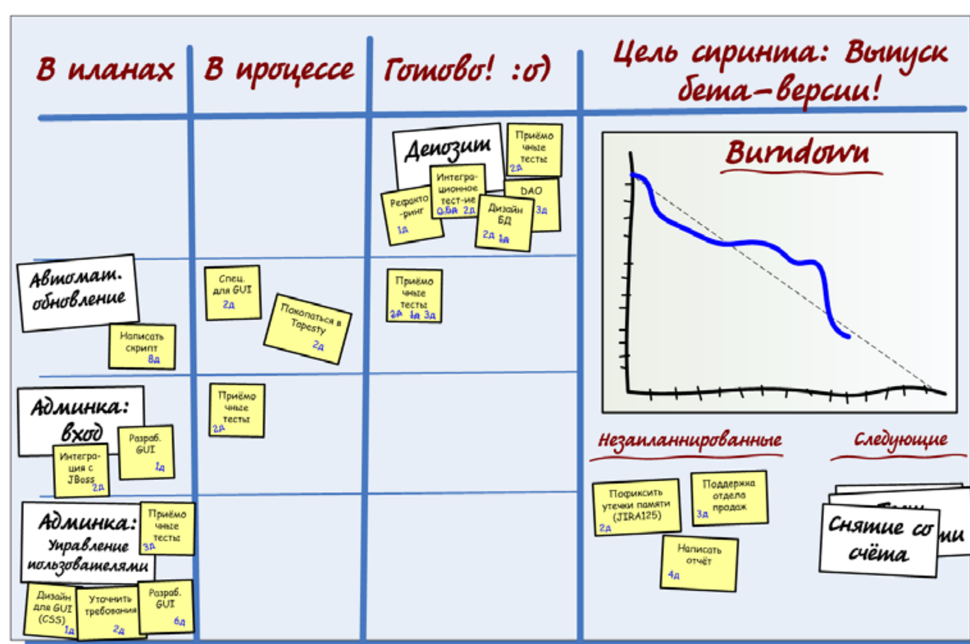
Битва за Agile

Scrum и XP: заметки с передовой

Как мы делаем Scrum



Хенрик Книберг

Предисловие от Джеффа Сазерленда и Майка Кона

InfoQ Enterprise Software Development Series

Yes, we did!

Чтобы прочитать эту книгу вам понадобится всего лишь два-три часа. Чтобы её перевести участникам сообщества Agile Ukraine потребовалось 4 месяца. Поверьте, мы не халтурили и делали свою работу от всей души.

К сожалению, на благодарности нам выделили всего лишь страничку. Поэтому я постараюсь представить всех наших активистов в фактах.

- Максим Харченко умудрялся переводить даже на море. Спасибо Гипер.NET.
- Дима Данильченко – директор и по совместительству (да и такое бывает ☺) один из самых активных переводчиков в нашем проекте.
- Если по ходу книги вам очень понравился перевод, и вы заулыбались, то, скорее всего, эту главу переводил Артём Сердюк.
- Боря Лебеда автоматизировал конвертацию оригинала из word'a в wiki формат. Я и понятия не имел, что это можно сделать так легко.
- Ярослав Гнатюк знает Хенрика лично.
- Антон Марюхненко – самый молодой и перспективный.
- Сергей Петров – самый старший и опытный.
- Марина Кадочникова – наша единственная девушка-переводчица.

Серёжа Мовчан, конечно же, я про тебя не забыл ☺. Просто хотел сказать тебе отдельное спасибо. Ты был тем вторым локомотивом, благодаря которому нам удалось дотянуть до победного конца. Ведь как говорится в одной японской поговорке: "Начинать легко, продолжать сложно".

Спасибо Марине Зубрицкой и Лёше Мамчию за финальную вычитку и редактуру. Им удалось найти свыше ста ошибок, в уже, казалось бы, вылизанном тексте. Нет предела совершенству.

Не забудем мы и про тех, у кого было желание, но, как часто это бывает, не было времени: Сергей Евтушенко, Артём Марченко, Алексей Тигарев, Тим Евграшин, Александр Кулик.

Лёша Солнцев,
инициатор и координатор первого украинского краудсорсингого проекта, Certified Scrum Master

P.S. Оригинал книги вы можете скачать по адресу http://www.infoq.com/minibooks/scrum-xp-from-the-trenches

Оглавление

Предисловие Джеффа Сазерленда 7

Предисловие Майка Кона 8

Предисловие – Эй! А Scrum-то работает! 9

Вступление 10

- Отказ от ответственности 11
- Зачем я это написал 11
- Так что же такое Scrum? 11

Как мы работаем с product backlog'ом 12

- Дополнительные поля для user story 13
- Как мы ориентируем product backlog на бизнес 14

Как мы готовимся к планированию спринта 15

Как мы планируем спринт 16

- Почему без product owner'a не обойтись 16
- Почему качество не обсуждается 17
- Планирование спринта, которое никак не заканчивается 18
- Распорядок встречи по планированию спринта 19
- Определяем длину спринта. 19
- Определение цели спринта 20
- Выбор историй, которые войдут в спринт 20
- Как product owner может влиять на то, какие истории попадут в спринт? 21
- Как команда принимает решение о том, какие истории включать в спринт? 22
  - Планирование, основанное на интуиции 22
  - Планирование, основанное на методе оценки производительности 22
  - Какую технику мы используем для планирования? 25
- Почему мы используем учетные карточки 26
- Критерий готовности 28
- Оценка трудозатрат с помощью игры в planning poker 28
- Уточнение описаний историй 29
- Разбиение историй на более мелкие истории 30
- Разбиение историй на задачи 30
- Выбор времени и места для ежедневного Scrum'a 31
- Когда пора остановиться 32
- Технические истории 32
- Как мы используем систему учёта дефектов для ведения product backlog'a 34
- Свершилось! Планирование спринта закончено! 34

Как мы доносим информацию о спринте до всех в компании 35

Как мы создаём sprint backlog ... 37
Формат sprint backlog'a ... 37
Как работает доска задач ... 38
Пример 1 – после первого ежедневного Scrum'a ... 38
Пример 2 – еще через пару дней ... 39
Как работает burndown-диаграмма ... 40
Тревожные сигналы на доске задач ... 40
Эй, как насчет отслеживания изменений? ... 41
Как мы оцениваем: в днях или часах? ... 41
Как мы обустроили комнату команды ... 42
Уголок обсуждений ... 42
Усадите команду вместе ... 43
Не подпускайте product owner'a слишком близко ... 44
Не подпускайте менеджеров и тренеров слишком близко ... 44
Как мы проводим ежедневный Scrum ... 45
Как мы обновляем доску задач ... 45
Как быть с опоздавшими ... 45
Как мы поступаем с теми, кто не знает, чем себя занять ... 46
Как мы проводим демо ... 48
Почему мы настаиваем на том, чтобы каждый спринт заканчивался демонстрацией ... 48
Памятка по подготовке и проведению демо ... 48
Что делать с "недемонстрируемыми" вещами ... 49
Как мы проводим ретроспективы ... 50
Почему мы настаиваем на том, чтобы все команды проводили ретроспективы ... 50
Как мы проводим ретроспективы ... 50
Как учиться на чужих ошибках ... 51
Изменения. Быть или не быть ... 52
Типичные проблемы, которые обсуждают на ретроспективах ... 52
«Нам надо было больше времени потратить на разбиение историй на подзадачи» ... 52
«Очень часто беспокоят извне» ... 52
«Мы взяли огромный кусок работы, а закончили только половину» ... 53
«У нас в офисе бардак и очень шумно» ... 53
Отдых между спринтами ... 54
Как мы планируем релизы и составляем контракты с фиксированной стоимостью ... 56
Определяем свою приёмочную шкалу ... 56
Оцениваем наиболее важные истории ... 57
Прогнозируем производительность ... 58
Сводим всё в план релиза ... 58

Корректируем план релиза 59
Как мы сочетаем Scrum с XP 60
Парное программирование 60
Разработка через тестирование (TDD) 61
TDD и новый код 61
TDD и существующий код 61
Эволюционный дизайн 62
Непрерывная интеграция (Continuous integration) 62
Совместное владение кодом (Collective code ownership) 63
Информативное рабочее пространство 63
Стандарты кодирования 63
Устойчивый темп / энергичная работа 63
Как мы тестируем 65
Скорее всего, вам не избежать фазы приёмочного тестирования 65
Минимизируйте фазу приёмочного тестирования 66
Повышайте качество, включив тестировщиков в Scrum-команду 66
Тестировщик – это "последняя инстанция". 66
Чем занимается тестировщик, когда нечего тестировать? 67
Повышайте качество – делайте меньше за спринт! 68
Стоит ли делать приёмочное тестирование частью спринта? 68
Соотношение спринтов и фаз приёмочного тестирования 69
Подход №1: "Не начинать новые истории, пока старые не будут готовы к реальному использованию" 70
Подход №2: "Начинать реализовывать новые истории, но наивысшим приоритетом ставить доведение старых до ума" 70
Неправильный подход: "Клепать новые истории" 71
Не забывайте об ограничении системы 71
Возвращаясь к реальности 71
Как мы управляем несколькими Scrum-командами 72
Сколько сформировать команд 72
Виртуальные команды 72
Оптимальный размер команды 73
Синхронизировать спринты или нет? 74
Почему мы ввели роль "тимлида" 75
Как мы распределяем людей по командам 76
Нужны ли узкоспециализированные команды? 77
Подход №1: команды, специализирующиеся на компонентах 77
Подход №2: универсальные команды 78

Стоит ли изменять состав команды между спринтами? 78
Участники команды с частичной занятостью 79
Как мы проводим Scrum-of-Scrums 79
Scrum-of-Scrums уровня продукта 80
Scrum-of-Scrums уровня компании 80
Чередование ежедневных Scrum'ов 80
«Пожарные» команды 81
Разбивать product backlog или нет? 82
Подход первый: Один product owner – один backlog 82
Подход второй: Один product owner – несколько backlog'ов 84
Подход третий: Несколько product owner'ов – несколько backlog'ов 84
Параллельная работа с кодом 85
Ретроспектива для нескольких команд 86
Как мы управляем географически распределёнными командами 87
Оффшорная разработка 87
Члены команды, работающие дома 88
Памятка ScrumMaster'a 90
В начале спринта 90
Каждый день 90
В конце спринта 90
Заключительное слово 91
Список рекомендованной литературы 92
Об авторе 94

Предисловие Джеффа Сазерленда

Командам необходимо знать основы Scrum'a. Как создать и оценить product backlog? Как получить из него sprint backlog? Как работать с burndown-диаграммой и вычислять производительность(velocity) своей команды? Книга Хенрика – это базовое руководство для начинающих, которое поможет командам перейти из состояния "мы пробуем Scrum" в состояние "мы успешно работаем по Scrum'у".

Хорошая реализация Scrum'a становится всё важнее и важнее для команд, которые хотят получить инвестиции. Я выступаю в качестве тренера по гибким методологиям для группы компаний с венчурными инвестициями, помогая им в стремлении вкладывать деньги только в настоящие Agile-компании. Глава группы инвесторов требует от компаний, составляющих инвестиционный портфель, ответа на вопрос, знают ли они производительность своих команд. Многих этот вопрос ставит в тупик. Будущие инвестиции требуют от команд знания собственной производительности разработки программного обеспечения.

Почему это так важно? Если команда не знает собственной производительности, следовательно product owner не может разработать стратегический план развития продукта с достоверными датами релизов. Без такого плана компанию может постичь неудача, в результате чего инвесторы потеряют свои деньги.

С этой проблемой сталкиваются разнообразные компании: большие и маленькие, старые и новые, с финансированием и без. Во время недавнего обсуждения реализации Scrum'a компанией Google на лондонской конференции я решил узнать у аудитории, состоящей из 135 человек, кто из них использует Scrum? Я получил утвердительный ответ лишь от тридцати человек. Затем я поинтересовался, соответствует ли их процесс Nokia-стандарту итеративной разработки. Итеративная разработка – это ключевое положение Agile Manifest'a: "Постарайтесь предоставлять версии работающего программного обеспечения как можно чаще и раньше". В результате проведения ретроспектив с сотнями Scrum-команд в течение нескольких лет, Nokia выработала некоторые базовые требования к итеративной разработке:

- Итерации должны иметь фиксированную длину и не превышать шести недель.
- К концу каждой итерации код должен быть протестирован отделом качества (QA) и работать как следует.

Из тридцати человек, которые сказали, что работают по Scrum'у, лишь половина подтвердила, что их команды придерживаются первого принципа Agile Manifest'a и соответствую Nokia-стандарту.

Затем я спросил их, придерживаются ли они Scrum-стандарта, разработанного Nokia:

- У Scrum-команды должен быть один product owner и команда должна знать, кто это.
- У product owner'a должен быть один product backlog с историями и их оценками, выполненными командой.
- У команды должна быть burndown-диаграмма, а сама команда должна знать свою производительность.
- На протяжении спринта никто не должен вмешиваться в работу команды.

Из тридцати команд, внедряющих Scrum, только у трёх процесс разработки соответствовал стандартам Nokia. Я думаю, что только эти три команды получат дальнейшие инвестиции от венчурных капиталистов.

Основная ценность книги Хенрика состоит в том, что если вы будете следовать его советам, то у вас будет и product backlog, и оценки для product backlog'a, и burndown-диаграмма. Вы также будете знать производительность вашей команды и сможете использовать все наиболее важные практики высокоэффективных Scrum-команд. Вы пройдёте Nokia Scrum-тест, за что инвесторы оценят вас по достоинству. Если вы – начинающая компания, то, возможно, вы получите такие жизненно важные для вашего проекта финансовые вливания. Возможно вы – будущее разработки программного обеспечения, вы – создатель нового поколения программ, которые станут лидерами рынка.

Джефф Сазерленд,
доктор наук, соавтор Scrum

Предисловие Майка Кона

И Scrum, и XP (экстремальное программирование) требуют от команд завершения вполне осязаемого куска работы, который можно предоставить пользователю в конце каждой итерации. Эти итерации планируются таким образом, чтобы быть короткими и фиксированными по времени. Такая целенаправленность на выпуск рабочего кода за короткий промежуток времени означает только одно: в Scrum и XP нет места теории. Agile-методологии не гонятся за красивыми UML моделями, выполненными при помощи специальных case-средств, за созданием детализированных спецификаций или написанием кода, который сойдёт на все случаи жизни. Вместо этого Scrum и XP команды концентрируются на том, чтобы завершить необходимые задачи. Эти команды могут мириться с ошибками в ходе работы, но они понимают, что лучшее средство выявить эти ошибки – это перестать думать о софте на теоретическом уровне анализа и дизайна, и, закатав рукава, полностью посвятить себя созданию продукта.

Именно акцент на действии, а не на теории ярко выделяет эту книгу среди прочих. То, что Хенрик разделяет эти взгляды, видно с самых первых страниц книги. Он не предлагает нам длинное описание того, что такое Scrum; вместо этого он просто ссылается на необходимые веб-ресурсы. Первым делом Хенрик начинает с описания того, как его команда работает со своим product backlog'ом. Затем он проходит по всем элементам и практикам правильно поставленного agile-проекта. Без теоретизирования. Без справочных данных. Ничего этого не нужно: книга Хенрика – не философское объяснение, почему Scrum работает или почему мы должны делать так, а не иначе. Это описание того, как работает одна успешная agile-команда.

Хенрик предлагает набор избранных практик и описывает живые примеры, чтобы помочь нам понять, как использовать Scrum и XP на передовой.

Майк Кон
Автор книг *Agile Estimating and Planning* и *User Stories Applied for Agile Software Development*.

Предисловие – Эй! А Scrum-то работает!

Scrum работает! По крайней мере, нам он подошёл (я имею в виду проект моего клиента из Стокгольма, чьё имя я не хотел бы называть). Надеюсь, вам он тоже подойдёт! И, возможно, именно эта книга поможет вам на пути освоения Scrum'а.

Это был первый случай в моей жизни, когда я увидел как методология (ну да, Кен [Кен Швебер – соавтор Scrum'а], – фреймворк) 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.

Не хочеться говорить, что я удивлён, но... так и есть. После беглого знакомства с парой книг по теме, Scrum произвёл на меня хорошее впечатление, даже слишком хорошее, чтобы быть похожим на правду. Так что неудивительно, что я был настроен слегка скептически. Однако после года использования Scrum'а, я настолько впечатлён (и большинство людей в моих командах тоже), что, скорее всего, буду использовать Scrum во всех новых проектах, ну, разве что кроме случаев, когда есть веская причина не делать этого.

1

Вступление

Собираетесь начать практиковать Scrum у себя в компании? Или вы уже работаете по Scrum'у пару месяцев? У вас уже есть базовые понятия, вы прочитали несколько книг, а, возможно, даже получили сертификат Scrum Master'a? Поздравляю!

Однако, согласитесь, какая-то неясность всё равно остаётся.

По словам Кена Швебера, Scrum – это не методология, это фреймворк. А это значит, что Scrum не дает готовых рецептов, что делать в тех или иных случаях. Вот незадача!

Но у меня для вас есть хорошая новость: я расскажу, как именно я практикую Scrum... очень подробно и со всеми деталями. Однако, и без плохой новости не обойдётся: я расскажу всего лишь о том, как практикую Scrum я. Это значит, что вам не обязательно делать всё точно так же. На самом деле, в зависимости от ситуации, я и сам могу делать что-то по-другому.

Достоинство Scrum'a и одновременно самый большой его недостаток в том, что его необходимо адаптировать к вашей конкретной ситуации.

Моё видение Scrum'a формировалось на протяжении целого года и стало результатом экспериментов в команде численностью около 40-ка человек. Одна компания попала в крайне сложную ситуацию: постоянные переработки, авралы, серьёзные проблемы с качеством, проваленные сроки и прочие неприятности. Эта компания решила перейти на Scrum, но внедрить его толком у неё не получалось. В итоге эта задача была поручена мне. К тому времени для большинства сотрудников слово «Scrum» было просто набившим оскомину термином, эхо которого звучало время от времени в коридорах без какого-либо отношения к их повседневной работе.

Через год работы мы внедрили Scrum на всех уровнях компании: поэкспериментировали со всевозможными размерами команд (от 3 до 12 человек), попробовали спринты различной длины (от 2 до 6 недель) и разные способы определения критерия готовности, разнообразные форматы product и sprint backlog'ов (Excel, Jira, учетные карточки), разные стратегии тестирования, способы демонстрации результатов, способы синхронизации Scrum-команд и так далее. Также мы опробовали множество XP практик: с разными способами непрерывной интеграции, с парным программированием, TDD (разработкой через тестирование), и т.д. А самое главное – разобрались, как все это дело сочетается со Scrum'ом.

Scrum подразумевает постоянный процесс развития, так что история на этом не заканчивается. Я уверен, упомянутая мной компания будет продолжать двигаться вперед (если в ней и дальше будут проводить ретроспективы спринтов) и постоянно находить новые и новые способы эффективного применения Scrum'a, учитывая особенности каждой из сложившихся ситуаций.

Отказ от ответственности

Эта книга не претендует на звание «единственно правильного» учебного пособия по Scrum'у! Она всего лишь предлагает вам пример удачного опыта, полученного на протяжении года в результате постоянной оптимизации процесса. Кстати, у вас запросто может возникнуть ощущение, что мы всё поняли не так.

Эта книга отражает моё субъективное мнение. Она никоим образом не является официальной позицией Crisp'a [от переводчика – консалтинговая компания, в которой работает Хенрик] или моего нынешнего клиента. Именно поэтому я специально избегал упоминания каких-либо программных продуктов или людей.

Зачем я это написал

Во время изучения Scrum'a я читал книги, посвящённые Scrum'y и Agile'y, рылся по различным сайтам и форумам, проходил сертификацию у Кена Швебера, засыпал его вопросами и проводил кучу времени, обсуждая Scrum с моими коллегами. Однако одним из наиболее ценных источников знаний стали реальные истории внедрения Scrum'a. Они превращают Принципы и Практики в... ну, в то, как вы это делаете. Они помогли мне предвидеть типичные ошибки Scrum-новичков, а иногда и избежать их.

Итак, вот и выпал мой шанс поделиться чем-то полезным. Это моя реальная история.

Так что же такое Scrum?

Ой, простите, я совсем забыл про новичков в Scrum'e и XP. Если вы к ним относитесь, вам не мешало бы посетить следующие веб-ресурсы:

- http://agilemanifesto.org/
- http://www.mountaingoatsoftware.com/scrum
- http://www.xprogramming.com/xpmag/whatisxp.htm

Нетерпеливые же могут продолжать читать книгу дальше. Я объясню все основные Scrum'овские термины по ходу изложения.

Я надеюсь, что эта книга станет стимулом для тех, кто так же не против поделиться своими мыслями на счёт Scrum'a. Пожалуйста, не забывайте сообщать мне о них!

2

Как мы работаем с product backlog'ом

Product backlog – это основа Scrum'а. С него все начинается. По существу, product backlog является списком требований, историй, функциональности, которые упорядочены по степени важности. При этом все требования описаны на понятном для заказчика языке.

Элементы этого списка мы будем называть "*историями*", *user story*, а иногда *элементами backlog'a*.

Описание каждой нашей истории включает в себя следующие поля:

- **ID** – уникальный идентификатор – просто порядковый номер. Применяется для идентификации историй в случае их переименования.
- **Название** – краткое описание истории. Например, "Просмотр журнала своих транзакций". Оно должно быть однозначным, чтобы разработчики и product owner (владелец продукта) могли примерно понять, о чем идет речь, и отличить одну историю от другой. Обычно от 2 до 10 слов.
- **Важность** (Importance) – степень важности данной задачи, по мнению product owner'a. Например, 10. Или 150. Чем больше значение, тем выше важность.
  - Я предпочитаю не использовать термин "приоритет", поскольку обычно в этом случае 1 обозначает наивысший приоритет. Это очень неудобно: если впоследствии вы решите, что какая-то история еще более важна, то какой "приоритет" вы тогда ей поставите? Приоритет 0? Приоритет -1?
- **Предварительная оценка** (initial estimate) – начальная оценка объема работ, необходимого для реализации истории по сравнению с другими историями. Измеряется в story point'ах. Приблизительно соответствует числу "идеальных человеко-дней".
  - Спросите вашу команду: "Если собрать команду из оптимального количества людей, то есть не слишком большую и не слишком маленькую (чаще всего из двух человек), закрыться в комнате с достаточным запасом еды и работать ни на что не отвлекаясь, то, сколько дней тогда понадобится на разработку завершённого, протестированного продукта, готового к демонстрации и релизу? ". Если ответ будет "Для трёх человек, закрытых в комнате, на это потребуется 4 дня", это значит, что изначальная оценка составляет 12 story point'ов.
  - В этом случае важно получить не максимально точные оценки (например, для истории в 2 story point'a потребуется 2 дня), а сделать так, чтобы оценки верно отражали относительную трудоёмкость историй (например, на историю, оцененную в 2 story point'a потребует примерно в два раза меньше работы по сравнению с историей в 4 story point'a).
- **Как продемонстрировать** (how to demo) – краткое пояснение того, как завершённая задача будет продемонстрирована в конце спринта. По сути, это простой тестовый сценарий типа "Сделайте это, сделайте то – должно получиться то-то".
  - Если у вас практикуется Test Driven Development (разработка через тестирование или кратко TDD) , то это описание может послужить псевдокодом [пр. Переводчика: в программировании специальный способ записи любого алгоритма] для приемочного теста.
- **Примечания** – любая другая информация: пояснения, ссылки на дополнительные источники информации, и т.д. Обычно она представлена в форме кратких тезисов.

Product backlog (пример)					
ID	Название	Важность	Предварительная оценка	Как продемонстрировать	Примечания
1	Депозит	30	5	Войти в систему, открыть страницу депозита, положить на счет €10, перейти на страницу баланса и проверить, что он увеличился на €10.	Нужна UML диаграмма последовательности. Пока что не стоит беспокоиться про шифрование данных.
2	Просмотр журнала личных транзакций	10	8	Войти в систему; перейти на страницу транзакций; положить деньги на счет; вернуться на страницу транзакций; проверить, что новая транзакция появилась в списке.	Чтобы избежать больших запросов к базе данных, стоит воспользоваться постраничным выводом информации. Дизайн такой же, как и у страницы просмотра пользователей.

Мы экспериментировали и с другими полями, но в итоге именно эти 6 оказались для нас самыми применимыми.

Обычно product backlog хранится в Excel таблице с возможностью совместного доступа (несколько пользователей могут редактировать файл одновременно). Хотя официально документ принадлежит product owner'у, мы не запрещаем другим пользователям редактировать его. Ведь разработчикам довольно часто приходится заглядывать в product backlog, чтобы что-то уточнить или изменить оценку предполагаемых трудозатрат.

По этой же причине, мы не помещаем product backlog в систему контроля версий, а храним его на сетевом диске. Этот простейший способ позволяет избежать взаимных блокировок доступа и конфликтов синхронизации изменений.

Однако почти все остальные артефакты хранятся у нас в системе контроля версий.

Дополнительные поля для user story

Иногда мы используем дополнительные поля в product backlog'е. В основном для того, чтобы помочь product owner'у определиться с его приоритетами.

- **Категория** (track). Например, "панель управления" или "оптимизация". При помощи этого поля product owner может легко выбрать все пункты категории "оптимизация" и установить им низкий приоритет.
- **Компоненты** (components) – указывает, какие компоненты (например, база данных, сервер, клиент) будут затронуты при реализации истории. Данное поле состоит из группы checkbox'ов, которые отмечаются, если соответствующие компоненты требуют изменений. Поле "компоненты" окажется особенно полезным, если у вас несколько Scrum команд, например, одна, которая работает над панелью управления и другая, которая отвечает за клиентскую часть. В данном случае это поле существенно упростит для каждой из команд процедуру выбора истории, за которую она могла бы взяться.
- **Инициатор запроса** (requestor). Product owner может захотеть хранить информацию о всех заказчиках, заинтересованных в данной задаче. Это нужно для того, чтобы держать их в курсе дела о ходе выполнения работ.
- **ID в системе учёта дефектов** (bug tracking ID) – если вы используете отдельную систему для учёта дефектов (например. Jira), тогда в описании истории полезно хранить ссылки на все дефекты, которые к ней относятся.

Как мы ориентируем product backlog на бизнес

Если product owner – технарь, то он вполне может добавить историю вроде "Добавить индексы в таблицу Events". Зачем ему это нужно? Да потому, что реальной целью этой истории, скорее всего, будет "ускорение поиска событий в панели управления".

И вообще, может оказаться, что не индексы были узким местом, приводящим к медленной работе поисковой формы. Причиной могло быть что-то абсолютно другое. Обычно команде виднее, каким образом лучше решить подобную проблему, поэтому product owner должен ставить цели с точки зрения бизнеса (т.е. что надо).

Когда я вижу технические истории подобные этой, я обычно задаю product owner'у вопросы вроде "Да, но зачем?". Я делаю это до тех пор, пока не проявится истинная причина появления истории (в приведенном примере – повысить скорость поиска событий в панели управления). Первоначальное техническое описание проблемы можно поместить в колонку с примечаниями: "Индексирование таблицы Events может решить проблему".

3

Как мы готовимся к планированию спринта

Не успеешь оглянуться – как наступил день планирования нового спринта. Мы не раз приходили к одному и тому же выводу:

Вывод: Убедитесь, что product backlog находится в нужной кондиции, прежде чем начинать планирование.

Что значит в *нужной кондиции*? Что все user story отлично описаны? Что все оценки трудозатрат корректны? Что все приоритеты расставлены? Нет, нет и ещё раз нет! Это значит:

- Product backlog должен существовать! (Кто бы мог подумать?)
- У каждого продукта должен быть *один* product backlog и *один* product owner.
- Все наиболее важные задачи должны быть классифицированы по уровню важности, а их числовые значения не должны совпадать.
    - Не волнуйтесь, если задачи с низким уровнем важности имеют одинаковые значения, скорее всего, они не попадут в текущий спринт, а, следовательно, не будут обсуждаться.
    - Все user story, которые, по мнению product owner'а имеют гипотетическую возможность попасть в следующий спринт, должны иметь уникальное значение важности.
    - Уровень важности используется исключительно для упорядочивания историй. Т.е. если история А имеет уровень важности 20, а история Б важность 100, это означает что Б важнее А. Это *не* означает, что Б в пять раз важнее А. Если Б присвоить важность 21 – смысл не поменяется!
    - Полезно оставлять промежутки из целых чисел между значениями на тот случай, если появится история В, более важная, чем А, но менее важная, чем Б. Конечно, можно выкрутиться, присвоив ей уровень важности 20.5, но выглядит это коряво, поэтому для промежутков мы решили использовать только целые числа!
- Product owner должен *понимать* каждую историю (чаще всего он их автор, хотя иногда другие члены команды тоже могут вносить предложения, и тогда product owner обязан назначить им приоритетность). Он не обязан знать во всех подробностях, что конкретно следует сделать, но он должен понимать, почему эта user story была включена в product backlog.

Примечание: Хотя заинтересованные лица могут добавлять user story в product backlog, они не имеют права присваивать им уровень важности. Это прерогатива product owner'а. Они также не могут добавлять оценки трудозатрат, поскольку это прерогатива команды.

Мы также попробовали:

- Использовать Jira (нашу систему учёта дефектов) для хранения product backlog'а. Но для большинства product owner'ов навигация по Jira была слишком обременительна. В Excel'е манипулировать историями намного проще и приятней. Вы можете раскрашивать текст, переставлять пункты, добавлять, в случае необходимости, новые колонки, комментировать, импортировать и экспортировать данные и т.д.
- Использовать программы, заточенные под Agile процесс, такие как VersionOne, ScrumWorks, Xplanner и т.д. Но толком до них руки так и не дошли, хотя, возможно, когда-нибудь мы их всё-таки попробуем.

4

Как мы планируем спринт

Как по мне, планирование спринта – наиболее важная часть Scrum'а. Плохо проведённое планирование может испортить весь спринт.

Цель планирования заключается в том, чтобы, с одной стороны, дать команде достаточно информации для спокойной работы в течение нескольких недель, а с другой – убедить product owner'a в том, что команда сможет сделать свою работу.

