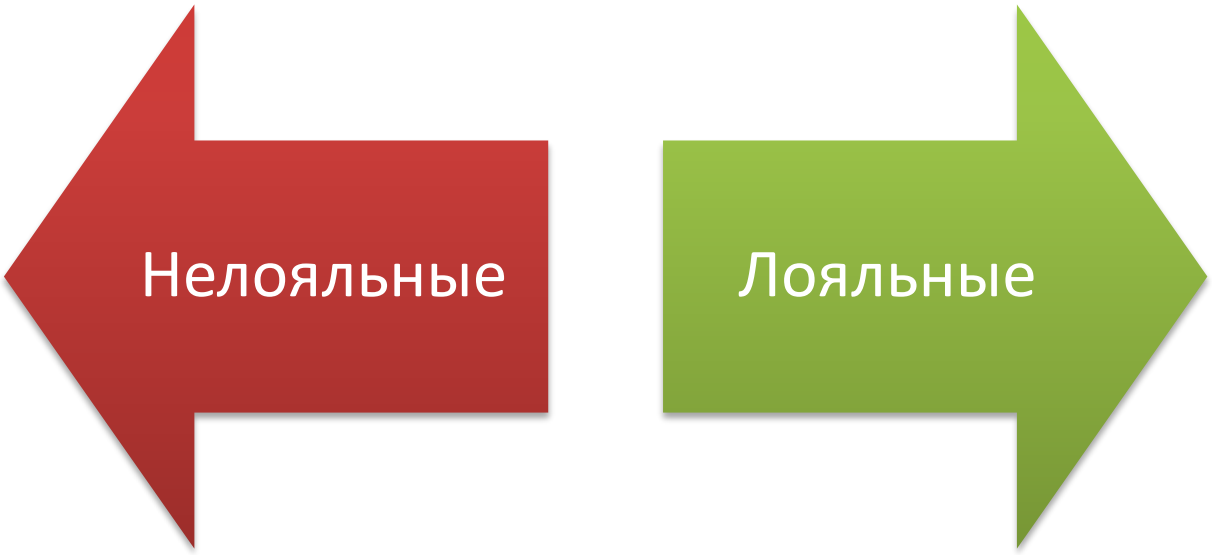
В отличие от классических методов определения срока завершения проекта или релиза, при гибком планировании мы имеем дело не с конкретной датой, а с вероятностными распределениями. И владелец продукта должен сознательно принимать решение о мере риска, на которую он и команда готова пойти.

«Вредные» клиенты

Поскольку гибкий подход подразумевает выстраивание по-настоящему партнерских отношений с заказчиком, необходимо выработать стратегию работы с «вредными» заказчиками. Ведь построить с ними партнерские взаимоотношения чрезвычайно сложно.

Давайте подумаем, как дифференцировать клиентов по «вредности» и как (и стоит ли) работать с «вредными» клиентами. Сначала определимся, кого считать «вредным»

клиентом. Обычно используется в отношении клиентов термин — лояльность. В самом простом случае клиентов можно разделить на три группы: нелояльные, средние и лояльные:



Лояльный клиент доставляет меньше всего проблем, обычно делает заказ на большую сумму, чем средний клиент. К тому же они часто делают повторные заказы. Таких клиентов надо холить и лелеять, для чего можно разработать программу повышения лояльности: от предоставления скидок до специальных условий работы.

Значительную часть клиентов можно отнести к категории нормальных или средних. С ними не возникает особых проблем, но и чудес лояльности они не покажут.

Самым плохим вариантом являются нелояльные или проблемные клиенты. Для каждой организации нелояльность клиентов определяется по-разному. Во всех сферах деятельности к таким клиентам относят тех, кто не соблюдает в той или иной форме договоренности, и, прежде всего, это касается финансовой части. Если говорить про разработку, то здесь стоит выделить клиентов, которые сами срывают сроки: не предоставляют материалов, не успевают проверять сделанную работу и прочее.

Сначала необходимо нелояльного клиента определить до появления проблем. Это можно сделать, наведя справки и в процессе непосредственного общения. Далее необходимо рассчитать возможные риски: проблемы с оплатой, различные задержки. В самом простом случае надо пометить в CRM, что с данным клиентом могут быть проблемы.

Следует хорошо подумать: нужен ли вам этот клиент, ведь с ним не получиться работать на 100% по Agile. Здесь подход должен быть комплексным: во внимание нужно принять как стоимость и сроки проекта, так и возможные риски.

7.Управление рисками

Традиционный Scrum не содержит такой дисциплины, поэтому необходимо адаптировать классические подходы по управлению рисками, не забывая про принципы Agile.

Хотя выделенной практики по управлению в Scrum нет, надо отметить, что как любая итеративная и инкрементальная методология, Scrum значительно снижает риски за счет получения ранней обратной связи от заказчика. Еще в Scrum есть очень хорошая практика проведения ретроспектив в конце спринта, она поможет выработать реакцию на риски, но, к сожалению реактивную, после того, как риски реализовались.

Работа с рисками ведется в несколько этапов, которые изображены на этой схеме:

Рисунок 30. Цикл управления рисками

Первый мозговой штурм по рискам можно включить в нулевой спринт (кстати, его можно провести в виде инновационной игры Speed Boat), и затем каждый спринт проводить дополнительный анализ и вырабатывать контрмеры. Отмечу, что контрмеры должны быть именно превентивными, так получается дешевле. Но ни в коем случае не делайте больше, чем надо, свято соблюдая принципы KISS и YAGNI. В мозговом штурме могут участвовать и представители заказчика, озвучивая своё видение возможных проблем.

Для стимуляции мозгового штурма крайне рекомендую использовать один из следующих риск-воркшопов:

- SEI Taxonomy-Based Risk Identification – таксономия рисков и опросник от Software Engineering Institute

- Дисциплина Управления Рисками MSF вер 1.1 – более легковесная версия категоризации софтверных рисков от Microsoft

Риски надо визуализировать, чтобы их знала вся команда (и заказчик) и полноценно участвовала в управлении ими. Самый плохой вариант сделать Excel-файл с рисками (в самом укромном уголке) и при провале проекта сказать: «Этот риск у меня есть в реестре под номером 37». Самый гибкий вариант – сделать доску с рисками и отслеживать их жизненный цикл.

Но рисками важно не переборщить, особенно привлекая к этой работе заказчика, ведь ему и команде может показаться, что проект состоит из одних потенциальных проблем. Очень ярко эта ситуация проявляется в командах, которые до этого не проводили подробный анализ рисков, а просто с завязанными глазами наступали на грабли.

Давайте чуть подробней остановимся на этапе анализа и приоритезации рисков. Мы договорились делать процесс максимально простым, поэтому предлагаю найти золотую середину между качественной и количественной оценкой рисков. Количественные оценки и математическое моделирование – вещь достаточно условная и необходимо хорошо понимать свойства построенной модели.

Возьмем только три градации для оценки вероятности и угроз (сколько ущерба он принесет) риска, при этом не будем использовать денежные оценки:

Вероятность / Угроза	Низкая=1	Средняя=2	Высокая=3
Высокая=3	3	6	9
Средняя=2	2	4	6
Низкая=1	1	2	3

Безусловно, максимум внимания необходимо уделить «красным» рискам, мало того, что они наиболее вероятны, но и ущерб от них обещает быть максимальным.

Активности, связанные с оперативным мониторингом рисков и коррекцией последствиями рисков очень удобно проводить на ежедневных скрам-митингах: теперь команда будет оперировать не виртуальными проблема, а конкретными рисками.

Что касается Lessons Learned, то для этого просто идеально подходит ретроспектива. Только замечу, что эти уроки и лучшие практики также необходимо распространять между командами, например, на Scrum of Scrum.

8.Инженерные практики

Инженерные практики представляют собой проверенные временем решения, связанные непосредственно с реализацией требований заказчика. Большинство практик, которые мы рассмотрим ниже, взяты из экстремального программирования, но они будут дополнены инспекциями кода и разработкой с тестами.

Непрерывная интеграция

Практика непрерывной интеграции заключается в использовании специального программного обеспечения, которое получает свежую версию исходного кода проекта, и производит сборку. В случае наличие проблем выводится и рассылается соответствующее сообщение. Отмечу, что в сборку проекта обязательно входит запуск автоматических тестов. Побочным продуктом являются разного рода отчеты о проекте, которые позволяют проводить анализ внутреннего качества проекта.

Непрерывная интеграция является своеобразным скелетом экстремального программирования, на который затем добавляются «мускулы» в виде других практик.

Разработка через тестирование и разработка с тестами

Сначала обсудим более традиционную практику – разработку с тестами. При таком подходе программист пишет код, а затем автоматизированные тесты для него для проверки корректности.

Экстремальное программирование идет дальше и превращает проверку качества в инструмент для создания спецификации и архитектуры. Для этого этап написания тестов переноситься в начала цикла разработки:

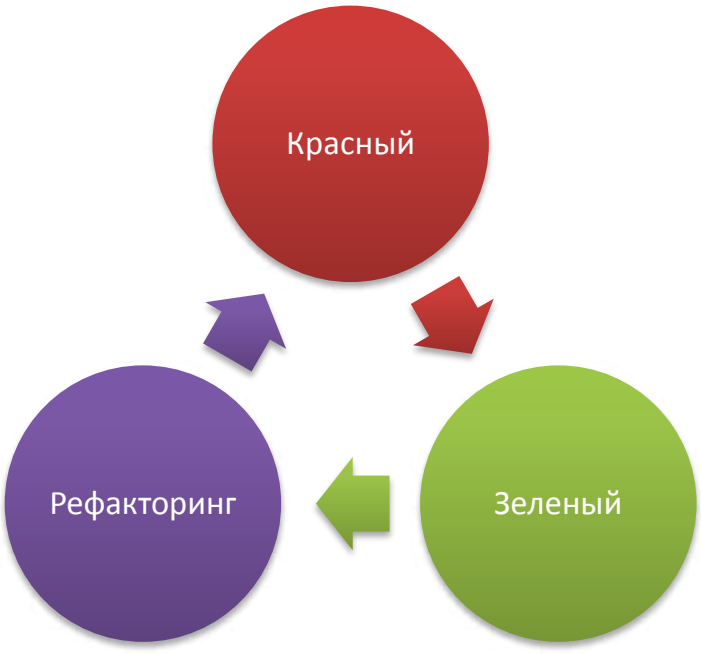
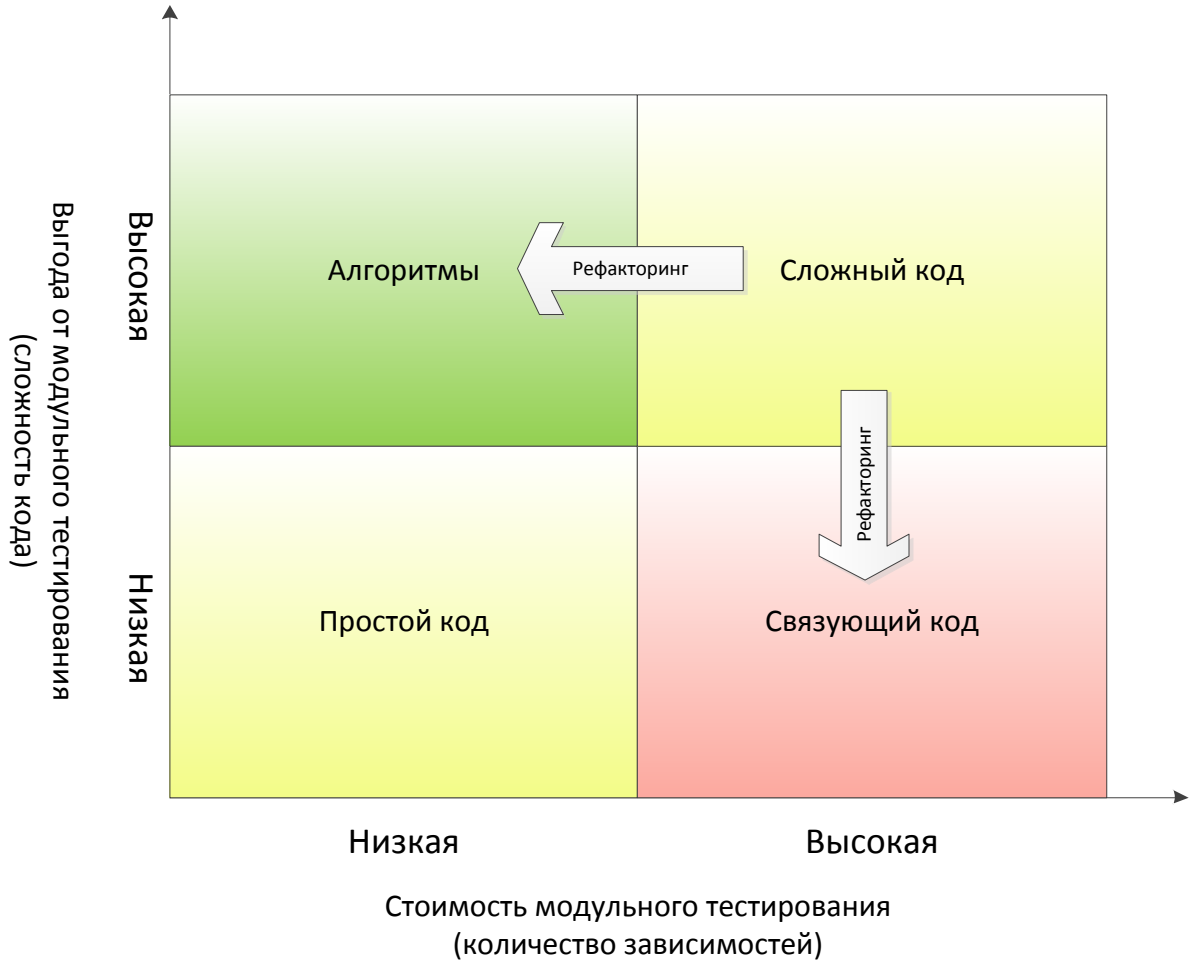


Рисунок 31. Цикл разработки в рамках TDD

Такой подход называется разработка через тестирование или разработка через тесты (Test Driven Development). Процесс работы разбивается на три этапа:

- красный – написание неработающего теста;
- зеленый – минимальными усилиями заставляем тест работать;
- рефакторинг – устраняем дублирования и приводим код в порядок.

Для выбора кода, который следует покрыть тестами, можно использовать следующую схему:



- Простой код – это самый тривиальный код, в котором сложно допустить ошибки, и который фактически не требует тестирования. Писать тесты для него необходимо только в минимальном количестве.
- Алгоритмы – это код, реализующий разного рода алгоритмы и бизнес-логику. Он достаточно независим от других частей, и тестировать его необходимо максимально тщательно.

> При разработке с тестированием хорошо сразу включить наличие тестов в критерии готовности истории пользователя. Это дисциплинирует разработчиков.
>
> Шабакаева Лия, разработчик

– Связующий код – это код, с максимальным количеством зависимостей, что сильно повышает стоимость поддержки модульных тестов для него, поэтому их необходимо писать в минимальных количествах.
– Сложный код – достаточно сложный и запутанный код, но для него нужны тесты. Как правило, его можно отрефакторить и сосредоточить в итоге свои усилия на алгоритмах.

В рамках TDD используется следующая практика из экстремального программирования – рефакторинг.

Рефакторинг

Рефакторинг – это изменения исходного кода без изменения функциональности для улучшения внутреннего качества (простота кода, гибкость архитектуры и так далее). Для проведения рефакторинга желательно знать «запахи кода» и непосредственно приемы рефакторинга (Фаулер, 2009):

Рисунок 32. Структура процесса рефакторинга

Парное программирование

При парном программирование код пишется двумя разработчиками за одним компьютером, причем один из разработчиков играет роль «пилота», а второй роль «штурмана»:

Рисунок 33. Роли разработчиков при работе в паре

Формальные инспекции кода

Формальные инспекции кода не относятся к экстремальному программированию, эту практику занимает парное программирование. Но по статистике эта практика позволяет находить наибольшее число дефектов.

Для проведения формальный инспекции кода используют специальные чек-листы, в которых указаны правила, которые должен соблюдать программист при разработке кода. Отметим, что эти правила должны быть нетривиальными, и не стоит включать в этот список правила, которые можно проверить автоматически при сборке проекта.

Простота архитектуры и метафора системы

Поскольку мы работаем итеративно, нам важно иметь максимально простую архитектуру, в которую будет возможно быстро и дешево внести изменения.

С другой стороны мы можем для улучшения понимания системы придумать метафору, которая будет ее описывать. Или, в крайнем случае, подобрать понятие из предметной области нашего приложения. Хорошим примером здесь может служить «корзины» в Интернет магазинах или «окна» в графическом интерфейсе операционных систем.

Коллективное владение кодом и стандарт кодирования

Коллективное владение кодом обеспечивает кроссфункциональность самих участников команды и позволяет реализовывать это важное свойство Scrum. Важным преимуществом такого подхода является быстрое распространение знаний между участниками команды.

Для реализации этой практики необходимо использовать стандарты кодирования, чтобы код, написанный разными участниками команды, был одинаков с точки зрения оформления. Проверку на соответствие стандартам лучше всего осуществлять на этапе сборки проекта в автоматическом режиме.

40-часовая рабочая неделя

Последней практикой, которую мы опишем, будет фиксированная 40-часовая рабочая неделя. Это гарантия для команды от перегрузок, одного из вида потерь в бережливом производстве. Надо очень четко понимать, что количество отработанных часов не равно количеству сделанного функционала, как и в любой интеллектуальной и инженерной деятельности.

