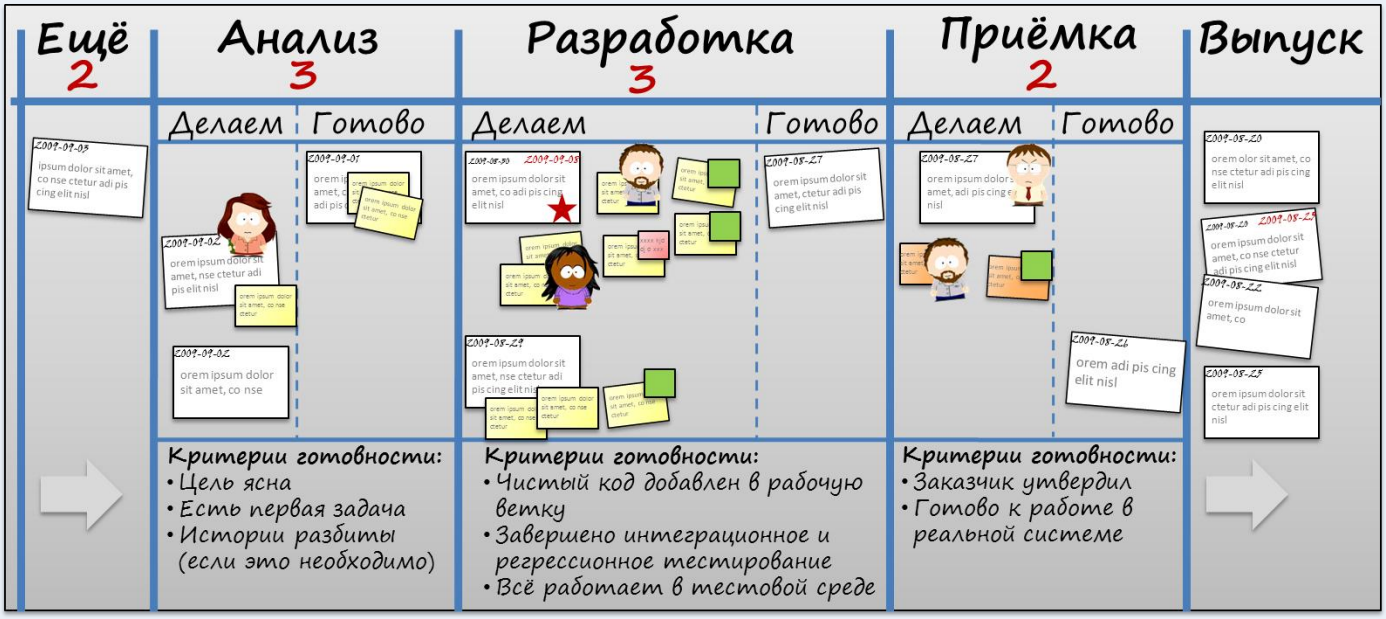


Scrum и Kanban: выжимаем максимум

Хенрик Книберг и Маттиас Скарин

Предисловие от Мери Поппендик и Дэвида Андерсона

ENTERPRISE SOFTWARE DEVELOPMENT SERIES

InfoQueue

Kanban и Scrum: выжимаем максимум

Хенрик Книберг и Маттиас Скарин
Предисловие от Мэри Поппендик и Дэвида Андерсона

Перевод выполнен сообществом Agile Ukraine: Алексей Солнцев, Мария Евграшина, Сергей Мовчан, Татьяна Кобзар, Артём Сердюк, Борис Лебеда, Роман Кононов, Тимофей Евграшин, Алексей Гончаренко, Алина Марусик, Александр Жовнуватый, Лина Шишкина, Ярослав Гнатюк, Андрей Бибичев

C4Media, Издательство InfoQ.com.

Эта книга является частью серии книг InfoQ Enterprise Software Development.

Для получения информации о покупке данной или других книг InfoQ, пишите по адресу books@c4media.com.

Главный редактор: Диана Плеса
Оформление обложки: Bistrian IOSIP
Вёрстка: Accurance
Перевод выполнен сообществом Agile Ukraine: Алексей Солнцев, Мария Евграшина, Сергей Мовчан, Татьяна Кобзар, Артём Сердюк, Борис Лебеда, Роман Кононов, Тимофей Евграшин, Алексей Гончаренко, Алина Марусик, Александр Жовнуватый, Лина Шишкина, Ярослав Гнатюк, Андрей Бибичев

Каталог изданий Библиотеки Конгресса США
ISBN: 978-0-557-13832-6

Оглавление

Предисловие Мэри Поппендик

Хенрик Книберг – один из тех немногих людей, которые умеют разобраться в сложной ситуации, отыскать значимые идеи в груде мелочей и предоставить настолько доходчивое объяснение, что все становится ясно, как дважды два. В этой книге, Хенрик блестяще объясняет разницу между Scrum и Kanban. Он дает понять, что это всего лишь пара инструментов из того полноценного инструментария, который вам в действительности необходим наряду с пониманием сильных и слабых сторон каждого инструмента и умением воспользоваться ими в нужный момент.