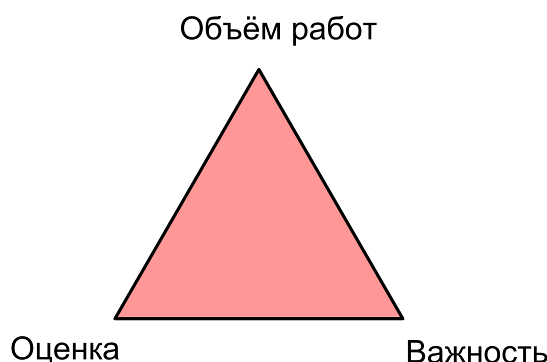
Хорошо-хорошо, это было весьма расплывчатое определение. Давайте лучше поговорим о том, что должно быть итогом планирования:

- Цель спринта.
- Список участников команды (и степень их занятости, если она не стопроцентная).
- Sprint backlog (список историй, которые вошли в спринт).
- Дата демонстрации.
- Место и время проведения ежедневного Scrum'а.

Почему без product owner'a не обойтись

Иногда product owner с большой неохотой тратит своё время на планирование вместе с командой: "Ребята, я уже перечислил всё, что мне нужно. Я больше не могу тратить время на ваше планирование". Это, между прочим, очень серьёзная проблема.

Команде и product owner'у просто необходимо планировать вместе, потому что каждая user story имеет три параметра, которые очень тесно связаны между собой.



Объём работ и приоритеты задач определяются product owner'ом. Зато оценка трудозатрат – это прерогатива команды. Благодаря взаимодействию команды и product owner'a в ходе планирования спринта вырабатывается оптимальное соотношение всех трех переменных.

Чаще всего product owner начинает планирование следующего спринта с описания основных целей и наиболее значимых историй. После этого команда производит оценку трудозатрат всех user story, начиная с самой важной. В процессе у команды возникают очень важные вопросы по поводу объёма предстоящих

работ. Например, “Подразумевает ли история “удалить пользователя” удаление всех его незавершённых транзакций?”. Иногда ответ на этот вопрос будет большим сюрпризом для команды и потребует пересмотра всех оценок для данной user story.

В некоторых случаях время, которое понадобится на выполнение user story, не будет совпадать с ожиданиями product owner’а. Следовательно, он захочет пересмотреть приоритет для story или изменить объём работы. Это, в свою очередь, заставит команду выполнить переоценку и так далее, и так далее.

Такая взаимная зависимость является основой Scrum’а, да, в принципе, и всего Agile’а.

Но что если product owner всё-таки упорно отказывается выделить пару часов на планирование спринта? В таком случае я обычно пытаюсь последовательно применить следующие стратегии:

- Попытайтесь донести до product owner’а, почему его участие крайне важно – а вдруг до него дойдёт.
- Попробуйте найти в своих рядах добровольца, который смог бы стать представителем product owner’а. Скажите своему product owner’у: “У вас нет времени на планирование, Джеф будет исполнять вашу роль. У него будут все полномочия на изменение приоритетов и объёмов работ. Советую вам обсудить с ним как можно больше нюансов до начала планирования. Если вы против Джефа, тогда выберите кого-то другого, но только с условием, что он будет присутствовать на планировании от начала до конца”.
- Попробуйте убедить менеджмент найти вам нового product owner’а.
- Отложите начало спринта до того момента, пока у product owner’а не появится свободная минутка для совместного планирования. А пока не берите на себя никаких новых обязательств. Пусть в это время ваша команда займётся любой другой полезной работой.

Почему качество не обсуждается

В предыдущей главе я намеренно не показал на треугольнике четвертую переменную – *качество*.

Попытаюсь объяснить разницу между *внутренним качеством* и *внешним качеством*.

- *Внешнее качество* – это то, как пользователи воспринимают систему. Медленный и неинтуитивный пользовательский интерфейс – это пример плохого внешнего качества.
- *Внутреннее качество* касается вещей, которые как правило не видны пользователю, но при этом оказывают огромное значение на удобство сопровождения системы. Это продуманность дизайна системы, покрытие тестами, читаемость кода, рефакторинг и т.д.

По правде говоря, у системы с высоким *внутренним* качеством иногда может быть довольно низкое *внешнее*. Но наоборот бывает крайне редко. Сложно построить что-то хорошее на прогнившем фундаменте.

Я рассматриваю внешнее качество, как часть общего объема работ. Ведь с точки зрения бизнеса бывает весьма целесообразно как можно быстрее выпустить версию системы с немного корявым и медленным пользовательским интерфейсом, и лишь потом подготовить версию с доработками и исправлениями. Здесь право выбора должно оставаться за product owner’ом, так как именно он отвечает за определение объёма работ. И напротив – внутреннее качество не может быть предметом дискуссии. Команда постоянно должна следить за качеством системы, поэтому оно попросту не обсуждается. Никогда.

(Ну ладно, почти никогда)

Так как же нам различать задачи, связанные с внутренним и внешним качеством?

Представьте, что product owner говорит: “Хорошо ребята, я понимаю, почему вы оценили эту задачу в 6 story point’ов, но я уверен, что, если вы чуточку помозгуете, то сможете по-быстрому “залатать” проблему”.

Ага! Он пытается использовать внутреннее качество как переменную! Как я догадался? Да, потому что он хочет, чтобы мы уменьшили оценку задач, не уменьшив при этом объём работ. Слово “заплатка” должно вызывать у вас тревогу.

Почему же мы так жестко стоим на своем?

По моему личному опыту, жертвовать внутренним качеством – это практически всегда очень и очень плохая идея. Сэкономленное время ничтожно мало по сравнению с той ценой, которую вам придётся заплатить как в ближайшем будущем, так и в перспективе. Как только качество вашего кода ухудшится, восстановить его будет очень тяжело.

В этом случае я стараюсь перейти к обсуждению объема задач. "Раз вам так важно получить эту историю как можно раньше, тогда может быть стоит сократить объем задач, чтобы мы могли сделать её побыстрее? Возможно, стоит упростить обработку ошибок и сделать "Улучшенную обработку ошибок" отдельной историей оставив ее на будущее? Или может понизить приоритет остальных историй, чтобы мы могли сосредоточить все свои усилия на этой?".

Планирование спринта, которое никак не заканчивается

Самая большая сложность при планировании спринта состоит в следующем:

1. Люди не расчитывают, что это займёт так много времени
2. … но так оно и происходит!

В Scrum'е всё ограничено по времени. Мне очень нравится это простое правило, и мы всячески пытаемся его придерживаться.

Так что же мы делаем, когда ограниченное по времени планирование спринта близится к концу, а цель спринта или sprint backlog всё ещё не определены? Просто обрываем планирование? Продлеваем на час? Или, быть может, мы завершаем собрание и продолжаем его на следующий день?

Это случается снова и снова, особенно в новых командах. Как вы обычно решаете эту проблему? Я не знаю. А как решаем её мы? Ну, обычно, я бесцеремонно обрываю встречу. Заканчиваю её. Пусть спринт пострадает. Точнее, я говорю команде и product owner'у: «Итак, встреча заканчивается через 10 минут. Мы, до сих пор, полностью не спланировали спринт. Можем ли мы начать работать с тем, что у нас есть, или назначим ещё одно 4-х часовое планирование спринта на завтра в 8 утра?». Можете догадаться, что они отвечают… :o)

Я несколько раз давал возможность совещанию затянуться, но обычно это, ни к чему не приводило. Главная причина этому – усталость участников встречи. Если команда не выработала подходящий план спринта за 2 – 8 часов (зависит от конкретно ваших ограничений по времени), они, скорее всего, не управятся с ним и в дополнительное время. Второй вариант, по правде, достаточно хорош: назначить новую встречу на следующий день. За исключением того, что люди обычно нетерпеливы и хотят быстрее начать спринт, а не тратить ещё пару часов на планирование.

Итак, я урезаю продолжительность встречи. Да, спринт от этого страдает. Но с другой стороны, команда получила очень ценный урок, и следующее планирование спринта пройдёт более эффективно. Кроме того, когда вы в следующий раз предложите увеличить продолжительность встречи, люди будут возмущаться гораздо меньше.

Учитесь оставаться в рамках установленного времени, учитесь давать реалистичные оценки. Это касается как продолжительности встреч, так и продолжительности спринта.

Распорядок встречи по планированию спринта

Наличие хотя бы примерного расписания значительно увеличит ваши шансы закончить планирование спринта в отведённое для этого время.

Например, наше расписание выглядит так:

Планирование спринта: с 13:00 до 17:00 (после каждого часа перерыв на 10 минут)

- **13:00 – 13:30.** Product owner разъясняет цель спринта и рассказывает про бизнес-процессы из product backlog'а. Обсуждается время и место проведения демо.
- **13:30 – 15:00**. Команда проводит оценку времени, которое потребуется на разработку бизнес-процессов и, при необходимости дробит их на более мелкие. В некоторых случаях product owner может изменить приоритет их исполнения. Выясняем все интересующие нас вопросы. Для наиболее важных заполняем поле «Как продемонстрировать».
- **15:00 – 16:00.** Команда определяет набор user story, которые войдут в следующий спринт. Чтобы проверить насколько это реально, вычисляем производительность команды.
- **16:00 – 17:00.** Договариваемся о времени и месте проведения ежедневного Scrum'а (если они изменились по сравнению с прошлым спринтом). После чего приступаем к разбиению user story на задачи.

Наличие расписания ни в коем случае не подразумевает наличия жестких ограничений. В зависимости от того, как проходит встреча, ScrumMaster может увеличить, или уменьшить продолжительность каждого этапа.

Определяем длину спринта.

Одна из основных задач планирования спринта – это определение даты демо. А это значит, что вам придётся определиться с длиной спринта.

Какая же длина оптимальна?

Короткие спринты – удобны. Они позволяют компании быть максимально “гибкой”, а значит готовой часто корректировать свои планы. Короткий спринт = короткий цикл обратной связи = частые релизы = быстрые отзывы от клиентов = меньше времени тратится на работу в неправильном направлении = быстрое обучение, совершенствование и т.д.

Но с другой стороны длинные спринты тоже хороши. У команды остаётся больше времени, чтобы набрать темп, больше пространства для манёвров, чтобы решить возникшие проблемы, а также больше времени для достижения цели спринта, а у вас меньше накладных расходов, таких как планирование спринта, демо и т.д.

В основном короткие спринты больше нравятся product owner'у, а длинные – разработчикам. Поэтому длина спринта – это всегда компромисс. Мы долго экспериментировали и выбрали нашу любимую длину: 3 недели. Большинство наших команд (но не все) используют трёхнедельный спринт. Достаточно короткий, чтобы предоставить адекватную корпоративную “гибкость”, но в тоже время достаточно длинный, для того чтобы команда смогла достигнуть максимальной производительности и успеть решить все вопросы, которые возникнут по ходу спринта.

Мы пришли к важному выводу: *экспериментировать* с длиной спринта *нужно* на начальном этапе. Не тратьте слишком много времени на *анализ*, просто выберите подходящую длину и используйте её на протяжении одного или двух спринтов, после чего можете выбрать новую.

Однако, как только вы найдёте подходящею длину, надолго *зафиксируйте её*. После нескольких месяцев экспериментов нам подошла длина в 3 недели, поэтому теперь мы следуем этому правилу и используем трёхнедельные спринты. Иногда спринт будет казаться слишком длинным, иногда – слишком коротким. Но сохранение фиксированной длины спринта позволяет выработать корпоративный ритм, в котором все чувствуют себя достаточно комфортно. К тому же исчезнут споры, насчёт даты релиза, так как все знают, что как ни крути, а выпуск новой версии продукта каждые 3 недели.

Определение цели спринта

Это случается практически всегда, когда в ходе нашего планирования я задаю вопрос: “Итак, какова же цель спринта?”. Все начинают смотреть на меня удивлёнными глазами, а product owner – морщить лоб, почёсывая свой подбородок.

Почему-то сформулировать цель спринта бывает *довольно непросто*. Но я до сих пор убеждён, что усилия, потраченные на попытки сформулировать цель, оправдывают себя. Лучше паршивая цель, чем её отсутствие. Например, цели могут быть следующие: “заработать больше денег”, “завершить три истории с наивысшими приоритетами”, “удивить исполнительного директора”, “подготовить систему к бета-тестированию”, “добавить возможность администрирования” или что-нибудь в этом духе. Самое главное, чтобы цель была обозначена в терминах бизнеса, а не в технических терминах. То есть языком, понятным даже людям вне команды.

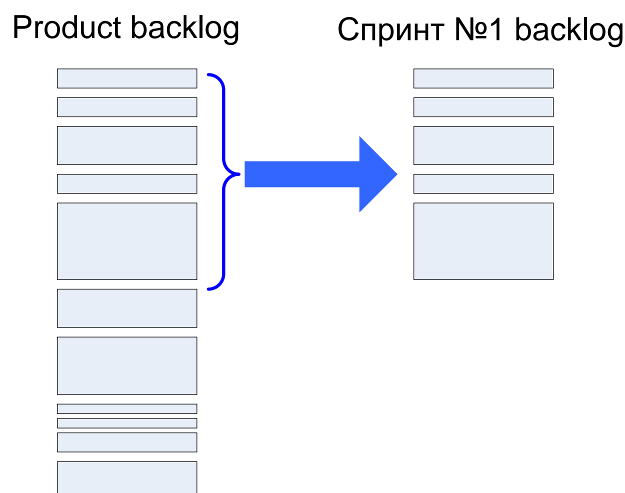
Цель спринта должна отвечать на главный вопрос “Зачем мы работаем над этим спринтом? Почему мы все просто не уйдём в отпуск?”. На самом деле, самый простой способ вытянуть цель спринта из product owner’а – напрямую задать ему этот вопрос.

Целью должно быть что-то, что не было ещё достигнуто. “Удивить исполнительного директора” может быть неплохой целью. Но только не в том случае, когда он и так в восторге от текущего состояния системы. В этом случае, все могут просто собраться и пойти домой, а цель спринта всё равно будет достигнута.

Цель спринта может показаться слегка глупой и надуманной на протяжении всего планирования. Но чаще всего, основная её ценность начинает проявляться к середине спринта, когда люди начинают забывать чего они хотят достичь в этом спринте. Если у вас работают несколько Scrum-команд (как у нас) над разными продуктами, очень полезно иметь возможность просмотреть список целей спринтов для всех команд на одной wiki-странице (или ещё где-нибудь), а также вывесить их на видном месте, чтобы все (а не только топ-менеджеры) знали, чем занимается компания и зачем!

Выбор историй, которые войдут в спринт

Основное в планировании спринта – процедура выбора историй, которые войдут в спринт. А точнее, выбор историй, которые нужно скопировать из product backlog’а в sprint backlog.



Взгляните на картинку. Каждый прямоугольник представляет собой историю, расположение которой соответствует уровню её важности. Наиболее важная история находится наверху списка. Размер истории (т.е. количество story point’ов) определяет размер каждого прямоугольника. Высота голубой скобки обозначает *прогнозируемую производительность команды*, т.е. количество историй, которое команда собирается завершить в следующем спринте.

Честно говоря, sprint backlog – это выборка историй из product backlog'a. Он представляет собой список историй, которые команда обязалась выполнить в течение спринта.

Именно *команда* решает, сколько историй войдёт в спринт. Ни product owner, ни кто-нибудь ещё.

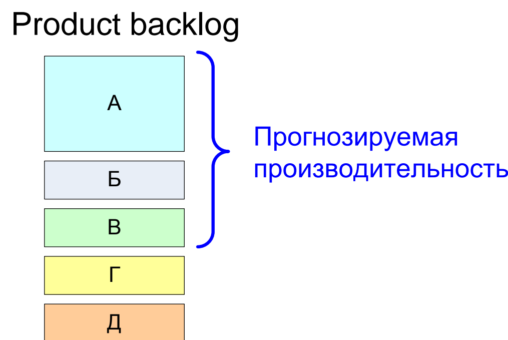
В связи с этим, возникают два вопроса:

1. Каким образом команда решает, какие истории попадут в спринт?
2. Как product owner может повлиять на их решение?

Начну со второго вопроса.

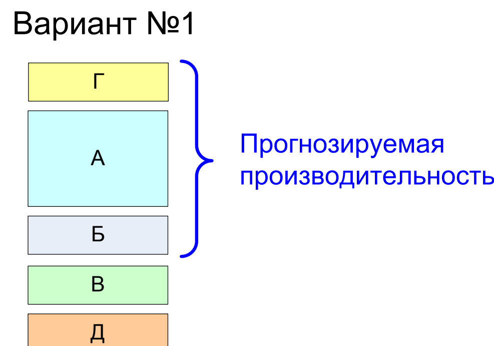
Как product owner может влиять на то, какие истории попадут в спринт?

Допустим, на планировании спринта возникла следующая ситуация:

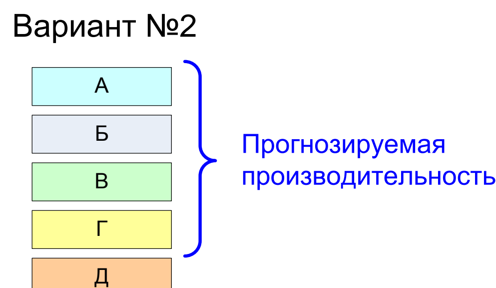


Product owner'а разочаровал тот факт, что история "Г" не попадает в спринт. Что он может сделать в ходе совещания?

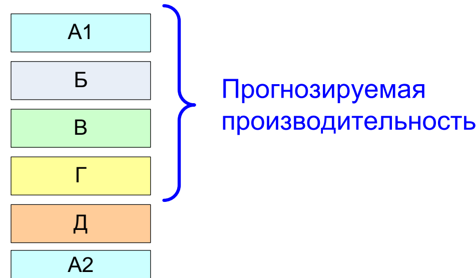
Первый вариант – изменение приоритетов. Если product owner назначит истории "Г" более высокий приоритет, то команда будет обязана включить её в спринт первой (исключив при этом историю "В").



Второй вариант – изменение объёма работ: product owner начинает уменьшать объём истории "А" до тех пор, пока команда не решит, что историю "Г" можно втиснуть в спринт.



Третий вариант – разбиение истории. Product owner может решить, что некоторые части истории “А” не так уж и важны. Таким образом, он разбивает историю “А” на две истории “А1" и “А2", а затем назначает им разный приоритет.



Итак, несмотря на то, что в большинстве случаев product owner не может контролировать прогнозируемую производительность, у него существует множество способов повлиять на то, какие истории попадут в спринт.

Как команда принимает решение о том, какие истории включать в спринт?

Мы используем два подхода:

1. на основе интуиции
2. на основе подсчёта производительности

Планирование, основанное на интуиции

ScrumMaster: “Ребята, мы закончим историю “А” в этом спринте?” (Показывает на самую важную историю в product backlog’е)
Лиза: “Конечно, закончим. У нас есть три недели, а это довольно тривиальная функциональность”.
ScrumMaster: “Хорошо. А как на счёт истории “Б”?” (Показывает на вторую по важности историю)
Том и Лиза одновременно: “Легко!”
ScrumMaster: “Хорошо. Как на счёт историй “А”, “Б” и “В”?”
Сэм (обращаясь к product owner): “Нужно ли реализовывать расширенную обработку ошибок для истории “В”?”
Product owner: “Нет. Пока хватит базовой”.
Сэм: “В таком случае историю “В” мы тоже закончим”.
ScrumMaster: “Хорошо, как на счёт истории “Г”?”
Лиза: “Хмм…”
Том: “Думаю, что сделаем”.
ScrumMaster: “Вероятность 90% или 50%?”
Лиза и Том: “скорее 90%.”
ScrumMaster: “Хорошо, значит, включаем историю “Г” в этот спринт. Что скажете на счет истории “Д”?”
Сэм: “Возможно”.
ScrumMaster: “90%? 50%?”
Сэм: “Ближе к 50%”.
Лиза: “Сомневаюсь”.
ScrumMaster: “В таком случае, не включаем историю “Д”. Обязуемся реализовать истории “А”,”Б”,”В” и “Г”. Конечно, если успеем, то реализуем и историю “Д”, однако не стоит на это расчитывать. Поэтому историю “Д” исключаем из плана спринта. Согласны?”
Все: “Согласны”.
Интуитивное планирование хорошо работает для маленьких команд и коротких спринтов.

Планирование, основанное на методе оценки производительности

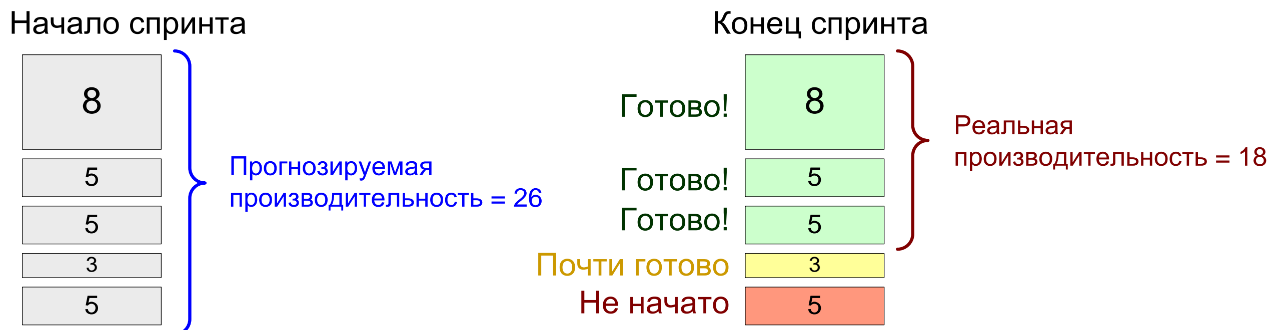
Этот подход включает в себя два этапа:

1. Определить *прогнозируемую производительность*.

2. Посчитать, сколько историй вы можете добавить без превышения прогнозируемой производительности.

Производительность является мерой “количества выполненной работы”. Она рассчитывается как сумма первоначальных оценок всех историй, которые были реализованы в течение спринта.

На следующем рисунке показан пример *прогнозируемой производительности* в начале спринта и *реальной производительности* в конце спринта. Каждый прямоугольник обозначает историю, число внутри прямоугольника – это его начальная оценка.



Помните, что реальная производительность расчитывается на основании *начальной оценки* каждой истории. Любые изменения оценки в течение спринта игнорируются.

Я уже слышу ваши возражения: “Какая от этого польза? Высокий или низкий уровень производительности зависит от миллиона факторов! Недалёкие программисты, неправильная начальная оценка, изменение объёма работ, незапланированные потрясения в ходе спринта и т.д.”

Согласен, производительность – это приблизительная величина. Но, тем не менее, очень полезная. Она лучше, чем ничего. Производительность даёт нам следующее: “Независимо от причин, мы имеем разницу между запланированным и выполненным объемом работ”.

А что, если история была *почти* закончена? Почему мы не используем дробные значения для таких историй при подсчете реальной производительности? Потому, что Scrum (как и гибкая разработка (agile), да и бережливое производство (lean)) ориентирован на то, чтобы создавать законченный, готовый к поставке продукт! Ценность реализованной наполовину истории нулевая (а то и отрицательная). См. книгу “Managing the Design Factory” автора Donald Reinertsen или одну из книг авторов Mary Poppendieck и Tom Poppendieck.

Так каким же магическим способом мы оцениваем производительность?

Проще всего оценить производительность, проанализировав предыдущие результаты команды. Какая производительность была в течение нескольких последних спринтов? Приблизительно такой же она будет и в следующем спринте.

Этот подход известен под названием *вчерашняя погода*. Он оправдан для тех команд, которые уже провели несколько спринтов (т.е. имеются статистические данные) и планируют следующий спринт без существенных изменений, т.е. тот же состав команды, рабочие условия и т.д. Конечно, это не всегда возможно.

Более продвинутый вариант оценки производительности заключается в определении доступных ресурсов. Допустим, мы планируем трёхнедельный спринт (15 рабочих дней). Команда состоит из 4-ёх человек. Лиза берёт два отгула. Дэйв сможет уделить проекту только 50% времени плюс берёт один отгул. Сложим всё вместе …

Доступные дни	
Том	15
Лиза	13
Сэм	15
Дэйв	7
	50 доступных человеко-дней

... получаем 50 доступных человеко-дней на спринт.

Получили ли мы прогнозируемую производительность? Нет! Потому что наша единица измерения – это *story point*, который, в нашем случае приблизительно равен “идеальному человеко-дню”. Идеальный человеко-день – это максимально продуктивный день, когда никто и ничто не отвлекает от основного занятия. Такие дни – редкость. Кроме того, нужно принимать во внимание, что в ходе спринта может быть добавлена незапланированная работа, человек может заболеть и т.д.

Без всякого сомнения, наша прогнозируемая производительность будет менее пятидесяти. Вопрос в том, насколько? Для ответа на него введём определения фокус-фактора.

Прогнозируемая производительность этого спринта

(доступные человеко-дни) x (фокус-фактор) = (прогнозируемая производительность)

Фокус-фактор – это коэффициент того, насколько команда сфокусирована на своих основных задачах. Низкий фокус-фактор может означать, что команда ожидает неоднократного вмешательства в свою работу или предполагает, что оценки слишком оптимистичны.

Выбрать разумный фокус-фактор лучше всего, взяв его из последнего спринта (а ещё лучше – среднее значение за несколько последних спринтов).

Фокус-фактор последнего спринта

$$(\text{фокус-фактор}) = \frac{(\text{действительная производительность})}{(\text{доступные человеко-дни})}$$

За *реальную производительность* принимается сумма начальных оценок для тех историй, которые были завершены в ходе последнего спринта.

Допустим, в ходе последнего спринта командой из трёх человек в составе Тома, Лизы и Сэма реализовано 18 story point’ов. Продолжительность спринта была 3 недели, что составляет 45 человеко-дней. Необходимо спрогнозировать производительность команды на будущий спринт. Слегка усложним задачу появлением Дэйва – нового участника команды. Принимая во внимание отгулы членов команды и другие вышеупомянутые обстоятельства, получим 50 человеко-дней.

Фокус-фактор последнего спринта:

$$40\% = \frac{18 \text{ story point'ов}}{45 \text{ человеко-дней}}$$

Прогнозируемая производительность этого спринта:

50 человеко-дней x 40% = 20 story point’ов

Таким образом, наша прогнозируемая производительность будущего спринта – это 20 story point’ов. Это означает, что команде следует включать истории в план спринта до тех пор, пока их сумма не будет примерно равна 20.

В нашем случае команда может выбрать 4 наиболее важные истории (что составляет 19 story point'ов) или 5 наиболее важных историй (24 story point'а). Остановимся на четырёх историях, т.к. их сумма близка к 20. Если возникают сомнения, выбирайте меньше историй.

Ввиду того, что выбранные 4 истории составляют 19 story point'ов, окончательная прогнозируемая производительность будущего спринта составляет 19.

Техника "вчерашней погоды" очень удобна, однако использовать её нужно, полагаясь на здравый смысл. Если последний спринт был необычайно плохим вследствие того, что все члены команды болели в течение недели, это вовсе не означает, что подобная ситуация повторится в ходе следующего спринта. Таким образом, фокус-фактор может быть увеличен. Если команда недавно внедрила сверхбыструю систему непрерывной интеграции, фокус-фактор также может быть увеличен. В случае, если к команде присоединился новый участник, фокус-фактор нужно уменьшить, принимая во внимание время, необходимое ему на то, чтобы влиться в проект, и на обучение. И т.д.

Для того, чтобы получать более достоверные оценки, по возможности используйте усредненные данные за последние несколько спринтов.

Что если команда новая и не имеет никакой статистики? В этом случае можно использоать фокус-фактор других команд, которые работают в похожих условиях.

Нет возможности взять данные других команд? Выберите фокус-фактор наугад. Хорошая новость состоит в том, что фокус-фактор придётся угадывать лишь для первого спринта. После первого спринта вы будете располагать статистическими данными и сможете непрерывно измерять и совершенствовать ваш фокус-фактор и прогнозируемую производительность.

В качестве "значения по умолчанию" фокус-фактора для новых команд мы обычно используем 70%, т.к. это именно тот предел, которого нашим командам удавалось достичь за всё время их работы.

Какую технику мы используем для планирования?

Я упоминал несколько подходов подсчёта производительности: на основе интуиции, на основе "вчерашней погоды" и на основе определения доступных ресурсов с учётом фокус-фактора.

Какой же подход используем мы?

Обычно мы совмещаем все перечисленные подходы. На это не требуется много времени.

Мы анализируем фокус-фактор и реальную производительность последнего спринта. И, проанализоровав доступные ресурсы, получаем фокус-фактор будущего спринта. Обсуждаем любые различия этих двух фокус-факторов и вносим необходимые коррективы.

Как только станет известен приблизительный список историй на будущий спринт, мы проводим поверку с использованием подхода, основанного на *интуиции*. Я прошу команду на некоторое время забыть о цифрах и просто прикинуть, насколько реально реализовать эти истории в одном спринте. Если кажется, что их много, то исключаем одну-две истории. И наоборот.

Вобщем, цель – принять решение о том, какие истории включать в спринт. А фокус-фактор, доступные ресурсы и прогнозируемая производительность являются лишь инструментами её достижения.

Почему мы используем учетные карточки

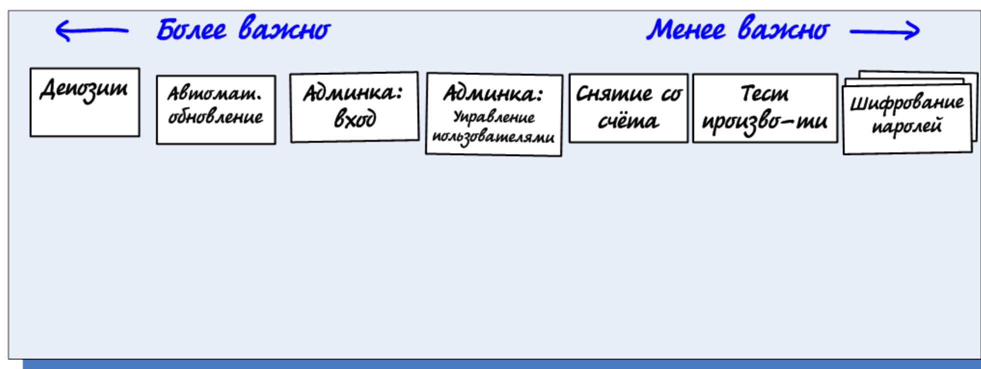
Большинство встреч по планированию спринта посвящены историям из product backlog'а: их оценке, расстановке приоритетов, уточнению требований, разбиению на части и т.д.