9.Контроль и обеспечение качества

Интеграция контроля и обеспечения качества в Scrum

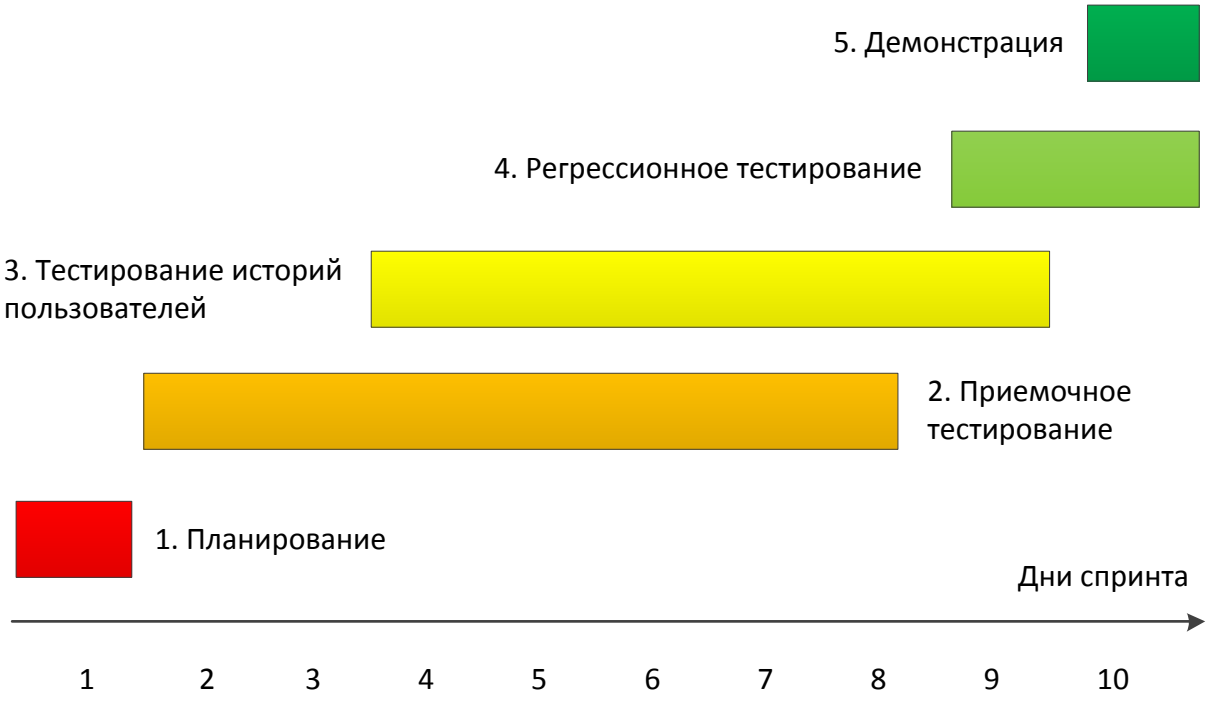
В гибких методологиях за обеспечение качества отвечает вся команда, но большая часть работы, безусловно, ложиться на тестировщиков, хотя их роль изменяется по сравнению с классическим подходом. У тестировщиков в Scrum есть две основные функции:

- Регрессионное тестирование функционала с предыдущих спринтов;
- Приемочное тестирование спринтов и релизов.

Обычно функционал растет очень быстро, и проекты испытывают острый недостаток в тестировщиках, поэтому для регрессионного тестирования необходимо применять автоматические тесты. Даже при достаточном количестве тестировщиков, они не смогут вручную тестировать регрессию по всё возрастающему количеству функционала продукта.

Это же касается тестирования функционала очередного спринта, иначе вы рискуете попасть в «регрессионную спираль смерти», но его можно разбавить и ручным тестированием.

Структура спринта для тестировщиков



В рамках спринта у тестировщиков есть пять основных активностей:

1. Планирование итерации
2. Автоматизация приемочного тестирования
3. Тестирование историй пользователей
4. Регрессионное тестирование
5. Демонстрация

Тестировщики обязательно должны участвовать вместе с командой в планировании спринта и оценке задач. Необходимо, чтобы к старту спринта тестировщики четко понимали суть и рамки каждой задачи. При наличии времени можно составить краткие сценарии тестирования историй пользователей, которые затем превратятся в автоматические тесты.

На этапе приемочного тестирования прорабатываются автоматические тесты для документирования нового функционала. Такой подход используется, прежде всего, в разработке через тестирование, где тесты пишутся до кода, который они проверяют. Затем на этапе тестирования историй пользователя происходит написание большего числа автоматических тестов.

К концу спринта происходит регрессионное тестирование, чтобы убедиться, что добавленный новый функционал не внес ошибки в старый функционал. В завершении спринта происходит демонстрация и ретроспектива. На ретроспективе тестировщик должен озвучить параметры качества, которых достигла команда в текущем спринте, и обсудить наиболее неприятные и критичные дефекты.

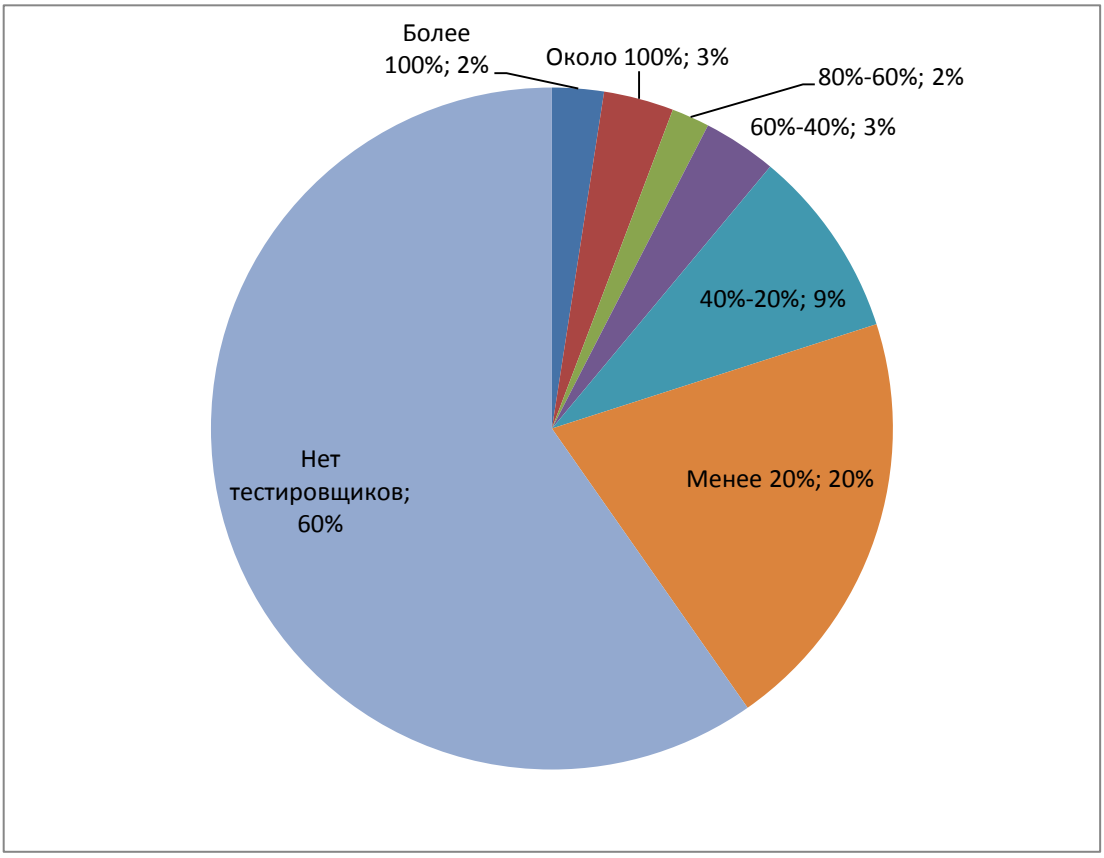
Сколько необходимо тестировщиков?

Многие профессионалы в нашей отрасли считают, что делать программы без дефектов невозможно. С этим очень трудно спорить, ведь зачастую в командах количество тестировщиков больше, чем разработчиков.

23 августа 2010 на сайте habrahabr.ru среди 1835 человек, из которых 551 человек воздержался, был проведен опрос по соотношению количества разработчиков и тестировщиков. Стоит отметить, что на сайте достаточно большое количество представителей небольших компаний/веб-студий и фрилансеров, но общую картину всё равно видно.

Количество тестировщиков определяется многими факторами:

- количеством разработчиков;
- наличием автоматических приемочных тестов;
- наличием практики написания модульных тестов разработчиками;
- общим качеством продукта;



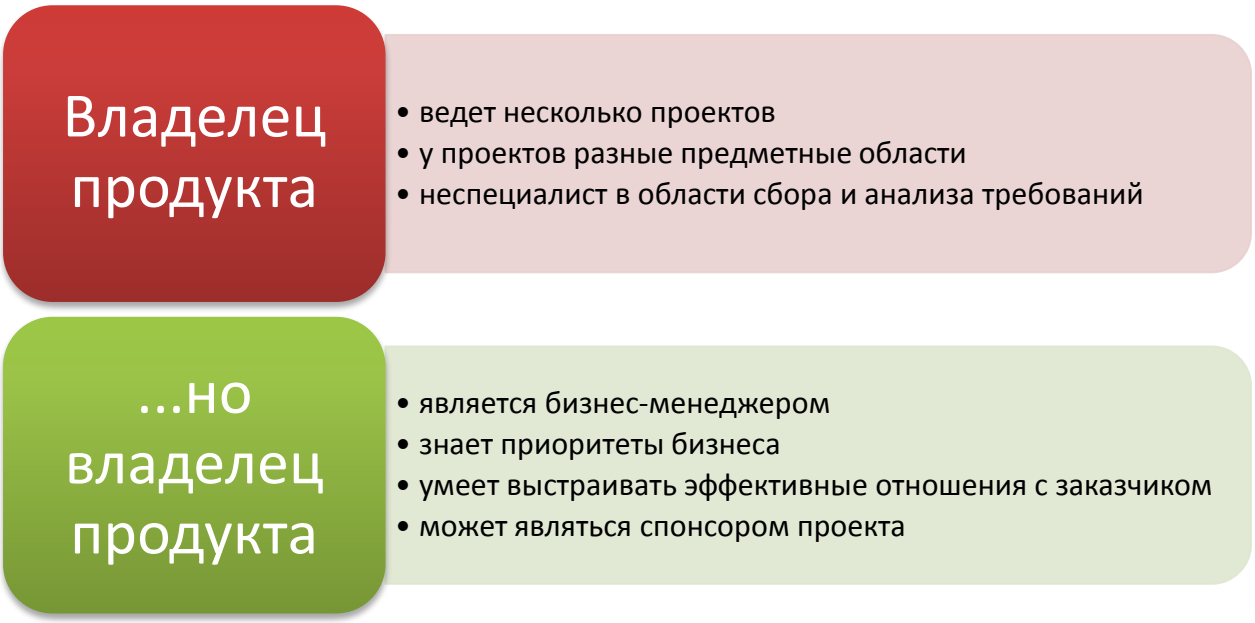
Из данной диаграммы видно, что в большинстве случаев тестировщиков в той или иной мере не хватает, поэтому их работу необходимо максимально автоматизировать, чтобы на выполнение тестов уходило минимум «ручного» труда.

10. Анализ требований

Scrum предоставляет гибкое и легковесное решение для управления требованиями, но зачастую есть проекты, для которых это решение приходиться расширять. Самым ярким примером здесь могут послужить проекты со сложной бизнес-логикой, которую необходимо сначала формализовать, чтобы реализовать функциональность, которую она описывает. Обычно описания такого рода хранятся в трекере либо в вики, а анализ требований ведет выделенный специалист – системный аналитик.

Роль системного аналитика

Чтобы понять, зачем вводится отдельная роль системного аналитика, давайте взглянем внимательнее на обязанности владельца продукта:



Чтобы разгрузить владельца продукта, часть его обязанностей, а именно – анализ и детальная проработка требований, отдается системному аналитику. Обращу ваше внимание, что расстановка приоритетов остается и становится главной обязанностью владельца продукта.

> Аналитик не выступает стеной между заказчиком и командой. Аналитик – член команды, который помогает заказчику понять, чего тот хочет на самом деле.
>
> Колосова Ксения, владелец продукта

UML

Классическим подходом к описанию требований в виде модели является UML, который позволяет описать буквально каждый аспект системы в визуальном виде. Но такой подход не является гибким:

- UML-диаграммы – это не конечный продукт, пользователям он не принесет ценность;
- UML-диаграммы – быстро теряют актуальность при начале разработки;
- UML очень объемен (более 10 видов диаграмм, 900-страничное официальное руководство) и избыточен, так как часть диаграмм фактически дублирует друг друга;
- UML описывает систему слишком подробно, часть знаний можно хранить и передавать в устном виде либо в виде текста;
- UML неявно подразумевает водопадную модель разработки;

> Если мы говорим о гибких процессах, то применение тех или иных средств визуализации системы должно основываться на здравом смысле. Ни одна диаграмма не заменит коммуникации в команде. Диаграммы подходят для описания сложных бизнес-процессов со сложной логикой поведения.
>
> Лукьянчикова Наталья, аналитик

Процесс ICONIX

Одним из вариантов гибкого анализа требований (и частично моделирования) является использование и адаптация процесса ICONIX, который подробно описан в (Rosenberg, и др., 2005) и в (Rosenberg, et al., 2007). ICONIX – это методология анализа требований, основанная на прецедентах использования. В рамках этого процесса используется небольшое подмножество UML, что делает его более легковесным:

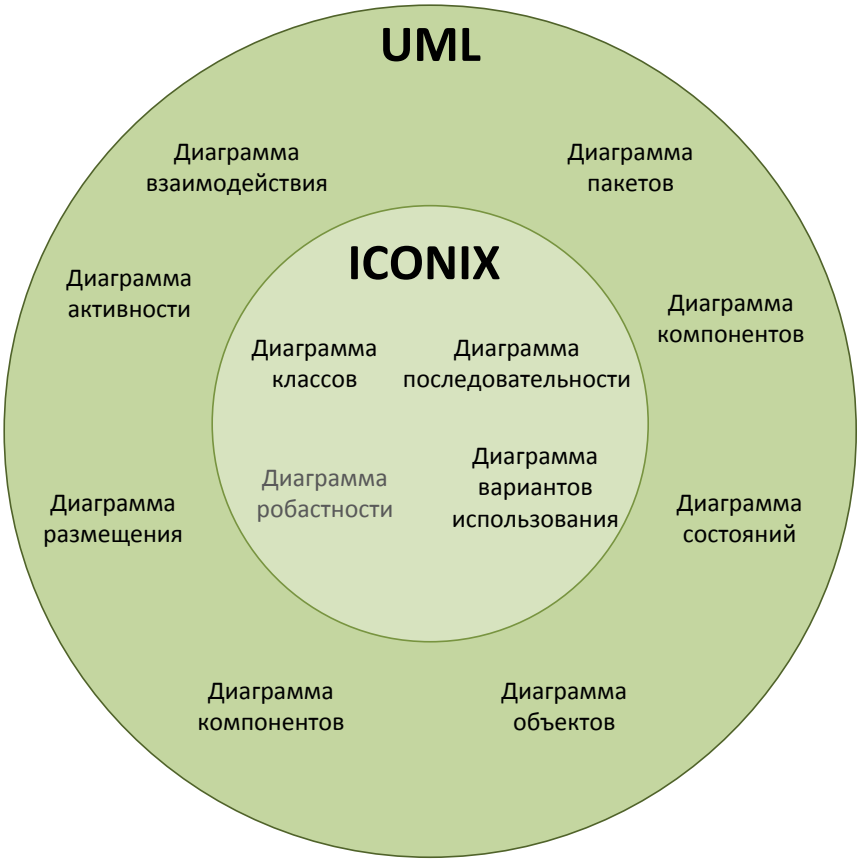


Рисунок 34. ICONIX - подмножество UML

Сам процесс разработки продукта в ICONIX носит практически водопадный характер, поэтому его необходимо адаптировать для итеративной методологии, к которой относиться Scrum:

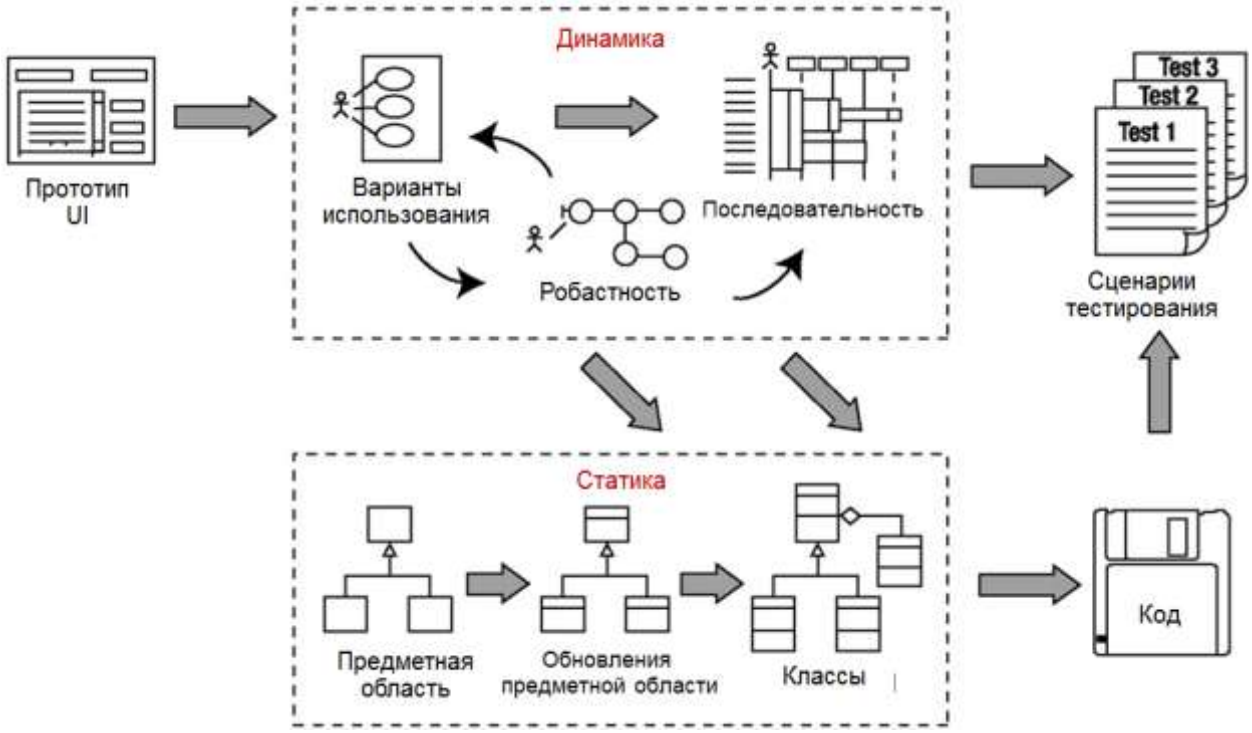
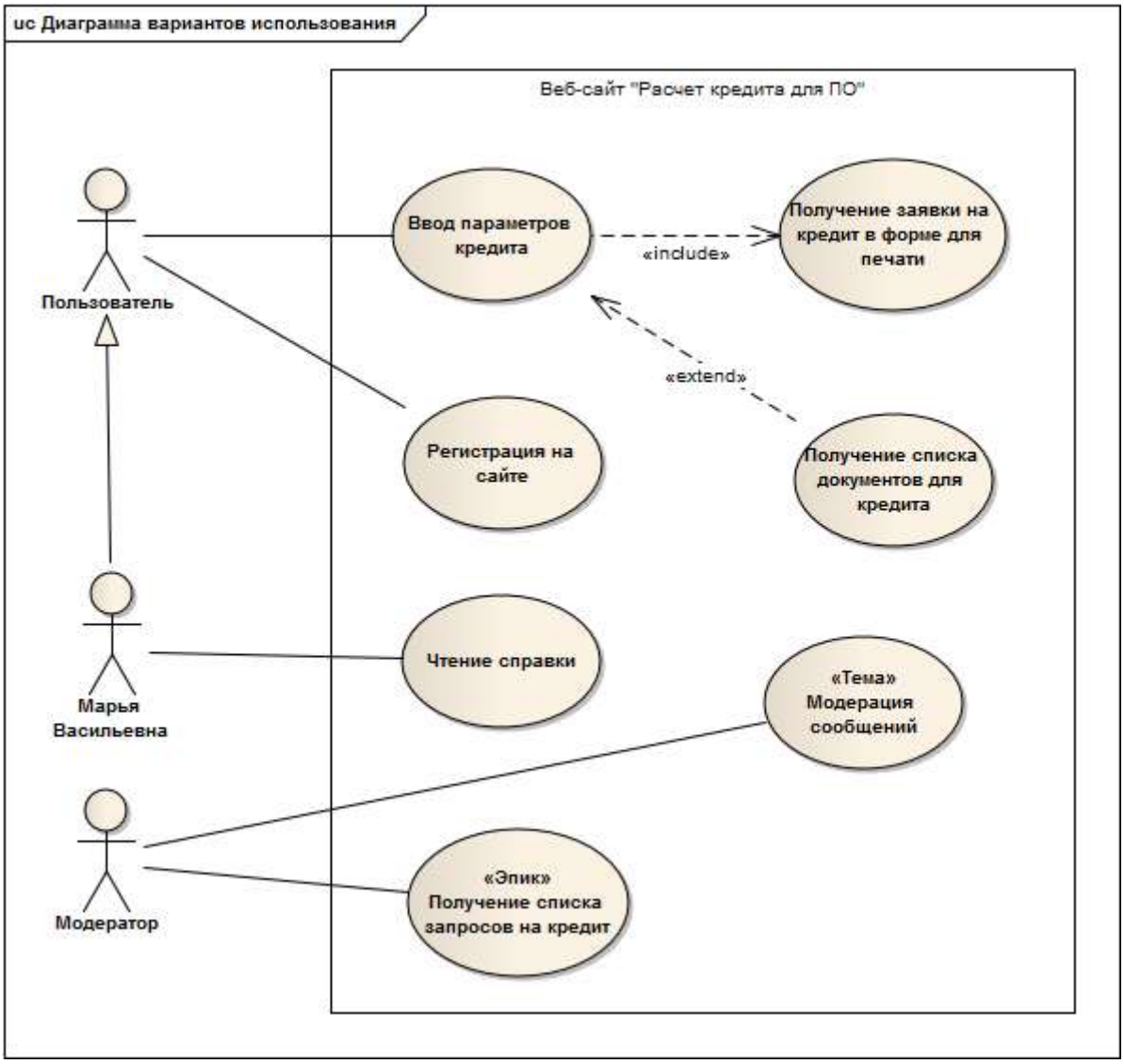