Как это происходит у нас?

Обычно команда включает проектор, который показывает backlog в Excel'е. Кто-то (обычно product owner или ScrumMaster) садится за клавиатуру, быстро зачитывает историю и начинает обсуждение. По мере того, как команда и product owner обсуждают приоритеты и детали, парень за клавиатурой обновляет историю прямо в Excel'е.

Неплохо, да? А вот и нет! На самом деле это фигня. И что ещё хуже, команда обычно не замечает, что это фигня, аж до конца встречи, когда оказывается, что ещё куча историй не обработана!

Есть способ получше – сделать *учетные карточки* и *прикрепить их на стену* (или на большой стол).



Такой "пользовательский интерфейс" выигрывает по сравнению с компьютером и проектором, по следующим причинам:

- Люди вынуждены вставать и ходить, поэтому они более сконцентрированы и не засыпают.
- Каждый чувствует себя причастным к процессу (а не только парень за клавиатурой).
- Можно редактировать несколько историй одновременно.
- Изменить приоритеты очень просто – достаточно поменять местами учетные карточки.
- После окончания встречи учетные карточки можно забрать в комнату, где работает команда, и использовать их на доске задач (см. стр. 38 "Как мы создаём sprint backlog?").

Вы можете заполнить учетные карточки собственноручно или (как мы обычно делаем) сгенерировать версию для печати прямо из product backlog'а, используя простой скрипт.

Задача №55 из backlog'a

Депозит

Примечания

Нужна UML диаграмма последовательности. Пока что не стоит беспокоиться про криптографическую защиту.

Как продемонстрировать

Войти в систему, открыть страницу депозита, положить на счет €10, перейти на страницу баланса и проверить, что он увеличился на €10.

Важность

30

Оценка

PS – скрипт можно найти в моём блоге на http://blog.crisp.se/henrikkniberg.

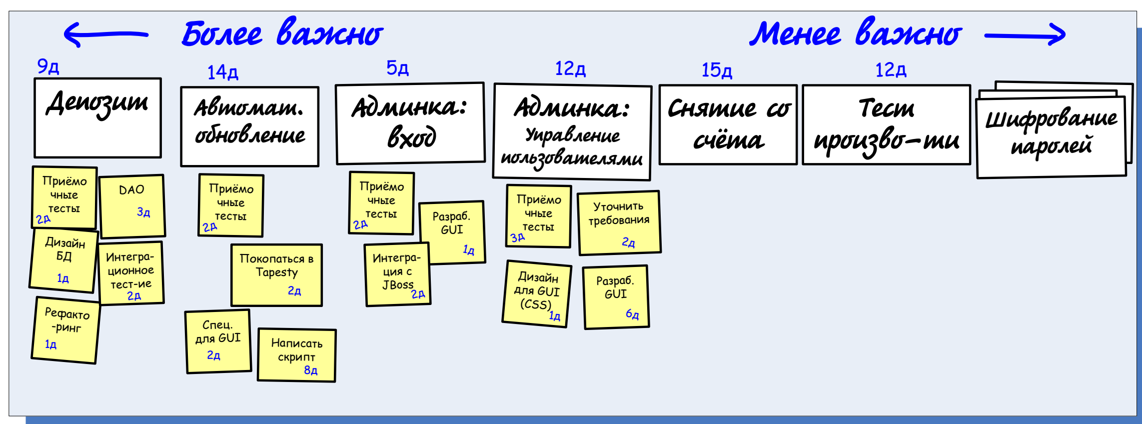
Важно: После планирования спринта наш ScrumMaster вручную обновляет product backlog в Excel'е, чтобы учесть все изменения, которые были сделаны на карточках. Да, это определённая морока, но мы на это согласны с учетом того, насколько эффективнее проходят встречи по планированию спринта с использованием бумажных учетных карточек.

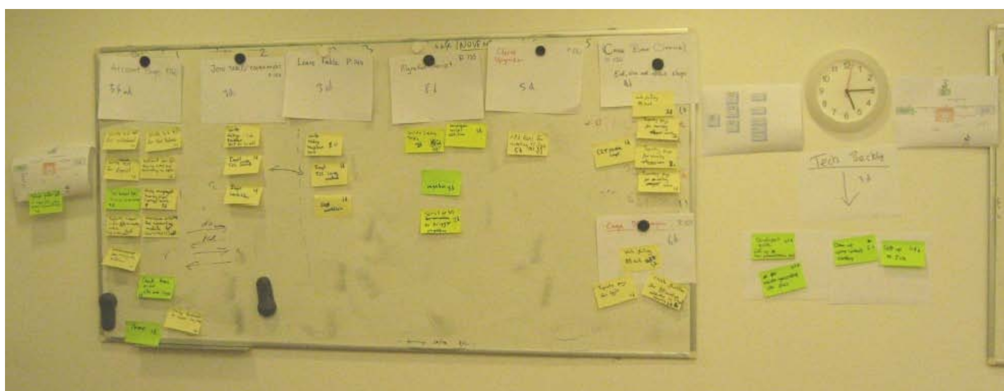
Несколько слов о поле "Важность" (Importance). Это значение "важности" из product backlog'а в Excel'е на момент распечатки карточки. Её обозначение на карточке помогает нам отсортировать карточки на стене. Обычно мы располагаем более важные элементы левее, а менее важные – правее. Однако, когда карточки уже на стене, можно забыть о значении поля "важность" и, вместо этого, использовать порядок, чтобы показать относительную важность истории. Если product owner меняет местами карточки, не тратьте время на обновление значений на бумажках. Просто после встречи убедись, что обновили значения степени важности историй в product backlog'е.

Историю обычно можно оценить легче и точнее, если она разбита на задачи. На самом деле мы использует термин "действие" (activity), потому что слово "задача" (task) значит на шведском кое-что *совершенно* другое :о) [прим. переводчика – шведско-русский онлайн словарь находится по адресу http://lexin.nada.kth.se/swe-eng.html]

Использование карточек также упрощает процедуру разбиения историй на задачи. Можно разбить команду на пары, тогда они смогут одновременно разбивать истории на задачи – каждая свою.

На нашей доске мы отображаем задачи в виде маленьких стикеров под каждой историей. Каждый стикер соответствует одной задаче в рамках этой истории.





Мы не добавляем задачи в product backlog в Excel'е по двум причинам:

- Список задач обычно часто меняется: к примеру, задачи могут изменяться и пересматриваться на протяжении sprint'а. Чтобы синхронизировать с ними product backlog в Excel'е нужно слишком много усилий.
- Скорее всего, на этом уровне детализации product owner не будет участвовать в процессе.

Так же, как и учетные карточки, стикеры с задачами можно использовать в sprint backlog'е (см. стр. 38 "Как мы создаём sprint backlog?").

Критерий готовности

Важно, чтобы и product owner, и команда совместными усилиями определили критерий готовности. Можно ли считать историю готовой, если весь код был добавлен в репозиторий? Или же она считается готовой, лишь после того как была развёрнута на тестовом сервере и прошла все интеграционные тесты? Где только это возможно, мы используем следующее определение критерия готовности: “история готова к развёртыванию на живом сервере”, однако, иногда мы вынуждены иметь дело с определением типа “развёрнуто на тестовом сервере и готово к приёмочному тестированию”.

Поначалу мы использовали детализированные контрольные листы для определения того, что история готова. А сейчас мы часто просто говорим: “история готова тогда, когда так считает тестировщик из нашей Scrum-команды”. В этом случае проверка того, что пожелания product owner’а были правильно восприняты командой, остаётся на совести тестировщика. В его задачи также входит контроль того, что история достаточно “готова” для того, чтобы удовлетворить принятому критерию готовности.

В конце концов, мы осознали, что нельзя все истории ровнять под одну гребёнку. История “форма поиска пользователей” будет очень сильно отличаться от истории под названием “руководство по эксплуатации”. В последнем случае “готово” может просто означать “принято отделом поддержки клиентов”. Поэтому здравый смысл часто лучше, чем формальный контрольный лист.

Если вы часто путаетесь с определением критерия готовности (как это было поначалу у нас), то вам, наверное, стоит ввести поле “критерий готовности” для каждой истории.

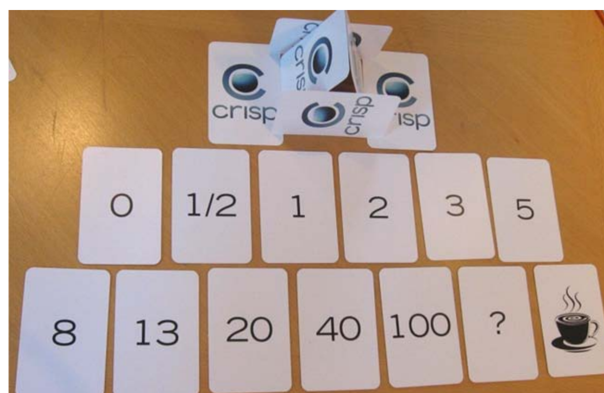
Оценка трудозатрат с помощью игры в planning poker

Оценка – это командная работа, и, зачастую, все члены команды участвуют в оценке каждой истории. Почему?

- Во время планирования мы обычно не знаем, кто будет выполнять ту или иную часть.
- Реализация историй обычно требует участия различных специалистов (дизайн пользовательского интерфейса, кодирование, тестирование, и т.д.).
- Для того, чтобы каждый участник команды мог выдать какую-то оценку, он должен более или менее понимать, в чём суть этой истории. Получая оценку от каждого члена команды, мы убеждаемся, что все понимают, о чём идёт речь. Это увеличивает вероятность взаимопомощи по ходу спринта. А также это увеличивает вероятность того, что наиболее важные вопросы по этой истории всплывут как можно раньше.
- При оценке истории совместными усилиями разностороннее видение проблемы приводит к сильному разбросу оценок. Такие разногласия лучше выявлять и обсуждать как можно раньше.

Если попросить всех оценить историю, то обычно человек, который понимает её лучше остальных, выдаст оценку первым. К несчастью это сильно влияет на оценки других людей.

Но существует прекрасная практика, которая позволяет этого избежать. Она называется planning poker (придуманная Майком Коном, насколько я знаю).



Каждый член команды получает колоду из 13-ти карт, таких же, как на картинке выше. Всякий раз, когда нужно оценить историю, каждый член команды выбирает карту с оценкой (в story point'ах), которая, по его мнению, подходит, и кладёт её на стол рубашкой наверх. Когда всё члены команды определились с оценкой, карты одновременно вскрываются. Таким образом, члены команды вынуждены оценивать самостоятельно, а не "списывать" чужую оценку.

Если получается большая разница в оценках, то эту разницу обсуждают и пытаются выработать общее понимание того, что должно быть сделано для реализации этой истории. Возможно, они разобьют задачу на более мелкие. После этого команда оценит историю заново. Этот цикл должен повторяться до тех пор, пока оценки не сойдутся, т.е. не станут примерно одинаковыми.

Очень важно напоминать всем членам команды, что они должны оценивать общий объём работ по истории, а не только "свою часть". Тестировщик должен оценивать не только работы по тестированию.

Заметьте, последовательность значений на картах – нелинейная. Вот, например, между 40 и 100 ничего нет. Почему так?

Это нужно, чтобы избежать появления ложного чувства точности для больших оценок. Если история оценивается примерно в 20 story point'ов, то нет смысла обсуждать должна ли она быть 20, или 18, или 21. Всё, что нам нужно знать, это то, что её сложно оценить. Поэтому мы примерно назначаем ей оценку в 20.

Если у вас возникло желание более детально переоценить эту историю, то лучше разбейте её на более мелкие части и оцените уже их!

И, кстати, жульничать, выкладывая карты 5 и 2, чтоб получить 7, нельзя. Вы должны выбрать или 5 или 8 – семёрки нет.

Есть ещё несколько специальных карт:

- 0 = или "история уже готова" или же её оценка "пара минут работы".
- ? = "Я понятия не имею. Абсолютно".
- Чашка кофе = "Я слишком устал, чтобы думать. Давайте сделаем перерыв".

Уточнение описаний историй

Нет ничего ужасней, чем ситуация, когда команда с пафосом демонстрирует новую функциональность продукта, а product owner тяжко вздыхает и говорит: "ну да – всё это красиво, вот только *не то, что я просил*!"

Как убедиться, что product owner и команда понимают историю одинаково? Или что все члены команды понимают все истории одинаково? Да никак. Есть простые способы выявить разницу в понимании. Наиболее простая практика – всегда заполнять все поля для каждой истории (точнее, для всех историй, которые могут попасть в текущий спринт).

Пример №1:

Команда и product owner вполне довольны планом на спринт и уже готовы закончить планирование, но тут ScrumMaster говорит: "Минуточку! У нас нет оценки для истории "добавить пользователя". Давайте-ка оценим!". После пары сдач в planning poker, команда сходится на оценке в 20 story point'ов, на что product owner вскакивает с криком: "ЧЕГООО?!?". Пара минут ожесточённых споров и вдруг выясняется, что команда имела в виду "удобный web-интерфейс для функций *добавить*, *редактировать*, *удалить* и *искать пользователей*", а product owner имел в виду только "добавлять пользователей напрямую в базу данных с помощью SQL-клиента". Команда оценивает историю заново и останавливается на 5-ти story point'ах.

Пример №2:

Команда и product owner вполне довольны планом на спринт и уже готовы закончить планирование, но тут Scrum master говорит: "Минуточку! Вот тут у нас есть история "добавить пользователя"… Как она может

быть продемонстрирована?”. Народ пошепчется и через минуту кто-то встанет и начнёт: “ну, для начала надо залогиниться на сайт, потом…”, а product owner тут же перебьёт: “залогиниться на сайт?! Не-не-не-не… эта функция вообще к сайту не должна иметь никакого отношения – это будет просто маленький SQL-скрипт, только для администраторов”.

Поле “как продемонстрировать” может (и должно) быть *очень кратким!* Иначе вы не успеете вовремя закончить планирование спринта. По сути, это лаконичное описание на обычном русском языке, как вручную выполнить наиболее общий тестовый пример: “Сделать это, потом это, потом проверить, что получилось так-то”.

И я понял, что такое простое описание *часто* позволяют обнаружить разное понимание объёма работ для историй. Хорошо ведь узнать об этом заранее, не так ли?

Разбиение историй на более мелкие истории

Истории должны быть не слишком маленькими, но и не слишком большими (в смысле оценок). Если вы получили кучу историй в половину story point’а, то вы наверняка падёте жертвой микроменеджмента. С другой стороны, история в 40 story point’ов несёт в себе риск того, что к концу спринта её успеют закончить лишь *частично*, а незавершённая история не представляет ценности для вашей компании, она только увеличивает накладные расходы. Дальше – больше: если ваша прогнозируемая производительность 70 story point’ов, а две наиболее важные истории оценены в 40, то планирование несколько усложнится. Команда станет перед выбором: или расслабиться (т.е. включить в спринт только одну историю), или взять на себя невыполнимые обязательства (т.е. включить обе).

Я считаю, что практически всегда есть возможность разбить историю на более мелкие. Однако, в этом случае нужно следить за тем, чтобы меньшие истории всё ещё представляли ценность с точки зрения бизнеса.

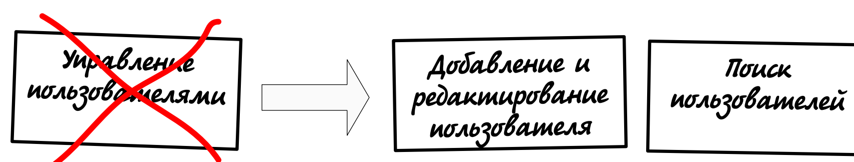
Обычно мы стремимся получить истории объёмом от двух до восьми человеко-дней. Производительность нашей среднестатистической команды обычно находится в пределах 40-ка – 60-ти человеко-дней, что позволяет нам включать в спринт примерно по 10 историй. Иногда всего 5, а иногда целых 15. Кстати, таким числом учётных карточек достаточно удобно оперировать.

Разбиение историй на задачи

Секундочку… В чём разница между “задачами” и “историями”? Очень правильный вопрос.

А различие очень простое: истории это нечто, что можно продемонстрировать, что представляет ценность для product owner’а, а задачи либо нельзя продемонстрировать, либо они не представляют ценности для product owner’а.

Пример разбиения истории на более мелкие:



Пример разбиения истории на задачи:

Несколько интересных наблюдений:

- Молодые Scrum-команды не любят тратить время на предварительное разбиение историй на задачи. Некоторые считают это “водопадным” подходом.
- Абсолютно понятные истории разбивать на задачи заранее так же легко, как и по мере их выполнения.
- Такая разбивка часто позволяет выявить дополнительную работу, которая увеличивает оценку, чем обеспечивается более реалистичный план на спринт.
- Такая предварительная разбивка заметно увеличивает эффективность ежедневного Scrum’а (см. стр. 46 “Как мы проводим ежедневный Scrum”).
- Даже неточная разбивка, которая будет изменяться по ходу работ, всё равно даёт нам все перечисленные выше выгоды.

Итак, чтобы успеть разбить истории на задачи, мы стараемся выделить достаточно времени на планирование спринта. Однако, если время поджимает, то разбиение на задачи мы можем и пропустить (см. следующую главу “Когда пора остановиться”).

Примечание: мы практикуем TDD (разработку через тестирование), из-за чего первой задачей почти каждой истории является “написать приёмочный тест”, а последняя – “рефакторинг” (улучшение читабельности кода и удаление повторений кода).

Выбор времени и места для ежедневного Scrum'а

Все часто забывают, что на планировании спринта, помимо всего прочего, необходимо выбрать время и место проведения ежедневного Scrum'а. Без этого ваш спринт обречён на неудачный старт. Ведь первый ежедневный Scrum – это, по большей части, ввод мяча в игру, когда каждый решает с чего начать работу.

Я предпочитаю проводить ежедневный Scrum утром. Хотя, должен признаться, мы особо и не пробовали проводить его в обед или ближе к вечеру.

Недостатки обеденного Scrum'а: приходя на работу утром, вам надо попытаться вспомнить, чем вы обещали команде заниматься сегодня.

Недостатки утреннего Scrum'а: приходя на работу утром, вам надо попытаться вспомнить, чем вы занимались вчера, чтобы можно было отчитаться об этом.

Мне кажется, первый недостаток хуже, так как наиболее важно то, что вы *собираетесь делать*, а не то, что вы *уже сделали*.

Мы обычно выбираем самое раннее время, которое не вызывает стонов ни у кого в команде. Обычно это 9:00, 9:30 или 10:00. Очень важно, чтобы все в команде искренне согласились на это время.

Когда пора остановиться

Ну вот, время заканчивается. Чем мы можем пожертвовать из всего того, что мы собирались сделать на планировании спринта, если время поджимает?

У меня, например, приоритеты для встречи по планированию спринта такие:

Приоритет №1: Цель спринта и дата демонстрации. Это тот минимум, с которым можно начинать спринт. У команды есть цель и крайний срок, а работать они могут прямо по product backlog'у. Да это нехорошо, и нужно запланировать ещё одну встречу по планированию sprint backlog'а на завтра, но если вам крайне необходимо стартовать спринт, то это, скорее всего, сработает. Хотя, если быть честным, я так никогда и не стартовал спринт, имея всего лишь цель и дату.

Приоритет №2: Список историй, которые команда включила в sprint backlog.

Приоритет №3: Оценки для каждой истории из sprint backlog'а.

Приоритет №4: Поле "Как продемонстрировать" для каждой истории из sprint backlog'а.

Приоритет №5: Расчёты производительности и ресурсов в качестве "испытания реальностью" для плана на спринт. Также нужен список членов команды с указанием их степени участия в проекте (без этого нельзя рассчитать производительность).

Приоритет №6: Определённое время и место проведения ежедневного Scrum'а. Вообще, выбрать время и место можно за одну минуту, но если время собрания уже закончилось, то ScrumMaster просто выбирает их сам после собрания и оповещает всех по электронной почте.

Приоритет №7: Истории, разбитые на задачи. Разбивать истории на задачи можно также и по мере их поступления в работу, совмещая это с ежедневным Scrum'ом, но такой подход слегка нарушает течение спринта.

Технические истории

А теперь ещё одна сложная проблема: технические истории. Это нефункциональные требования, или как их там ещё называют.

Я имею в виду всё, что должно быть сделано, но невидимо для заказчика, не относится ни к одной user story, и не даёт прямой выгоды product owner'у.

Мы называем это "техническими историями".

Например:

- **Установить сервер непрерывной интеграции**
    - Почему это надо сделать? Потому, что это сэкономит много времени разработчикам и снизит риск проблемы "большого взрыва" при интеграции в конце итерации.
- **Описать архитектуру системы**
    - Почему это надо сделать? Потому, что разработчики часто забывают общую архитектуру и поэтому пишут несогласованный код. Нужен документ, предоставляющий всем одинаковую общую картину.
- **Рефакторинг слоя доступа к данным**
    - Почему это надо сделать? Потому, что слой доступа к данным стал очень запутанным, и разработчики теряют много времени на то, чтобы разобраться и исправить возникающие дефекты. Рефакторинг сохранит время всей команды и повысит устойчивость системы.
- **Обновить Jira (систему учёта дефектов)**
    - Почему это надо сделать? Текущая версия очень нестабильная и медленная: обновление сохранит время всей команде.

Имеют ли смысл эти истории? Или это задачи не связанные ни с одной историей? Кто задаёт приоритеты? Нужно ли вовлекать сюда product owner'а?

Мы попробовали различные варианты работы с техническими историями. Мы пробовали считать их самыми обычными user story. Это была неудачная идея: для product owner'а приоритезировать их в product backlog'е было всё равно, что сравнить тёплое с мягким. По очевидным причинам технические истории получали самый низкий приоритет с объяснением: "Да, ребята, несомненно, ваш сервер непрерывной интеграции – очень важная штука, но давайте сперва реализуем кое-какие прибыльные функции? После этого вы можете прикрутить вашу техническую конфетку, окей?"

В некоторых случаях product owner действительно прав, но чаще все-таки нет. Мы пришли к выводу, что product owner не всегда компетентен, чтобы идти на компромисс. И вот что мы делаем:

1. Стараемся избегать технических историй. Ищем способ превратить техническую историю в нормальную историю с измеряемой ценностью для бизнеса. В таком случае у product owner'а больше шансов найти разумный компромисс.
2. Если мы не можем превратить техническую историю в обычную, смотрим нельзя ли включить эту работу в уже существующую историю. Например, "рефакторинг доступа к данным" мог бы стать частью истории "редактировать пользователя", поскольку она подразумевает работу с данными.
3. Если оба подхода не прошли, отмечаем это как техническую историю и храним список таких историй отдельно. Пусть product owner видит список, но не редактирует. В переговорах с product owner'ом используем параметры “фокус-фактора” и “прогнозируемой производительности” и выделяем время в спринте для реализации технических историй.

Пример (диалог очень похож на то, что случилось во время одного из наших планирований спринта).

Команда: “У нас есть кое-какие внутренние технические работы, которые должны быть сделаны. Мы бы хотели заложить на это 10% всего времени, т.е. снизить фокус-фактор с 75% до 65%. Это возможно?”

Product owner: “Естественно, нет! У нас нет времени!”

Команда: “Хорошо, давайте посмотрим на наш последний спринт (все бросают взгляд на график производительности на белой доске). Наша прогнозируемая производительность была 80, но реальная производительность оказалась 30, верно?”

Product owner: “Именно! У нас нет времени на внутренние технические работы! Нам нужен новый функционал!”

Команда: “Хорошо. Но *причина* ухудшения нашей производительность в том, что мы тратим слишком много времени на создание полной сборки для тестирования”.

Product owner: “Ну и что?”

Команда: “А то, что производительность и дальше будет такой низкой, если мы ничего не сделаем”.

Product owner: “Да ... и что вы предлагаете?”

Команда: “Мы предлагаем выделить примерно 10% следующего спринта на установку билд сервера и другие вещи, чтобы сделать интеграцию менее болезненной. Это, скорее всего, увеличит производительность всех последующих спринтов *как минимум* на 20%!”

Product owner: “Серьёзно? Почему же мы это не сделали на предыдущем спринте?!”

Команда: “Хм... потому что вы не захотели, чтобы мы это сделали...”

Product owner: “О! Ммм..., ладно, тогда логично, если вы это сделаете сейчас!”

Конечно, есть и другой вариант: не вести переговоры с product owner'ом по поводу технических историй, а просто поставить его перед фактом, что у нас фокус-фактор такой-то. Но это не правильно даже *не попытаться* достичь компромисса.

Если product owner оказался сообразительным и компетентным (нам в своё время с этим действительно повезло), я бы рекомендовал информировать его как можно лучше и дать ему возможность определять общие приоритеты. Ведь прозрачность – это один из основополагающих принципов Scrum'а, верно?

Как мы используем систему учёта дефектов для ведения product backlog'a

Есть ещё одна непростая задача. С одной стороны, Excel очень хороший формат для product backlog'a. С другой стороны, вам всё равно нужна система учёта дефектов, и Excel здесь явно не тянет. Мы используем Jira.

Итак, как мы переносим задачи из Jira в планирование спринта? Не можем же мы просто их проигнорировать и сосредоточиться только лишь на историях.

Мы пробовали следующие подходы:

1. Product owner распечатывает самые высокоприоритетные задачи из Jira, выносит их на планирование спринта и вешает их на стенку с другими историями (неявно указывая их относительный приоритет).
2. Product owner создаёт истории, соответствующие задачам из Jira. Например, “Исправить самые критические ошибки отчётности в админке, Jira-124, Jira-126, и Jira-180”.
3. Работы по исправлению ошибок не включаются в спринт, то есть команда определяет довольно низкий фокус-фактор (например, 50%), чтобы хватало времени на исправления. Затем, вводится предположение, что команда в каждую итерацию будет тратить определённую часть времени на ошибки в Jira.
4. Заносим product backlog в Jira (просто переносим из Excel'a). Считаем баги обычными историями.

Мы ещё не определились, какой подход для нас самый лучший; в действительности он может отличаться в разных командах и меняться от спринта к спринту. Я больше склоняюсь к первому подходу: он прост и понятен.

Свершилось! Планирование спринта закончено!

Ух, я и не думал, что глава по планированию спринта будет такой длинной! [прим. переводчика: я, если честно, тоже :)] Полагаю, этот факт отражает моё мнение: планирование спринта – самая важная вещь в Scrum'e. Вложите побольше усилий в планирование – и всё остальное пойдёт как по маслу.

Планирование спринта прошло успешно, если все (и команда, и product owner) с улыбкой завершают встречу, с улыбкой просыпаются следующим утром и с улыбкой проводят первый ежедневный Scrum.

Затем, конечно, всё может пойти криво, но вы, как минимум, не сможете списать всю вину на планирование спринта :o)

5

Как мы доносим информацию о спринте до всех в компании

Важно информировать всю компанию о том, что происходит в вашей команде. Если этого не делать, то остальные начнут жаловаться, или – что ещё хуже – придумывать всякие ужасы про вас.

Мы для этой цели используем “страницу с информацией о спринте”.

Команда ”Джокер”, спринт №15

Цель спринта

- Релиз, готовый к бета-тестированию!

Sprint backlog (в скобках – оценки)

- Депозит (3)
- Автоматическое обновление (8)
- Админка: вход (5)
- Админка: управление пользователями (5)

Прогнозируемая производительность: 21

Расписание

- Спринт: с 2006-11-06 по 2006-11-24
- Ежедневный scrum: 9:30 – 9:45, в главной комнате
- Демонстрация: 24/11/2006, 13:00, в кафетерии

Команда

- Джим
- Эрика (ScrumMaster)
- Том (75%)
- Ева
- Джон

Иногда мы также добавляем к названию истории поле ”как продемонстрировать”

Сразу же после встречи по планированию спринта эту страницу создаёт ScrumMaster. Он помещает её в wiki, и тут же спамит на всю компанию:

Тема письма: Спринт №15 у команды ”Джокер” начался

Всем привет! Команда ”Джокер” только что стартовала спринт №15. Наша цель – продемонстрировать релиз, готовый к бета-тестированию, 24-го ноября.

Страница информации о спринте доступна по адресу:
http://wiki.mycompany.com/jackass/sprint15

Кроме этого у нас есть ”панель” в wiki, в которой содержаться ссылки на текущие спринты всех команд.

Корпоративная панель

Текущие спринты

- Команда X, спринт №15
- Команда Y, спринт №12
- Команда Z, спринт № 1

В дополнение ко всему этому наш ScrumMaster распечатывает страницу информации о спринте и вывешивает её на стену в коридоре. Таким образом кто угодно, проходя мимо, может взглянуть на эту страницу и узнать, чем же занимается наша команда.

Когда спринт подходит к концу, ScrumMaster напоминает всем про приближающуюся демонстрацию.

Тема письма: Завтра в 13:00 команда ”Джокер” проводит демонстрацию в кафетерии

Всем привет! Все приглашаются на демонстрацию спринта №15 команды ”Джокер” завтра в 13:00 в кафетерии. Будет продемонстрирован релиз, готовый к бета-тестированию.

Страница информации о спринте доступна по адресу:

http://wiki.mycompany.com/jackass/sprint15

Если всё это делать, то ни у кого не получится сказать, что он *не мог узнать*, чем занимается команда.

6

Как мы создаём sprint backlog

Уже добрались до этой главы? Отлично!

Итак, мы только что закончили планирование и протрубили на весь мир про начало нашего новоиспечённого спринта. Теперь настал черёд ScrumMaster'а создать sprint backlog. Это необходимо сделать *после* планирования спринта, но *до* начала первого ежедневного Scrum'а.

Формат sprint backlog'а

Мы поэкспериментировали с разными форматами sprint backlog'а, включая Jira, Excel, не забыли попробовать и обычную настенную доску. Сперва в основном использовали Excel, в интернете можно найти довольно много Excel'евских шаблонов для sprint backlog'ов, с авто-генерацией burndown диаграмм и всякими другими примочками. Я мог бы долго рассказать про sprint backlog'и, созданные при помощи Excel, но я не буду. Даже не стану приводить примеров.

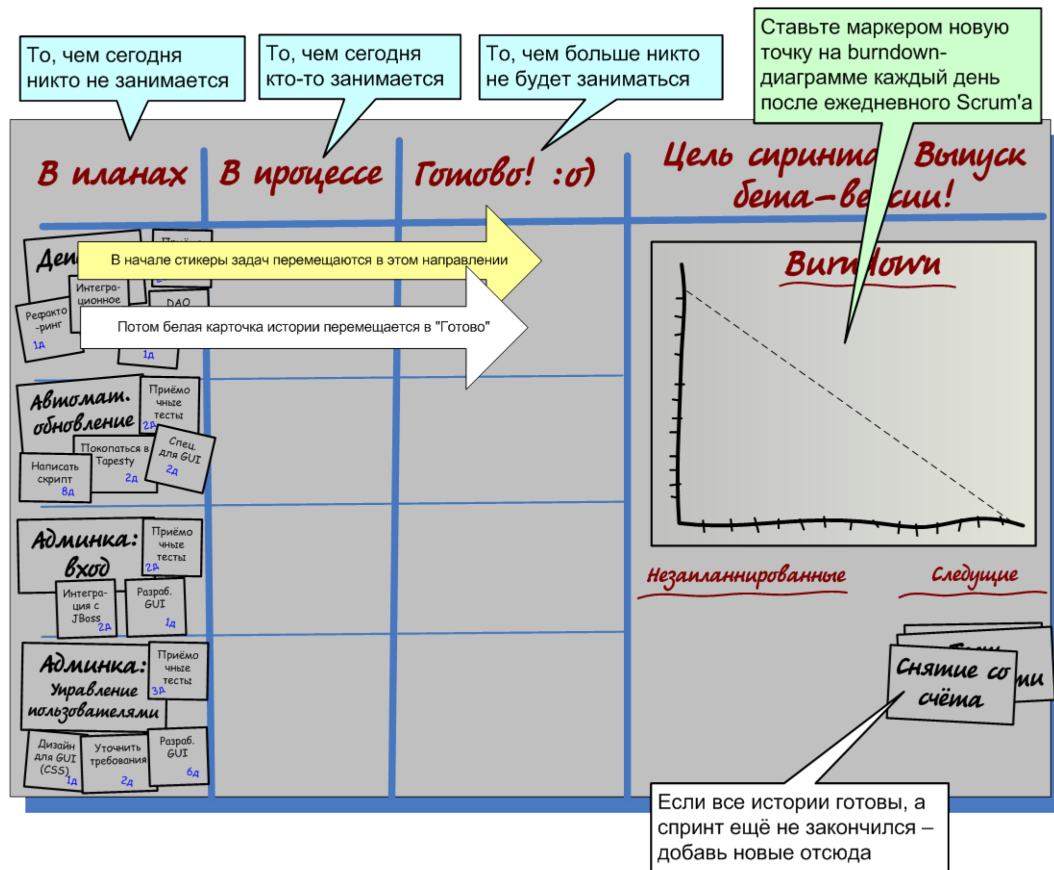
Вместо этого я опишу во всех подробностях формат sprint backlog'а, который мы сочли наиболее эффективным – доску задач на стене.

Найдите большую стену, на которой либо ничего нет, либо висит всякая ерунда вроде логотипа компании, старых диаграмм или ужасных картинок. Очистите её (если необходимо, спросите разрешения). Склейте скотчем большущее полотно бумаги (как минимум 2x2 или 3x2 метра для большой команды). А потом сделайте что-то вроде этого:

Как вариант, можно использовать белую доску. Но такое её использование неэффективно. Если это возможно, сохраните белую доску для набросков будущего дизайна, а в качестве доски для задач используйте "бумажные стены".

Примечание: если вы пользуетесь стикерами для задач, не забудьте прикрепить их скотчем, или же в один "прекрасный" день вы найдете их аккуратной кучкой на полу.

Как работает доска задач

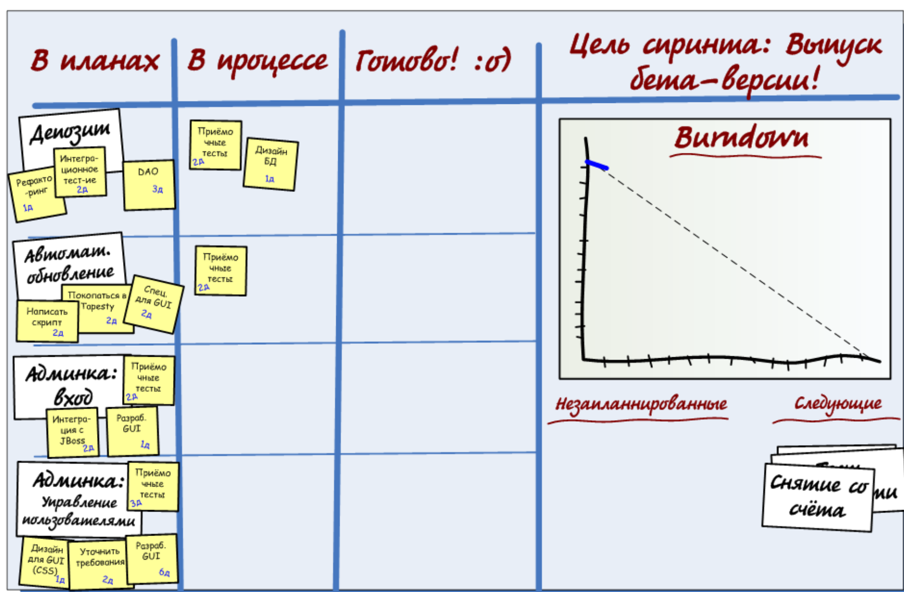


Вы, конечно, можете добавить любые дополнительные поля. Например, "В ожидании интеграционного тестирования" или "Отменённые". Но прежде чем всё усложнять, хорошенько подумайте, действительно ли эти дополнения *так уж* необходимы?

Я понял, что простота крайне ценна в такого рода вещах, поэтому я делаю усложнения только в случае, если цена *неделания* слишком велика.

Пример 1 – после первого ежедневного Scrum'a

После первого ежедневного Scrum'a доска задач может выглядеть примерно так:

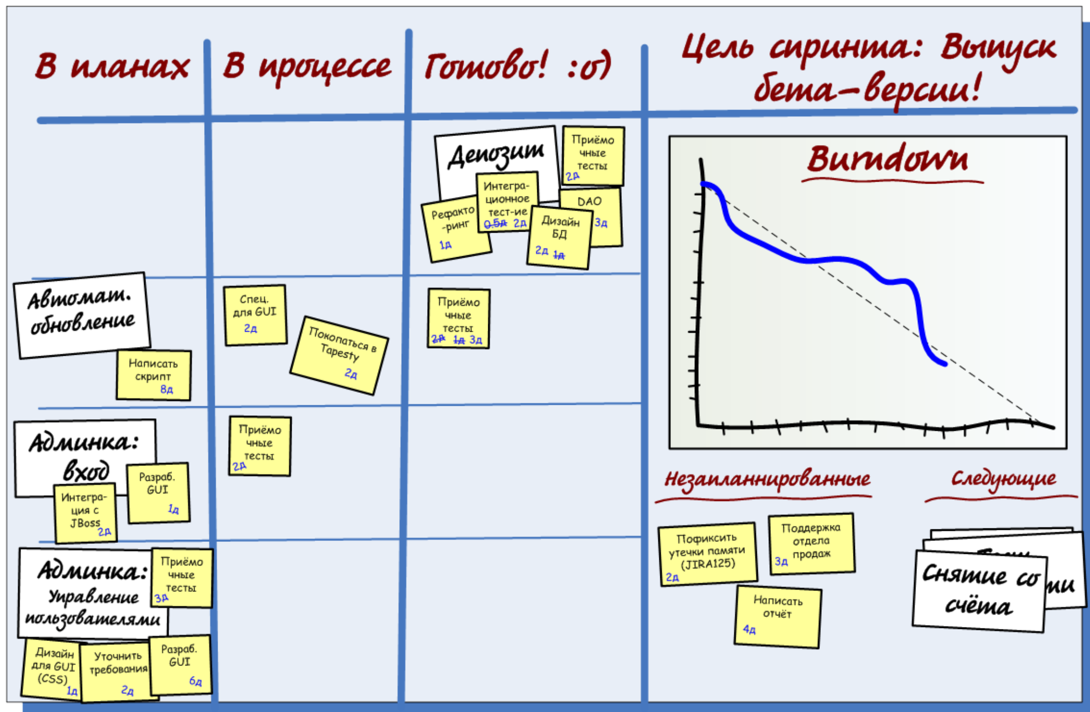


Как видно, три задачи находятся "в процессе", то есть команда будет заниматься ими сегодня.

Иногда, в больших командах, задача зависает в состоянии "в процессе" потому, что никто не помнит, кто над ней работал. Если такое случается часто, команда обычно принимает меры. Например, отмечает на карточке задачи имя человека, который взялся над ней работать.

Пример 2 – еще через пару дней

Через пару дней доска задач может выглядеть примерно так:



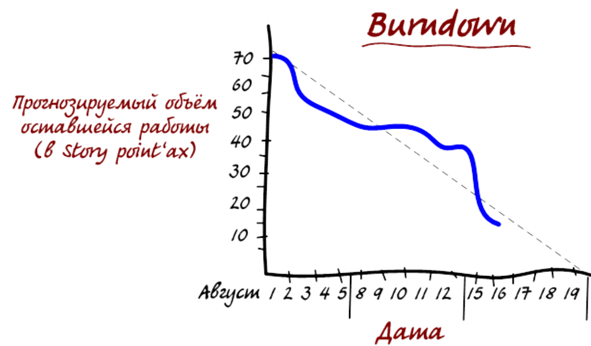
Как видно, мы закончили историю "Депозит" (т.е. она была зафиксирована в системе контроля версий, протестирована, отрефакторена и т.д.) "Автоматическое обновление" сделано частично, "Админка: вход" начат, а "Админка: управление пользователями" еще нет.

У нас возникло 3 незапланированные задачи, как видно справа внизу. Об этом полезно будет вспомнить на ретроспективе.

Вот пример настоящего sprint backlog'a ближе к концу спринта. Он, в самом деле, становится довольно беспорядочным по ходу спринта, но ничего страшного, так как это ненадолго. На каждый новый спринт мы начинаем sprint backlog с чистого листа.

Как работает burndown-диаграмма

Давайте присмотримся к burndown-диаграмме:



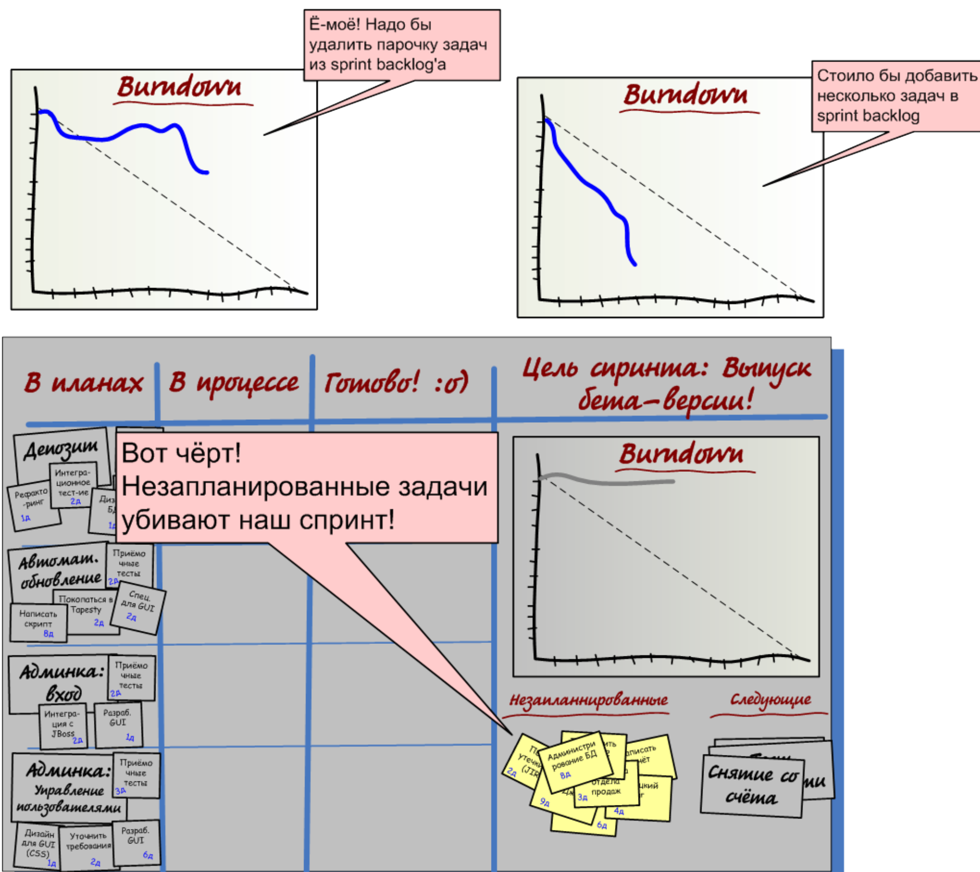
Вот о чем она говорит:

- В первый день спринта, 1-го августа, команда определила, что работы осталось примерно на 70 story point'ов. Это как раз и есть *прогнозируемая производительность* на этот спринт.
- 16-го августа работы осталось примерно на 15 story point'ов. Пунктирная направляющая показывает, что они на верном пути, то есть в таком темпе они успеют закончить все задачи к концу спринта.

Мы пропускаем выходные по оси X, так как в это время редко что-то делается. Раньше мы их включали, но burndown при этом выглядел страннovато, так как он "выравнивался" на выходных, и это походило на повод для беспокойства.

Тревожные сигналы на доске задач

Беглый взгляд на доску задач должен дать возможность любому человеку понять, насколько успешно продвигается итерация. ScrumMaster несёт ответственность за то, чтобы команда принимала соответствующие меры при обнаружении первых тревожных симптомов:



Эй, как насчет отслеживания изменений?

Лучший вариант отслеживания изменений, который я могу предложить при данном подходе – это делать фотографию доски задач каждый день. Делайте так, если это необходимо. Я тоже иногда так делаю, хотя ещё никогда не возникало необходимости пересматривать эти фотографии позже.

Если отслеживание изменений для вас очень важно, тогда возможно подход с доской задач вам вообще не подходит.

И все-таки, я бы предложил вам сначала оценить значение детального отслеживания изменений в спринте. После того как спринт закончен и рабочий код вместе с документацией был отослан заказчику, будет ли кто-нибудь разбираться, сколько историй было закончено на 5й день спринта? Будет ли кто-нибудь волноваться о том, какова была реальная оценка для задачи "Написать приемочный тест для истории Депозит" после того, как история была закончена?

Как мы оцениваем: в днях или часах?

В большинстве книг и статей, посвящённых Scrum'у, для оценки времени выполнения задач используются часы, а не дни. И мы так раньше делали. Общая формула была такой: один эффективный человеко-день равен шести эффективным человеко-часам.

Однако мы отказались от этой практики по следующим причинам:

- Оценки в человеко-часах чересчур мелкие, что приводит к появлению большого количества крохотных задач по часу или два, и, как результат, к микроменеджменту (micromanagement).
- Оказалось, что всё равно все оценивают в человеко-днях, а когда нужно получить оценку в человеко-часах, просто умножают дни на шесть. "Хм, эта задача займёт примерно день. Ага, я должен дать оценку в часах. Что ж, значит шесть часов".
- Две разных единицы измерения вводят в заблуждение. "Это оценка в человеко-днях или человеко-часах?"

Поэтому мы используем человеко-дни в качестве основной единицы при оценке времени (мы называем их story point'ами). Наше наименьшее значение – 0.5. Т.е. любые задачи, оцененные менее чем в 0.5, либо удаляются, либо комбинируются с другими, либо оценка остаётся 0.5 (ничего страшного в слегка завышенной оценке нет). Изящно и просто.

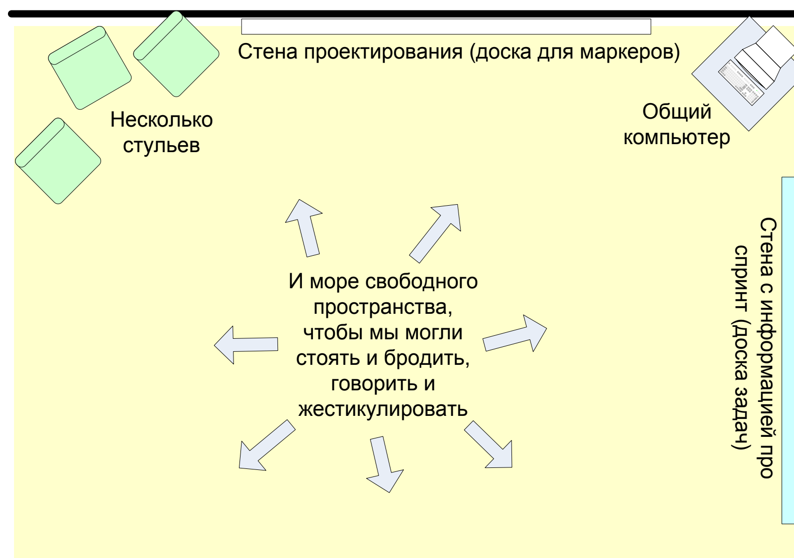
7

Как мы обустроили комнату команды

Уголок обсуждений

Я заметил, что большинство интересных и полезных обсуждений возникают спонтанно, прямо перед доской задач.

Поэтому мы пытаемся оформить это место как отдельный "уголок обсуждений"



Это и в самом деле полезно. Нет лучшего способа посмотреть на систему в целом, чем стать в уголок обсуждений, посмотреть на обе стены, а потом сесть за компьютер и пощелкать последний билд системы (в случае, если вам повезло, и у вас внедрена практика "непрерывной интеграции" системы (см. стр. 62" Как мы сочетаем Scrum с XP")

"Стена проектирования" – это просто большая доска, на которой развешены самые важные для разработки наброски и распечатки (блок-схемы, эскизы интерфейса, модели предметной области и т.п.)

На фотографии: ежедневный Scrum в вышеупомянутом уголке.

Хммммм... Эта burndown диаграмма выглядит подозрительно красивой и гладкой, правда? Но команда настаивает на том, что это правда :o)

Усадите команду вместе

Когда приходит время расставить столы и рассадить команду, есть одно правило, которое сложно переоценить.

Усадите команду вместе!

Чуть поясню, что я имею в виду:

Усадите команду вместе!

Людям не нравится переезжать. По крайней мере, в тех компаниях, в которых работал я. Они не хотят собирать все свои вещички, выдергивать шнуры из компьютера, переносить весь свой скарб на другой стол, и снова втыкать все шнуры. Чем меньше расстояние, тем больше недовольство. "Да ладно, шеф, НА КОЙ ЧЕРТ меня пересаживать всего на 5 метров вправо"?

Но когда мы строим эффективную команду по Scrum'у, другого выхода нет. Просто соберите команду вместе. Даже если придется им угрожать, самому переносить все их имущество, и самому вытирать застарелые следы от чашек на столах. Если для команды нет места – найдите место. Где угодно. Даже если придется посадить команду в подвале. Передвиньте столы, подкупите офис-менеджера, делайте всё, что нужно. Но как бы то ни было, соберите команду вместе.

Как только вы соберете всю команду вместе, результат появится незамедлительно. Уже после первого спринта команда согласится, что это была хорошая идея – собраться всем в одном месте (по моему опыту так и есть, хотя нет никаких гарантий, что команда не окажется слишком упрямой, чтобы это признать).

Да, кстати, а что значит "вместе"? Как должны стоять столы? Ну, лично у меня нет однозначного мнения о наилучшем расположении столов. Но даже если бы и было, я думаю, у большинства команд нет такой роскоши – решать, как будут расставлены столы в их комнате. Всегда существуют физические ограничения – соседняя команда, дверь в туалет, здоровый автомат с батончиками и напитками посреди комнаты – да что угодно.

"Вместе" значит:

- **В пределах слышимости**: каждый в команде может поговорить с любым другим членом команды без крика и не вставая из-за своего стола.

- **В пределах видимости**: каждый член команды может увидеть любого другого. Каждый может видеть доску задач. Не обязательно быть достаточно близко, чтобы *читать*, но, по крайней мере, *видеть*.
- **Автономно**: если вдруг вся ваша команда поднимется и начнет внезапную и очень оживлённую дискуссию об архитектуре системы, никого из не-членов команды не окажется достаточно близко, чтобы ему это помешало. И наоборот.

"Автономно" не значит, что команда должна быть полностью изолирована. В пространстве, разделённом на секции, вполне может хватить отдельной секции для команды, с достаточно высокими стенами, чтобы не пропускать *большую часть* шума извне.

А как быть с распределённой командой? Ну, тогда вам не повезло. Чтобы уменьшить негативные последствия, используйте как можно больше технических средств: видеоконференции, вебкамеры, средства для совместного использования рабочего стола и т.п.

Не подпускайте product owner'а слишком близко

Product owner должен находиться настолько близко к команде, чтобы в случае возникновения вопросов, команда могла бы спросить его лично, и чтобы он имел возможность на своих двоих подойти к доске задач. Но он не должен сидеть в одной комнате с командой. Почему? Потому что есть вероятность, что он не сможет не вдаваться в подробности, а команда не сможет правильно сработаться (т.е. не достигнет состояния полной автономности, самомотивации и сверхпродуктивности).

Если честно, то это всего лишь догадки: в действительности, я сам никогда не видел, чтобы product owner сидел рядом с командой, а, значит, у меня нет оснований говорить, что это плохая идея. Мне это просто подсказывает внутреннее чутьё и другие ScrumMaster'а.

Не подпускайте менеджеров и тренеров слишком близко

Эту главу мне писать немного тяжело, ведь я был как менеджером, так и тренером ...

Тесная работа с командами входила в мои непосредственные обязанности. Я создавал команды, переходил из одной команды в другую, программировал с ними в парах, тренировал ScrumMaster'ов, организовывал планирования спринтов и т.д. Оглядываясь назад можно сказать, что я творил Хорошие Дела. И это всё благодаря моему опыту в гибкой разработке программного обеспечения.

Но потом меня назначили (звучит тема Дарта Вейдера) руководителем отдела разработки. В общем, я стал менеджером. Это означало, что моё присутствие автоматически делало команду менее самоорганизующейся. "О! Шеф тут. У него, небось, есть куча идей о том, что надо сделать, и кто должен этим заняться. Давай-ка послушаем".

Я придерживаюсь следующей точки зрения: Если вы Scrum-тренер (и возможно совмещаете эту роль с ролью менеджера), не бойтесь очень плотно работать с командой. Но только в течение определённого промежутка времени, а потом оставьте команду в покое и дайте ей возможность сработаться и самоорганизоваться. Время от времени контролируйте её (однако не очень часто). Это можно делать, посещая демо, изучая доску задач или принимая участие в ежедневном Scrum'е. Если вы увидите как можно улучшить процесс, отведите ScrumMaster'а в сторонку и дайте ему дельный совет. Не стоит поучать его на глазах у всей команды. Посещение ретроспективы (см. стр. 51 "Как мы проводим ретроспективы") тоже будет не лишним. Если степень доверия к вам со стороны команды невелика, то сделайте доброе дело, не ходите на ретроспективы, а то ваше присутствие заставит команду молчать о своих проблемах.

Если Scrum-команда работает хорошо, обеспечьте её всем необходимым, а потом оставьте в покое (за исключением демо, разумеется).

8

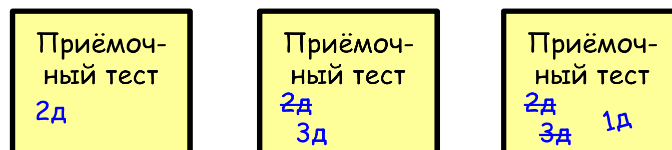
Как мы проводим ежедневный Scrum

Наш ежедневный Scrum очень похож на то, как это описывают в книгах. Каждый день он начинается в одно и то же время, в одном и том же месте. Вначале, мы предпочитали проводить его в отдельной комнате (это были дни, когда мы использовали sprint backlog'и в электронном формате), однако сейчас мы проводим ежедневный Scrum у доски задач в комнате команды. Нет ничего лучше!

Обычно мы проводим встречу стоя, поскольку это уменьшает вероятность того, что продолжительность нашей "планёрки" превысит 15 минут.

Как мы обновляем доску задач

Обычно мы обновляем доску задач во время ежедневного Scrum'а. По мере того, как каждый член команды рассказывает о том, что он сделал за вчерашний день и чем будет заниматься сегодня, он перемещает стикеры на доске задач. Как только рассказ касается какого-то незапланированного задания, то для него клеится новый стикер. При обновлении временных оценок, на стикере пишется новая оценка, а старая зачеркивается. Иногда стикерами занимается ScrumMaster, пока участники говорят.



В некоторых командах принято, что все члены команды обновляют доску задач *перед* каждой встречей. Это тоже хорошо работает. Просто решите, что вам ближе, и придерживайтесь этого.

Независимо от того, какой формат sprint backlog'а вы используете, пытайтесь вовлечь *всю команду* в поддержание sprint backlog'а в актуальном состоянии. Мы пробовали проводить спринты, в которых только ScrumMaster занимался поддержкой sprint backlog'а. Он должен был каждый день обходить всех членов команды и спрашивать об оценках оставшегося до окончания времени. Недостатки этого подхода в том, что:

- ScrumMaster тратит слишком много времени на "бумажную работу", вместо того, чтобы заниматься поддержкой команды и устранением преград.
- Члены команды не в курсе состояния спринта, поскольку им не нужно заботиться о sprint backlog'е. Эта нехватка обратной связи уменьшает общую гибкость и сконцентрированность команды.

Если sprint backlog имеет надлежащий вид, то любой член команды может легко его обновить.

Сразу же после ежедневного Scrum'а, кто-то суммирует оценки всех задач на доске (конечно, кроме тех, которые находятся в колонке "Готово") и ставит новую точку на burndown-диаграмме.

Как быть с опоздавшими

Некоторые команды заводят специальную копилку. Если вы опоздали, даже на минуту, вы кидаете в копилку определённую сумму. Без вариантов. Даже если вы позвонили перед началом ежедневного Scrum'а и предупредили, заплатить всё равно придётся :o)

Отвертеться можно лишь в исключительных случаях. Например, визит к врачу, собственная свадьба или что-то не менее важное.

Деньги из копилки используются на общественные нужды. Например, на них можно заказать пиццу, когда мы решаем поиграть вечерком :o)

Этот подход работает неплохо. Но пользоваться им нужно лишь в том случае, когда люди часто опаздывают. Некоторым командам это просто не нужно.

Как мы поступаем с теми, кто не знает, чем себя занять

Иногда кто-то говорит: "Вчера я делал то-то и то-то, а сегодня нет четкого представления у меня, чем занять себя". Наши действия?

Допустим, Джо и Лиза не знают, чем сегодня заняться.

Если я выступаю в роли ScrumMaster'a, я просто передаю слово следующему. При этом беру на карандаш тех, кто не знает, чем ему заняться. После того, как все высказались, я пробегаюсь вместе с командой по доске задач и проверяю, что данные на доске задач актуальные и что все понимают смысл каждой истории. Далее я предлагаю каждому участнику команды приклеить новые стикеры. После этого возвращаюсь к тем, кто не знал, чем себя занять, с вопросом "после того, как мы прошлись по доске, не появилось ли у вас представление о том, чем заняться?" Надеюсь, к тому времени оно уже появится.

Если же нет, то я выясняю, есть ли возможность для парного программирования. Допустим, сегодня Никлас планирует реализовать интерфейс для админки. Вы можете предложить Джо или Лизе поработать в паре с Никласом над этой функциональностью. Обычно это работает.

Если нет, то вот вам следующий приём.

> **ScrumMaster**: "Так, и кто хочет продемонстрировать нам готовую бета-версию?" (предполагая, что выпуск бета-версии – цель спринта)
> **Команда**: недоумевающая тишина
> **ScrumMaster**: "Мы что – не готовы?"
> **Команда**: "ммм... нет".
> **ScrumMaster**: "Почему? Что ещё осталось незаконченным?"
> **Команда**: "Так у нас даже нет тестового сервера. Кроме того нужно починить билд-скрипт".
> **ScrumMaster**: "Ага" (наклеивает два новых стикера). "Джо и Лиза, вам всё еще нечем заняться сегодня?"
> **Джо**: "Хм... Думаю, что я попробую раздобыть тестовый сервер".
> **Лиза**: "... а я постараюсь починить билд-скрипт."

Если повезёт, кто-то действительно сможет продемонстрировать готовый к выпуску бета-версии релиз. Отлично! Цель спринта достигнута. Но что делать, если прошла только половина времени, отведённая на спринт. Всё просто. Поздравьте команду за отличную работу, возьмите одну-две истории из стопки "Следующие" и поместите их в колонку "В планах". После этого повторно проведите ежедневный Scrum. Уведомите product owner'a о том, что вы добавили несколько новых историй в спринт.

Что же делать, если команда не достигла цели спринта, а Джо с Лизой всё ещё не могут определиться с тем, какую пользу они могут принести? Я обычно пользуюсь одной из нижеперечисленных методик (все они не очень хорошие, но всё же это последнее средство):

- **Пристыдить**: "Ладно, если не знаешь, как принести пользу команде, иди домой, почитай книгу и т.д. Или просто сиди здесь, пока кому-то не потребуется твоя помощь".
- **По старинке**: Просто назначить им задачу.
- **Моральное давление**: Скажите им: "Джо и Лиза! Не смею вас больше задерживать. А мы все просто постоим тут, пока у вас не появятся идеи, как помочь нам в достижении цели".
- **Закабалить**: Скажите им: "Вы сможете помочь команде, исполняя роль прислуги сегодня. Готовьте кофе, делайте массаж, вынесите мусор, приготовьте обед: делайте всё, о чём вас может попросить

команда". Вы будете удивлены, насколько быстро Джо и Лиза найдут для себя полезные технические задачи :o)

Если у вас есть человек, который часто заставляет вас заходить так далеко, возможно, вы должны отвести этого товарища в сторонку и культурно объяснить ему, что он не прав. Если и после этого проблема не решится, нужно понять, важен ли этот человек для команды или нет?

Если он *не очень важен*, постарайтесь исключить его из своей команды.