Рисунок 35. Структура процесса ICONIX

Давайте более подробно посмотрим, какие диаграммы предлагает нам ICONIX, и как будет происходить процесс анализа требований и моделирования. В качестве примера рассмотрим небольшое приложение по расчету кредита на веб-сайте.

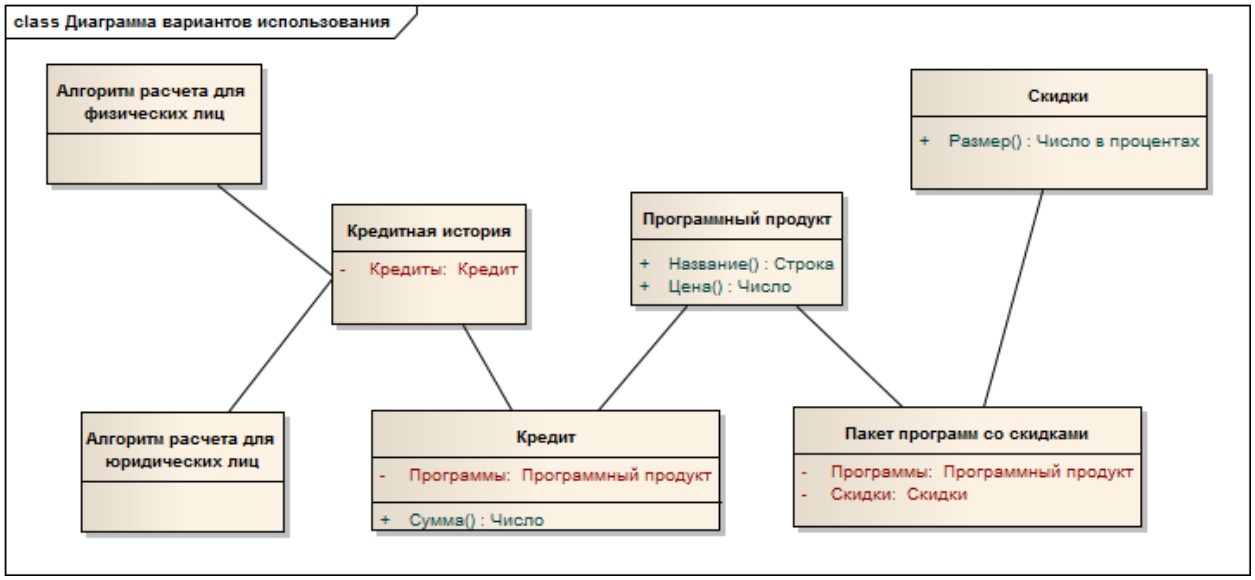
На первом этапе происходит анализ вариантов использования, который является своеобразным аналогом сторимаппинга:



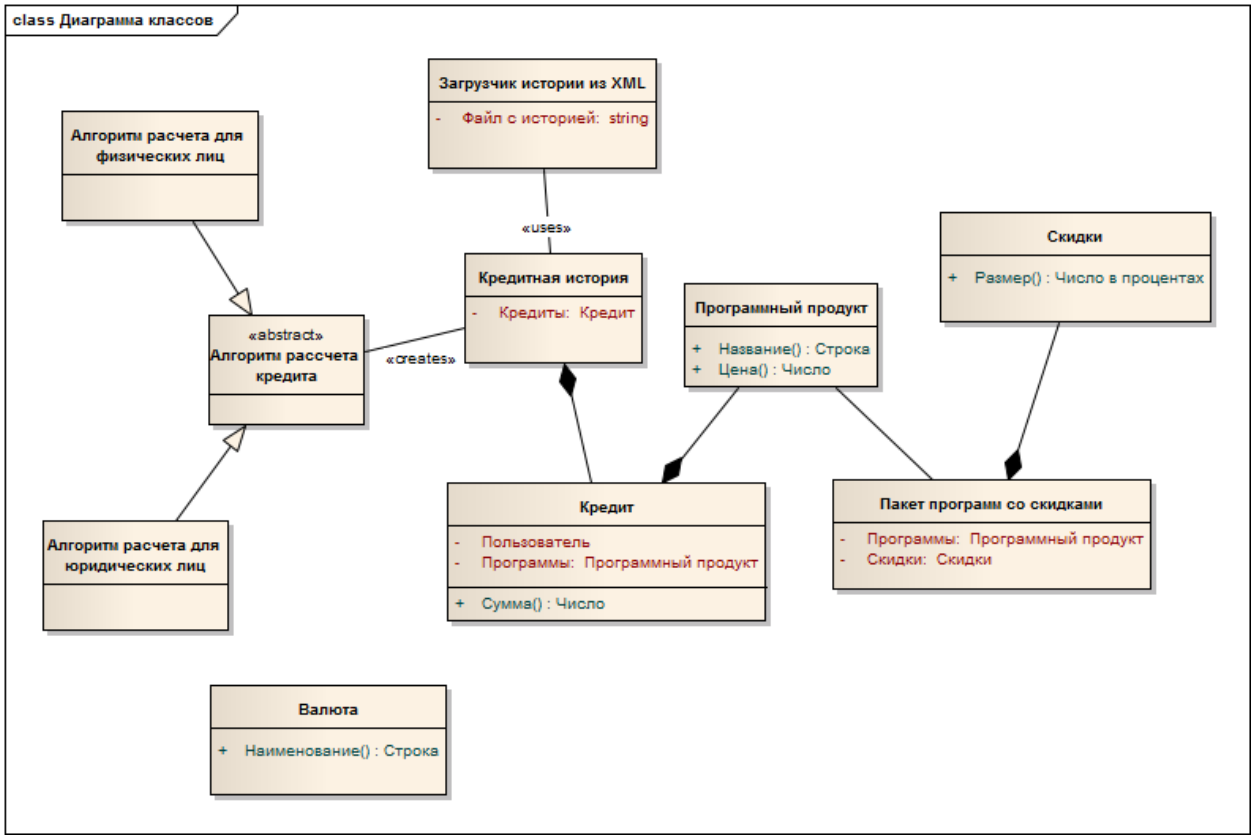
На ней отображаются роли пользователей (персоны из сторимаппинга) и варианты использования, которые фактически являются более тяжеловесным вариантом историй пользователей. Обратите внимание на стереотипы «Эпик» и «Тема», которыми обозначены два варианта использования.

Теперь можно провести анализ предметной области, хотя он часто проводиться параллельно. Для этого можно начать с соответствующей диаграммы, на которой мы

обозначим основные элементы нашего домена с полями и наметим связи между ними:



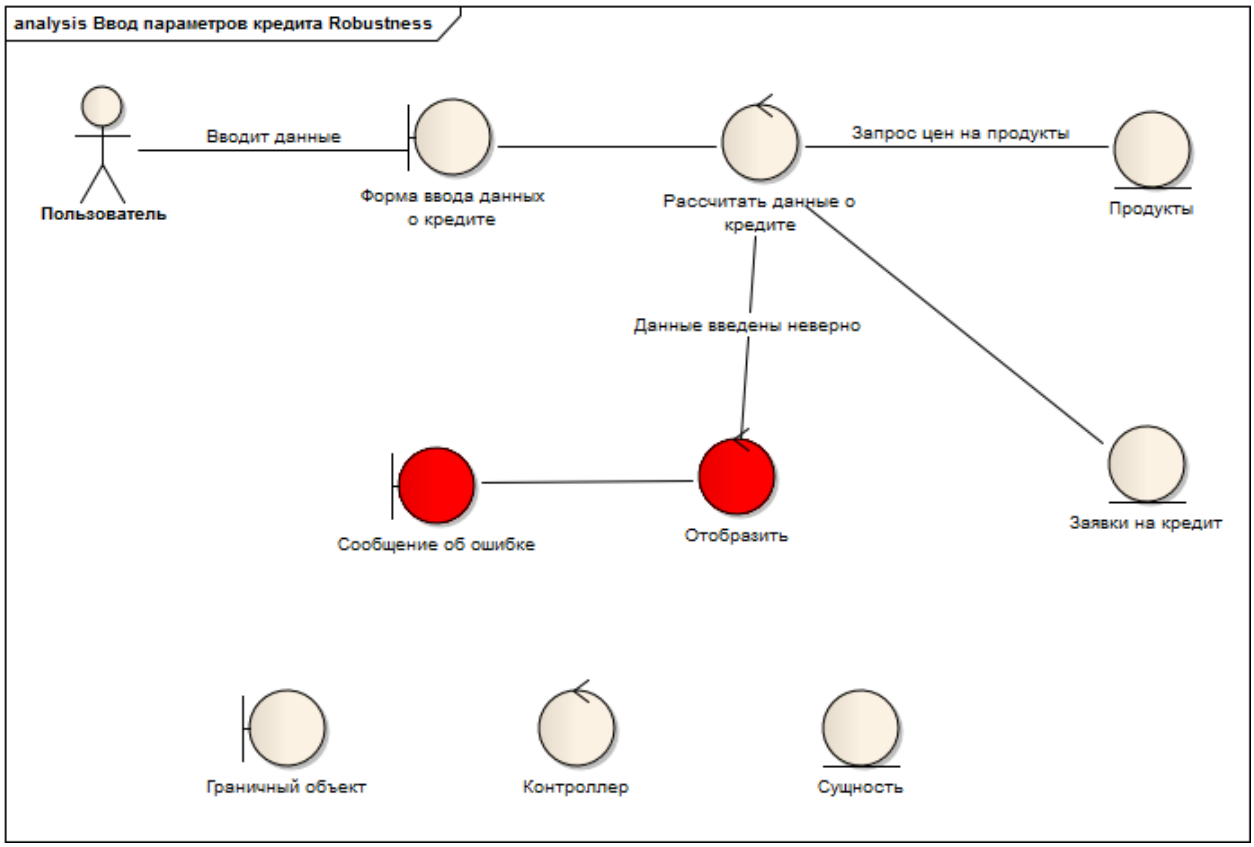
Затем можно продолжить анализ и добавить вспомогательные и абстрактные классы, которые могут напрямую не относиться к предметной области. Таким образом, мы получим набросок архитектуры нашего приложения в виде диаграммы классов:



Для примера разберем один из вариантов использования и опишем его более подробно в виде диаграммы робастности, на которой будут отражены:

- граничные объекты – интерфейс взаимодействия с пользователями;
- котроллеры – бизнес-логика и различные алгоритмы;
- сущности – данные приложения.

Можно заметить, что данная диаграмма очень похожа на шаблон проектирования Model-View-Controller:

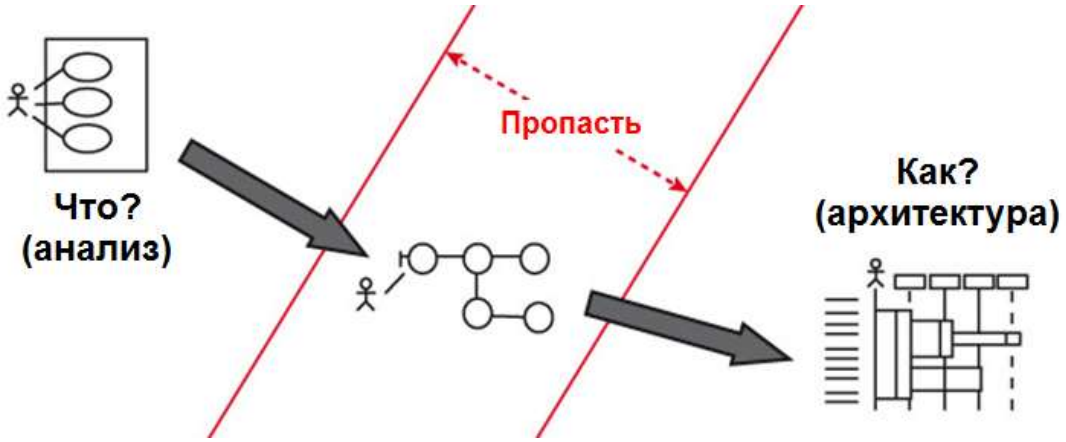


Красным выделены альтернативные (ошибочные) цепочки действий, про которые обычно забывают при анализе, хотя им нужно уделить максимальное внимание.

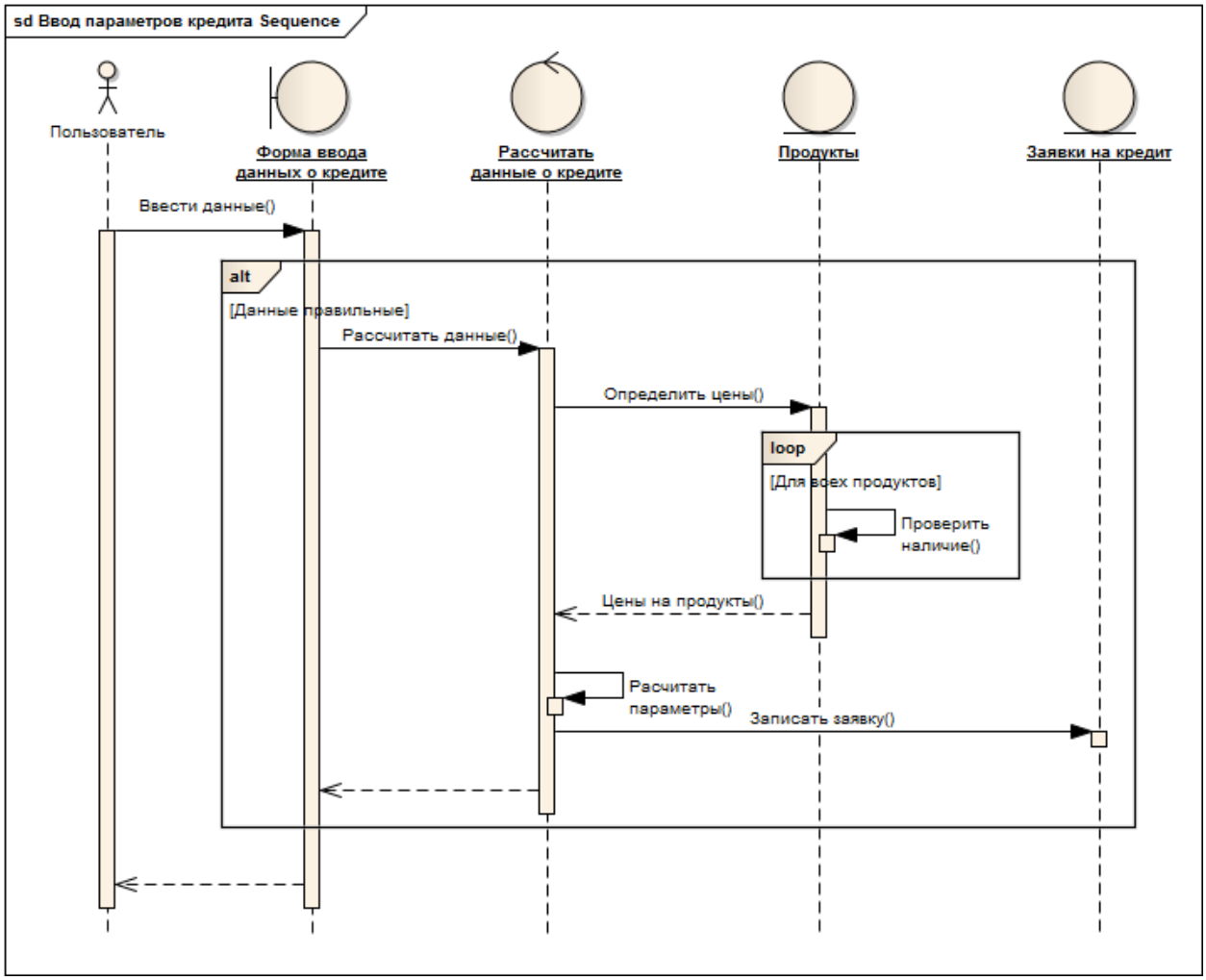
Диаграмма робастности нужна, чтобы:

- осуществить проверку полноты вариантов использования;
- выявить дополнительные объекты;
- проработать архитектуру на высоком уровне.