Если же он *важен* для команды, попробуйте найти ему напарника-"наставника". Джо может быть классным программистом и отличным архитектором, просто он предпочитает, чтобы ему другие люди говорили, что он должен делать. Отлично. Назначьте Никласа наставником Джо. Или будьте сами им. Если Джо действительно важен для команды, такой будет цена достижения цели. У нас были похожие ситуации, и этот подход более-менее срабатывал.

9

Как мы проводим демо

Демонстрация спринта – очень важная часть Scrum'a, которую многие все же недооценивают.

"Ой, а нам что, *обязательно* делать демо? Мы все равно ничего интересного не покажем!"

"У нас нет времени на подготовку разных &%$# демо!"

"У меня куча работы, не хватало еще смотреть чужие демо!"

Почему мы настаиваем на том, чтобы каждый спринт заканчивался демонстрацией

Хорошо выполненное демо оказывает огромное воздействие, даже если оно не показалось захватывающим.

- Положительная оценка работы *воодушевляет* команду.
- Все остальные узнают, чем занимается ваша команда.
- На демо заинтересованные стороны обмениваются жизненно важными отзывами.
- Демо проходит в дружеской атмосфере, поэтому разные команды могут свободно общаться между собой и обсуждать насущные вопросы. Это ценный опыт.
- Проведение демо заставляет команду *действительно доделывать задачи* и выпускать их (даже если это всего лишь на тестовый сервер). Без демо мы постоянно оказывались с кучей на 99% сделанной работы. Проводя демо, мы можем получить меньше сделанных задач, но они будут *действительно закончены*, что (в нашем случае) намного лучше, чем куча функционала, который *"типа" сделан* и будет болтаться под ногами в следующем спринте.

Если команду заставлять проводить демо, когда у них ничего толком не работает, им будет не по себе. Команда будет запинаться и спотыкаться, показывая функциональность, и хорошо, если в конце вы услышите жиденькие аплодисменты. Людям будет жаль эту команду, а некоторых может даже разозлить то, что они только потеряли время на этом вшивом демо.

Это очень неприятно. Но это действует, как горькая пилюля. *В следующем спринте* команда действительно постарается все *доделать*! Они будут думать "ладно, может, в следующем спринте стоит показать всего две вещи вместо пяти, но, черт возьми, в этот раз они будут РАБОТАТЬ!". Команда знает, что демо придется проводить не смотря ни на что, и благодаря этому шансы увидеть там что-то пристойное значительно возрастают. Я несколько раз был этому свидетелем.

Памятка по подготовке и проведению демо

- Постарайтесь как можно более чётко озвучить цель данного спринта. Если на демо присутствуют люди, которые ничего не знают о вашем продукте, то не поленитесь уделить пару минут, чтобы ввести их в курс дела.
- Не тратьте много времени на подготовку демо, особенно на создание эффектной презентации. Выкиньте всё ненужное и сконцентрируйтесь на демонстрации только реально работающего кода.
- Следите, чтобы демо проходило в быстром темпе. Сконцентрируйтесь на создании не столько красивого, сколько динамичного демо.

- Пусть ваше демо будет бизнес-ориентированным, забудьте про технические детали. Сфокусируйтесь на том "что мы сделали", а не на том "как мы это делали".
- Если это возможно, дайте аудитории самой попробовать поиграть с продуктом.
- Не нужно показывать кучу исправлений мелких багов и элементарных фич. Вы можете упомянуть о них, но демонстрировать их не стоит, потому что это заберёт у вас много времени и снизит внимание к более важным историям.

Что делать с "недемонстрируемыми" вещами

Член команды: "Я не собираюсь демонстрировать эту задачу, потому что её невозможно продемонстрировать. Я говорю про историю 'Улучшить масштабируемость системы так, чтобы она могла обслуживать одновременно 10 000 пользователей'. Я, по-любому, не смогу пригласить на демо 10 000 пользователей".

ScrumMaster: "Так, ты закончил с этой задачей?"

Член команды: "Ну, конечно".

ScrumMaster: "А как ты узнал, что оно потянет?"

Член команды: "Я сконфигурировал нашу систему в среде, предназначенной для тестирования производительности, и нагрузил систему одновременными запросами с восьми серверов сразу".

ScrumMaster: "Так у тебя есть данные, которые подтверждают, что система может обслужить 10 000 пользователей?"

Член команды: "Да. Хоть тестовые сервера и слабенькие, однако, в ходе тестирования они всё равно справились с 50 000 одновременных запросов".

ScrumMaster: "Так, а откуда у тебя эта цифра?"

Член команды (расстроенный)**:** "Ну, хорошо, у меня есть отчёт! Ты можешь сам глянуть на него, там описано как всё это дело было сконфигурировано и сколько запросов было отослано!"

ScrumMaster: "О, отлично. Это и есть твоё "демо". Просто покажи этот отчёт аудитории и вкратце пробегись по нему. Всё же лучше, чем ничего, правда?"

Член команды: "А что, этого достаточно? Правда он выглядит как-то корявенько, надо бы его немножко шлифануть".

ScrumMaster: “Хорошо, только не трать на это слишком много времени. Он не обязан быть красивым, главное – информативным”.

10

Как мы проводим ретроспективы

Почему мы настаиваем на том, чтобы все команды проводили ретроспективы

Наиболее важная вещь в отношении ретроспектив – *это их проведение*.

По некоторым причинам команды не проявляют должного интереса к проведению ретроспектив. Без небольшого давления со стороны многие команды часто пропускают ретроспективу и сразу переходят к следующему спринту. Может быть, это особенность шведского менталитета, в чём я не уверен.

Хотя при этом все вроде соглашаются, что ретроспективы крайне полезны. Я бы даже сказал, что ретроспектива является вторым по значимости мероприятием в Scrum'е (первое – это планирование спринта), потому что *это самый подходящий момент для начала улучшений*!

Конечно, чтобы возникла хорошая идея, вам не нужна ретроспектива, она может прийти к вам в голову, когда вы дома в душевой кабинке! Но поддержит ли команда вашу идею? Может быть, но вероятность значительно выше, если идея рождается внутри команды, то есть, во время ретроспективы, когда каждый может сделать свой вклад в обсуждение идеи.

Без ретроспектив вы обнаружите, что команда наступает на одни и те же грабли снова и снова.

Как мы проводим ретроспективы

Хотя основной формат немного варьируется, но в основном мы делаем так:

- Выделяем 1-3 часа, в зависимости от того насколько долгая ожидается дискуссия.
- Участвуют: product owner, вся команда и я собственной персоной.
- Располагаемся либо в отдельной комнате с уютным мягким уголком, либо на террасе, либо в каком-то другом похожем месте, поскольку нам нравится вести дискуссию в спокойной и непринуждённой атмосфере.
- Зачастую мы стараемся не проводить ретроспективы в рабочей комнате, так как это рассеивает внимание участников.
- Выбираем кого-то в качестве секретаря.
- ScrumMaster показывает sprint backlog и при участии команды подводит итоги спринта. Важные события, выводы и т.д.
- Начинаем "серию" обсуждений. В этот момент каждый имеет шанс высказаться о том, что, по его мнению, было хорошего, что можно было бы улучшить и что бы он сделал по-другому в следующем спринте. При этом его никто не перебивает.
- Мы сравниваем прогнозируемую и реальную производительность. Если имеются существенные расхождения, то пытаемся проанализировать и понять, почему так получилось.
- Когда время подходит к концу, ScrumMaster пытается обобщить все конкретные предложения по поводу того, что мы можем улучшить в следующем спринте.

Вообще-то, наши ретроспективы не имеют чёткого плана проведения, но главная тема – всегда одна и та же: "Что мы можем улучшить в следующем спринте".

Вот вам пример доски с нашей последней ретроспективы:

У нас есть три колонки:

- **Хорошо**: Если нужно было бы повторить этот спринт ещё раз, то мы бы сделали это точно так же.
- **Могло бы быть и лучше**: Если нужно было бы повторить этот спринт ещё раз, то мы бы сделали это по-другому.
- **Улучшения**: Конкретные идеи о том, как в будущем можно что-то улучшить.

Таким образом, первая и вторая колонки относится к прошлому, тогда как третья – направлена в будущее.

После того как команда закончил свой мозговой штурм по поводу всех этих стикеров, они проводят "точечное голосование" для определения улучшений, которым следует уделить особое внимание в ходе следующего спринта. У каждого члена команды имеется три магнитика, которыми он может воспользоваться для голосования. Каждый член команды может лепить магнитики как ему вздумается, хоть все три сразу на одну задача.

Основываясь на этом голосовании, мы выбираем 5 улучшений, которые мы попытаемся внедрить в следующем спринте, а на следующей ретроспективе мы проверим, что у нас вышло.

Очень важно не переоценить свои возможности. Выберите всего несколько улучшений для следующего спринта.

Как учиться на чужих ошибках

Информация, которая всплывает в ходе ретроспектив, обычно крайне важна. Для команды настали нелёгкие времена, потому что менеджеры по продажам начали забирать программистов с работы на свои встречи, чтоб те играли роль "технических экспертов"? Это очень важная информация. Возможно, что и у других команд точно такие же проблемы. Может быть, нам стоит провести специальные тренинги по нашему продукту для отдела маркетинга, чтобы они самостоятельно смогли отвечать на все вопросы клиентов?

Возможные способы решения проблем, найденных командой на ретроспективе, могут оказаться полезными не только для неё самой, но и для остальных.

Как же собрать все эти результаты? Мы выбрали достаточно простой способ. Один человек (в этом случае я) принимает участие во всех ретроспективах в роли "связующего звена". Без всяких формальностей.

В качестве альтернативного варианта можно предложить каждой команде составлять отчёт о проведённой ретроспективе. Мы пробовали и этот способ, но поняли, что немногие читают такие отчёты, а ещё меньше тех, кто делают выводы из прочтённого. Поэтому мы остановились на самом простом способе.

Наиболее важные требования для человека, который будет "связующим звеном":

- Он должен быть хорошим слушателем.
- Если ретроспектива проходит очень вяло, он должен быть готов задать простой, но меткий вопрос, который подтолкнёт людей на дискуссию. Например: "Если бы можно было повернуть время вспять и переделать этот спринт с самого первого дня, чтобы вы сделали по-другому?".
- Он должен быть согласен тратить своё время на посещение всех ретроспектив всех команд.
- Он должен обладать необходимыми полномочиями, которые помогли бы ему взяться за выполнение предложенных командой улучшений, выходящих за пределы возможностей самой команды.

Такой подход работает достаточно хорошо, но это не значит, что нет подходов намного лучше. Как только найдёте что-то новенькое, дайте мне знать.

Изменения. Быть или не быть

Предположим, команда пришла к выводу, что "мы слишком слабо общались внутри команды, поэтому мы постоянно мешали друг другу и переделывали архитектурные решения".

Что нам с этим делать? Организовать ежедневные встречи для обсуждения архитектуры? Внедрить новые средства, чтобы упростить общение? Создать больше страниц в wiki? Может, и да. А может, и нет.

Оказалось, что достаточно всего лишь четко определить проблему, и она часто решается сама собой в следующем спринте. В особенности, если на стене в рабочей комнате повесить записи по ретроспективе спринта (что, к нашему стыду, мы так часто забываем сделать!) Имейте в виду, что каждое изменение имеет свою цену, поэтому перед тем как его внедрять, подумайте, может, стоит ничего не делать вообще и надеяться, что проблема станет меньше или исчезнет совсем.

Пример, приведенный выше ("мы так слабо общались внутри команды...") – это классический пример того, что решается лучше всего бездействием.

Если в ответ на каждую жалобу пытаться что-то делать, народ с неохотой будет рассказывать про свои даже самые мелкие проблемы, которые могут быть ужасными.

Типичные проблемы, которые обсуждают на ретроспективах

В этой главе я постараюсь описать стандартные проблемы, которые всплывают в ходе ретроспектив, и возможные пути их решения.

«Нам надо было больше времени потратить на разбиение историй на подзадачи»

О, это классика жанра. Каждый день на Scrum'е можно услышать, как люди произносят избитую до боли фразу: "Я не знаю, что мне сегодня делать". И вам приходится изо дня в день тратить кучу времени для того, чтобы после Scrum'а найти задачи для этих ребят. Мой совет – делайте это заранее.

Стандартные действия: никаких. Возможно, команда сама решит эту проблему на следующем планировании. Если же это повторяется из раза в раз, увеличьте время на планирование спринта.

«Очень часто беспокоят извне»

Стандартные действия:

- Попросите команду уменьшить фокус-фактор на следующий спринт, чтобы у них был более реалистичный план.
- Попросите команду более подробно записывать случаи вмешательства (кто и как долго). Потом будет легче решить проблему.
- Попросите команду переводить все внешние запросы на ScrumMaster'а или product owner'а.
- Попросите команду выбрать одного человека в качестве "голкипера" и перенаправлять на него все вопросы, которые могут отвлечь команду от работы. Это позволит остальной части команды

сконцентрироваться на своих задачах. В это роли может выступать, как ScrumMaster, так и любой член команды, которого нужно будет периодически менять.

«Мы взяли огромный кусок работы, а закончили только половину»

Стандартные действия: никаких. Скорее всего, в следующий раз команда не станет браться за нереальный объём работ. Или, по крайней мере, поскромнее оценит свои возможности.

«У нас в офисе бардак и очень шумно»

Стандартные действия:

- Попробуйте создать более благоприятную атмосферу или перевезите команду на другое место. Куда угодно. Можете снять комнату в отеле (см. стр. 43 "Как мы обустроили комнату команды").
- Если это возможно, попросите команду уменьшить фокус-фактор на следующий спринт с чётким описанием причины: шум и бардак в офисе. Может быть, это заставит product owner'a начать пинать вышестоящий менеджмент насчёт вашей проблемы.

Слава Богу, мне никогда не приходилось перевозить команду в другое место. Но если придётся – я это сделаю. :o)

11

Отдых между спринтами

В реальной жизни невозможно постоянно бежать как спринтер. Между забегами вам в любом случае нужен отдых. Если вы бежите с постоянной максимальной скоростью, то, по сути, вы просто бежите трусцой.

То же самое наблюдается в Scrum'e, да и в разработке программного обеспечения в целом. Спринты очень изматывают. Как у разработчика, у вас никогда нет времени, чтобы расслабиться, каждый день вы должны стоять на этом проклятом Scrum-митинге и рассказывать всем, чего вы добились вчера. Мало кто может похвастаться: "Я потратил большую часть дня, положив ноги на стол, просматривая блоги и попивая капучино".

Помимо собственно отдыха, есть ещё одна причина для паузы между спринтами. После демо и ретроспективы, как команда, так и product owner будут переполнены информацией и всевозможными идеями, которые им следовало бы обмозговать. Если же они немедленно займутся планированием следующего спринта, то есть риск, что они не смогут упорядочить всё полученную информацию или сделать надлежащие выводы. К тому же у product owner'a не хватит времени для корректировки его приоритетов после проведённого демо и т.д.

Плохо:

Понедельник
09-10 Спринт №1: демо **10-11** Спринт №1: ретроспектива **13-16** Спринт №2: планирование

Перед началом нового спринта (если быть точным, после ретроспективы спринта и перед планированием следующего спринта) мы стараемся добавлять небольшой промежуток свободного времени. Увы, у нас это не всегда получается.

Как минимум, мы стараемся добиться того, чтобы ретроспектива спринта и следующее планирование спринта не проходили в один и тот же день. Перед началом нового спринта каждый должен хорошенько выспаться, не думая при этом о спринтах.

Лучше:

Понедельник	Вторник
09-10 Спринт №1: демо **10-11** Спринт №1: ретроспектива	**09-13** Спринт №2: планирование

Еще лучше:

Пятница	Суббота	Воскресение	Понедельник
09-10 Спринт №1: демо **10-11** Спринт №1: ретроспектива			**09-13** Спринт №2: планирование

Один из вариантов для этого – "инженерные дни" (или как бы вы их не называли). Это дни, когда разработчикам разрешается делать по сути все, что они хотят. (ОК, я признаю, в этом виноват Google). Например, читать о последних средствах разработки и API, готовиться к сертификации, обсуждать компьютерные занудства с коллегами, заниматься своим личным проектом, ...

Наша цель – инженерный день между каждым спринтом. Так между спринтами появляется реальная возможность отдохнуть, а команда разработки получает хороший шанс поддерживать актуальность своих знаний. К тому же, это достаточно весомое преимущество работы в компании.

Лучше некуда?

Четверг	Пятница	Суббота	Воскресение	Понедельник
09-10 Спринт №1: демо **10-11** Спринт №1: ретроспектива	Инженерный день			**09-13** Спринт №2: планирование

Сейчас у нас один инженерный день между спринтами. Если конкретно, то это первая пятница каждого месяца. Почему же не между спринтами? Ну, потому что я считал важным, чтобы вся компания брала инженерный день в одно и то же время. Иначе люди не воспринимают его серьезно. И так как мы (пока что) не согласовывали спринты между всеми продуктами, я был вынужден выбрать инженерный день, независимый от спринтов.

Когда-нибудь мы можем попробовать согласовать спринты между продуктами (то есть одна и та же дата для начала спринта и одновременное окончание спринтов для всех продуктов и команд). В этом случае, мы точно поместим инженерный день между спринтами.

12

Как мы планируем релизы и составляем контракты с фиксированной стоимостью

Иногда нужно планировать дальше, чем на один спринт вперед. Это типичная ситуация для контрактов с фиксированной стоимостью, когда нам *приходится* планировать наперед, или же есть риск подписаться под нереальной датой поставки.

Как правило, планирование релиза для нас – это попытка ответить на вопрос: "*когда, в самом худшем случае*, мы сможем поставить версию 1.0".

Если вы *действительно* хотите разобраться, как планировать релиз, советую пропустить эту главу и купить книгу Майка Кона "Agile Estimating and Planning". Эх, прочитать бы мне эту книгу раньше... (она попалась мне уже *после* того, как мы на собственном опыте поняли, что к чему...). Мой способ планирования простой, как угол дома, но может послужить вам хорошей отправной точкой.

Определяем свою приёмочную шкалу

В дополнении к обычному product backlog'у, product owner определяет *приёмочную шкалу*, которая представляет собой ни что иное, как простое разбиение всех историй product backlog'a на группы в зависимости от их уровня важности в контексте контрактных обязательств.

Вот пример диапазонов из нашей приёмочной шкалы:

- Все элементы с важностью >= 100 *обязаны* быть включены в версию 1.0, иначе нас оштрафуют по полной программе.
- Все элементы с важность 50-99 *должны* быть включены в версию 1.0, но в случае чего мы *можем* выкатить эту функциональность в следующем дополнительном релизе.
- Элементы с важностью 25-49 необходимы, но могут быть сделаны в последующем релизе версии 1.1.
- Важность элементов < 25 весьма спорна, так как возможно, что они вообще никогда не пригодятся.

Вот пример product backlog'a, раскрашенного в соответствии с вышеописанными правилами:

Важность	Название
130	Банан
120	Яблоко
115	Апельсин
110	Гуава
100	Груша
95	Изюм
80	Арахис
70	Пончик
60	Лук
40	Грейпфрут
35	Папайя
10	Черника
10	Персик

Красные = *обязательно* должны быть добавлены в версию 1.0 (банан – груша)

Жёлтые = *желательно* включить в версию 1.0 (изюм – лук)

Зелёные = могут быть добавлены позже (грейпфрут – персик)

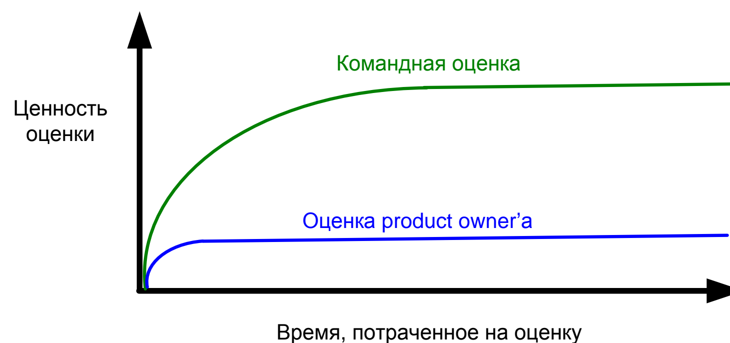
Итак, если к крайнему сроку мы закончим всё: от банана до лука, то нам бояться нечего. Если время будет нас поджимать, то мы ещё *успеем выкрутиться*, убрав изюм, арахис, пончик и лук. Всё, что ниже лука – бонус.

Оцениваем наиболее важные истории

Чтобы спланировать релиз, product owner'у нужны оценки, как минимум оценки всех включенных в контракт историй. Как и в случае планирования спринта, это – коллективный труд команды и product owner'a. Команда планирует, а product owner объясняет и отвечает на вопросы.

Оценка считается *ценной*, если впоследствии она оказалась близкой к реальности. Менее полезной, если отклонение составило, скажем, 30%. И абсолютно бесполезной, если она не имеет ничего общего с реально потраченным временем.

А вот что я думаю о зависимости ценности оценки от того, кто и как долго её делает.



Резюмируя вышесказанное:

- Пусть *команда* проведёт оценку.
- Не давайте им тратить на это много времени.
- Убедитесь, что команда понимает, что нужно получить приблизительные оценки, а не контракт, под которым надо ставить подпись.

Обычно product owner собирает всю команду, делает вводный обзор и сообщает, что целью этого совещания является оценка двадцати (например) наиболее значимых историй из product backlog'a. Он проходится по каждой истории и позволяет команде приступить к процессу оценки. Product owner остаётся с командой, чтобы, в случае необходимости, отвечать на вопросы и прояснить объём работ историй. Так же, как и при планировании спринта, колонка "как продемонстрировать" – отличное средство, чтобы избежать недопонимания.

Совещание должно быть строго ограниченным по времени – команды склонны тратить очень много времени на оценку всего нескольких историй.

Если product owner захочет потратить больше времени на оценку, он просто назначит другое совещание позже. Команда должна убедиться в том, что product owner осознаёт, что проведение подобных совещаний отразится на их текущем спринте. То есть, что он понимает, что за все (и за оценку в том числе) нужно платить.

Ниже приведен пример результатов оценки (в story point'ах):

Важность	Название	Оценка
130	Банан	12
120	Яблоко	9
115	Апельсин	20
110	Гуава	8
100	Груша	20
95	Изюм	12
80	Арахис	10
70	Пончик	8
60	Лук	10
40	Грейпфрут	14
35	Папайя	4
10	Черника	
10	Персик	

Прогнозируем производительность

Хорошо, теперь у нас есть приблизительные оценки для наиболее важных историй.

Следующий шаг – прогноз средней производительности команды.

Это значит, что для начала мы должны определить наш фокус-фактор (см. стр. 23 "Как команда принимает решение о том, какие истории включать в спринт?")

По большому счёту, фокус-фактор показывает: "насколько эффективно команда использует своё время для работы над выбранными историями". Это значение никогда не достигнет 100%, так как команда всегда тратит время на незапланированные задачи, помощь другим командам, переключение между задачами, проверку электронной почты, ремонт своих поломанных компьютеров, политические дебаты на кухне и т.д.

Предположим, что фокус-фактор нашей команды равен 50% (это достаточно низкое значение, у моей команды значение колеблется в районе 70%). Допустим также, что длина нашего спринта будет 3 недели (15 дней), а размер команды – 6 человек.

Таким образом, каждый спринт – это 90 человеко-дней, однако, в лучшем случае мы можем надеяться только на 45 человеко-дней (так как наш фокус-фактор составляет всего 50%).

Следовательно, прогнозируемая производительность составит 45 story point'ов.

Если у каждой истории оценка будет равна 5 дням (хотя такого и не бывает), тогда эта команда сможет выдавать на-гора примерно по 9 историй за спринт.

Сводим всё в план релиза

Сейчас, когда у нас есть оценки и прогнозируемая производительность, мы можем легко разбить product backlog на спринты:

Важность	Название	Оценка
		Спринт 1
130	Банан	12
120	Яблоко	9
115	Апельсин	20
		Спринт 2
110	Гуава	8
100	Груша	20
95	Изюм	12
		Спринт 3
80	Арахис	10
70	Пончик	8
60	Лук	10
40	Грейпфрут	14
		Спринт 4
35	Папайя	4
10	Черника	
10	Персик	

Каждый спринт состоит из набора историй, количество которых не превышает спрогнозированную производительность 45.

Теперь видно, что, скорее всего, нам потребуется 3 спринта для завершения всей обязательной и желательной функциональности.

3 спринта = 9 календарных недель = 2 календарных месяца. Станет ли это крайним сроком, который мы озвучим клиенту? Это полностью зависит от вида контракта, от того, насколько фиксирован объем работ, и т.д. Обычно мы берём время со значительным запасом, тем самым защищая себя от ошибочных оценок, возможных проблем, неоговоренного функционала и т.д. Значит, в этом случае мы установим срок поставки в 3 месяца, чтобы иметь месяц в резерве.

Очень хорошо, что мы можем демонстрировать клиенту что-нибудь пригодное к использованию каждые 3 недели и позволять ему изменять требования на протяжении всего времени сотрудничества (конечно в зависимости от того, как выглядит контракт).

Корректируем план релиза

Реальность не подстроится под план, поэтому приходится его корректировать.

По окончании спринта мы смотрим на реальную производительность команды. Если эта производительность существенно отличается от прогнозируемой, мы изменяем прогнозируемую производительность для будущих спринтов и обновляем план релиза. Если это грозит нам срывом срока поставки, product owner может начать переговоры с клиентом или начать искать путь уменьшения объема работ без нарушения контракта. Или, возможно, он и команда смогут увеличить производительность или фокус-фактор путём устранения серьёзных препятствий, которые были обнаружены во время спринта.

Product owner может позвонить клиенту и сказать: "Привет, мы слегка не вписываемся в график, но я полагаю, что мы сможем уложиться в срок, если уберём встроенный Тетрис, разработка которого занимает много времени. Мы можем добавить его в следующем релизе, который будет через три недели после первого релиза".

Пусть это и не самая лучшая новость, но, хотя бы, мы были честны и дали возможность клиенту заранее сделать выбор: или мы поставляем только самую важную функциональность в срок, или же всю полностью, но с задержкой. Обычно, это не очень сложный выбор. :o)

13

Как мы сочетаем Scrum с XP

То, что Scrum и XP (eXtreme Programming) могут быть эффективно объединены, не вызывает сомнений. Большинство рассуждений в интернете поддерживают это предположение, и я не хочу тратить время на дополнительные обоснования.

Тем не менее, одну вещь я всё-таки должен упомянуть. Scrum решает вопросы управления и организации, тогда как XP специализируется на инженерных практиках. Вот почему эти две технологии хорошо работают вместе, дополняя друг друга.

Тем самым я присоединяюсь к сторонникам мнения, что Scrum и XP могут быть эффективно объединены!

Я собираюсь рассказать про наиболее полезные практики из XP и про то, как мы используем их в нашей повседневной работе. Не все наши команды смогли применить практики XP в полном объеме, но мы провели большое количество экспериментов со многими аспектами комбинации XP/Scrum. Некоторые практики XP напрямую соответствуют практикам Scrum, например, "Whole team", "Sit together", "Stories" и "Planning game". В этих случаях мы просто придерживаемся Scrum.

Парное программирование

Недавно мы начали практиковать его в одной из наших команд. Как ни удивительно, работает довольно-таки хорошо. Большинство других наших команд до сих пор не очень много программирует парно, однако, попробовав эту практику в одной из наших команд для нескольких спринтов, я вдохновился идеей внедрить парное программирование и в других командах.

Вот пока несколько выводов после применения парного программирования:

- Парное программирование действительно улучшает качество кода.
- Парное программирование действительно увеличивает сосредоточенность команды (например, когда напарник говорит: “Слушай, а эта штуковина точно нужна для этого спринта?”)
- Удивительно, но многие разработчики, которые выступают против парного программирования, на самом деле не практиковали его, однако раз попробовав – быстро понимают все преимущества.
- Парное программирование выматывает, так что не стоит заниматься им целый день.
- Частая смена пар даёт хороший результат.
- Парное программирование действительно способствует распространению знаний внутри команды, заметно ускоряя этот процесс.
- Некоторые люди чувствуют себя некомфортно, работая в парах. Не стоит избавляться от хорошего программиста, только потому, что ему не нравится парное программирование.
- Ревью кода – хорошая альтернатива парному программированию.
- У “штурмана” (человека, который не пишет код) должен также быть свой компьютер, но не для разработки, а для выполнения мелких задач, когда это необходимо – просмотра документации, если “водитель” (человек, который пишет код) запнулся и так далее.
- Не навязывайте парное программирование людям. Вдохновите их, дайте необходимые инструменты и позвольте самим дойти до этого.

Разработка через тестирование (TDD)

Наконец-то! Разработка через тестирование для меня важнее, чем Scrum и XP вместе взятые. Можете отнять у меня дом, телевизор, собаку, но только попробуйте запретить использование TDD! Если вам не нравится TDD, тогда просто не подпускайте меня близко, иначе я всё равно привнесу его в проект втихую :)

Курс TDD за 10 секунд:

> *Разработка через тестирование означает, что вы сначала должны написать автоматизированный тест (который не проходит – прим. переводчика). После этого надо написать ровно столько кода, чтобы тест прошёл. Затем необходимо провести рефакторинг, в основном, чтобы улучшить читабельность кода и устранить дублирование. При необходимости повторить.*

Несколько фактов о TDD:

- Разработка через тестирование – это *непросто*. На деле оказывается, что демонстрировать TDD программисту практически бесполезно – часто единственный действенный способ *заразить* его TDD заключается в следующем. Программиста надо обязать работать в паре с кем-то, кто в TDD хорош. Но как только программист *вник* в TDD, то он уже заражен серьезно и о разработке другим способом даже слышать не хочет.
- TDD оказывает глубокое положительное влияние на дизайн системы.
- Чтобы TDD стало приносить пользу в новом проекте, необходимо приложить немало усилий. Особенно много тратится на интеграционные тесты методом "черного ящика". Но эти усилия окупаются *очень быстро*.
- Потрать достаточно времени, но сделай так, чтобы писать тесты *было просто*. То есть надо получить необходимый инструментарий, обучить персонал, обеспечить создание правильных вспомогательных и базовых классов и т.д.

Мы используем следующие инструменты для разработки через тестирование:

- jUnit / httpUnit / jWebUnit. Мы присматриваемся к TestNG и Selenium.
- HSQLDB в качестве встроенной БД в памяти (in-memory) для тестовых целей.
- Jetty в качестве встроенного web-контейнера в памяти (in-memory) для тестовых целей.
- Cobertura для определения степени покрытия кода тестами.
- Spring framework для написания различных типов тестовых фикстур (в т.ч. с использованием моков (mock-object) и без, с внешней БД и БД в памяти (in-memory) и т.д.)

В наших наиболее сложных продуктах (с точки зрения TDD) у нас реализованы автоматизированные приёмочные тесты методом "черного ящика". Эти тесты загружают всю систему в память, включая базы данных и web-сервера, и взаимодействуют с системой только через внешние интерфейсы (например, через HTTP).

Такой подход позволяет получить быстрые циклы "разработка-сборка-тест". Он так же выступает в качестве страховки, придавая разработчикам уверенность в успешности частого рефакторинга кода. В свою очередь, это обеспечивает простоту и элегантность дизайна даже в случае разрастания системы.

TDD и новый код

Мы используем TDD для всех новых проектов, даже если это означает, что фаза развёртывания рабочего окружения проекта потребует больше времени (потому что нужно больше усилий на настройку и поддержку тестовых утилит). Нетрудно понять, что выгода перевесит любой предлог *не внедрять* TDD.

TDD и существующий код

Я уже говорил, что TDD – это непросто, но что *действительно сложно*, так это пытаться применять TDD для кода, который изначально не был спроектирован для применения этого подхода! Почему? Эту тему

можно долго обсуждать, но я, пожалуй, остановлюсь. Приберегу мысли для следующей книги: "TDD: Заметки с передовой" :o)

В своё время мы потратили много времени в попытках автоматизировать интеграционное тестирование одной из сложных систем, код которой мы унаследовали в ужасном состоянии и без единого теста.

При выходе каждого релиза мы выделяли команду тестировщиков, которые выполняли весь набор сложных регрессионных тестов и тестов производительности. Регрессионное тестирование выполнялось преимущественно вручную, что существенно замедляло процессы разработки и выпуска релизов. Тогда нашей целью стало автоматизировать все эти тесты. Однако, побившись пару месяцев головой об стену, мы поняли, что не приблизились к цели ни на йоту.

Тогда мы изменили подход. Мы признали тот факт, что автоматизировать регрессионное тестирование нам не по силам, и задали себе вопрос: "Как можно уменьшить время тестирования?". Это была игровая система, и мы выяснили, что большую часть времени тестирующая команда тратит на тривиальные задачи, такие как настройку тестового турнира или ожидание начала турнира. Поэтому мы создали утилиты для выполнения этих операций. Маленькие, доступные нажатием горячих клавиш скрипты, которые выполняли всю подготовительную работу, позволяя тестировщикам сосредоточиться непосредственно на тестировании.

А вот эти усилия уже окупились сполна! По уму мы должны были поступить так с самого начала. Но мы были настолько зациклены на автоматизации, что забыли про эволюционный подход, в котором первым шагом было бы создание окружения, позволяющего сделать *ручное* тестирование более эффективным.

Усвоенный урок: Если от ручного регрессионного тестирования отказаться нельзя, но очень хочется его автоматизировать – лучше не надо (ну разве что это действительно очень просто). Вместо этого сделайте всё для облегчения процесса ручного тестирования. *А уже после этого* можно подумать и про автоматизацию.

Эволюционный дизайн

Это значит начать с простого дизайна и постоянно улучшать его, а не пытаться сделать всё идеально с первого раза и больше ничего и никогда не трогать.

С этим мы справляемся достаточно хорошо. Мы уделяем достаточно времени рефакторингу и улучшению существующего дизайна, и очень редко занимаемся детальным проектированием на годы вперёд. Иногда мы, конечно, можем напортачить, в результате чего плохой дизайн настолько врастает в систему, что рефакторинг превращается в действительно большую задачу. Но, в целом, мы полностью довольны этим подходом.

Если практиковать TDD, то, по большому счёту, постоянное улучшение дизайна получается само собой.

Непрерывная интеграция (Continuous integration)

Чтобы внедрить непрерывную интеграцию нам пришлось для большинства наших продуктов создать достаточно сложное решение, построенное на Maven'e и QuickBuild'e. Это архиполезно и экономит массу времени. К тому же это позволило нам раз и навсегда избавится от классической фразы: "но у меня же это работает!". Наш сервер непрерывной интеграции является "судьёй" или эталоном, по которому определяется работоспособность всего исходного кода. Каждый раз, когда кто-то сохраняет свои изменения в системе контроля версий, сервер непрерывной интеграции начинает собирать заново все доступные ему проекты и прогоняет все тесты на сервере. Если хоть что-то пойдёт не так, то сервер обязательно разошлёт всем участникам команды уведомления. Такие электронные письма содержат в себе информацию про то, какие именно изменения поломали сборку, ссылку на отчёты по тестам и т.д.

Каждую ночь сервер непрерывной интеграции пересобирает каждый проект заново и публикует на наш внутренний портал последние версии бинарников (EAR'ы, WAR'ы и т.д. [пр. переводчика – формат файлов, который используется в J2EE для компоновки модулей]), документации, отчётов по тестам, по покрытию тестами, по зависимостям между модулями и библиотеками и ещё много чего полезного. Некоторые проекты также автоматически устанавливаются на тестовых серверах.

Чтобы это всё заработало, пришлось потратить *уйму времени*, но, поверьте мне, это того стоило.

Совместное владение кодом (Collective code ownership)

Мы выступаем за совместное владение кодом, хотя ещё не все наши команды внедрили у себя эту практику. На собственном опыте мы убедились, что парное программирование и постоянная смена пар автоматически увеличивают уровень совместного владения кодом. А команды, у которых совместное владение кодом на высоком уровне, доказали свою высочайшую надёжность. К примеру, они никогда не проваливают спринты из-за того, что у них кто-то заболел.

Информативное рабочее пространство

У всех команд есть доступ к доскам и незанятым стенам, которыми они действительно пользуются. Во многих комнатах стены завешаны всякого рода информацией о продукте и проекте. Самая большая проблема у нас – постоянно накапливающееся старье на стенах. Уже подумываем завести роль "домработницы" в каждой команде.

Хотя мы и поощряем использование доски задач, еще не все команды внедрили их (см. стр. 43"Как мы обустроили комнату команды")

Стандарты кодирования

Недавно мы начали определять стандарты кодирования. Очень полезно – жаль не сделали этого раньше. Это не займёт много времени, начни с простого и постепенно дополняй. Записывай только то, что может быть понятно не всем, при этом, по возможности, не забудь сослаться на существующие материалы.

У большинства программистов есть свой индивидуальный стиль кодирования. Мелкие детали: как они обрабатывают исключения, как комментируют код, в каких случаях возвращают null и так далее. В одних случаях эти различия не играют особой роли, в других могут привести к серьёзному несоответствию дизайна системы и трудно читаемому коду. Стандарты кодирования – идеальное решение этой проблемы, если они, конечно, регламентируют важные моменты.

Вот небольшая выдержка из наших стандартов кодирования:

- Вы можете нарушить любое из этих правил, но на то должна быть веская причина и это должно быть задокументировано.
- По умолчанию используйте стандарты кодирования Sun: http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html
- Ни при каких обстоятельствах не перехватывайте исключения без сохранения полного стека вызовов программы (stack trace), либо повторной генерации исключения (rethrow). Допустимо использование log.debug(), только не потеряйте стек вызовов.
- Для устранения тесного связывания между классами применяйте внедрение зависимостей на основе сеттеров (Setter Based Injection) (разумеется, за исключением случаев, когда такое связывание просто необходимо).
- Избегайте аббревиатур. Общеизвестные аббревиатуры, такие как DAO, допустимы.
- Методы, которые возвращают коллекции или массивы, не должны возвращать null. Возвращайте пустые коллекции и массивы вместо null.

Устойчивый темп / энергичная работа

Множество книг по Agile-разработке программного обеспечения утверждают, что затянувшаяся переработка ведёт к падению продуктивности.

После некоторых не вполне добровольных экспериментов на эту тему, я могу только согласиться с этим всем сердцем!

Около года назад одна из наших команд (самая большая) очень-очень много работала сверхурочно. Качество кода было ужасное, и большую часть времени команда "тушила пожары". У тестировщиков (которые тоже работали сверхурочно) не было ни малейшей возможности нормально протестировать систему. Наши пользователи были в ярости, а газетчики готовы были нас растерзать.

Спустя несколько месяцев мы смогли уменьшить количество переработки до приемлемого уровня. Люди перестали работать сверхурочно (кроме редких кризисов в проекте), и – вот сюрприз! – и производительность, и качество кода заметно улучшились.

Конечно, уменьшение количества рабочих часов не было *единственной* причиной повышения производительности и улучшения кода. Но мы уверены, что это сыграло существенную роль.

14

Как мы тестируем

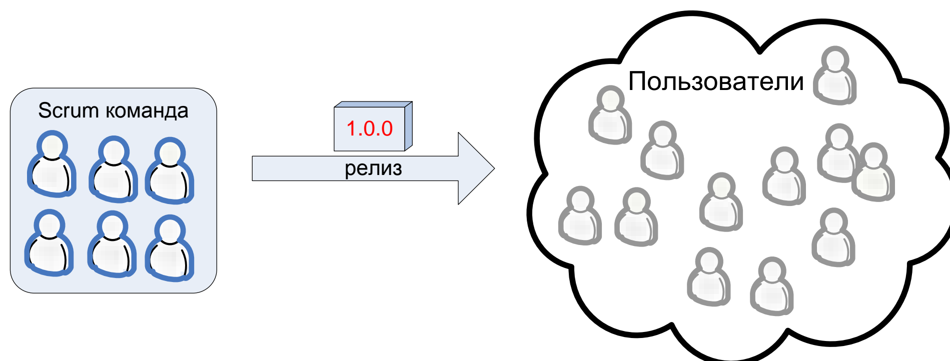
Это самая сложная часть. Вот только я не уверен, то ли это самая сложная часть Scrum'а, то ли разработки программного обеспечения в целом.

Организация тестирования может достаточно сильно отличаться в различных компаниях. Всё зависит от количества тестировщиков, уровня автоматизации тестирования, типа системы (просто сервер + интернет приложение или, возможно, вы выпускаете «коробочные» версии программ?), частоты релизов, критичности ПО (блог-сервер или система управления полётами) и т.д.

Мы довольно много экспериментировали, чтобы понять, как организовать процесс тестирования в Scrum'е. Сейчас я попытаюсь рассказать о том, что мы делали и чему мы успели научиться за это время.

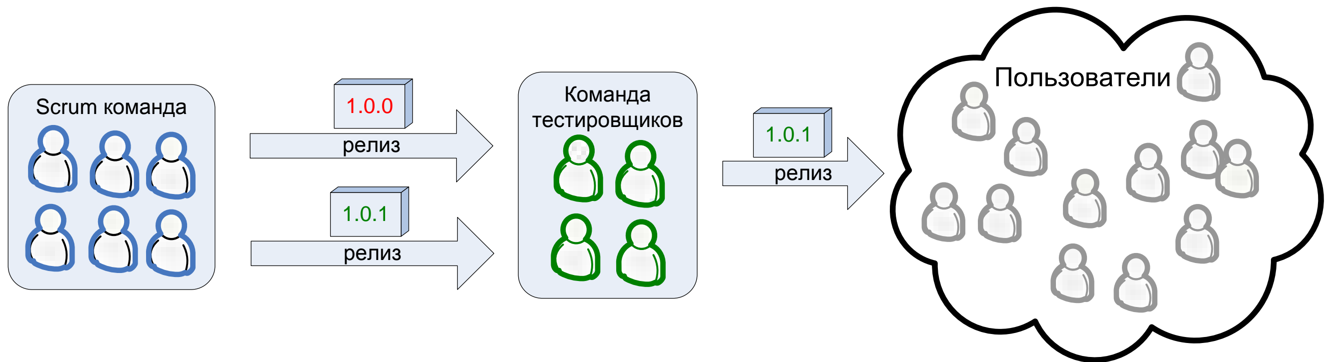
Скорее всего, вам не избежать фазы приёмочного тестирования

В идеальном мире Scrum'а результатом каждого спринта должна быть система, потенциально готовая к использованию. Бери и устанавливай, да?



А вот и нет!

По нашему опыту, такой подход обычно не работает. Там будет куча противных багов. Если "качество" для вас хоть что-нибудь значит, тогда придётся позаботиться о ручном приёмочном тестировании. Это когда специально выделенные тестировщики, которые *не являются* частью команды, бомбят систему теми видами тестов, о которых Scrum-команда не могла даже и подумать, или на которые у неё не было времени, или соответствующего оборудования. Тестировщики работают с системой точно так же, как и пользователи, а значит, что это нужно делать вручную (если, конечно же, ваша система разработана для людей).



Команда тестировщиков найдёт баги, Scrum-команде придётся подготовить версию с исправленными багами, и рано или поздно (будем надеяться что рано) вы сможете выпустить для своих пользователей версию 1.0.1 с необходимыми исправлениями. Она будет куда более стабильной, чем глючная 1.0.0.

Когда я говорю "фаза приёмочного тестирования", я имею в виду весь период тестирования, отладки и перевыпуска, пока версия не будет достаточно хороша для выхода в свет готового продукта.

Минимизируйте фазу приёмочного тестирования

Приёмочное тестирование, мягко говоря, приносит некоторые неудобства. Оно определённо не имеет ничего общего с гибкими методологиями. И, несмотря на то, что мы не можем избавиться от этой фазы, мы можем попробовать свести её к минимуму. А если точнее, уменьшить *количество времени*, которое для него необходимо. Этого можно достигнуть следующими способами:

- Максимально улучшить качество исходного кода, создаваемого Scrum-командой.
- Максимально увеличить эффективность ручного тестирования (т.е. найти лучших тестировщиков, обеспечить их лучшими инструментарием и убедиться, что они сообщают о тех задачах, которые отнимают много времени и могут быть автоматизированы).

Так как же мы можем поднять качество кода команды? Ну, вообще-то способов существует очень много. Я остановлюсь на тех, которые, как нам показалось, действуют лучше всего:

- Включите тестировщиков в Scrum-команду.
- Делайте меньше за спринт.

Повышайте качество, включив тестировщиков в Scrum-команду

О, я уже слышу эти возражения:

- "Но это же очевидно! Scrum-команды должны быть *кросс-функциональными*!"
- "В Scrum-команде не должно быть выделенных ролей! У нас не может быть человека, занимающегося *только* тестированием!"

Я бы хотел кое-что разъяснить. В данном случае, под "тестировщиком" я подразумеваю человека, главная специализация которого тестирование, а не человека, чья роль – это исключительно тестирование.

Разработчики достаточно часто бывают отвратительными тестировщиками. *Особенно*, когда они тестируют свой собственный код.

Тестировщик – это "последняя инстанция".

Кроме того, что тестировщик – обычный член команды, он ещё и выполняет очень важную функцию. Он – человек, за которым всегда "последнее слово". Ничто не может считаться готовым в спринте, пока *он* не подтвердит, что это действительно так. Я обнаружил, что разработчики часто говорят, что что-то готово, хотя в действительности это не так. Даже, если у вас имеется очень чёткое определение критерия готовности (которое у вас должно быть, см. стр. 29 "Критерий готовности"), в большинстве случаев, разработчики будут

часто его забывать. Мы, программисты, люди очень нетерпеливые и стремимся перейти к следующему пункту списка как можно быстрее.

Так как же мистер Т (наш тестировщик) узнает, что что-то действительно готово? Ну, прежде всего, он может это (сюрприз!) *протестировать*! В большинстве случаев оказывается, что то, что разработчик считает готовым, на самом деле даже *невозможно протестировать*! Либо по причине того, что оно не было зафиксировано в репозитории, либо не установлено на сервер, либо не может быть запущено, или ещё что-нибудь. Как только мистер Т завершил тестирование, первым делом он должен пройтись по критериям готовности (если таковые у вас имеются) и, желательно, вместе с разработчиком. К примеру, если критерий готовности требует наличия заметок к релизу, то мистер Т проверяет их наличие. Если же необходима более формальная спецификация функционала (хотя это бывает редко), мистер Т проверяет и это тоже. И так далее.

Приятный дополнительный эффект этого подхода состоит в том, что у команды появляется человек, который отлично подходит на роль организатора демонстрации результатов спринта.

Чем занимается тестировщик, когда нечего тестировать?

Этот вопрос никогда не теряет своей актуальности. Мистер Т: "Scrum-мастер, сейчас нечего тестировать. Чем *я* могу заняться?". Может пройти неделя, пока команда закончит первую историю, так чем же будет заниматься тестировщик *всё это* время?

Для начала, ему следует заняться *подготовкой к тестированию*. А именно: написанием спецификаций тестов, подготовкой тестового окружения и так далее. Таким образом, когда у разработчика появится что-нибудь готовое к тестированию, мистер Т должен быть готов начать тестирование.

Если команда практикует TDD, люди с первого дня заняты написанием тестирующего кода. В этом случае, тестировщик может заняться парным программированием с разработчиками, пишущими тестирующий код. Если же тестировщик вообще не умеет программировать, ему следует работать в паре с разработчиком в роли "штурмана", дав напарнику возможность печатать. У хорошего тестировщика обычно получается выдумать больше разных тестов, чем у хорошего разработчика, поэтому они друг друга дополняют. Если же команда не занимается TDD или же количества подлежащих написанию тестов недостаточно, чтобы полностью загрузить тестировщика, он просто может делать всё что угодно, чтобы помочь команде достичь цели спринта. Как и любой другой член команды. Если тестировщик умеет программировать – отлично. Если нет, команде придется выявить все задания, не требующие навыков программирования, но которые необходимо выполнить за спринт.

Когда на планировании спринта истории разбиваются на задачи, то в первую очередь команда старается сфокусироваться на задачах, требующих *программирования*. Однако, как правило, существует множество задач, которые *не требуют программирования*, но, тем не менее, подлежат выполнению по ходу спринта. Если вы в течение планирования спринта потратите немного времени на *определение задач, которые не требуют программирования*, то мистер Т получит возможность сделать для достижения цели спринта очень много. Даже если он не умеет программировать или в этот момент нечего тестировать.

Вот некоторые примеры задач, которые не требуют программирования, но которые часто должны быть закончены до конца спринта:

- Установить и настроить тестовое окружение.
- Уточнить требования.
- Детально обсудить процесс установки.
- Написать документы по установке (заметки к релизу, readme.txt или что там требуется в вашей компании).
- Пообщаться с подрядчиками (например, с дизайнерами пользовательского интерфейса).
- Улучшить скрипты автоматизированной сборки.
- Последующее разбиение историй на задачи.
- Собрать ключевые вопросы от разработчиков и проследить, чтобы они не остались без ответов.

С другой стороны, что мы делаем в случае, когда мистер Т оказывается "узким местом"? Скажем, сейчас последний день спринта и большая часть функционала уже реализована, а мистер Т не имеет возможности протестировать всё необходимое. Что мы делаем в этом случае? Ну, мы можем сделать всех членов команды помощниками мистера Т. Он решает, что он может сделать самостоятельно, а простые задачи поручает остальным членам команды. Вот, что такое кросс-функциональная команда!

Итак, мистер Т *действительно* играет особую роль в команде, но он, кроме того, может выполнять работу других членов команды, так же, как и другие члены команды могут делать его работу.

Повышайте качество – делайте меньше за спринт!

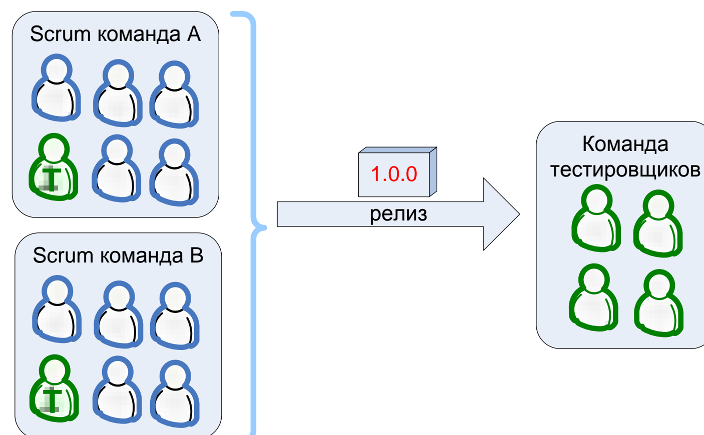
Это решается ещё на планировании спринта. Проще говоря, не пытайтесь сделать как можно больше историй за спринт. Если у вас существуют проблемы с качеством или вам приходится тратить слишком много времени на приёмочное тестирование, просто делайте меньше за спринт! Это автоматически приведёт к повышению качества, уменьшит продолжительность приёмочного тестирования и количество багов, которые вылезут у конечного пользователя. В конце концов, это должно поднять производительность всей команды, ведь она сможет сконцентрироваться на новых задачах, вместо того, чтобы постоянно тратить время на исправление старого кода, который постоянно ломается.

Почти всегда получается дешевле сделать меньше, но качественнее, чем больше, но потом в панике латать дыры.

Стоит ли делать приёмочное тестирование частью спринта?

Тут у нас полный «разброд и шатание». Некоторые наши команды включают приёмочное тестирование в спринт, однако, большинство – нет. И вот почему:

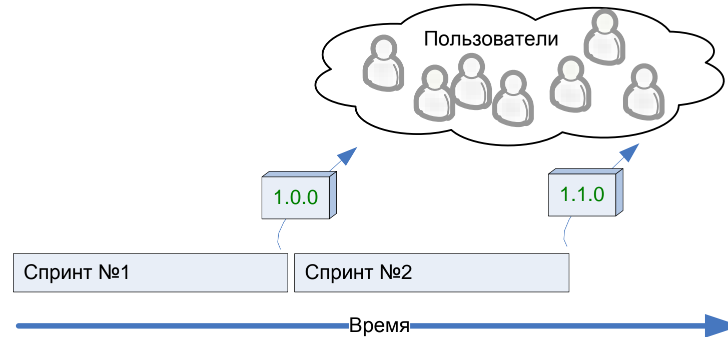
- Спринт ограничен во времени. Приёмочное тестирование (которое, если брать во внимание моё определение, включает отладку и повторный выпуск продукта), довольно сложно втиснуть в чёткие временные рамки. Что если время уже вышло, а в системе остался критический баг? Что тогда? Собираетесь ждать завершения ещё одного спринта? Или выпустите готовый продукт с этим багом? В большинстве случаев оба вариант неприемлемы. Именно поэтому мы выносим ручное приёмочное тестирование за пределы спринта.
- Если две Scrum-команды работают над одним продуктом, тогда ручное приёмочное тестирование необходимо проводить, собрав результаты работы обеих команд. Если обе команды включают в спринт ручное приёмочное тестирование, тогда всё равно нужна команда, которой придётся протестировать финальный релиз, который получается после интеграции результатов работы обеих команд.



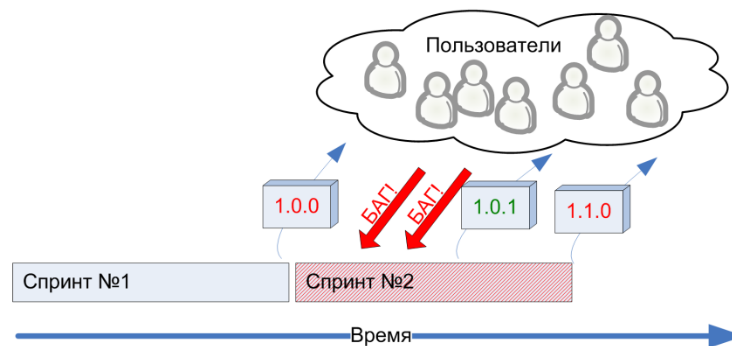
Это далеко не идеальное решение, но в большинстве случаев оно нас устраивает.

Соотношение спринтов и фаз приёмочного тестирования

В идеальном Scrum-мире фаза приёмочного тестирования не нужна, так как каждая Scrum-команда после каждого спринта выдаёт новую, готовую к реальному использованию версию системы



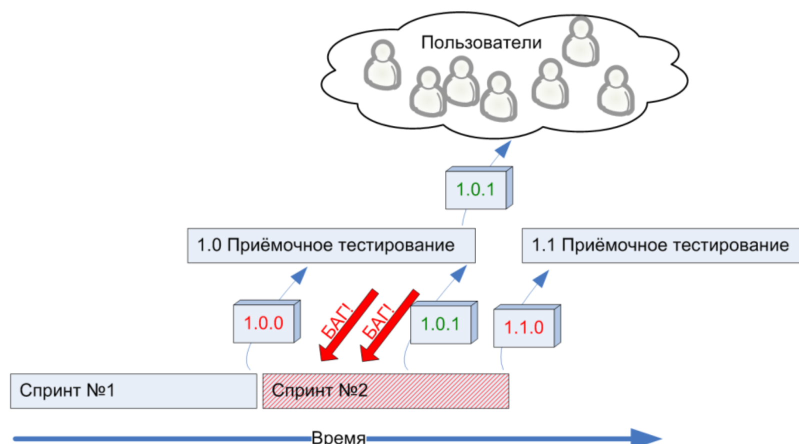
Ну, а на самом деле, всё выглядит чуть-чуть по-другому:



После первого спринта выпускается глючная версия 1.0.0. Во время второго спринта начинают поступать сообщения об ошибках, и команда большую часть времени занимается отладкой, а потом выпускает версию с исправлениями 1.0.1 в середине спринта. Потом, в конце второго спринта выходит версия 1.1.0 с новым функционалом, которая, естественно, оказывается ещё более глючной, так как у команды просто не хватило времени довести её до ума из-за того, что приходилось подчищать хвосты, оставшиеся с прошлого спринта. И так по кругу.

Наклонная красная штриховка второго спринта символизирует хаос.

Неприглядная картина, да? А самое грустное в том, что эта проблема остаётся даже при наличии команды приёмочного тестирования. Единственная разница состоит в том, что основная масса сообщений об ошибках поступает от команды тестирования, а не от негодующих пользователей. Но эта разница просто огромна с точки зрения бизнеса, хотя для разработчиков ничего и не меняется. Ну, кроме только того, что тестировщики обычно менее агрессивны, чем конечные пользователи. Обычно.



Простого решения этой проблемы мы так и не нашли. Но наэкспериментировались с разными подходами вдоволь.

Перво-наперво, опять же, необходимо обеспечить наивысшее качество кода, который создаёт Scrum-команда. Стоимость раннего обнаружения и исправления ошибки (в пределах спринта) несравнимо ниже стоимости обнаружения и исправления ошибки после окончания спринта.

Но факт остаётся фактом: как бы мы не уменьшали количество ошибок, они обязательно найдутся и после завершения спринта. Так что же с этим делать?

Подход №1: "Не начинать новые истории, пока старые не будут готовы к реальному использованию"

Звучит классно, не так ли? Вас тоже греет эта мысль? :)

Несколько раз мы уже было решались на этот подход, и даже рисовали теоретические модели того, как бы могли это сделать. Но каждый раз мы останавливались, когда рассматривали обратную сторону медали. Нам бы пришлось ввести неограниченную по времени итерацию между спринтами, в которую бы мы только тестировали и исправляли ошибки до тех пор, пока у нас на руках не было бы готового к использованию релиза.

Наличие неограниченной во времени итерации между спринтами нам не нравилось в основном из-за того, что она бы нарушила регулярность спринтов. Мы не смогли бы больше заявлять: "Каждые три недели мы начинаем новый спринт". А кроме того она не решает проблему. Даже если мы введём такую итерацию, всё равно время от времени будут появляться ошибки, срочно требующие внимания, и нам надо быть к этому готовыми.

Подход №2: "Начинать реализовывать новые истории, но наивысшим приоритетом ставить доведение старых до ума"

Мы предпочитаем этот подход. По крайней мере, до сих пор так и было.

По сути, он состоит в следующем: когда мы заканчиваем спринт, мы переходим к следующему, но учитываем, что в следующем спринте нам потребуется время на исправление багов прошлого спринта. Если следующий спринт оказывается перегружен работой над исправлением дефектов прошлого, то мы пытаемся понять причину такого количества дефектов и выработать способ поднять качество. И мы выбираем длину спринта достаточной, чтобы успеть справиться с приличным объёмом работы по исправлению багов прошлого спринта.

Постепенно, за несколько месяцев, количество работы по устранению дефектов прошлых спринтов уменьшилось. Кроме того, мы смогли добиться, чтобы на устранение дефекта требовалось отвлекать меньше людей, то есть, нет нужды беспокоить всю команду по поводу каждого бага. Теперь хаос в наших спринтах снизился до приемлемого уровня.

При планировании спринта, чтобы учесть то время, которое мы планируем потратить на устранение дефектов, мы устанавливаем уменьшенное значение фокус-фактора. Со временем команды начинают очень хорошо определять нужное значение фокус-фактора. В этом также очень помогает статистика реальной производительности (см. стр. 23 "Как команда принимает решение о том, какие истории включать в спринт?")

Неправильный подход: "Клепать новые истории"

Если перефразировать, то это значит: "реализовывать новые истории, *вместо того, чтобы довести старые до ума*". Казалось бы – да кому такое может прийти в голову? А, тем не менее, мы частенько допускали эту ошибку в начале и, я уверен, что и многие другие компании тоже. Эта неадекватность связана со стрессом. На самом деле многие менеджеры не понимают, что когда кодирование закончено, то, мы, как правило, всё ещё далеки от релиза, готового к использованию. Поэтому менеджер (или product owner) просит команду наращивать функционал продукта в то время, как груз почти-готового-к-выпуску кода становится всё тяжелее и тяжелее, замедляя весь процесс.

Не забывайте об ограничении системы

Допустим приемочное тестирование – это ваше самое узкое место. Например, у вас слишком мало тестировщиков или фаза приемочного тестирования забирает много времени из-за ужасного качества кода.

Допустим, команда, которая выполняет приемочное тестирование, успевает проверить 3 фичи в неделю (не подумайте, мы не используем "фичи в неделю" как метрику; я взял это для примера). Также допустим, что ваши разработчики могут сделать 6 новых фич в неделю.

Для менеджера или product owner'а (даже для самой команды) будет искушением запланировать разработку 6-ти новых фич в неделю.

Только не это! Витать в облаках долго не получится, а когда упадёте – будет больно.