Последний пункт очень важен, ведь между анализом и архитектурой существует практически пропасть, которую эта диаграмма призвана заполнить:



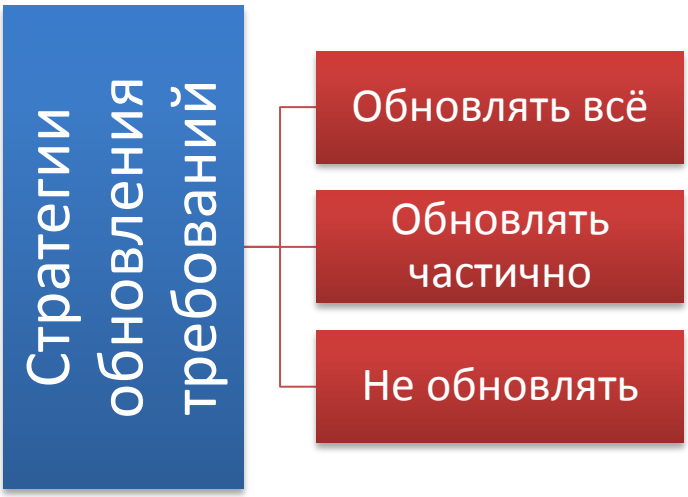
После такого анализа можно описать подробности алгоритма или логики в диаграмме последовательности:



Стратегия актуализации документации

Если рассматривать проектную документацию, в том числе требования, с позиций бережливого производства, то она является «мудой» – потерей при производстве (подробнее потери при производстве описаны в главе «Бережливое производство»). Аналогичный принцип действует и в гибких методологиях: прогресс измеряется не по документации, а по конечному продукту, который приносит ценность для заказчика.

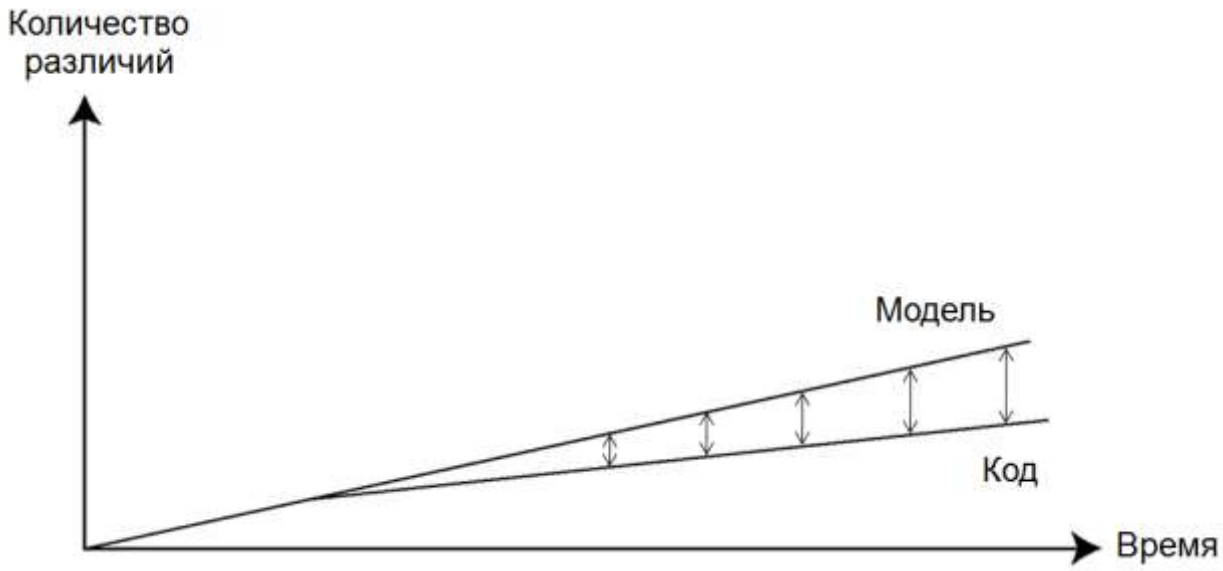
Фактически аналитик должен выбрать одну из трех стратегий актуализации требований:



Несмотря на то, что agile процессы ставят готовый продукт выше документации, роль документации не исключается как таковая. Принятые решения, ограничения системы, описание бизнес-процессов должны быть зафиксированы и быть доступны любому участнику команды.

Лукьянчикова Наталья, аналитик

Если документация не обновляется полностью, то с какого-то момента начинают накапливаться различия между моделью (требованиями) и кодом:



Роль аналитика в Scrum

Наиболее распространенной практикой интеграции аналитика в Scrum является подготовка требований до старта спринта, что позволяет провести более качественную оценку и обсуждения. Для этого аналитик берет еще не взятые в спринт требования и

проводит их детальный анализ. Если проводилась предварительная оценка требований, то их можно набирать в соответствии со скоростью работы команды:

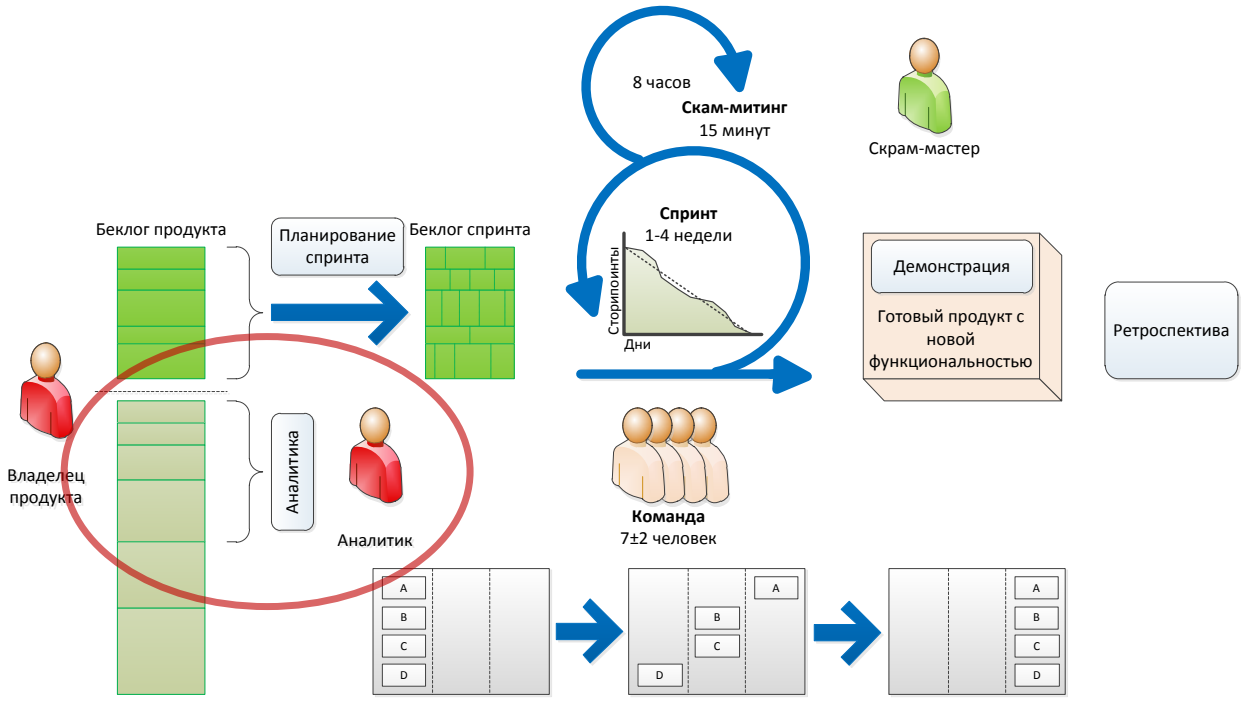
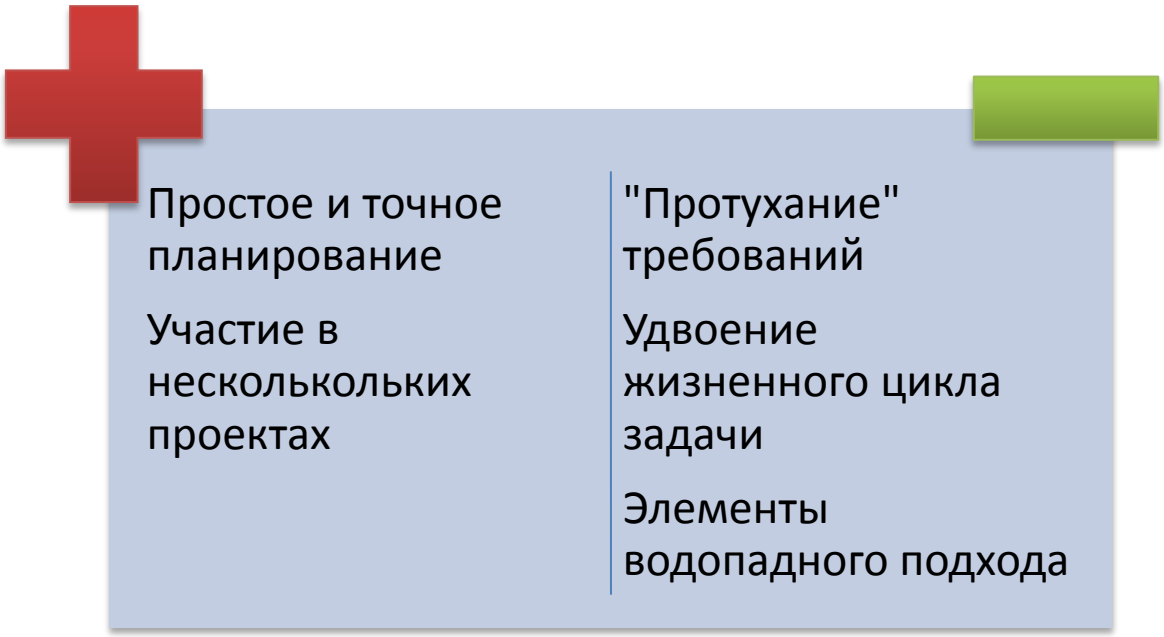


Рисунок 36. Место аналитики в Scrum

Получается, что аналитик как бы опережает команду на один спринт. У такого подхода есть свои плюсы и минусы:



Роль аналитика в канбане

Плюсы и минусы работы в Scrum в канбане меняются местами: здесь аналитик работает наравне со всеми членами команды в потоке задач. Обычно ему выделяют дополнительный столбец на доске:

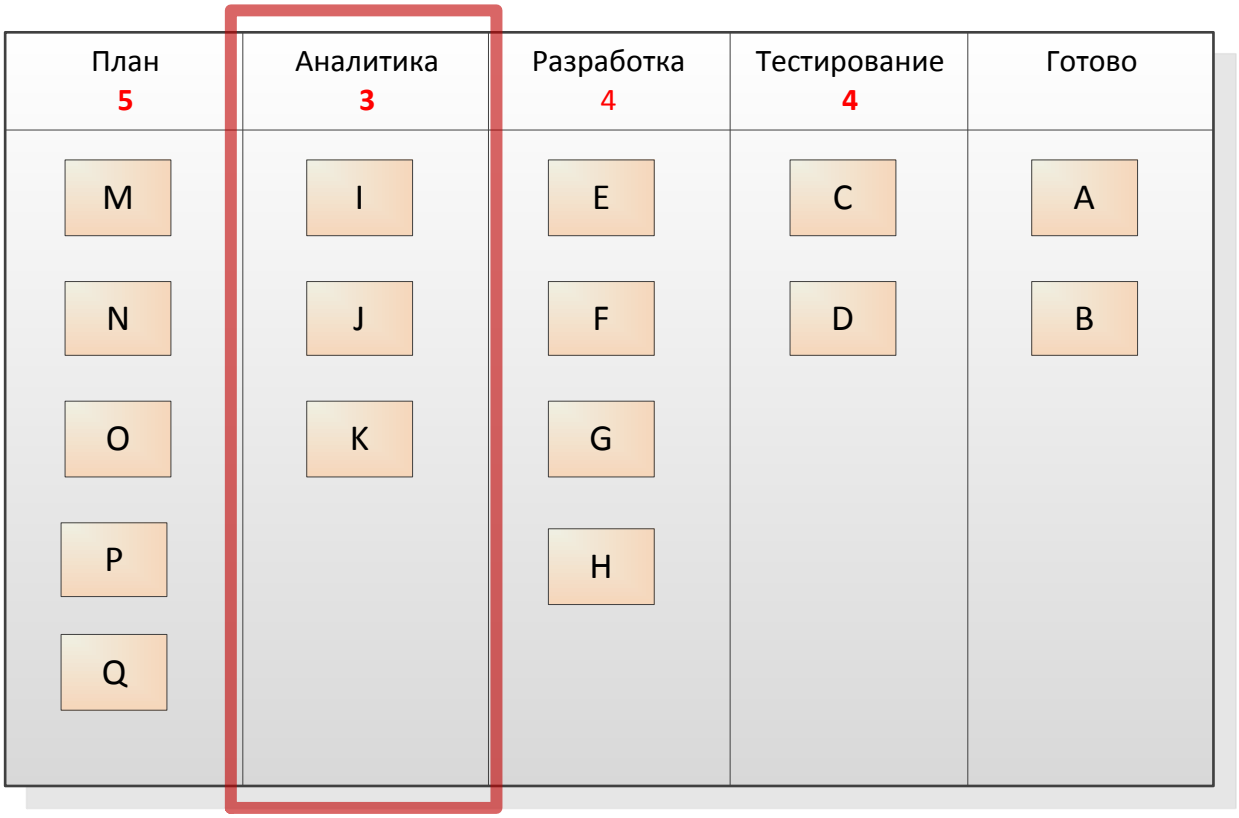


Рисунок 37. Столбец "Аналитика" для соответствующего состояния

Аналитик также попадает под третье правило канбана по оптимизации процесса с целью сокращения времени жизни задачи. По этой причине и из-за отсутствия спринтов время жизни при введении дополнительного этапа для аналитики в канбане так сильно не растягивается.

Прототипы

> Для заказчика прототипы играют такую же важную роль, какую диаграммы играют для команды – особенно в том случае, когда именно от аналитика поступает предложение, как воплотить в жизнь то, что хочет заказчик
>
> Колосова Ксения, владелец продукта

Важным элементом работы системного аналитика является создание прототипа. В зависимости от используемых бизнес-процессов и наличия специалистов прототип может быть более общим, отражая лишь функциональность, либо более конкретным и даже интерактивным:

Общий прототип подразумевает, что в дальнейшем он будет проработан специалистом по интерфейсу пользователя либо дизайнером в случае веб-разработки. Более конкретные прототипы часто могут быть взяты в работу непосредственно разработчиками, так как они содержат уже необходимую информацию:

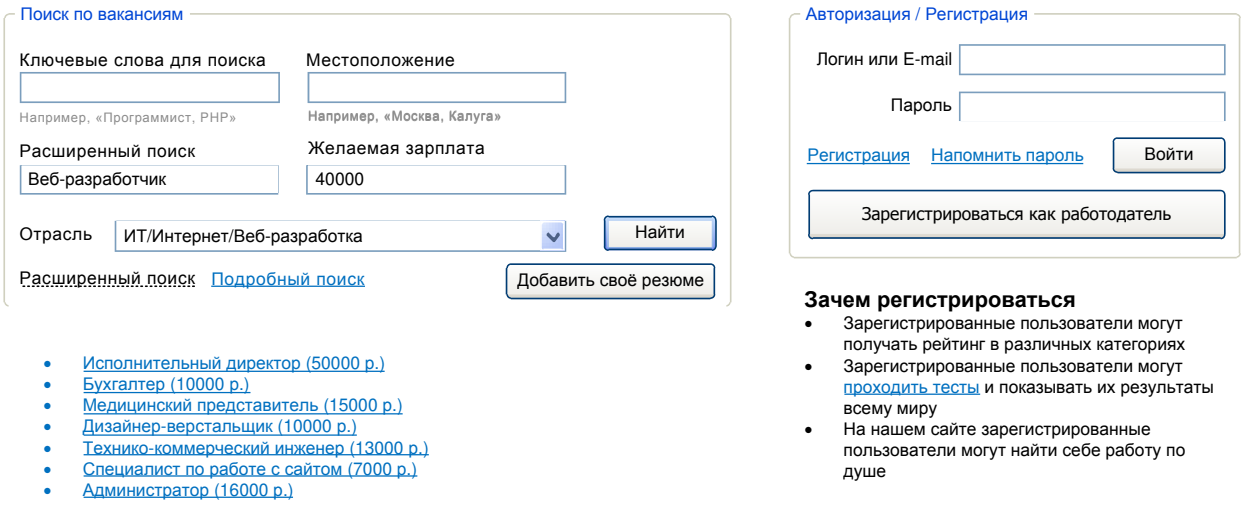


Рисунок 38. Прототип для веб-страницы

В рамках Scrum прототипы, рекомендуется, делать к конкретным историям пользователей в сочетании с текстовым описанием и диаграммами.

11. Масштабирование Agile

Классическая Agile-команда состоит из небольшого числа участников, обычно 7±2 человека. Это максимальное число людей, при котором возможно гибкое взаимодействие. При дальнейшем увеличении количества членов команды, резко увеличиваются издержки на коммуникации (количество возможных коммуникаций находится в квадратной зависимости от количества участников коммуникации).

Как было сказано выше, в Agile используется командный подход, таким образом, для масштабирования этих методологий на уровень предприятия необходимо выстроить систему взаимодействия отдельных команд.

Организационные структуры

Организационная структура (оргструктура) – это способ упорядочить сотрудников организации для достижения ее целей. Прежде всего, оргструктура создается и поддерживается для:

- координации деятельности сотрудников;
- обозначения границ ответственности.

Хочу отметить, что оргструктура зависит от многих факторов и описанный здесь вариант можно (и нужно) адаптировать в соответствии с окружением и бизнес-процессами организации. И конечно, оргструктура – это не монолит, который остается на весь жизненный цикл компании: она меняется с ней с течением времени и развитием компании.