Лучше при планировании рассчитывать на 3 фичи в неделю, а оставшееся время направить на устранение узкого места. Например:

- Заставить разработчиков потестировать (приготовьтесь к "благодарности" за это...).
- Внедрить новые инструменты и скрипты, которые упростят тестирование.
- Добавить больше автоматизации.
- Сделать длиннее спринт и включить приемочное тестирование в спринт.
- Выделить несколько "тестовых спринтов", где вся команда будет работать над приемочным тестированием.
- Нанять больше тестировщиков (даже если это означает уволить некоторых разработчиков).

Мы пробовали все эти решения (кроме последнего). С точки зрения долгосрочной перспективы лучшими являются пункты 2 и 3, а именно: более эффективные инструменты, скрипты и автоматизация тестирования.

А для выявления узких мест лучше всего подходят ретроспективы.

Возвращаясь к реальности

У вас, наверное, сложилось впечатление, что у нас есть тестеры во всех Scrum-командах, и что мы обзавелись ещё одной огромной командой тестеров, которые после каждого спринта проводят приёмочное тестирование всех готовых продуктов.

Ну ... это не так совсем.

У нас *всего несколько раз* получилось выделить время на все эти процедуры, но тогда мы на собственном опыте убедились насколько это полезно. Могу сказать, что на данный момент мы всё ещё далеки от желаемого процесса обеспечения качества, и нам по-прежнему есть чему учиться.

15

Как мы управляем несколькими Scrum-командами

Когда у вас над одним продуктом работают сразу несколько Scrum-команд, всё намного сложнее. Это общая проблема и она характерна не только для Scrum'а. Чем больше разработчиков, тем больше проблем.

Мы и с этим экспериментировали (как обычно). Максимальный размер команды, работавшей у нас над одним проектом, был порядка 40-ка человек.

Как оказалось, ключевыми являются следующие вопросы:

- Сколько сформировать команд?
- Как распределить людей по командам?

Сколько сформировать команд

Если настолько сложно работать по Scrum'у с несколькими командами, то зачем вообще заморачиваться? Почему просто не собрать всех в одну команду?

Самая большая Scrum-команда, которая у нас когда-либо была, состояла из 11-ти человек. И это работало, правда, не очень хорошо. Ежедневный Scrum всегда длился больше 15-ти минут. Членам команды было сложно держать в голове информацию о том, чем каждый из них занимается, из-за этого могли возникать недоразумения. ScrumMaster'у тоже было сложно направить работу в нужное русло, и было сложно найти время, чтобы разобраться со всеми возникшими трудностями.

Как вариант, можно выделить две команды. Но будет ли это лучше? Не факт.

Если команда опытная, ей комфортно работать по Scrum'у, и вы знаете, как правильно разделить работу на два отдельных направления, что позволит избежать одновременной работы над одними и теми же исходниками, тогда я скажу, что это хорошая идея: разделить на отдельные команды. В противном же случае, не смотря на все минусы работы в большой команде, я бы подумал о том, как оставить одну большую команду.

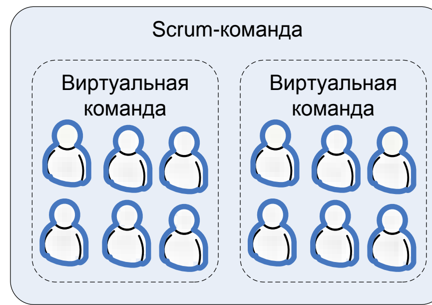
Мой опыт показывает, что намного лучше иметь несколько больших команд, чем много маленьких, которые постоянно будут мешать друг другу. Создавайте маленькие команды только тогда, когда они не нуждаются во взаимодействии друг с другом.

Виртуальные команды

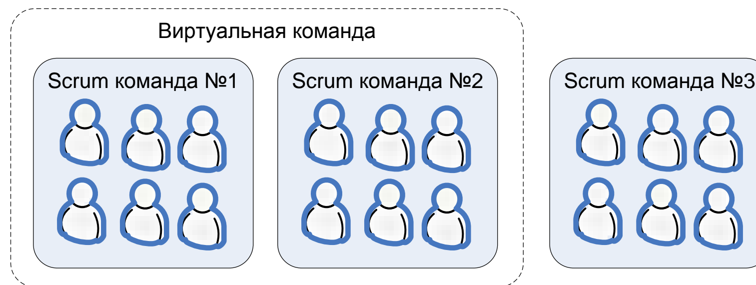
Как же понять, насколько правильно были оценены преимущества и недостатки при выборе между "большой командой" или "несколькими командами"?

Будьте предельно внимательны, и вы заметите формирование "виртуальных команд".

Пример 1: Вы остановились на одной большой команде. Но как только начнёте наблюдать за тем, кто с кем общается на протяжении спринта, то заметите, что команда фактически разбивается на две подкоманды.



Пример 2: Вы решили сделать три небольшие команды. Но как только начнёте прислушиваться, кто и с кем говорит в ходе спринта, то заметите, что первая и вторая команды общаются между собой, тогда как третья работает сама по себе.



Что бы это значило? Что ваша стратегия разделения была неправильной? И да (если виртуальные команды постоянные) и нет, (если виртуальные команды временные).

Взгляните ещё раз на первый пример. Если состав обеих виртуальных подкоманд меняется (т.е. люди переходят из одной виртуальной подкоманды в другую), тогда возможно вы приняли правильное решение, создав одну большую Scrum-команду. Если же две виртуальные подкоманды в ходе спринта остаются неизменными, то, возможно, для следующего спринта их следует разбить на две настоящие независимые Scrum-команды.

Теперь вернёмся ко второму примеру. Если первая и вторая команды общаются друг с другом (но не с третьей) на протяжении всего спринта, тогда возможно для следующего спринта следует объединить первую и вторую команду в одну Scrum-команду. Если первая и вторая команда очень плотно общаются в первой половине спринта, а во второй половине спринта уже первая и третья команды ведут оживлённые беседы друг с другом, тогда думаю, вам стоило бы рассмотреть возможность объединения всех трёх команд в одну, или всё-таки оставить эти команды как есть. Поднимите этот вопрос на ретроспективе и дайте возможность командам самим решить его.

Разбиение на команды – это действительно одна из самых сложных задач в Scrum'е. Не пытайтесь копать очень глубоко или заниматься очень сложной оптимизацией. Экспериментируйте, наблюдайте за виртуальными командами, и не забывайте уделять обсуждению этого вопроса достаточно времени на ретроспективах. Рано или поздно вы найдёте для себя правильное решение. Однако запомните, что вам следует сделать всё, чтобы команды чувствовали себя комфортно и не мешали друг другу слишком часто.

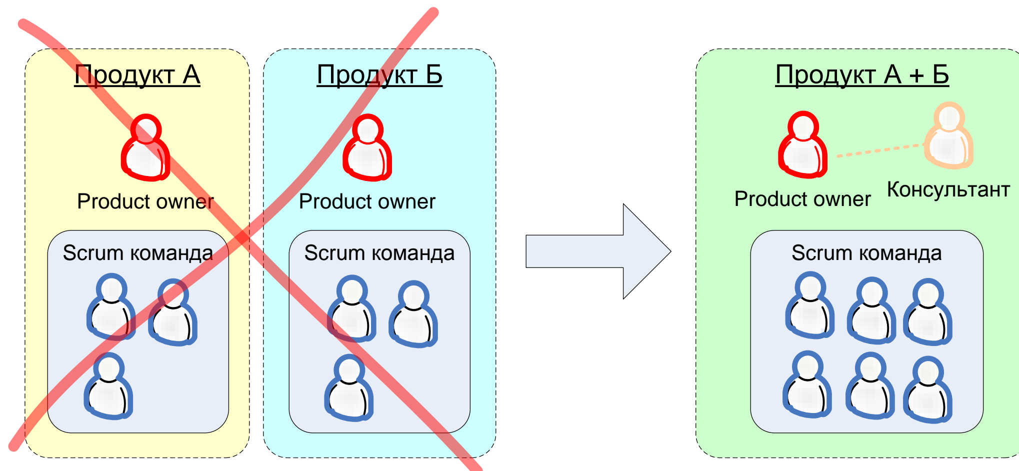
Оптимальный размер команды

Практически во всех книгах, которые я читал, утверждается, что "оптимальный размер" команды составляет примерно 5-9 человек.

Исходя из того, что я видел, остаётся только согласиться с этим мнением. Хотя, как по мне, всё-таки лучше иметь команду от 3-ёх до 8-ми человек. Поднапрягитесь, но создайте команды такого размера.

Допустим, у вас есть одна Scrum-команда из 10-ти человек. Подумайте, может быть стоить выкинуть двух наиболее слабых участников команды. Ой-йо-йой, я что – правда, сказал это?

Предположим, вы разрабатываете два разных продукта, у вас по три человека в каждой команде, однако обе команды работают очень медленно. *Может быть,* их следует объединить в одну команду из 6-ти человек. И пусть эта команда одновременно создаёт оба продукта. В этом случае одному из двух product owner'ов придётся уйти (или начать играть роль консультанта, ну или ещё что-то наподобие этого).

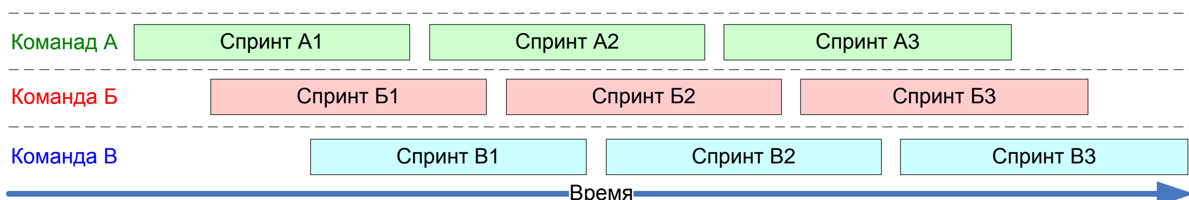


Допустим, у вас одна Scrum-команда из 12 человек, которую нереально разбить на две независимые команды только потому, что исходный код страшно запущен. Вам придётся приложить максимум усилий, чтобы отрефакторить (вместо того чтобы клепать новый функционал) исходный код до такой степени, чтобы над ним могли работать независимые команды. Поверьте мне, что, скорее всего, ваши "инвестиции" окупятся с лихвой.

Синхронизировать спринты или нет?

Предположим, есть три Scrum команды, которые работают над одним проектом. Должны ли их спринты быть синхронизированными, т.е. начинаться и заканчиваться одновременно? Или же они должны пересекаться друг с другом?

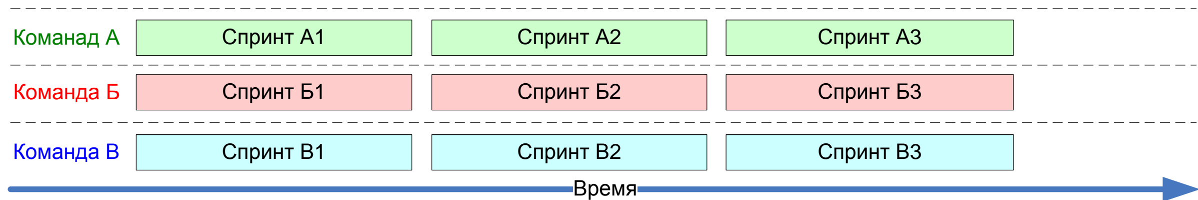
Сначала мы решили, что нужны пересекающиеся спринты (по времени).



Это звучало круто. В любой момент времени у нас был бы спринт, который вот-вот закончится, и спринт, который вот-вот начнётся. Нагрузка на product owner'а была бы распределена равномерно по времени. Постоянные релизы продукта. Демонстрации каждую неделю. Аллилуйя.

Да-да, утопия... но тогда это *действительно* звучало убедительно!

Мы только-только начали так работать, но тут мне подвернулась возможность пообщаться с Кеном Швабером (Ken Schwaber) (в рамках моей Scrum-сертификации). Он указал на то, что это *неправильно*, и что было бы гораздо лучше синхронизировать спринты. Я точно не помню его доводов, но в ходе нашей дискуссии он меня убедил в этом.



С тех пор мы использовали это решение и ни разу об этом не пожалели. Я никогда не узнаю, провалилась бы стратегия пересекающихся спринтов или нет, но думаю, что да. Преимущества синхронизированных спринтов в следующем:

- Появляется естественная возможность перетасовывать команды между спринтами! При пересекающихся спринтах нет возможности реорганизовывать команды так, чтобы не побеспокоить ни одной команды в разгаре спринта.
- Все команды могут работать на одну цель в течение спринта и проводить планирование спринта вместе, что приводит к лучшему сотрудничеству между командами.
- Меньше административной мороки, например меньшее количество встреч для планирования спринта, демонстраций и релизов.

Почему мы ввели роль "тимлида"

Предположим, что у нас над одним продуктом работают три команды.

Красным помечен product owner. Чёрным – Scrum Master'a. А остальные это ~~пехота~~... ну то есть... почтенные члены команды.

Кто при таком распределении ролей должен решать, какие люди будут отнесены к какой команде. Может, product owner? Или может все три ScrumMaster'a вместе? Или вообще каждый человек сам решает, в какой команде работать? Но если так, то что делать в случае, когда все захотят в одну и ту же команду (потому что там *красивая* ScrumMaster'ша)?

А что если потом окажется, что над этим кодом больше двух команд работать не смогут, и нам придётся трансформировать три команды по 6 человек в две по 9? Это значит, что будет всего 2 ScrumMaster'a. Так кого же из трёх текущих ScrumMaster'ов надо лишить титула?

Во многих компаниях эти вопросы надо решать очень деликатно.

Есть соблазн отдать распределение людей по командам и их последующее перераспределение на откуп product owner'у. Но ведь это не совсем то, чем должен заниматься product owner, правда же? Product owner это специалист по предметной области, который указывает команде направление движения. В общем случае его не должно волновать всё остальное. Особенно, учитывая его роль "цыплёнка" (это если вы слышали про метафору "свиней и цыплят", а если не слышали, то погуглите).

Мы решили проблему вводом роли "тимлид". Человека в этой роли можно описать как "Scrum-of-Scrums master", "босс" или "главный ScrumMaster". Ему не нужно возглавлять какую-либо команду, но он отвечает за

вопросы, не входящие в компетенцию команд, такие как, например, "кого назначить ScrumMaster'ом", "как распределить людей по командам" и т.д.

Придумать действительно подходящие название для этой роли у нас так толком и не получилось. А "тимлид" оказалось наименее неподходящим, из тех, что мы перепробовали.

Такой подход сработал в нашем случае, и я его рекомендую (не зависимо от того, как вы решите назвать эту роль у себя).

Как мы распределяем людей по командам

На случай, когда у вас несколько команд работают над одним и тем же продуктом, существует две стратегии распределения людей по командам.

- Позволить специально назначенному человеку провести распределение, например «тимлиду», про которого я писал выше, product owner'у или любому другому менеджеру (если у него достаточно информации, чтобы с этим справиться).
- Позволить командам каким-то способом самоорганизоваться.

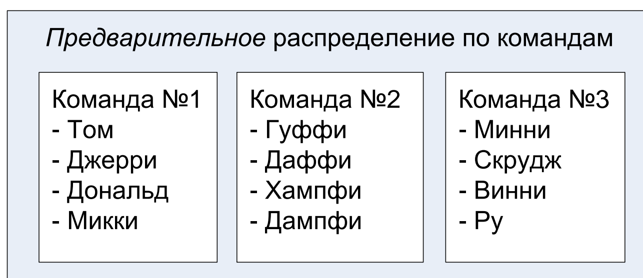
Мы попробовали все три стратегии. Три?!? Ну да – стратегию №1, стратегию №2 и их комбинацию.

И оказалось, что комбинация двух стратегий работает лучше всего.

Перед планированием спринта тимлид приглашает product owner'а и всех ScrumMaster'ов на совещание по поводу формирования команд. Мы обсуждаем прошлый спринт и решаем, есть ли необходимость в переформатировании команд. Возможно, нам нужно объединить две команды или перевести несколько человек из одной команды в другую. Мы решаем, что именно нам нужно, записываем это на листик и несём на планирование спринта как *предварительное распределение по командам*.

Первым делом на планировании спринта мы проходимся по наиболее приоритетным историям из product backlog'а. А потом тимлид говорит что-то вроде:

"Всем привет. Вот как мы предлагаем сформировать команды на следующий спринт".



"Как видно, мы собираемся уменьшить количество команд с четырёх до трёх. Составы команд указаны. Пожалуйста, сгруппируйтесь согласно спискам и выберите себе подходящую стену для планирования".

(тимлид ждёт пока народ побродит по комнате, и через некоторое время появляются три группы людей, каждая соберётся возле своей части стены).

"Текущее распределение – только *прикидка*! Просто чтобы было с чего начать. По мере планирования спринта любой из вас волен переходить в любую другую команду, можно разделять команды, можно, наоборот, объединять их... В общем, делайте всё, что подскажет здравый смысл, учитывая приоритеты, которые озвучивает product owner."

Описанный выше подход оказался для нас наиболее эффективным. Немного директивного управления, которое оптимизируется разумной долей самоуправления.

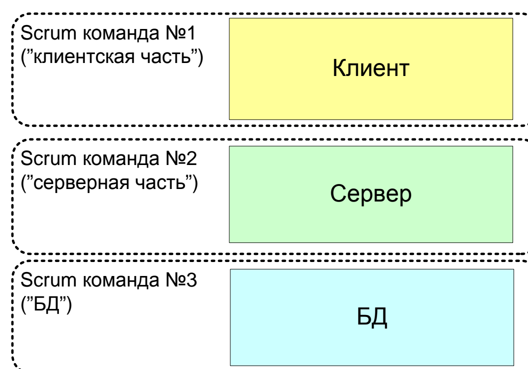
Нужны ли узкоспециализированные команды?

Предположим, ваша система состоит из трёх основных компонентов:

Допустим, что над вашим продуктом работают 15 человек, и вам не очень хочется собирать их в одну Scrum-команду. Как же разделить людей на команды?

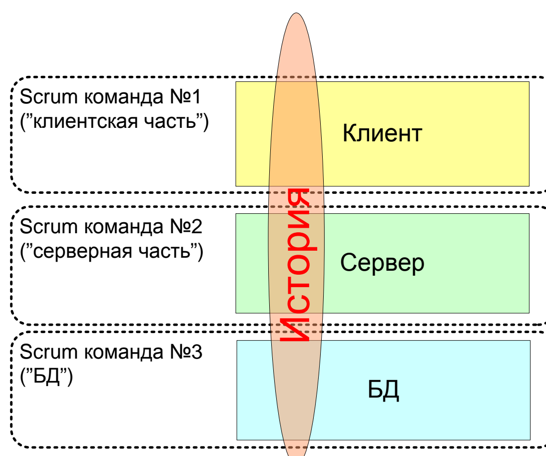
Подход №1: команды, специализирующиеся на компонентах

Можно создать команды, специализирующиеся на конкретных компонентах. Тогда мы получим "команду для клиентской части", "команду для серверной части" и "команду для базы данных".



Именно с этого подхода мы когда-то начинали. Работает не очень хорошо, по крайней в том случае, когда большинство историй затрагивают сразу несколько компонентов.

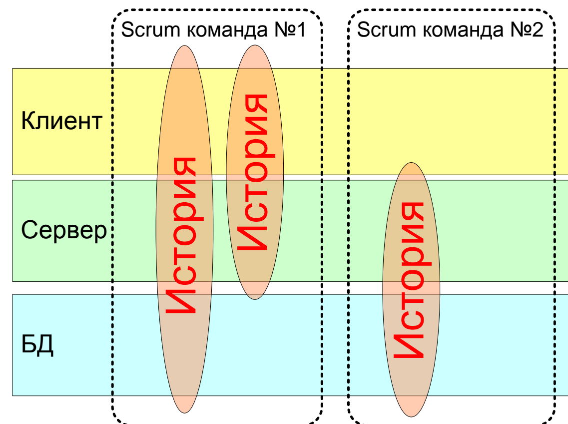
К примеру, возьмём, историю, которая называется "доска объявлений, где пользователи могут оставлять друг другу сообщения". Для создания такой доски объявлений нам придётся обновить пользовательский интерфейс в клиентской части, добавить бизнес-логику на стороне сервера, и добавить парочку таблиц в базу данных.



Это значит, что всем трём командам придётся довольно плотно сотрудничать, чтобы закончить эту историю. Не очень удобно.

Подход №2: универсальные команды

А можно создать универсальные команды, то есть команды, которые не заточены на работу всего лишь с одним специфическим компонентом.



Если большая часть всех историй предполагает работу над несколькими компонентами, тогда такое разделение на команды работает намного лучше. Каждая команда сможет реализовать историю целиком: клиентскую часть, серверную и базу данных. Благодаря чему команды смогут работать намного более независимо друг от друга, что на самом деле ОЧЕНЬ ХОРОШО.

Когда мы начали внедрять Scrum, первым делом мы сделали из всех наших специализированных команд (подход №1) универсальные команды (подход №2). Это уменьшило количество ситуаций "мы не можем закончить задачу, так как ждём, пока эти ребята закончат серверную часть".

Однако иногда нам всё-таки приходится собирать временные команды, специализирующиеся на разработке отдельных компонентов.

Стоит ли изменять состав команды между спринтами?

Обычно каждый спринт обладает своими собственными особенностями в зависимости от того, какого рода задачи мы пытаемся решить. Как следствие, оптимальный состав команды для каждого спринта может отличаться.

Фактически, почти каждый спринт нам приходилось говорить себе что-то вроде: "*этот* спринт не совсем *обычный* спринт, потому что (ля-ля-ля)...". Через некоторое время мы прекратили использовать понятие "обычный" спринт. Обычных спринтов просто нет. Так же как нет "обычных" семей или "обычных" людей.

Для одного спринта может показаться хорошей идеей, создать команду, которая занимается клиентской частью приложения и включает в себя всех, кто хорошо знает код клиента. Для другого спринта хорошей идеей может быть создание двух универсальных команд и разделение специалистов по клиентской части между ними.

Одним из ключевых аспектов Scrum'а является "сработанность команды", т.е. если члены команды работают вместе в течение многих спринтов, они обычно становятся *очень сплоченными*. Они научатся входить в *групповой поток,* и достигнут невероятного уровня продуктивности. Но чтобы достичь этого требуется несколько спринтов. Если вы будете часто изменять состав команды, то вы никогда не достигнете настоящей командной сработанности.

Поэтому, если вы решили изменить состав команды, учитывайте все последствия. Будут ли это долговременные или кратковременные изменения? Если кратковременные, стоит их пропустить. На долговременные изменения можно пойти.

Есть одно исключение: большая команда, которая только-только начала работать по Scrum'у. В этом случае возможны некоторые эксперименты с разделением команды на подкоманды, пока не будет найден вариант, который полностью устраивал бы всех. Удостоверьтесь, что все понимают, что отрицательный результат – тоже результат, что первые несколько итераций могут быть комом – и это нормально, при условии, что вы работаете над улучшениями.

Участники команды с частичной занятостью

Могу только подтвердить то, что говорят книги, посвящённые Scrum'у: наличие в Scrum-команде участников с частичной занятостью – не очень хорошая идея.

Предположим, вы рассматриваете возможность взять Джо в свою команду как участника с частичной занятостью. Сначала хорошо всё обдумайте. Действительно ли Джо необходим вашей команде? Уверены, что не можете заполучить его на полный день? Какие у него ещё обязанности? Можно ли передать обязанности Джо кому-то другому и перевести его на роль консультанта? Можно ли заполучить Джо на полный день, начиная со *следующего* спринта, а пока передать его обязанности кому-то другому [участнику вашей команды – прим. переводчика]?

Но иногда просто нет выбора. Вам позарез нужен Джо потому, что он единственный администратор баз данных (DBA) во всём здании. Другим командам он нужен так же сильно, как и вам, поэтому он никак не может работать с полной занятостью в вашей команде. Кроме того, компания не может себе позволить нанять ещё одного DBA. Ну и ладно. Это аргументированный случай, чтобы взять его на неполную занятость (что, кстати, было у нас). Но прежде, чем сделать это, всегда проводите подобный анализ.

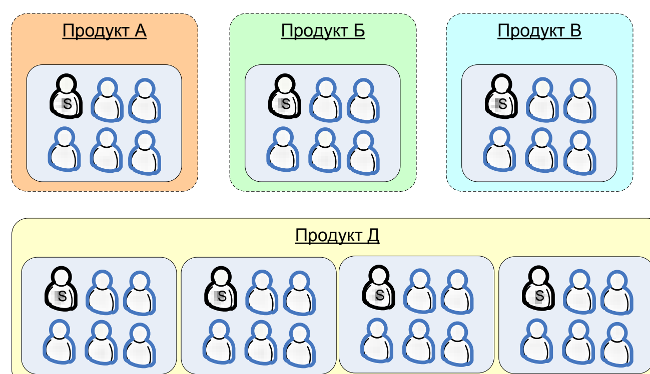
В обычной ситуации я бы предпочёл команду, состоящую из трёх участников с полной занятостью, чем из восьми, но с частичной.

Если у вас есть человек, являющийся участником нескольких команд, как, например вышеупомянутый DBA, всё равно неплохо бы его закрепить за одной командой. Определите, какая команда нуждается в нём больше всего, и назначьте её в качестве "домашней команды". Когда его никто не будет дёргать, он будет присутствовать на ежедневных Scrum'ах, планированиях спринтов, ретроспективах и т.д. этой команды.

Как мы проводим Scrum-of-Scrums

Scrum-of-scrums – это регулярные встречи, цель которых – обсуждение различных вопросов между Scrum-мастерами.

Как-то мы работали над четырьмя продуктами. Над тремя из них работало по одной Scrum-команде, а над четвёртым – 25 человек, которые были разделены на несколько Scrum-команд. Это выглядело следующим образом:



У нас было два уровня Scrum-of-Scrums: "уровня продукта", который проводился с участием команд продукта Д, и "уровня компании" для участников всех команд.

Scrum-of-Scrums уровня продукта

Эта встреча была очень важной. Мы проводили её не реже одного раза в неделю. Мы обсуждали проблемы интеграции, балансировки команд, подготовку к следующему планированию спринта и т.д. Мы выделяли на это 30 минут, но часто нам их не хватало. В качестве альтернативы можно было бы проводить ежедневный Scrum-of-Scrums, однако, мы так и не собрались опробовать его.

Наша повестка дня имела следующий вид:

1. Каждый по очереди рассказывал, что его команда сделала на прошлой неделе, что планирует закончить на этой недели, и с какими трудностями они столкнулись.
2. Любые другие проблемы, относящиеся к компетенции нескольких команд одновременно, которые нужно обсудить. Например, вопросы интеграции.

На самом деле повестка дня Scrum-of-Scrums не так уж и важна – важнее, чтобы эта встреча проводилась регулярно.

Scrum-of-Scrums уровня компании

Мы назвали эту встречу "Пульсом". Мы пробовали проводить её в разных форматах с разными участниками. В конце концов, мы отказались от всех остальных идей в пользу еженедельного собрания продолжительностью 15 минут, в котором участвует весь коллектив (вообще-то все те, кто участвуют в процессе разработки).

Чегооо? 15 минут? Весь коллектив? Все участники всех продуктовых команд? И это работает?

Да – работает, если вы (или ответственный за проведение этого собрания) очень строги относительно того, чтобы собрание было сжатым.

Формат собрания:

1. Новости и уточнения со стороны руководителя разработки. Например, информация о предстоящих мероприятиях, событиях.
2. "Карусель". Один человек из каждой продуктовой группы [группы команд, вовлечённых в разработку одного продукта – прим. переводчика] отчитывается в том, что было сделано за прошлую неделю, что планируется сделать на этой неделе и о проблемах. Некоторые другие люди так же отчитываются (например, начальник отдела по работе с клиентами, начальник отдела контроля качества).
3. Все, кто хочет, могут свободно высказаться и задать любые вопросы.

Это собрание для подачи сжатой информации, а не для дискуссий или рефлексии. Если этим и ограничиться, то 15-ти минут вполне хватает. Иногда оно занимает больше времени, но очень редко больше 30ти минут. Если завязывается интересная дискуссия, я её приостанавливаю и предлагаю всем заинтересованным остаться и продолжить её после собрания.

Почему мы проводим "Пульс" всем коллективом? Потому, что мы заметили, что Scrum-of-Scrums уровня компании посвящен преимущественно отчётности. На нём очень редко возникали дискуссии. Кроме того, информация, озвучиваемая на Scrum-of-Scrum’е, очень интересна и многим из тех, кто на него не попадает. Обычно командам интересно, чем занимаются другие команды. И мы посчитали, если всё равно нужно тратить время на информирование друг друга, то почему бы не присутствовать всем.

Чередование ежедневных Scrum'ов

Если у вас есть много Scrum команд, работающих над одним продуктом, и у всех ежедневный Scrum происходит в одно и то же время, может возникнуть проблема. Product Owner'ы (и не в меру надоедливые

люди вроде меня) обладают физической возможностью посещать только один ежедневный Scrum в один момент времени.

Поэтому мы просим команды разнести время проведения ежедневных Scrum'ов.

	Комната №1	Комната №2
9:00	Команда №1	
9:15		Команда №2
9:30	Команда №3	
9:45		Команда №4
10:00	Команда №5	

Этот пример расписания относится к тому периоду, когда у нас был ежедневный scrum в разных комнатах, а не в комнате совещаний команды. Обычно встречи планируются на 15 минут, но каждая команда получала 30 минутный временной слот на случай, если ей надо было немного задержаться.

Это *очень удобно* по двум причинам:

1. Люди, подобные мне и product owner'у, могут посетить *все* ежедневные scrum'ы за одно утро. Нет лучшего способа получить представление о том, как проходит спринт, и в чем основные проблемы.
2. Команды могут посещать ежедневные Scrum'ы друг друга. Не очень часто, но иногда случается, что две группы работают в смежных областях, и участники групп заглядывают друг к другу на ежедневные Scrum'ы, чтобы быть в курсе происходящего.

Обратная сторона медали заключается в ограничениях для команды – теряется возможность изменить время проведения ежедневного Scrum'а. Хотя, на самом деле, для нас это не составляло проблемы.

«Пожарные» команды

Мы столкнулись с ситуацией, когда было невозможно внедрить Scrum в большом проекте из-за того, что его команда постоянно тушила пожары, т.е. в панике устраняла дефекты преждевременно выпущенной системы. Это был порочный круг: из-за того, что всё время уходило на постоянную борьбу с пожарами, не было времени на их предотвращение (т.е. на улучшение архитектуры, внедрение автоматического тестирования, создание систем мониторинга и оповещения, и т.п.)

Мы разрубили этот гордиев узел тем, что выделили специальную «пожарную» команду и отдельную Scrum-команду.

Задачей Scrum-команды (с благословления product owner'а) была стабилизация системы и предотвращение потенциальных пожаров. А «пожарная» команда (её мы назвали «командой поддержки») выполняла две задачи:

1. Тушить пожары.
2. Прикрывать Scrum-команду от всяких раздражителей, вроде неожиданных запросов на изменение функционала, которые непонятно откуда берутся.

«Пожарную» команду мы разместили поближе к дверям, а Scrum-команду – подальше в комнате. Таким образом, «пожарная» команда могла даже *физически защищать* Scrum-команду от раздражителей типа жаждущих общения "продажников" или разозлённых клиентов.

В каждую команду были включены старшие разработчики, чтобы команды не слишком зависели друг от друга.

В основном этот подход был решением проблемы внедрения Scrum'а. Как вообще можно начать практиковать Scrum, если команда не может спланировать свою работу больше, чем на один день вперёд? Нашим ответом на это, как сказано выше, было разделение команд.

Сработало это достаточно хорошо. Так как Scrum-команде дали возможность планировать свою работу, она, в конце концов, смогла стабилизировать систему. А тем временем пожарная команда полностью отказалась от попыток что-либо планировать и работала полностью по запросам, то есть только устраняла очередной проявившийся ужасный дефект.

Естественно, *полностью* оставить Scrum-команду в покое не получилось. Частенько пожарная команда вынуждена была привлекать к решению вопросов отдельных людей из Scrum-команды, или, в худшем случае, всю команду.

В любом случае через пару месяцев система стала настолько стабильной, чтобы мы могли отказаться от пожарной команды в пользу дополнительных Scrum-команд. «Пожарные» были просто счастливы сдать свои шлемы и присоединиться к обычным Scrum-командам.

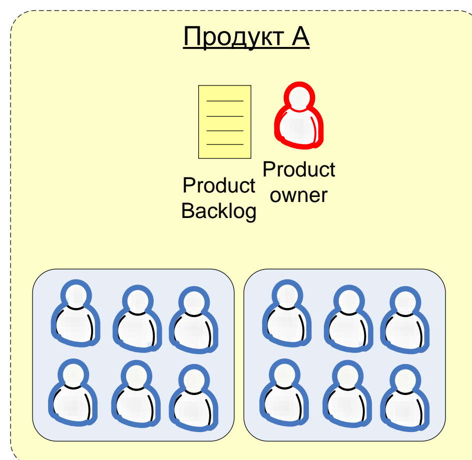
Разбивать product backlog или нет?

Предположим, у вас есть один продукт и две Scrum-команды. Сколько вам нужно product backlog'ов? Сколько product owner'ов? Выбор достаточно сильно повлияет на то, как будут проходить встречи по планированию спринта. Мы оценили три возможных подхода.

Подход первый: Один product owner – один backlog

"Должен остаться только один". Наш любимый подход.

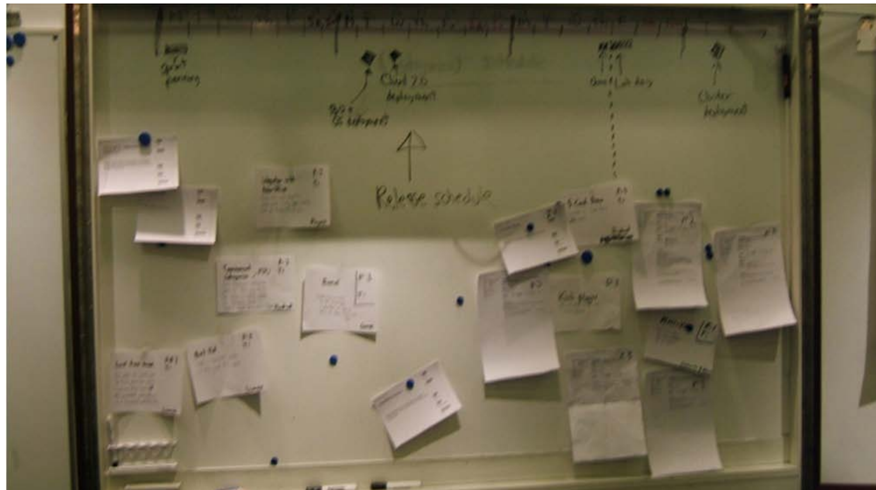
Преимущество этого подхода в том, что можно разрешить команде самой планировать работу на основе приоритетов, расставленных product owner'ом. Product owner может сосредоточиться на том, *что ему нужно*, и предоставить командам самим, разбивать истории на задачи.



Ближе к делу. Давайте посмотрим, как проходит встреча по планированию спринта для этой команды.

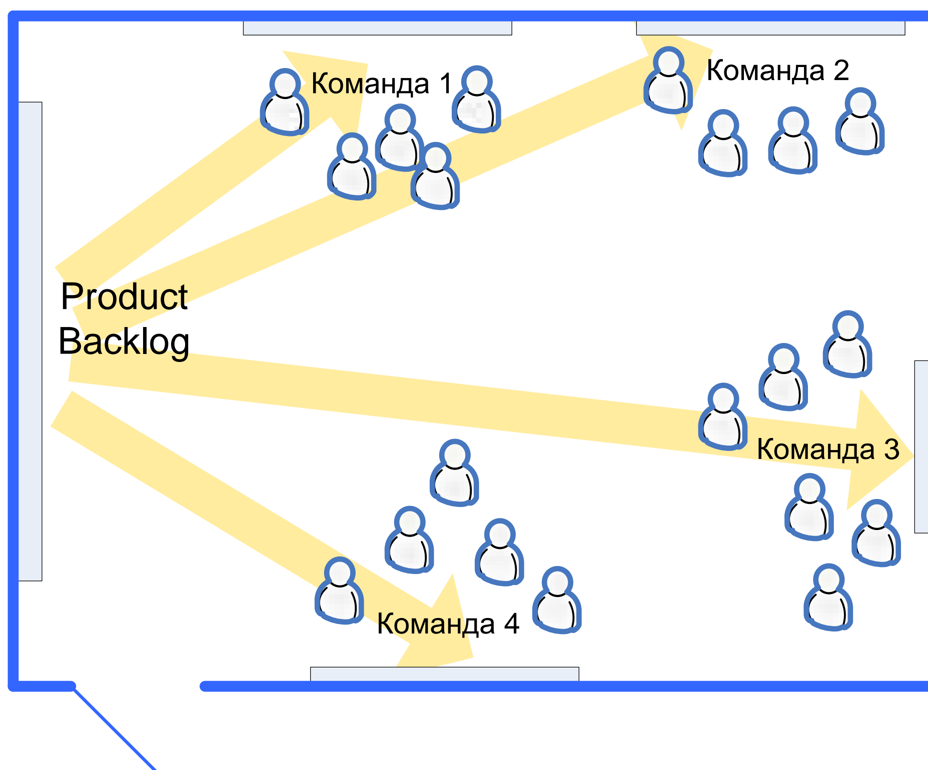
Встреча по планированию спринта проходит на нейтральной территории.

Прямо перед встречей product owner называет одну из стен "стеной product backlog'a" и развешивает на ней истории (учетные карточки), отсортированные по приоритету. Он продолжает вешать их, пока не займёт всю стену. Как правило, такого количества историй более чем достаточно для спринта.



Каждая Scrum-команда выбирает пустой участок стены и вешает там своё название. Это будет их "командная стена". После этого каждая команда отклеивает истории со "стены product backlog'a", начиная с самых важных, и переклеивает их на свою "командную стену".

На рисунке ниже показана описанная ситуация. Желтые стрелки изображают движение учетных карточек со "стены product backlog'a" на стены команд.



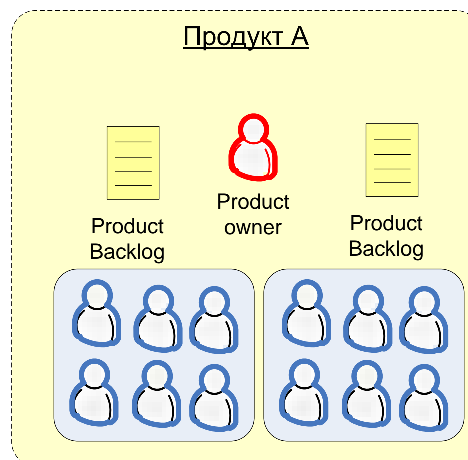
По ходу встречи product owner и команды торгуются за учетные карточки, двигают их между командами, передвигают карточки вверх-вниз, меняя приоритеты, разбивают их на части и т.п. Где-то через час каждая из команд получает предварительную версию sprint backlog'a на своей "командной стене". Дальше команды работают независимо, оценивая истории и разбивая их на подзадачи.

Да, хоть этот дурдом и забирает массу сил, но зато это эффективно, прикольно и способствует общению. Когда время заканчивается, у всех команд, как правило, достаточно информации, чтобы начать спринт.

Подход второй: Один product owner – несколько backlog'ов

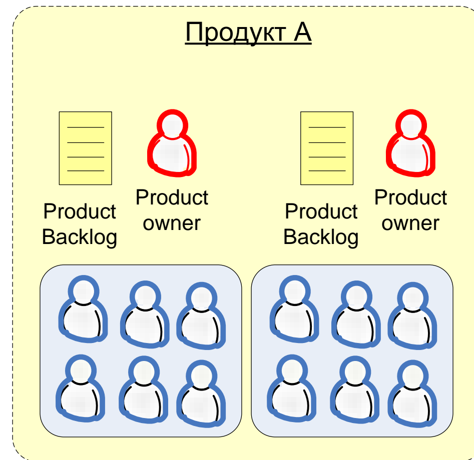
В этом подходе product owner ведёт *несколько* product backlog'ов, по одному на команду. Мы на самом деле не применяли этот подход, но мы делали что-то похожее. Это был наш запасной план на случай, если первый подход не сработает.

Недостатком этого подхода является то, что здесь истории командам раздает product owner, хотя команды, вероятно, справились бы с этим лучше сами.



Подход третий: Несколько product owner'ов – несколько backlog'ов

Похоже на второй вариант, по отдельному product backlog'у на команду, только ещё и с отдельным *product owner'ом* на каждую команду.



Мы не пробовали так делать, и, скорее всего, пробовать не будем.

Если два product backlog'а касаются одного и того же исходного кода, это скорее всего приведёт к серьёзному столкновению интересов между product owner'ами.

Если же два product backlog'а имеют дело с разными исходниками, то это, по большому счету, всё равно, что разбить продукт на отдельные подпродукты, и разрабатывать их независимо. И это значит, что мы вернулись к ситуации "одна команда разрабатывает один продукт", которая для нас приятна и понятна.

Параллельная работа с кодом

При наличии нескольких команд, одновременно работающих над одним исходным кодом, нам неизбежно придется иметь дело с параллельными ветками кода в системе SCM (software configuration management). Есть много книг и статей, рассказывающих, как обеспечить одновременную работу с кодом для нескольких людей, поэтому я не буду вдаваться в детали. Вряд ли мне удастся добавить что-то новое. Однако я хотел бы вкратце поделиться наработками нашей команды:

- Всегда поддерживайте основную ветку проекта в рабочем состоянии. Это, как минимум, означает, что все должно компилироваться, и все юнит-тесты должны проходить. Таким образом, мы получаем возможность в *любой момент* выпустить рабочий релиз. Желательно, чтобы сервер непрерывной интеграции строил и автоматически устанавливал готовый продукт в тестовом окружении каждую ночь.
- Помечайте каждый релиз тэгом. Всякий раз, когда для приемочного тестирования или реального использования выпускается очередной релиз, соответствующая версия кода должна быть помечена тэгом. Это позволит вам при необходимости в любой момент создать в этой точке отдельную ветку для поддержки выпущенного продукта.
- Создавайте новые ветки кода только тогда, когда это действительно необходимо. Хорошим практическим правилом будет создание новой ветки *только* при условии, если вы вынуждены нарушать стратегию использования текущий ветки. Если у вас есть сомнения, то не делайте ветвлений. Почему? Потому что каждое ветвление требует дополнительной работы и последующей поддержки.
- Используйте ветвление для разделения кода *разных стадий разработки*. Вы можете создать для каждой Scrum команды отдельную ветку разработки. Но если вы смешаете краткосрочные изменения с долгосрочными изменениями в одной и той же ветке, вам очень сложно будет выпускать небольшие заплатки!
- Чаще синхронизируйтесь. Если вы работаете в отдельной ветке, синхронизируйтесь каждый раз, когда вы хотите что-либо построить. Каждый день, когда вы начинаете работу, синхронизируйтесь с главной ветвью разработки, так чтобы ваша ветка была в адекватном состоянии и учитывала все изменения, сделанные другими группами. Даже если это приведет к кошмару слияния (merge-hell), учтите, что все могло бы быть еще хуже, если бы вы затянули слияние.

Ретроспектива для нескольких команд

Как мы проводим ретроспективы в случае, если над одним продуктом работает сразу несколько команд?

Сразу же после того как закончилась демонстрация спринта и отшумели бурные овации, команды расходятся по своим комнатам или отправляются в какое-то удобное местечко. И каждая команда проводит свою ретроспективу как это описано на странице 50 "Как мы проводим ретроспективы".

А на планировании (где присутствуют все команды, так как мы стараемся синхронизировать спринты всех команд, которые работают над одним продуктом), мы первым делом выслушиваем представителей каждой команды, на тему наиболее важных моментов последней ретроспективы. Выходит примерно по 5 минут на команду. После этого проводим свободное обсуждение минут на 10-20. И, собственно, только потом начинается планирование спринта.

Мы не пробовали никаких других способов, потому что и такой подход работает достаточно хорошо. Однако, самый большой недостаток этого подхода, состоит в отсутствии возможности немного передохнуть между ретроспективой и началом планирования (см. стр. 55 «Отдых между спринтами»).

В ходе планирования с одиночной командой мы не подводим итоги ретроспективы. В этом нет необходимости, ведь каждый член команды присутствовал на ретроспективе.

16

Как мы управляем географически распределёнными командами

Что же делать, если участники команды находятся в географически разнесённых местах? Большая часть "магии" Scrum'а и XP основана на совместном тесном взаимодействии участников команд, которые программируют парно и встречаются лицом к лицу каждый день.

У нас есть географически распределённые команды. Так же некоторые участники работают дома время от времени.

Наша политика относительно распределённых команд очень простая. Мы используем все возможные способы, чтобы максимизировать пропускную способность средств связи между физически разделёнными участниками команды. Под "пропускной способностью средств связи" я понимаю не только Mbit/sec (хотя это тоже важный показатель). Я имею в виду пропускную способность средств связи в более широком смысле:

- Возможность практиковать парное программирование.
- Возможность встречаться лицом к лицу в ходе ежедневного Scrum'а.
- Возможность увидеть друг друга в любой момент.
- Возможность личных встреч и живого общения.
- Возможность проводить незапланированные совещания всей командой.
- Возможность видеть одни и те же версии sprint backlog'а, sprint burndown'а, product backlog'а и других источников информации по проекту.

Вот некоторые меры, предпринятые нами (или предпринимаемые в настоящий момент) для достижения цели:

- Web-камера и наушники с микрофоном на каждой рабочей станции.
- Комната для проведения телеконференций, оборудованная web-камерами, микрофонами, компьютерами со всем необходимым ПО – для общего доступа к рабочему столу, проведения телеконференций и т.д.
- "Удалённые окна". Большие мониторы в каждом офисе, на которых постоянно можно видеть, что происходит в других офисах. Что-то типа виртуального окна между двумя отделами. Можно стоять перед ним и наблюдать. Например, кто на своём рабочем месте или кто с кем разговаривает. Всё для того, чтобы создать ощущение, что все находятся вместе.

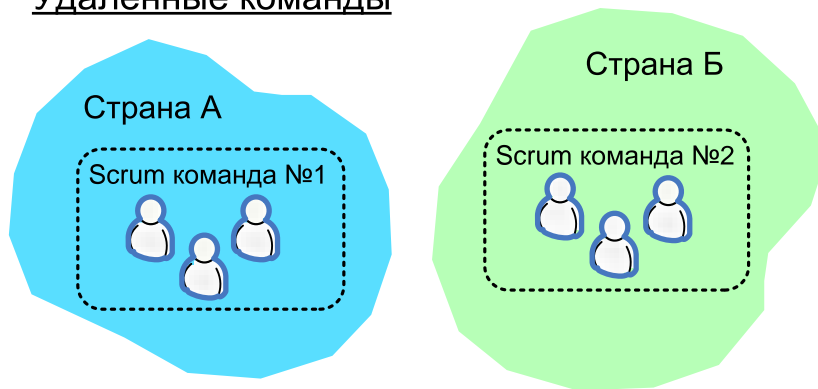
Используя эти и другие техники, мы медленно, но уверенно начинаем понимать, как проводить планирование спринтов, демонстрации, ежедневные scrum'ы и пр. в географически распределённых командах.

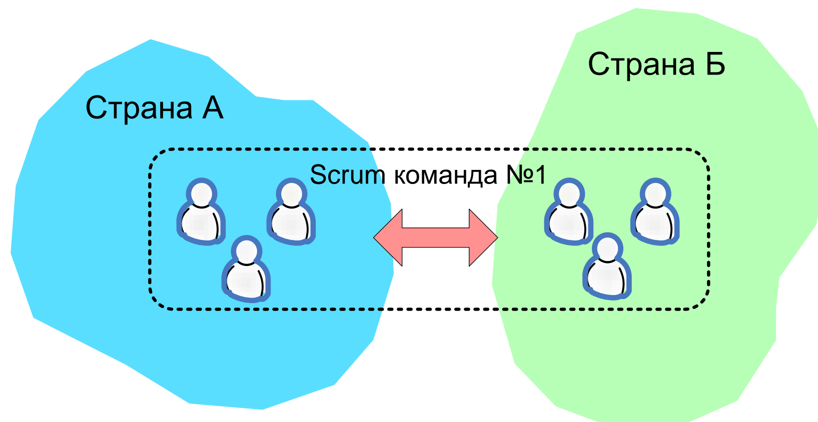
Как обычно, главное – не прекращать экспериментировать. Обсуждение => адаптация => обсуждение => адаптация => обсуждение => адаптация => обсуждение => адаптация => обсуждение => адаптация.

Оффшорная разработка

У нас есть несколько оффшорных команд. Мы экспериментировали с тем, как эффективно ими управлять, используя Scrum.

Существуют две основных стратегии: удаленные команды и удаленные участники команд.





Первая стратегия – удалённые команды – очевидный выбор. И всё-таки, мы начали с использования второй стратегии – удалённые участники команд. На то существует несколько причин.

1. Мы хотим, чтобы участники команд лучше узнали друг друга.
2. Мы хотим, чтобы была налажена эффективная коммуникация, и чтобы команды поняли её важность.
3. На начальной стадии оффшорная команда слишком маленькая, чтобы самоорганизоваться в эффективную scrum-команду.
4. Мы хотим, чтобы был период интенсивного обмена знаниями, прежде чем задействовать независимые оффшорные команды.

В долгосрочной перспективе мы вполне можем перейти к стратегии "удалённые команды".

Члены команды, работающие дома

Работа дома иногда может быть действительно эффективной. Порой, за один день дома можно сделать больше, чем за всю неделю на работе. По крайней мере, если у вас нет детей :o)

Однако, усадить команду вместе – одна из основополагающих идей Scrum'a. И что же делать в этом случае?

Чаще всего мы позволяем команде решать, как часто и когда именно её члены могут работать дома. Некоторые люди регулярно работают дома, так как живут далеко. И всё-таки, мы стараемся делать так, чтобы большую часть времени команда проводила вместе.

Когда члены команды работают дома, они участвуют в ежедневном Scrum'e, используя Skype-конференции (иногда видео конференции). Они доступны в чате в течение всего дня. Не так хорошо, как находиться в той же комнате, но этого достаточно.

Как-то мы опробовали идею выделения среды, как специального дня для работы на дому. По сути это означало: "Хочется поработать дома? Нет проблем, но только по средам. И согласуйте это с командой". Для

команды, на которой ставился эксперимент, это сработало отлично. По средам большая часть команды обычно оставалась дома и выполняла значительный объём работ, будучи при этом на связи друг с другом. Один такой день не сильно нарушал синхронизацию людей в команде. Но по каким-то причинам с другими командами этот подход не сработал.

В целом, люди, работающие дома, не стали для нас проблемой.

17

Памятка ScrumMaster'а

Напоследок я познакомлю вас с нашей памяткой ScrumMaster'а. Она содержит наиболее важные административные задачи, за которые отвечает ScrumMaster. Есть вещи, про которые очень легко забыть. Но есть и очевидные, такие как "устранять препятствия на пути команды", которые мы не включаем в наш список.

В начале спринта

- После планирования создать “страницу с информацией о спринте”.
  - На стартовой странице wiki-портала поместить ссылку на созданную страницу.
  - Распечатать эту страницу и повесить её на стене, которая у всех на глазах.
- Разослать e-mail'ы с уведомлением о начале нового спринта. Не забыть указать цель спринта и дать ссылку на "страницу с информацией о спринте".
- Обновить статистику спринтов. Добавить оценку предварительной производительности, размера команды, длины спринта и т.д.

Каждый день

- Следить за тем, чтобы ежедневный Scrum начинался и заканчивался вовремя.
- Следить за тем, чтобы в случае добавления или удаления истории из sprint backlog'а все было сделано, как положено, чтобы эти изменения не сорвали график работ.
  - Следить за тем, чтобы product owner знал про эти изменения.
- Следить за тем, чтобы команда постоянно обновляла burndown-диаграмму.
- Следить за тем, чтобы все проблемы решались. Как вариант можно проинформировать о них product owner'а и/или начальника отдела разработки.

В конце спринта

- Провести открытую демонстрацию результатов спринта.
- За несколько дней до демонстрации напомнить всем про её проведение.
- Провести ретроспективу при участии всей команды и product owner'а. Пригласить начальника отдела разработки, чтобы он помог найти оптимальное решение проблем.
- Обновить статистику спринтов. Внести значение реальной производительности и основные тезисы прошедшей ретроспективы.

18

Заключительное слово

Фух! Вот уж не ожидал, что это займёт столько времени.

Надеюсь, эта книга навеяла вам несколько полезных идей, будь вы новичками или уже бывалыми специалистами.

Поскольку Scrum всё равно необходимо подстраивать под каждую конкретную среду, спор по поводу общих практик заведомо будет неконструктивен. Однако мне всё же интересно получить от вас отзывы и предложения. Расскажите мне, чем ваши подходы отличаются от наших. Поделитесь идеями, как стать лучше!

Пишите мне на **henrik.kniberg@crisp.se**.

Кроме того, я заглядываю сюда: **scrumdevelopment@yahoogroups.com**.

Если вам понравилась эта книга, вы, вероятно, захотите иногда заглядывать на мой блог. Я надеюсь и впредь писать там заметки по Java и разработке софта:

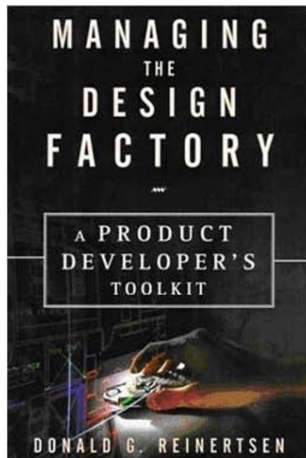
http://blog.crisp.se/henrikkniberg/

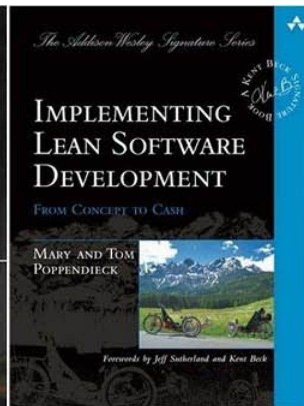
А ещё никогда не забывайте ...

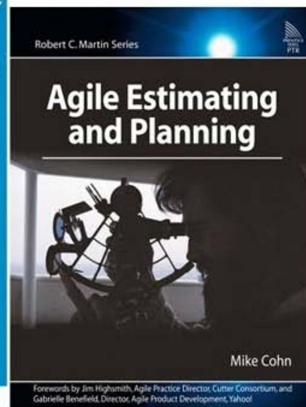
Это ведь просто работа, правда?

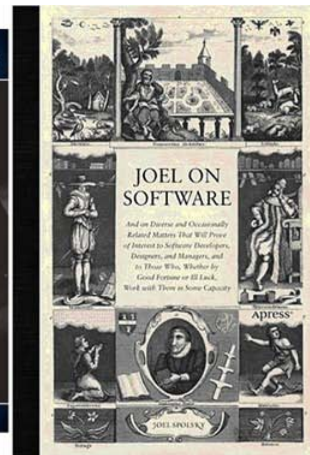
Список рекомендованной литературы

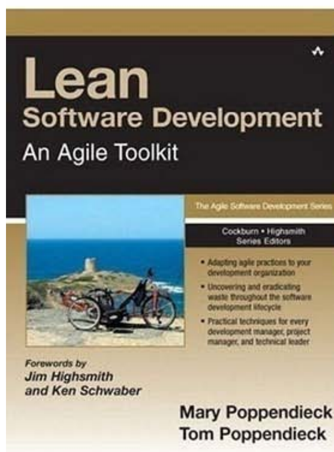
Ниже перечислены книги, которые стали для меня источниками идей и вдохновения. Настоятельно рекомендую!

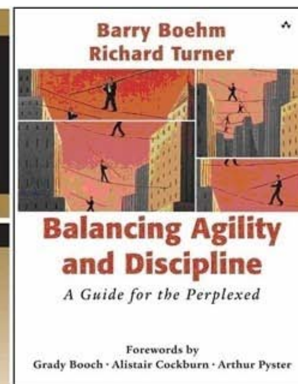


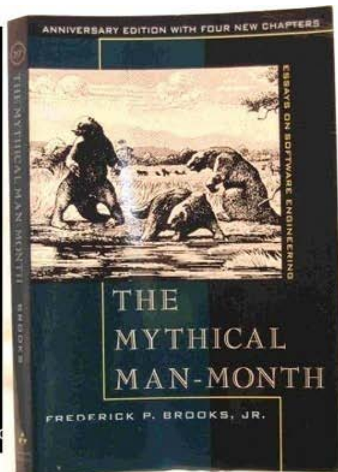


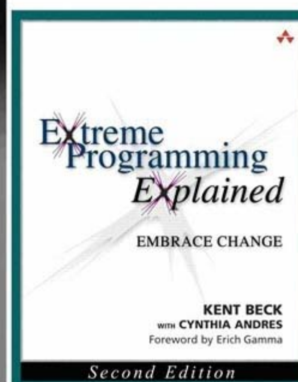


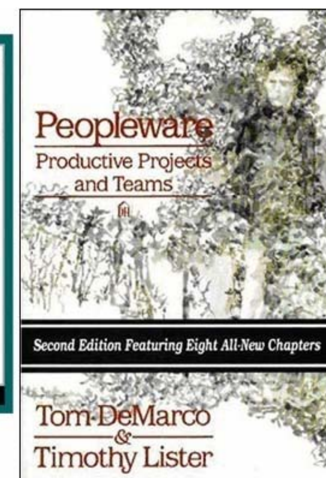












От переводчика: большая часть рекомендованных Хенриком книг не переведена на русский язык. Приведенные переводы достаточно точны, качественные и не меняют смысла написанного в оригинале.

- Donald G. Reinertsen "Managing the Design Factory"
- Mary Poppendieck, Tom Poppendieck "Implementing Lean Software Development"
- Mike Cohn "Agile Estimating and Planning"
- Джоэл Спольски "Джоэл о программировании"
- Mary Poppendieck, Tom Poppendieck "Lean Software Development. An Agile Toolkit"
- Barry Boehm, Richard Turner, "Balancing Agility and Discipline"

- Ken Schwaber, Mike Beedle, "Agile Software Development with Scrum"
- Фредерик Брукс "Мифический человеко-месяц"
- Ken Schwaber, "Agile Project Management with Scrum"
- Кент Бек "Экстремальное программирование"
- Том ДеМарко, Тимоти Листер "Человеческий фактор"

Об авторе

Хенрик Книберг (henrik.kniberg@crisp.se) – консультант компании Crisp в Стокгольме (www.crisp.se), специализируется на Java и Agile-разработке.

С тех пор как увидели свет agile-манифест и первые книги по XP, Хенрик начал следовать agile принципам и изучать, как наиболее эффективно их применять в организациях различного типа. Будучи соучредителем и генеральным директором компании Goyada в 1998-2003 гг. он получил прекрасную возможность поэксперементировать с TDD и другими agile-практиками, поскольку он начинал с нуля, управлял технической платформой и отделом в тридцать человек.

В конце 2005 года Хенрик по контракту стал главой отдела разработки в шведской игровой компании. Компания была в кризисе и нуждалась в срочном разрешении организационных и технических проблем. Используя инструменты Scrum и XP, Хенрик помог компании преодолеть кризис, внедрив принципы гибкой и бережливой (lean) разработки на всех уровнях.

Однажды в пятницу в ноябре 2006 Хенрик лежал дома с температурой и решил написать несколько коротких заметок о том, что ему удалось сделать за прошлый год. Однако, как только он начал писать, он уже не смог остановиться, и через три дня неистовой графомании первоначальные заметки выросли до восьмидесятистраничной статьи “Scrum and XP from the Trenches”, которая вскоре переросла в эту книгу.

Хенрик применяет целостный подход и с удовольствием выступает в роли менеджера, программиста, ScrumMaster'а, учителя или тренера. Помогая компаниям создать прекрасный софт и крепкую команду, он с энтузиазмом берётся за любую роль, которая в данной ситуации необходима.

Хенрик вырос в Токио, а сейчас живёт в Стокгольме со своей женой Софией и двумя детьми. Он увлечённо занимается музыкой в свободное время, сочиняет, играет на бас-гитаре и клавишных инструментах в местных группах. Чтобы узнать биографию более детально, посетите http://www.crisp.se/henrik.kniberg