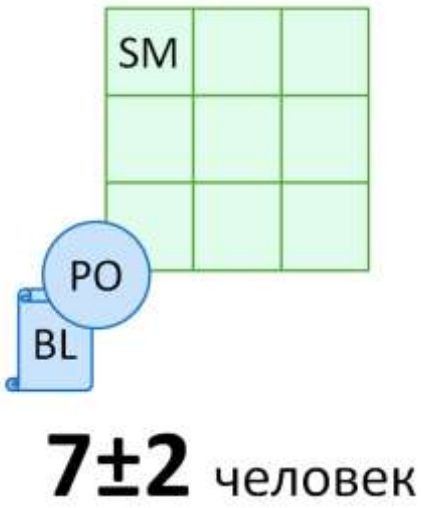
Виды оргструктур

Давайте кратко посмотрим, какие организационные структуры существуют, и какие мы можем использовать для масштабирования Scrum:

- **Дивизионная оргструктура** объединяет в дивизионы полный коллектив для разработки связанного семейства продуктов или для организации работ по географическому признаку.
- **Функциональная оргструктура** объединяет в функциональные единицы (обычно отделы) сотрудников одной специальности.
- **Командная (проектная) оргструктура** объединяет в небольшую группу сотрудников разных специальностей для реализации проекта.
- **Матричная оргструктура** представляет собой смесь функциональной и командой оргструктуры. В зависимости от степени участия/подчиненности/отчетности матричные оргструктуры бывают сильными, когда сотрудник больше относиться к команде, и слабыми, когда сотрудник больше относиться к функциональному отделу.

На разных уровнях компании могут использоваться разные виды организационных структур в зависимости от потребностей.

Scrum-команда: состав



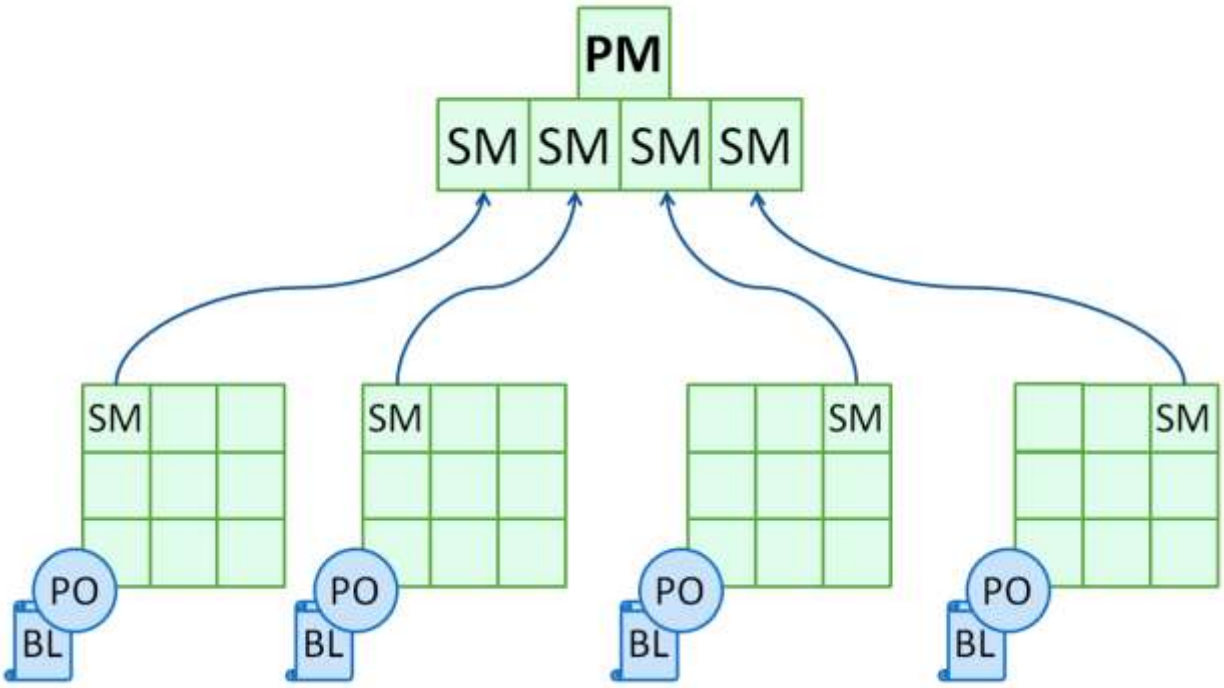
Команда в Scrum состоит из небольшого числа людей от 5 до 9 человек, чтобы можно было осуществлять эффективные коммуникации. Один из членов команды становиться скрам-мастером (SM), ответственным за процессы и социальный климат в команде.

Владелец продукта (PO) не является членом команды в прямом смысле этого слова, но он ставит команде цели с помощью требований в беклоге (BL).

Координация деятельности происходит ежедневно на скрам-митинге, который проводит скрам-мастер и в котором участвует вся команда. Основная цель скрам-митинга – координация работы команды на уровне отдельных историй пользователя.

Масштабирование Scrum

Небольшие команды показывают чаще хорошие результаты, чем большие, поэтому необходимо по возможности вести разработку компактными командами. К сожалению, часто бывает так, что размер проекта и сроки его реализации просто не позволяют вести разработку 5-9 людьми и приходиться задействовать несколько команд.



Для организации разработки больших проектов или портфеля проектов необходимо масштабировать Scrum на следующий уровень. Со стороны команд разработки – это выливается в проведении Scrum of Scrum.

На этот митинг собираются скрам-мастера (SM), в качестве представителей конкретных команд. Организует собрание руководитель программы (Program Manager – PM). В случае использования дивизионной организационной структуры на данном уровне, он также может являться руководителем соответствующего дивизиона (подразделения).

В качестве базовой структуры митинга можно предложить каждому скрам-мастеру ответить на следующие вопросы:

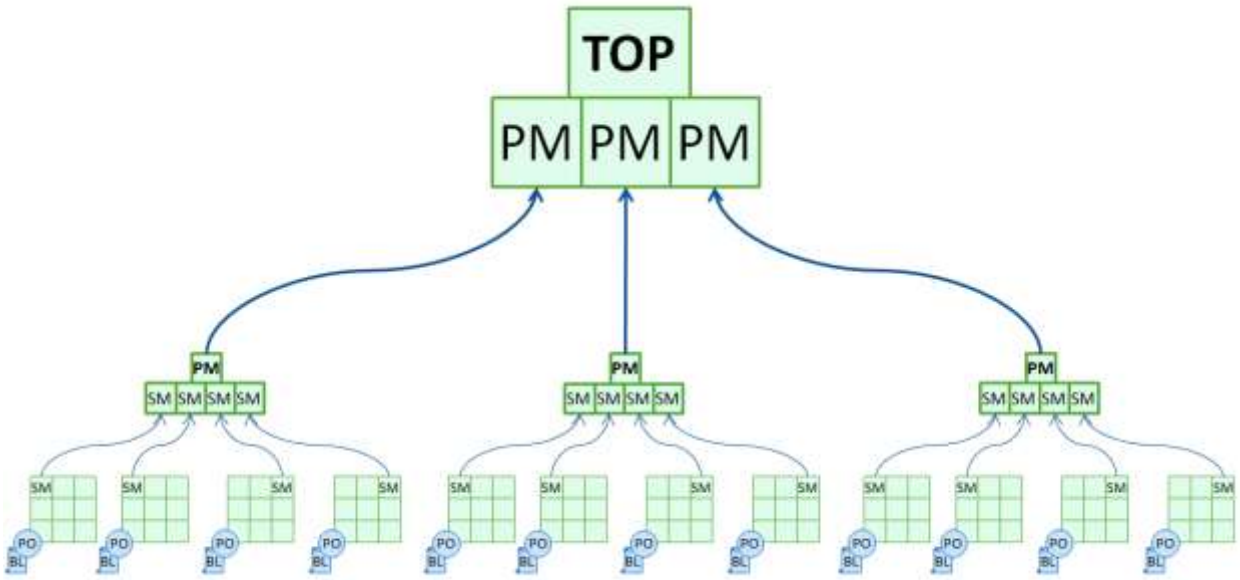
1. Что было сделано с прошлого Scrum of Scrum?
2. Какие были проблемы?
3. Что будет сделано к следующему Scrum of Scrum?

При этом акцент надо сделать на проблемах, которые команда не может решить сама, и вынуждена эскалировать выше.

Scrum of Scrum of Scrum

Следующим уровнем масштабирование является Scrum of Scrum of Scrum. Для осуществления этого проводится соответствующее мероприятие, на которое собираются менеджеры программ и топ-менеджер (обычно технический директор подразделения).

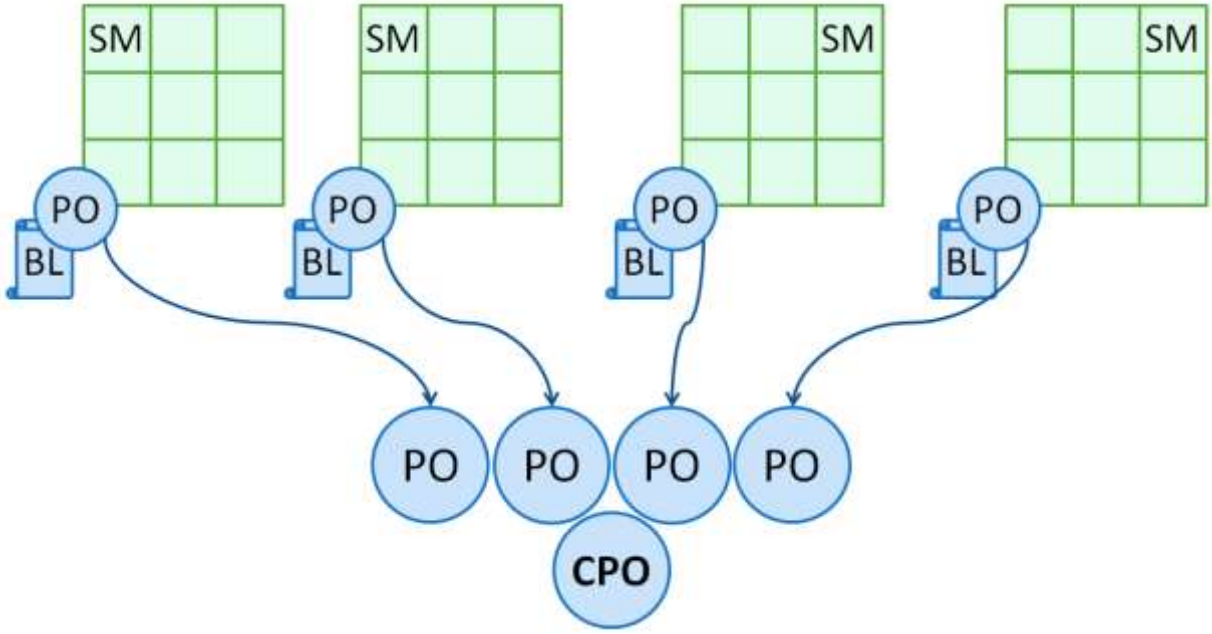
Обсуждение на данном уровне ведется в масштабе спринтов и релизов. Из проблем обсуждаются только самые крупные.



Очень удобно использовать данное мероприятие для финального обсуждения и принятия производственных стандартов.

Управление продуктами

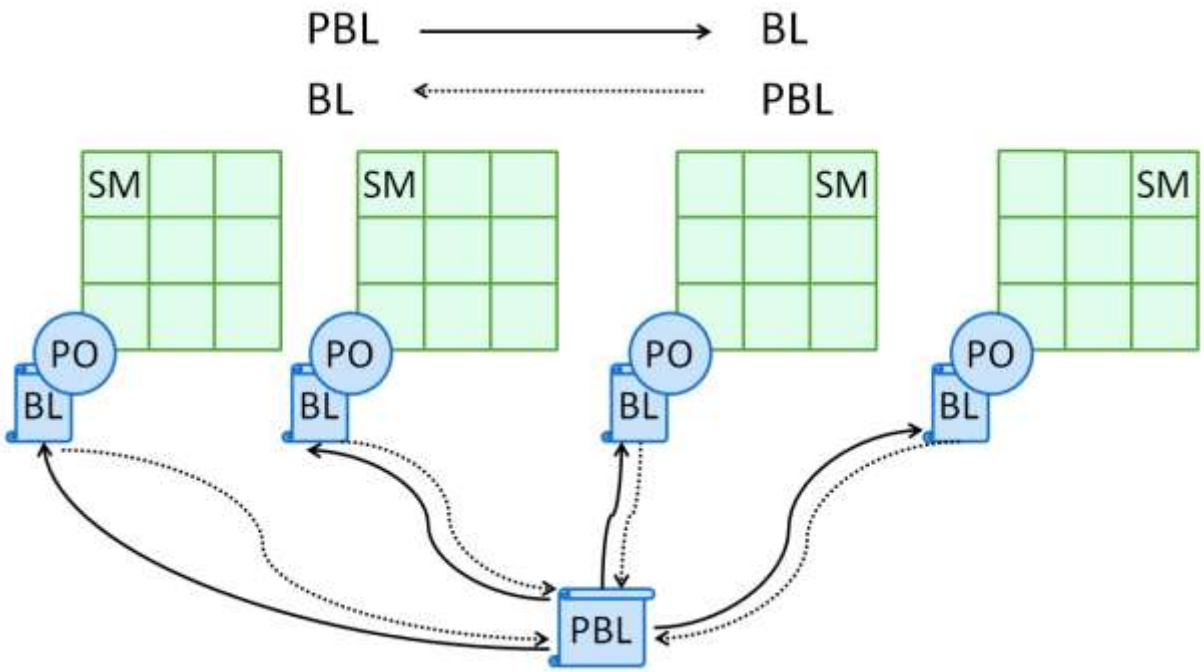
Для масштабирования деятельности продуктовых команд владельцы продуктов (Product Owner – PO) проводят Meta Scrum под руководством Chief Product Owner.



Обсуждение требований происходит на уровне беклогов (BL) спринтов, а не отдельных историй пользователей. Соответственно определяются и высокоуровневые приоритеты.

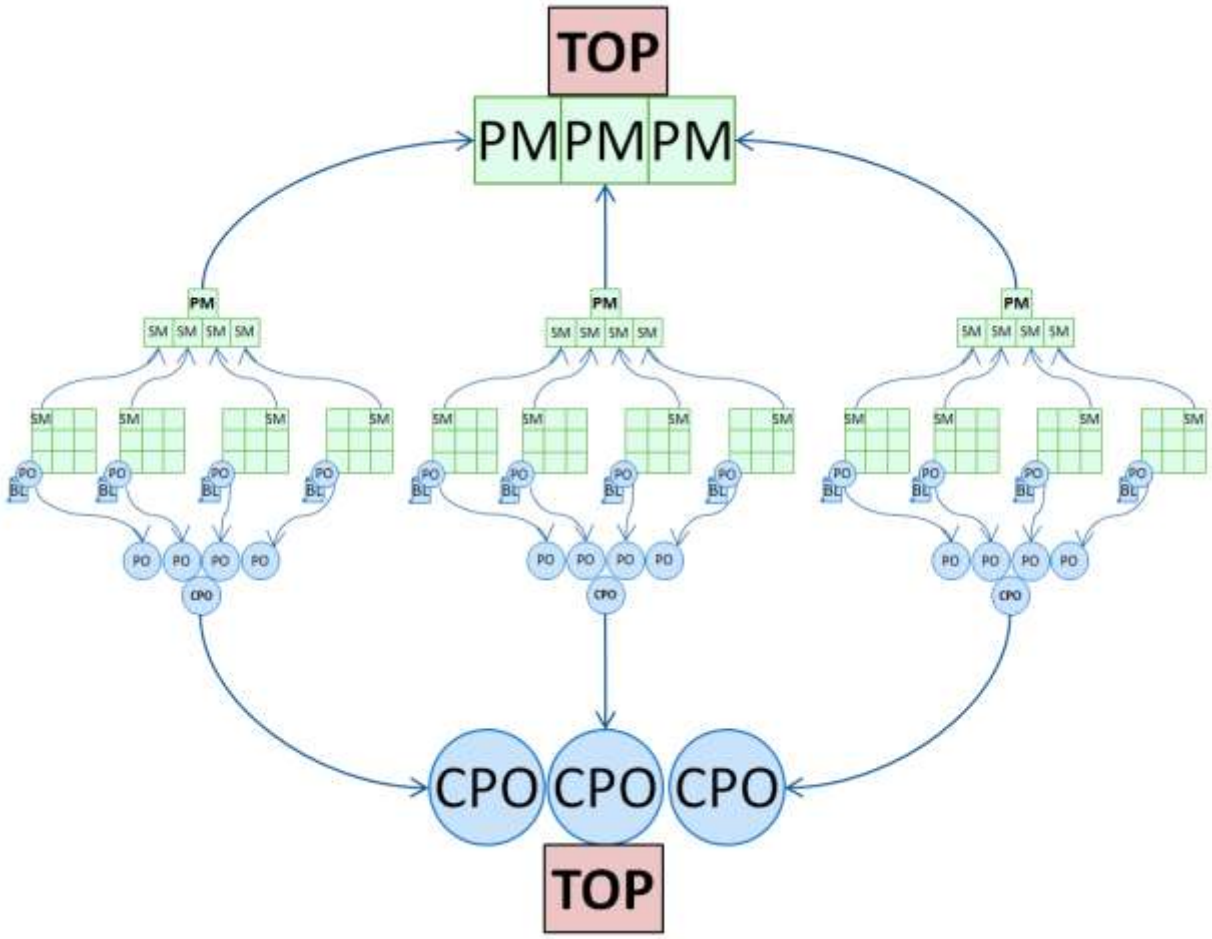
Также на данном уровне можно рассмотреть беклог программы (Program Backlog – PBL). Этот беклог может быть как реальным, так и виртуальным (также возможны сочетания обоих подходов). Реальный беклог программы получается при реализации большого

проекта несколькими командами, а виртуальный при реализации множества независимых проектов:



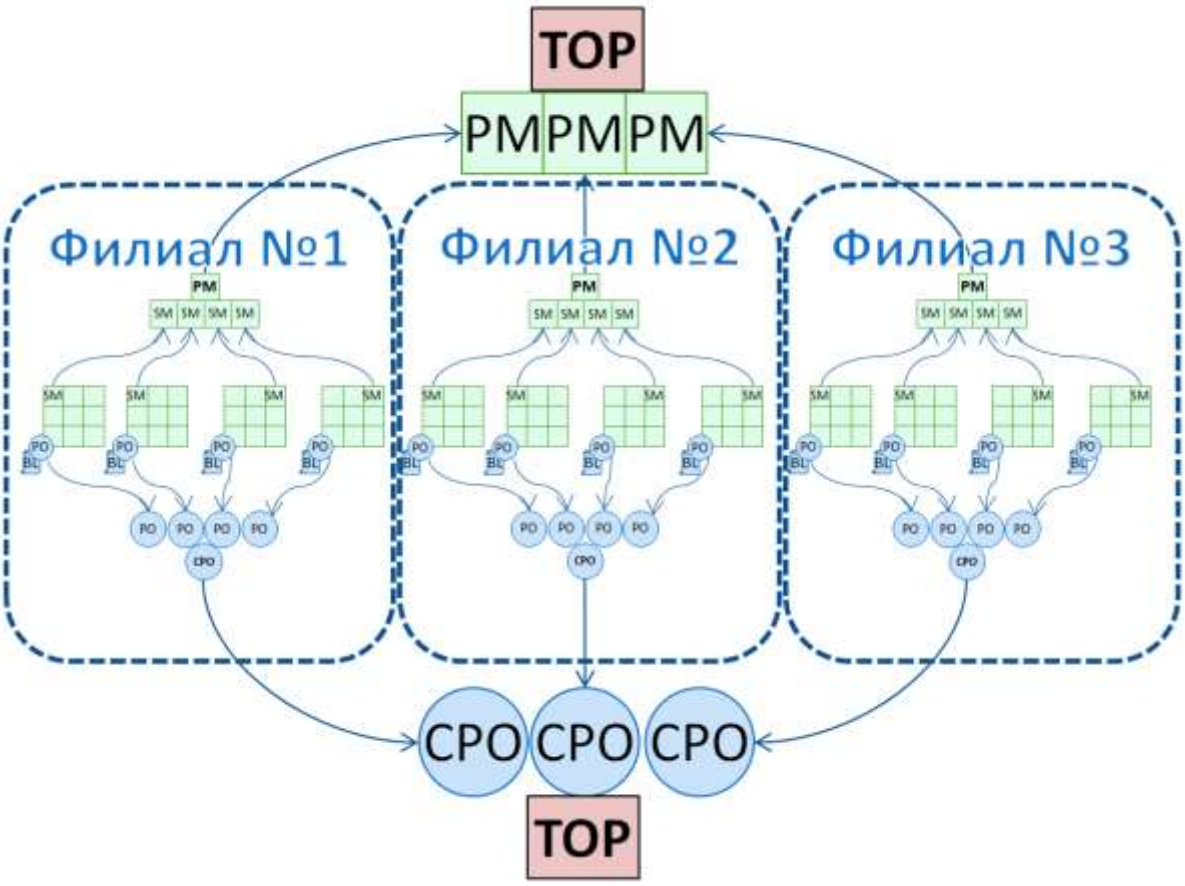
Scrum на уровне предприятия

Если объединить продуктовую и производственную схему масштабирования, то получим полноценную схему масштабирования Scrum на уровне предприятия:



Распределенный скрам

Стоит отметить, что вышеописанные схемы подходят для организации распределенной разработки, для этого необходимо ввести региональные дивизионы и закрепить за ними руководителя программы и главного владельца продукта:


TOP
PM PM PM
Филиал №1
Филиал №2
Филиал №3
PM
SM
PO
BL
CPO
CPO CPO CPO
TOP

12. Бережливое производство

Бережливое производство пришло к нам из США. Фактически бережливое производство – это осмысление производственной системы Тойоты американскими учеными. Бережливой такую систему организации труда называют, потому она нацелена на сокращение издержек в виде потерь производства (вообще, это достаточно узкий взгляд на данную систему).

Основателем производственной системы Тойоты был Тайити Оно, который еще в послевоенное время сделал первые попытки оптимизации производства. Его компания была вынуждена конкурировать с такими гигантами, как Ford, которые за счет эффекта масштаба производства имели достаточно низкую себестоимость произведенных автомобилей.

«Ценность» - основа бережливого производства

Чтобы понять концепцию бережливого производства, нужно представлять, что такое ценность для потребителя. Ценность – это те преимущества, которые потребитель получит от использования вашего продукта или услуги. Бережливое производство расценивает все, что не добавляет ценности продукту, потерями, которые на японском языке называются «му́да». Основой бережливого производства как раз и является устранение муды.

Виды потерь

Классически выделяются семь видов потерь:

1. перепроизводство;
2. ожидание;
3. ненужная транспортировка;
4. лишние этапы обработки;
5. лишние запасы;
6. ненужные перемещения;
7. дефекты.

В самой популярной книге о производственной системе Toyota «Дао Toyota» авторы обозначили восьмой вид потерь:

8. нереализованный творческий потенциал сотрудников

Также традиционно выделяют еще два вида потерь:

9. му́ри («перегрузка»)
10. му́ра («неравномерность»)

Инструменты бережливого производства

Бережливое производство ПО

Принципы бережливого производства можно применить и в разработке программного обеспечения:

- **Уменьшайте потери**
  Видов потерь в разработке программного обеспечения достаточно много: от переключения между задачами до реализации лишнего функционала. Помните правило 20/80: 20% функционала продукта приносят 80% ценности заказчику и именно гибкие методологии позволяют эти 20% функционала выявить и назначить им максимальную важность.
- **Фокусировка на обучении**
  Весь процесс создания программного обеспечения фактически состоит из обучения: с каждой новой функцией мы узнаем что-то новое. Поэтому, как и для любой другой инженерной деятельности, необходимо сосредотачивать на обучении.
- **Встроенное качество**
  Необходимость высокого качества продуктов никто не оспаривает, но редко, кто действительно вкладывает в это хотя бы чуточку усилий. В нашей индустрии есть устоявшиеся практики по контролю и обеспечению качества – это автоматические тесты и разработка в стиле TDD.
- **Позднее принятие решений**
  Очень важно принимать решения как можно позже. Самым ярким примером здесь является создание архитектуры приложения. В противоположность подходу Big Up Front Design, мы создаем архитектуру только для текущего функционала, следуя принципам KISS и YAGNI.
- **Быстрая поставка**
  Бережливое производство ПО предполагает максимально быстрые и частые поставки для получения обратной связи.
- **Уважение к людям**
  Люди – это основа любой мыслительной деятельности, к которой, несомненно, относится разработка ПО. Мы должны уважительно относиться к командам, предоставляя им максимум полномочий и ответственности.
- **Оптимизация системы целиком**
  При оптимизационных мероприятиях мы должны оптимизировать всю систему целиком, не занимаюсь субоптимизацией отдельных сегментов.

Производственная система Тойоты (Toyota Production System – TPS)

14 принципов TPS

1. Философия долгосрочной перспективы: можно пойти на убытки для достижения отдаленной цели.
2. Производственный поток должен быть непрерывным.

3. Канбан: производство по системе «точно вовремя» без промежуточных запасов.
4. Хейдзунка: равномерное распределение нагрузки на всех этапах технологического процесса.
5. Андон и дзидока: автоматическая остановка производства с целью решения проблем.
6. Формализация накопленных знаний: достигнутое нужно делать новым стандартом.
7. Визуальный контроль: иногда простая лампочка эффективнее компьютерного монитора.
8. Внедрять только проверенные технологии.
9. Воспитывать собственных лидеров, искренне исповедующих философию компании.
10. Формировать и воспитывать рабочие команды, в которых каждый искренне исповедует философию компании.
11. Уважать и развивать партнеров-поставщиков.
12. Генти гаубицу: перед тем как начать разбираться в ситуации, увидеть все своими глазами.
13. Немаваси: принимать коллективные решения только после согласия большинства, но внедрять — немедленно.
14. Хансей и кайзен: любой процесс можно постоянно анализировать и совершенствовать.

Кайзен

«Совершенствоваться не обязательно.
Выживание – дело добровольное»

Э. Деминг

Японское слово «кайзен» состоит из двух иероглифов, которые можно перевести как «изменения, направленные на улучшения»:

Кай Изменение 改善 Зен Улучшение

На практике кайзен – это стратегия постоянного улучшения путем небольших изменений. Для реализации такого подхода необходимо следовать принципам кайзена, которые отлично согласуются с гибкими методологиями:

- **Фокус на клиентах** — мы должны быть сфокусированы на клиентах, причем не только на словах, но и на деле;
- **Непрерывные изменения** — все аспекты нашей производственной системы должны постоянно улучшаться небольшими шагами, для чего мы будем активно обсуждать процессы на ретроспективах;

- **Открытое признание проблем** — мы должны честно и открыто признавать и обсуждать все проблемы не для поиска виноватых, а для оптимизации процессов;
- **Создание сообществ** — организация сообществ (не обязательно формальных) является важной частью оптимизации производства, потому что позволяет организовать обсуждения для инициативы «снизу»;
- **Управление проектами при помощи межфункциональных команд** — команды в Scrum максимально кроссфункциональны и включают в себя всех необходимых специалистов для реализации проекта;
- **Формирование «поддерживающих взаимоотношений»** — для организации таких взаимоотношений необходимо уделять внимание социальной атмосфере в коллективе;
- **Развитие самодисциплины** — команды в Scrum самоуправляемы и могут определять, как они будут достигать целей, поставленных владельцем продукта;
- **Информирование каждого сотрудника** — информация (в том числе коммерческая) должна быть максимально прозрачна для каждого сотрудника;
- **Делегирование полномочий каждому сотруднику** — чем больше полномочий (и ответственности) делегировано конкретным исполнителям, тем больше возможность появления креативных и прорывных решений, и тем больше времени у руководства для проработки стратегических решений.

Инструменты кайзена

Кроме философии и культуры, которые, безусловно, важны в долгосрочной перспективе, кайзен предлагает использовать ряд инструментов для оптимизации производственных процессов. В этом разделе мы рассмотрим наиболее часто применяемые.

Карта потока создания ценности

Карта потока создания ценности (Value Stream Mapping) – инструмент, который отображает стадии производственного процесса и время между ними. Затем производится подсчет эффективности процесса, как частное от полезного времени, когда добавлялась ценность продукту, и общего времени работы процесса:

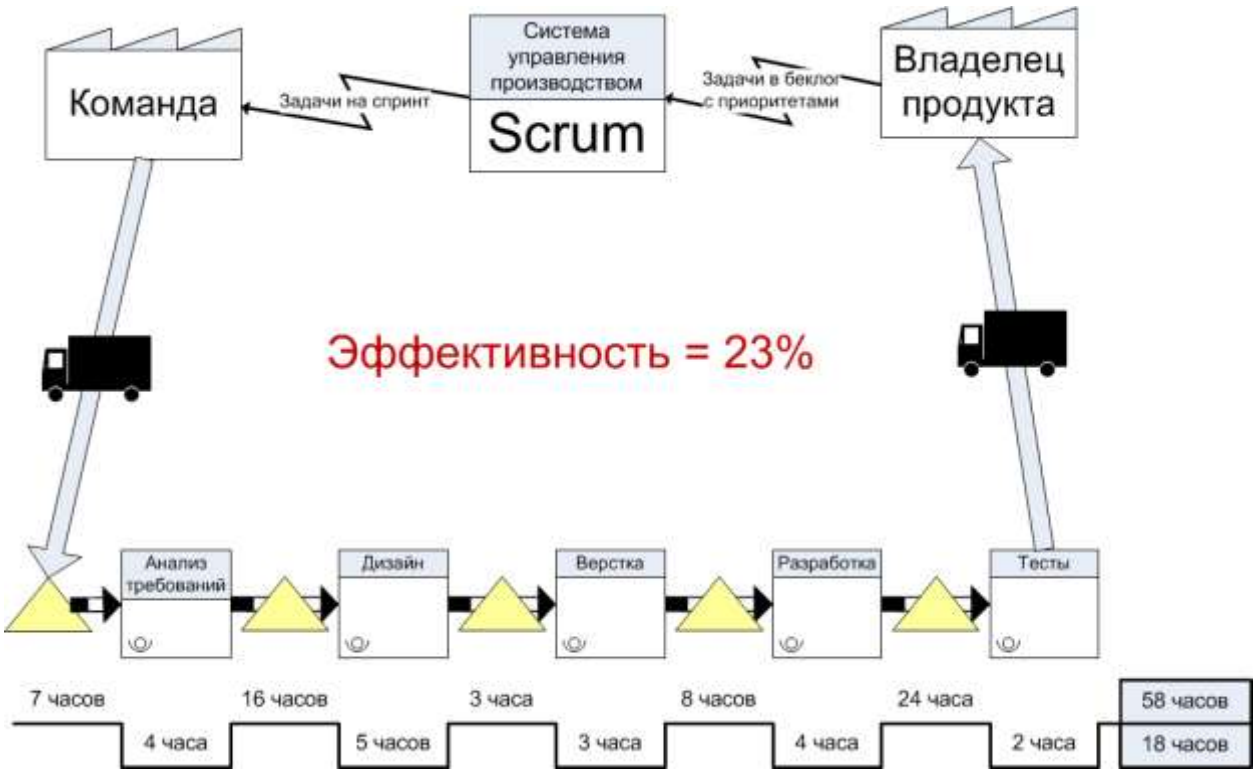


Рисунок 39. Пример карты потока создания стоимости

После этого процесс перестраивается с целью увеличить его эффективность. В рамках Scrum такой анализ имеет смысл проводить не по отдельным историям пользователям, а по эпика или темам.

«Пять почему»

Самый простой инструмент, который может применяться даже в устной форме. Его суть в поиске коренных причин проблем (например, дефектов) и принятие многоуровневых контрмер. Для этого по отношению к проблеме задается пять раз вопрос «Почему», чтобы проникнуть в ее суть:

Уровень	Симптом	Действие
1	Сайт не работает	– Оповестить команду
2	На сайте выдается сообщение об ошибке подключения к БД	– Проверить все ли в порядке с БД
3	В конфигурационном файле прописана тестовая БД	– Проверять работоспособность сайта после деплоймента; – Сделать автоматические тесты для быстрой проверки;

4	Разработчик забыл поменять конфигурационный при деплойменте	– Добавить в стандарт деплоймента проверку конфигурационных файлов; – Проинструктировать разработчиков по порядку выноса сайтов;
5	Недостаточная внимательность разработчика	– Заменить ручную смену конфигурационного файла на автоматическое определение окружения и выставления соответствующей БД;

Диаграммы причинно-следственной связи

Такие диаграммы является уже более тяжеловесным инструментом по сравнению с «Пять почему». На них уже отображается множество проблем и их связи, но цель та же – выявление коренных причин. В качестве нотации можно посоветовать упрощенную нотацию, предложенную Хенриком Книбергом (Книберг, 2009):

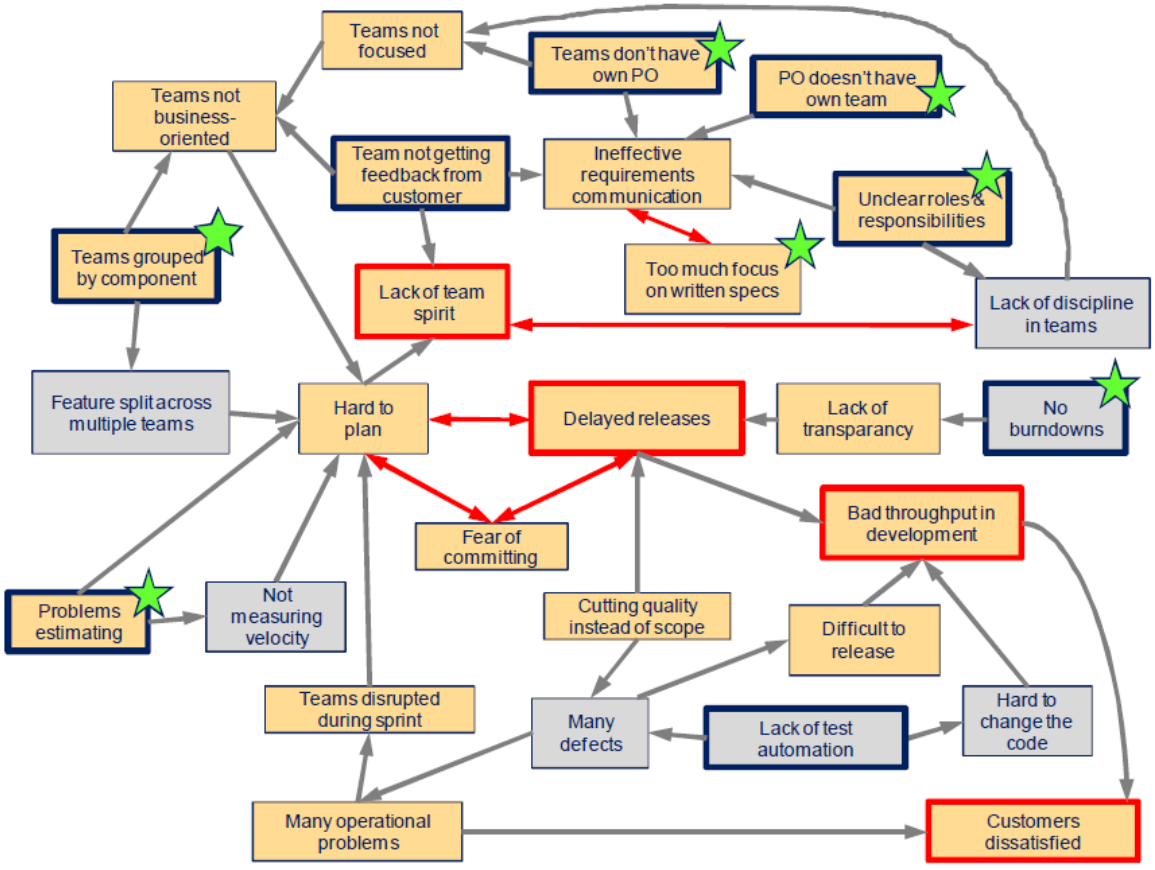


Рисунок 40. Пример нотации для анализа причинно-следственной связи

Диаграммы Исикавы

Еще одним инструментом для анализа проблем, но уже на уровне организации в целом является диаграмма Исикавы. На ней также отображаются проблемы, но они заранее сгруппированы по нескольким областям, например:

- Методология
- Требования
- Разработка
- Контроль качества

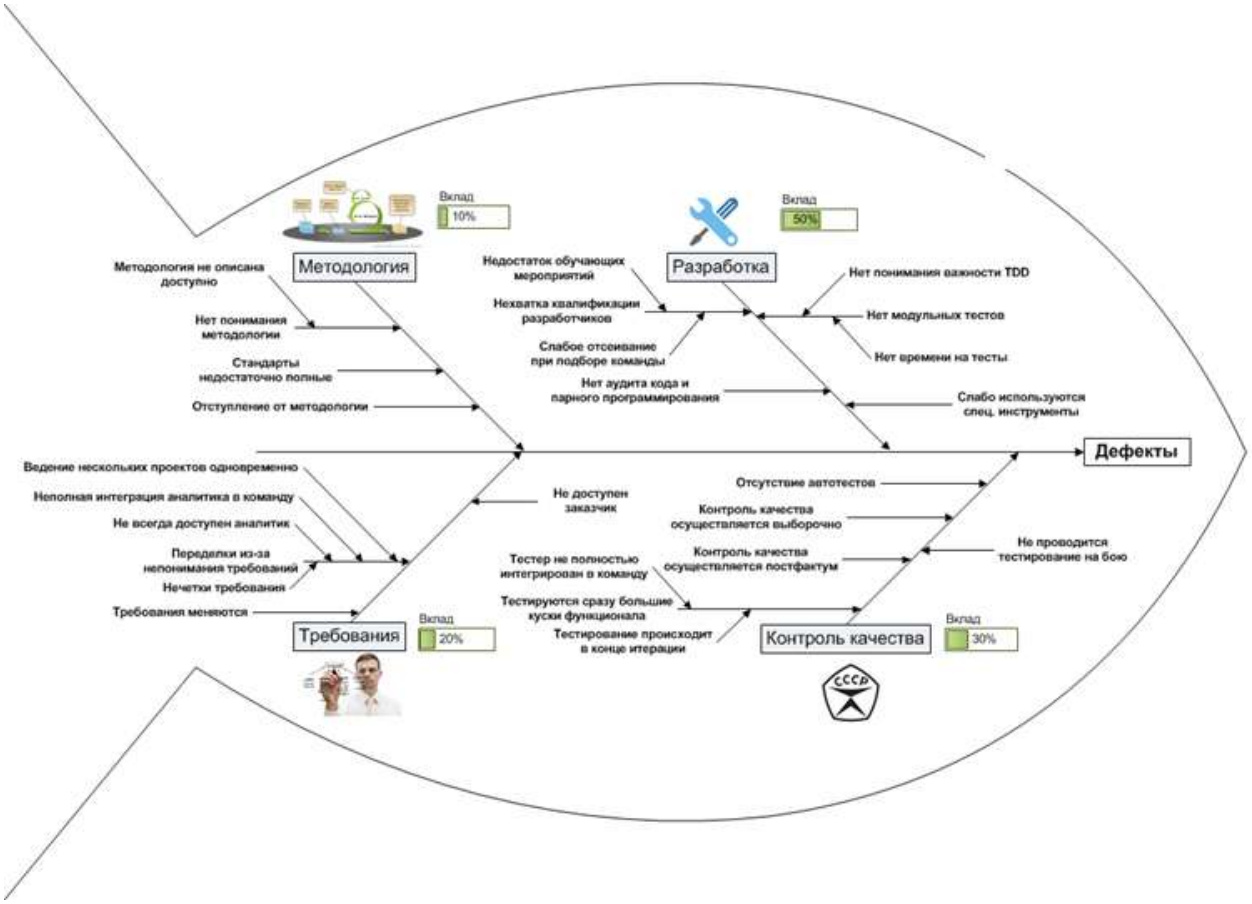
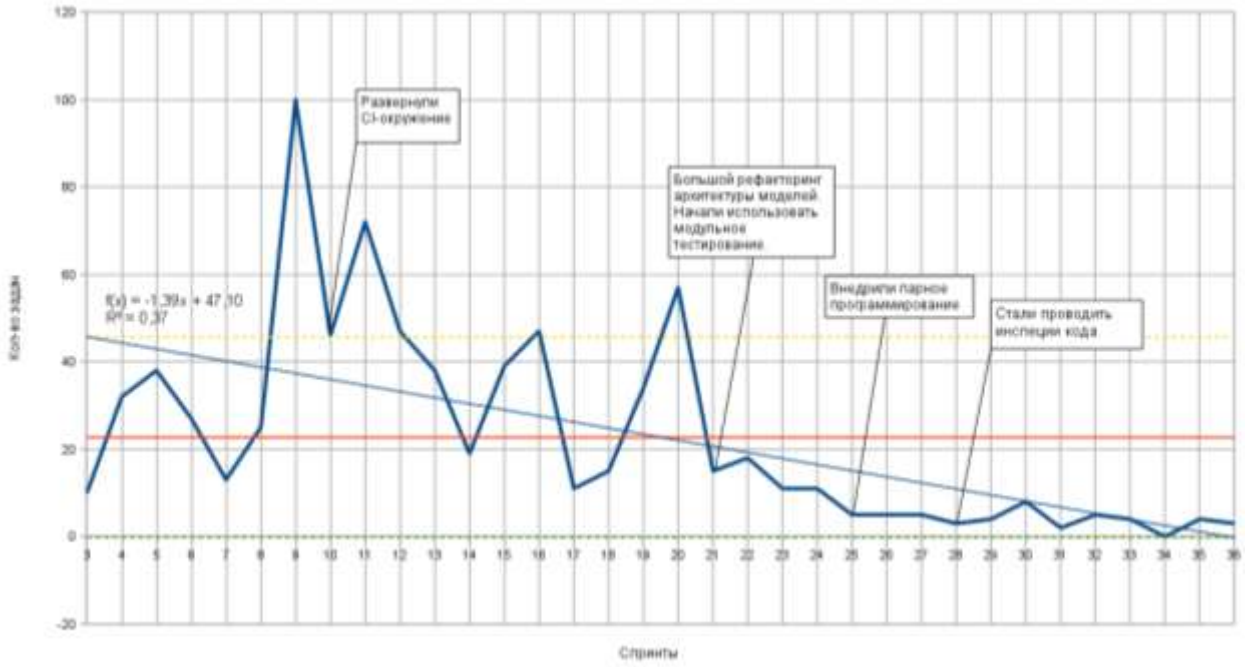


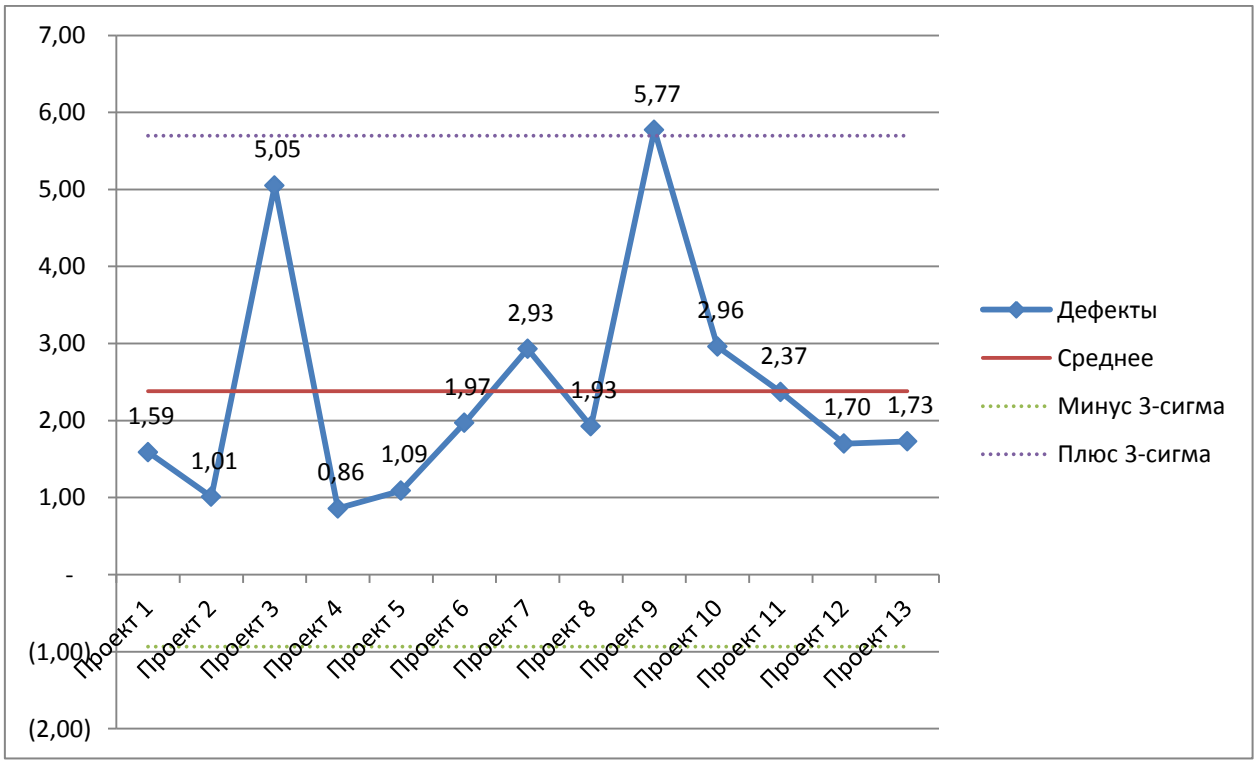
Рисунок 41. Пример диаграммы Исикавы

Контрольные карты

Этот инструмент используется для выявления несистематических отклонений и устранения вариаций. Для этого собираются данные и упорядочиваются, например, по времени. Затем вычисляется среднее значение и стандартное отклонение от него. В качестве примеру покажу контрольную карту по дефектам, которая показывает пользу от внедрения инженерных практик:



Классическим применением является сравнение некоторых элементов системы по выбранному показателю. Например, мы хотим определить, насколько наши проекты хорошо тестируются. Для этого можно собрать простую статистику: количество дефектов в финальной версии продуктов в равных временных промежутках на одного разработчика и построить по ним контрольную карту:



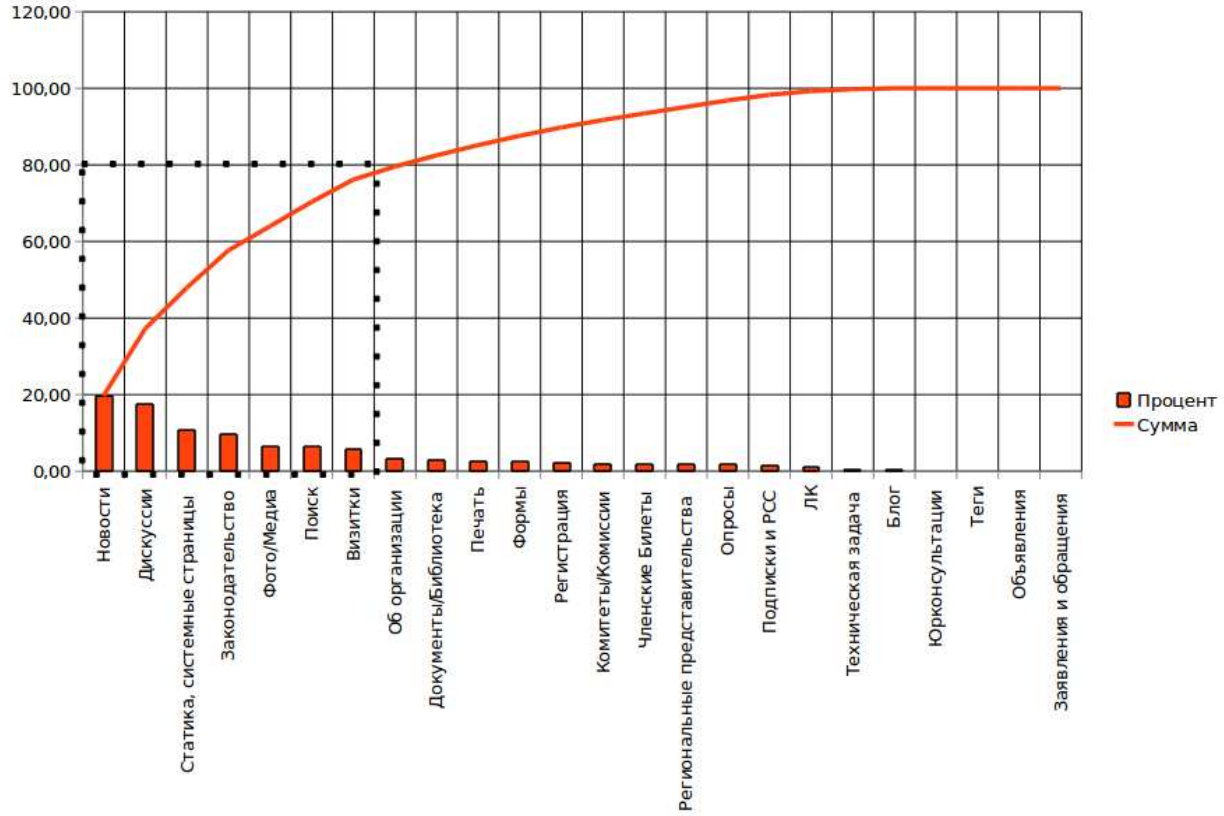
По ней видно, что проект №9 имеют несистемные отклонения. Если количество дефектов больше, чем три стандартных отклонения, то необходимо выработать ряд мер, прежде всего, процессных для улучшения качества.

Контрольные карты бывают разных видов, например, классические контрольные карты Шухарта (см. (Госстандарт России, 1999) и (Уилер , и др., 2009)). Различия касаются формул и значений для выбора среднего значения и контрольных пределов. В контрольных картах Шухарта для определения предела используется не стандартное отклонение, а размах. Также существует ряд правил для трактовки контрольных карт, например, правила Western Electric и правила Нельсона.

Отдельно подчеркну, что нельзя использовать такие карты напрямую для оценки сотрудников: большая часть проблем относиться к процессам и сотрудники часто просто не могут на эти причины повлиять.

Диаграмма Парето

Этот инструмент позволяет выявить модули, которые содержат определенный процент дефектов. Обычно получается соотношение близкое к 20/80: 20% модулей содержат 80% дефектов:



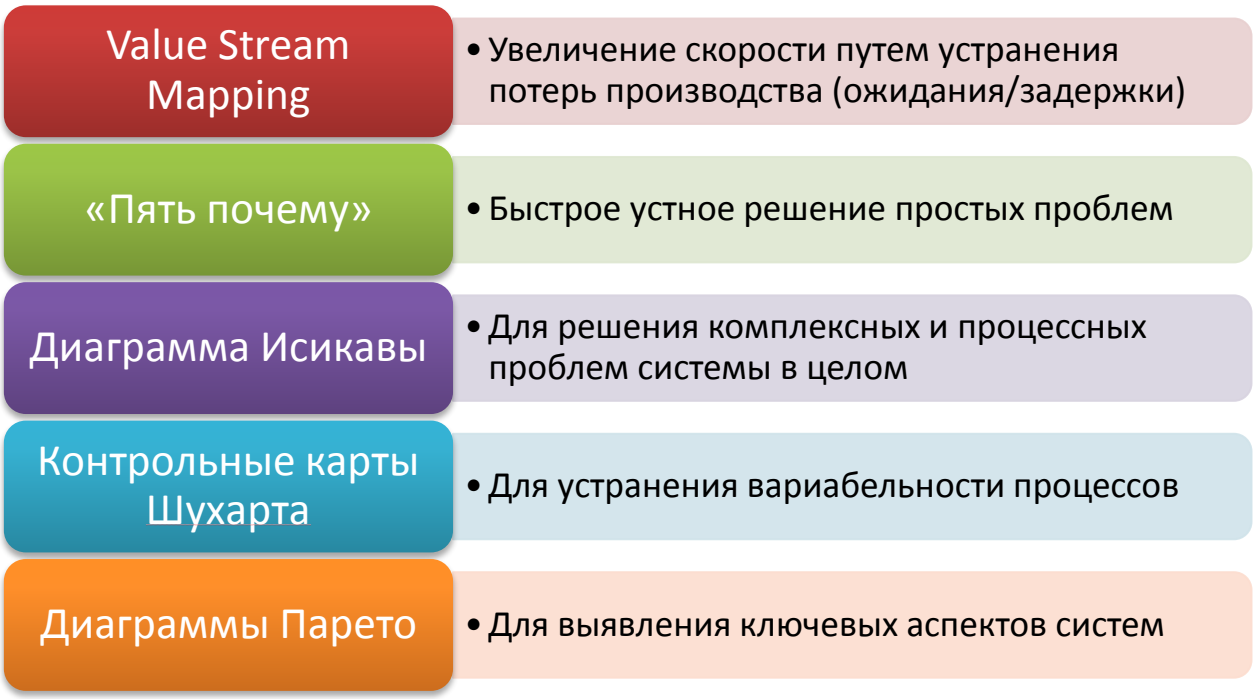
Именно на эти модули стоит обратить внимание:

- покрыть их дополнительно тестами;
- провести дополнительные инспекции кода.

Диаграмму Парето можно использовать для анализа модулей по дефектам, но можно этим и не ограничиваться: аналогичный анализ можно провести по количеству вносимых изменений. Модули, в которые вносятся часто изменение, экономически выгодно сделать более гибкими:

– сделать более гибкую и настраиваемую архитектуру;
– активно использовать шаблоны проектирования для дополнительной гибкости.

Итоги



13. Как внедрить Agile за 14 недель

Введение

Основная цель составления данного плана по внедрению Agile: дать четкую и краткую инструкцию по трансформации компании/подразделения в гибкую и эффектную бизнес-единицу по производству программного обеспечения.

План рассчитан на небольшие компании размером примерно от **20 до 50 человек** и нуждается в адаптации под конкретную компанию. Для компаний (или отдельных команд) меньшего размера можно использовать этот же план, за исключением масштабирования методологий.

Будем считать, что в компании по исторически сложившимся обстоятельствам используется «методология» Code&Fix. В качестве допущений будем использовать следующие положения

- длина спринта – 2 недели,
- длина релиза фиксирована – 3 итерации,
- внедрение Agile поддерживается руководством;

План внедрения рассчитан на то, чтобы серьезно не отрывать команды от производства и сделать внедрение управленческих и технологических инноваций частью корпоративной культуры компании.

Предполагается, что организацией внедрения будет заниматься один человек, работая полный день над этой задачей. Это может быть как внешний тренер/консультант, так и внутренний эксперт-внедренец. Для выбора внешнего тренера/консультанта можно воспользоваться главой «Консалтинг-компании и независимые тренеры».

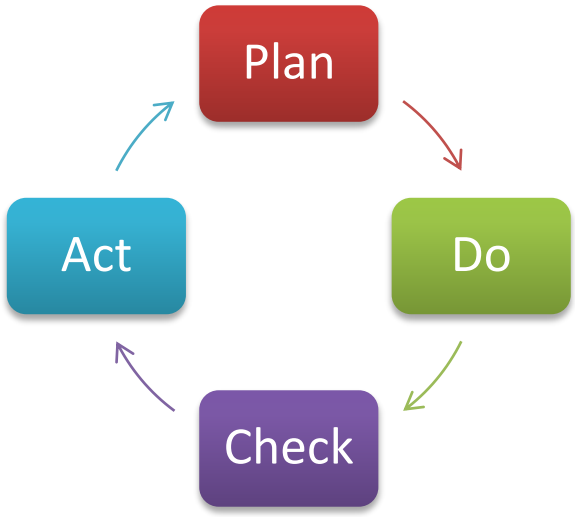
При выборе компании или тренера рекомендуется обратить внимание на следующие факторы:

1. **Реальный практический опыт работы по теме тренинга**
   Самым важным параметром при выборе компании и тренера является наличие практического опыта работы по теме тренинга. Дело в том что, «тренеры-теоретики» просто не способны понять многие вещи в силу отсутствия соответствующего опыта, даже при наличии всех необходимых сертификатов.
2. **Опыт консалтинговой деятельности и проведения тренингов**
   Одного опыта наличия практических навыков недостаточно, важно, чтобы у компании или тренера имелся определённый багаж знаний и навыков по консалтингу и внедрению соответствующих практик.

Принципы внедрения

Цикл Деминга (PDCA-цикл)

При организационных изменениях очень помогает использование здравого смысла и научного подхода. Традиционным методом в данном случае является цикл Деминга, который состоит из 4 шагов:



1. **Plan (планирование)**
   Производится анализ системы и вырабатываются возможные подходы к улучшениям и определяются желаемые результаты
2. **Do (исполнение)**
   Решения, выработанные на предыдущем шаге, реализуются.
3. **Check (проверка)**
   Производится анализ, полученных результатов, на предыдущем шаге.
4. **Act (корректировка)**
   Выполняются корректирующие действия, для уменьшения отклонений от плана.

ShuHaRi

Внедрение методологии и практик можно разбить на три этапа. Важно, чтобы компания и отдельные команды их прошли, не застряв на одном из них. Названия:

1. Shu (守:しゅ – «защита», «подчинение») — изучение традиционной мудрости — изучение методологии, работа строго по книжкам, руководствуясь предписаниями тренера/внедренца.
2. На (破:は – «отделение», «отклонение») — отступление от традиции — понимание методологии на очень глубоком уровне и ее адаптация под требования проектов/бизнеса/внешней среды
3. Ri (離:り – «покидание», «отделение») — превосходство над традицией — осознанное отступление от методологии, например переход со Scrum на Scrumban.

Важно пройти все этапы, не перепрыгивая их: достаточно стандартная ситуация, когда команда не может делать Scrum и сразу перепрыгивает на канбан, что в итоге выливается в классический Code&Fix.

График и содержание внедрения

План состоит из трех частей:

1. **Подготовка компании к трансформации:** сбор и анализ информации, получение знаний и навыков сотрудниками компании.

2. **Первый релиз:** знакомство с основными элементами Scrum и Lean
3. **Второй релиз:** адаптация Agile к бизнесу компании

Неделя №1 (подготовка к трансформации)

Цели: собрать и проанализировать основную информацию о компании, дать основным участникам базовые знания об Agile

1. Изучение и описание текущих бизнес-процессов компании
   1.1. Составление карты бизнес-процессов, касающихся разработки ПО/веб-сайтов (в графическом или текстовом виде)
2. Изучение проектов и организация их в портфель проектов
   2.1. Составление списка с проектов
   2.2. Разработка методологии приоритезации, принятия решений о запуске/завершения проектов
   2.3. Приоритезация и балансировка портфеля проектов
3. Буткемп по основам Scrum (однодневный тренинг по основам скрама с деловыми играми)
   3.1. Каждый участник тренинга должен понимать роли, процессы и артефакты Scrum
4. Продвинутое обучение скрам-мастеров (4-х часовой тренинг)
   4.1. Скрам-мастера должны получить дополнительные знания и навыки по процессам Scrum, навыки фасилитации и организации работы команд.
5. Продвинутое обучение владельцев продуктов (4 часовой тренинг)
   5.1. Владельцы продуктов должны получить дополнительные знания и навыки по управлению продуктами (выявление ролей пользователей, проведение сторимаппинга (story mapping), управление беклогом (backlog), управление релизами)

Неделя №2 (нулевой спринт)

Цели: выработать понимание продукта и создать высокоуровневую архитектуру

1. Исследование продукта
   1.1. Выявление ролей и персонажей по проектам
   1.2. Сторимаппинг (story mapping)
   1.3. Прототипирование основных интерфейсов
   1.4. Сессия для выявления основных рисков и выработки контрмер
2. Создание высокоуровневой архитектуры продукта
   2.1. Выбор платформы реализации
   2.2. Диаграмма предметной области / высокоуровневая диаграмма классов

Неделя №3 (старт первого «калибровочного» спринта)

Цели: отработать процессы по запуску спринта и проведению Scrum of Scrum

1. Старт первого спринта с командами
   1.1. Проведение планирования спринта и разбиение user story (юзер-стори) на задачи
   1.2. Проведение покер-планирования для оценки user story

2. Scrum of Scrum
    2.1. Определение сроков проведения Scrum of Scrum
    2.2. Проведение первого Scrum of Scrum
    2.3. Отработка механизма эскалации проблем
    2.4. Отработка механизма синхронизации деятельности команд

Неделя №4 (завершение первого «калибровочного» спринта)

Цели: отработать завершение спринта и провести ретроспективу на основе качественных показателей

1. Проведение демонстрации и получение обратной связи
2. Ретроспектива (что было сделано хорошо, что было сделано плохо, список улучшений)
    2.1. Определение скорости команды эмпирическим путем

Неделя №5 (старт второго спринта)

Цели: отработать старт спринта и планирования на основе количественных показателей, начать внедрение базовых практик экстремального программирования

1. Планирование и старт второго спринта
    1.1. Планируем, исходя из скорости предыдущего спринта
2. Тренинг и мастер-класс по практикам экстремального программирования
    2.1. Внедрение системы непрерывной интеграции: полная сборка продукта происходит автоматически и непрерывно
    2.2. Выработка и внедрение стандартов кодирования

Неделя №6 (завершение второго спринта)

Цели: отработать завершение спринта и провести ретроспективу на основе количественных показателей, использую инструменты бережливого производства

1. Изучение практики инструментов бережливого производства(Lean)
    1.1. Виды потерь при производстве
    1.2. Value Stream Mapping для текущего процесса
    1.3. «5 почему»
2. Демонстрация
3. Ретроспектива с применением инструментов бережливого производства
    3.1. Разбор причин опоздания по несделанным задачам
    3.2. «5 почему» по каждому дефекту

Неделя №7 (старт третьего спринта)

Цели: отработать старт предрелизного спринта и понять, как в будущем избежать таких «стабилизационных» спринтов, начать активно использовать автоматизированное тестирование

1. Планирование и старт третьего спринта

    1.1. Особое внимание уделяем недоделанным user stories, которые не успели сделать из-за ограничения по скорости команды
    1.2. Рассматриваем возможность снизить скорость команды, чтобы успеть всё к релизу
2. Внедрение модульных и приемочных тестов
    2.1. Проведения тренинга по приемочным тестам
    2.2. Покрытие 5% основного бизнес-функционала продукта приемочными тестами
    2.3. Проведения тренинга по модульным тестам
    2.4. Покрытие 50% кода, реализованного за спринт, модульными тестами
3. Внедрение рефакторинга

Неделя №8 (завершение третьего спринта)

Цели: сделать первый Agile-релиз продукта и выработать значительные меры по улучшению процессов на основе информации, полученной за три спринта.

1. Кайзен-сессия на ретроспективе
    1.1. Диаграмма Исикавы по глобальным проблемам проекта и выработка мер по устранению проблем
2. Завершение третьего спринта
    2.1. Первый релиз продукта: обязательно, чтобы его попробовали конечные пользователи и предоставили обратную связь.
3. Post-mortem релиза в рамках ретроспективы

Неделя №9 (старт четвертого спринта)

Цели: научиться планировать и управлять релизом

1. Планирование релиза
    1.1. Начало ведения диаграммы burndown-а релиза
    1.2. Отбор владельцем продукта user story для релиза
    1.3. Возможная переоценка беклога командой «на пальцах»
2. Внедрение (трех)четырехзвенной архитектуры
3. Планирование и старт четвертого спринта
    3.1. Скорость команды считаем эмпирически по трем предыдущим спринтам

Неделя №10 (завершение четвертого спринта)

Цели: внедрение статистического управления качеством

1. Завершение четвертого спринта
2. Внедрение основ статистического управления качеством
    2.1. Статистика по дефектам
    2.2. Диаграмма Парето по модулям
    2.3. Контрольные карты Шухарта

Неделя №11(старт пятого спринта)

Цели: внедрение канбана для команды поддержки

1. Планирование и старт пятого спринта

    1.1. Анализируем и изменяем scope по диаграмме сгорания релиза
2. Переход на Scrumban команды поддержки
    2.1. Тренинг по канбана (4 часа) для членов команды
    2.2. Отказ от жестких итераций
3. Внедрение разработки через тестирование
    3.1. Тренинг и мастер-класс по разработке через тестирование
    3.2. Покрытие тестами модулей ядра системы (не менее 50% строк кода)

Неделя №12 (завершение пятого спринта)

Цели: улучшение внутреннего качества ядра системы

1. Частичный рефакторинг модулей ядра системы
    1.1. Определение стратегии рефакторинга и выбор модулей
2. Завершение пятого спринта

Неделя №13 (старт шестого «идеального» спринта)

Цели: запуск идеального спринта

1. Планирование и старт шестого спринта
    1.1. Анализируем и изменяем scope по диаграмме сгорания релиза

Неделя №14 (завершение «идеального» шестого спринта)

Цели: завершение идеального спринта

1. Завершение шестого спринта
2. Релиз продукта
3. Post-mortem релиза в рамках ретроспективы
    3.2. Анализ релиз-бёрндауна

14. Гибкие компании-аутсорсеры

В этот список включены компании-аутсорсеры, которые работают по гибким методологиям, описанным в этой книге.

Название	Специализация	Кол-во сотрудников	Контакты
IndyCode	Автоматизация бизнес-процессов Электронный документооборот CRM- и ERP-системы	20	http://indycode.net Тел.: +7 (351) 281 0071 info@indycode.ru
Softline	Веб-разработка Поисковые решения	100	http://services.softline.ru/webdev Тел.: +7(495) 232-0023 * 0443 E-mail: webdev@softline.ru

Чтобы попасть в эту табличку свяжитесь с автором книги

15. Консалтинг-компании и независимые тренеры

Название	Специализация	Контакты
ScrumGuides	Сертификация скрам-мастеров Сертификация владельцев продуктов Инженерные практики Архитектура	http://scrumguides.com info@scrumguides.com Тел.: 050 358-9212
ScrumTrek	Сертификация скрам-мастеров Сертификация владельцев продуктов Управление требованиями Lean Инженерные практики	http://scrumtrek.ru info@scrumtrek.ru Тел.: +7 (495) 991 69 20
SkillTrek	Инженерные практики Архитектура Scrum GTD	http://skilltrek.ru info@scrumtrek.ru Тел.: +7 (495) 991 69 20
The Improved Methods	Scrum Управление требованиями Лидерство в командах Личный тайм-менеджмент Управление командами	http://tim.com.ua info@tim.com.ua Тел.: +380 (44) 22 88 934
XPInjection	Инженерные практики TDD в PHP/Java Канбан QA в Agile Управление рисками	http://xpinjection.com anna.alimenkova@xpinjection.com Тел: +380 (68) 46 111 56

Независимые тренеры и консультанты

Фамилия и имя	Специализация	Контакты
Бындю Александр 	Scrum Инженерные практики Принципы проектирования Domain Driven Design (DDD)	www.byndyu.ru Тел.: +7 (904) 305 5263 alexander.byndyu@gmail.com Twitter: alexanderbyndyu Skype: alexander.byndyu

Вольфсон Борис 	Scrum Управление продуктом Управление рисками	http://borisvolfson.moikrug.ru borisvolfson@gmail.com Twitter: borisvolfson Skype: borisvolfson

Чтобы попасть в эту табличку свяжитесь с автором книги

16. Предметный указатель

17. Список литературы

A Practical Guide to Feature-Driven Development [Книга] / авт. Stephen Palmer John Felsing.

A Practical Guide to Seven Agile Methodologies, Part 2 [В Интернете] / авт. Rod Coffin Derek Lane. - http://www.devx.com/architect/article/32836/1954.

Advanced Topics in Agile Planning [В Интернете] / авт. Cohn Mike. - http://www.mountaingoatsoftware.com/system/presentation/file/132/Advanced-Topics-Agile-Planning-Cohn-NDC2010.pdf?1276713148.

Agile Development with the ICONIX Process: People, Process and Pragmatism [Книга] / авт. Rosenberg Doug, Stephens Matt и Collins-Cop Mark. - 2005.

Agile Metrics / авт. Алименков Николай.

Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process [Книга] / авт. Ambler Scott W.. - 2002.

Agile Project Management with Scrum [Книга] / авт. Schwaber Ken.

Agile Retrospectives: Making Good Teams Great [Книга] / авт. Esther Derby Diana Larsen.

Agile Software Development with Scrum (Series in Agile Software Development) [Книга] / авт. Ken Schwaber Mike Beedle. - 2001.

Cause-effect diagrams [Книга] / авт. Книберг Хенрик. - 2009.

Crystal Clear – Alistair Cockburn (2005) [В Интернете] / авт. Björkholm Tomas. - http://www.crisp.se/rd/Crystal_Clear.pdf.

Crystal Clear: A Human-Powered Methodology for Small Teams [Книга] / авт. Cockburn Alistair. - 2004.

DSDM: Business Focused Development [Книга] / авт. DSDM Consortium.

Kano Model - How to delight your customers [Online] / auth. Phillips Lawrence (Laurie). - http://www.slideshare.net/LawrencePhillips/kano-model-rev-1.

Leading a Self-Organizing Team [В Интернете] / авт. Cohn Mike. - 2011 г.. - http://www.mountaingoatsoftware.com/presentations/142-leading-a-selforganizing-team.

Prioritizing Your Product Backlog [Online] / auth. Cohn Mike. - http://www.mountaingoatsoftware.com/system/presentation/file/127/Prioritizing-Product-Backlog-Cohn-ADP2010.pdf.

QA in Agile [В Интернете] / авт. Алименков Николай. - 2008 г.. - http://www.slideshare.net/alimenkou/qa-in-agile.

Retrospectives [В Интернете] / авт. Дмитриев Сергей. - 2011 г.. - http://www.slideshare.net/Blackie6/retrospectives-agiledays-2011.

Scaling Agile to Work with Distributed Teams [Online] / auth. Cohn Mike. - http://www.mountaingoatsoftware.com/system/presentation/file/133/Scaling-Distributed-Agile-Cohn-NDC2010.pdf.

Scrum и Kanban: выжимаем максимум [Книга] / авт. Книберг Хенрик и Скарин Матиас.

Scrum и XP: заметки с передовой [Книга] / авт. Книберг Хенрик.

Scrum. Гибкая разработка ПО [Книга] / авт. Кон Майк. - 2011.

Small Hyper-Productive Teams [В Интернете] / авт. Алименков Николай. - 2011 г.. - http://www.slideshare.net/alimenkou/small-hyper-productive-teams-it-brunch-10150627.

Test Driven Development for Embedded C [Книга] / авт. Grenning James.

The Enterprise and Scrum [Книга] / авт. Schwaber Ken.

The Pragmatic Programmer: From Journeyman to Master [Book] / auth. Andrew Hunt David Thomas. - 1999.

Use Case Driven Object Modeling with UML: Theory and Practice [Книга] / авт. Rosenberg Doug и Stephens Matt. - 2007.

User Stories [В Интернете] / авт. Cohn Mike. - http://www.mountaingoatsoftware.com/system/presentation/file/130/User-Stories-Cohn-NDC2010.pdf?1276712524.

User Stories Applied: For Agile Software Development [Книга] / авт. Cohn Mike.

Vision Crafting [В Интернете] / авт. Филиппов Никита и Лобасев Дмитрий. - 2011 г.. - http://www.slideshare.net/Nfilippov/2-bmg.

Бережливое производство программного обеспечения. От идеи до прибыли [Книга] / авт. Мэри Поппендик Том Поппендик. - 2010.

Бережливое производство. Как избавиться от потерь и добиться процветания вашей компании [Книга] / авт. Джеймс Вумек Дэниел Джонс.

Вальсируя с Медведями: управление рисками в проектах по разработке программного обеспечения [Книга] / авт. Том ДеМарко Тимоти Листер.

Выход из кризиса. Новая парадигма управления людьми, системами и процессами [Книга] / авт. Деминг Эдвардс.

Гибкое тестирование. Практическое руководство для тестировщиков ПО и гибких команд [Книга] / авт. Лайза Криспин Джанет Грегори.

ГОСТ Р 50779.42-99 (ИСО 8258-91) [Книга] / авт. Госстандарт России. - Москва : [б.н.], 1999.

Кано-взвешивание [В Интернете] / авт. Дмитриев Сергей. - http://www.slideshare.net/Blackie6/ss-7654382.

Машина, которая изменила мир [Книга] / авт. Вумек Джеймс и Джонс Даниел. - Москва : Попурри, 2007.

Подвижная мишень и дрожащие руки [В Интернете] / авт. Дорофеев Максим. - 2010 г.. - http://www.slideshare.net/Cartmendum/hitting-moving-target.

Построение бизнес-моделей. Настольная книга стратега и новатора [Книга] / авт. Александр Остервальдер Ив Пинье.

Практика дао Toyota. Руководство по внедрению принципов менеджмента Toyota [Книга] / авт. Майер жеффри Лайкер и Дэвид.

Практическое руководство по эКстремальному программированию [Книга] / авт. Дэвид Астелс Гранвилл Миллер, Мирослав Новак.

Рефакторинг баз данных. Эволюционное проектирование [Книга] / авт. Скотт В. Эмблер Прамодкумар Дж. Садаладж. - [б.м.] : Вильямс.

Рефакторинг с использованием шаблонов [Книга] / авт. Кериевски Джошуа. - [б.м.] : Вильямс.

Рефакторинг. Улучшение существующего кода [Книга] / авт. Фаулер Мартин. - 2009.

Статистическое управление процессами. Оптимизация бизнеса с использованием контрольных карт Шухарта [Книга] / авт. Уилер Дональд и Дэвид Чемберс. - Москва : Альпина Бизнес Букс, 2009.

Цель. Процесс непрерывного совершенствования [Книга] / авт. Элияху М. Голдрат Джефф Кокс.

Цель-2. Дело не в везении [Книга] / авт. Голдратт Элияху.

Человеческий фактор. Успешные проекты и команды [Книга] / авт. Листер Том Демарко и Тимоти.

Экстремальное программирование [Книга] / авт. Бек Кент.

Экстремальное программирование: планирование [Книга] / авт. Кент Бек Мартин Фаулер.

Экстремальное программирование: постановка процесса. С первых шагов и до победного конца [Книга] / авт. Кент Ауэр Рой Миллер.

Экстремальное программирование: разработка через тестирование [Книга] / авт. Бек Кент.

Эффективная работа с унаследованным кодом [Книга] / авт. Физерс Майкл К.. - [б.м.] : Вильямс, 2009.