В этой книге вы узнаете, что же такое Kanban, каковы его сильные и слабые стороны, и когда его применять. Вы также получите хорошее представление о том, как и когда, следует дополнять Scrum или любой другой инструмент, который вам посчастливится использовать. Хенрик показывает, что важен не начальный выбор инструмента, а постоянное совершенствование навыков его применения и расширение своего инструментария со временем.

Во второй части этой книги Маттиас Скарин предлагает ознакомиться с применением Scrum и Kanban в реальной жизненной ситуации, что делает книгу еще более полезной. Там вы увидите пример использования описываемых инструментов как порознь, так и в сочетании друг с другом, для улучшения процесса разработки программного обеспечения. Вы заметите, что ни один подход не может претендовать на звание "лучшего"; в каждой конкретной ситуации вам необходимо самостоятельно искать решение – только так вы сможете наметить очередной шаг в сторону улучшения разработки ПО.

Мэри Поппендик

Предисловие Дэвида Андерсона

В основе Kanban-а лежит очень простая идея. Количество незавершённой работы (НЗР) должно быть ограничено, и что-либо новое может начинаться только тогда, когда какой-то существующий кусок работы поставляется или – в терминах lean – вытягивается следующим элементом конвейера. Kanban (или сигнальная карточка) подразумевает, что производится некое визуальное оповещение о том, что можно "вытягивать" новую работу, так как текущий объём работ не равен принятым лимитам. На революцию это не тянет, также не похоже, что это может существенно повлиять на производительность, культуру, квалификацию и степень зрелости команды в частности и организации в целом. Тем более удивительно, что … таки влияет! Хоть и кажется, что Kanban – это всего лишь небольшое изменение, оно, тем не менее, способно в корне изменить весь ваш бизнес.

Мы пришли к пониманию того, что Kanban – это подход к управлению изменениями. Это не методология разработки программного обеспечения, не жизненный цикл или процесс управления проектом. Kanban – это подход, который позволит привнести изменения в существующий жизненный цикл разработки программного обеспечения или методологию управления проектом. Основное в Kanban-е это то, что вы начинаете с того, что сейчас имеете. Визуализация потока создания ценностей помогает понять текущий процесс, после чего вы устанавливаете ограничения незавершённой работы (НЗР) на каждом этапе этого процесса. А затем вы регулируете поток работ, вытягивая (pull) новую работу, только по соответствующему kanban-сигналу.

Kanban уже доказал свою полезность для команд, которые используют Agile-подходы к разработке программного обеспечения, но он также набирает популярность и для команд, которые ориентируются на более традиционные подходы. Kanban заявил о себе как часть инициативы Lean по трансформации организационной культуры и поощрению процесса постоянного улучшения.

Поскольку НЗР в Kanban-е ограничена, то любой заблокированный элемент может потенциально застопорить систему. Если же достаточно большое число элементов рабочего процесса будет заблокировано, то и весь процесс точно застопорится. Что, в свою очередь, заставит не только саму команду, но и организацию в целом сфокусироваться на решении возникшей проблемы, разблокировать эти элементы и восстановить поток.

Kanban использует механизм визуального контроля для отслеживания того, как работа проходит через различные этапы потока создания ценностей. Как правило, используется или доска со стикерами, или её электронный вариант. В идеале, наверное, лучше иметь и то и другое. Вытекающая из этого прозрачность процесса также вносит свой вклад в изменение организационной культуры. Agile методы хороши для обеспечения прозрачности состояния незавершённой и выполненной работы, а также для генерации всевозможных метрик, таких, например, как производительность (количество работы завершённой за одну итерацию.) Однако, Kanban делает ещё шаг вперёд и добавляет прозрачности не только самому процессу, но и потоку создания ценностей. Kanban показывает узкие места, очереди, отклонения и потери процесса. В общем, всё то, что влияет на эффективность работы организации в разрезе количества полезной работы, а также времени, требуемого для того, чтобы эту работу выполнить. Kanban наглядно демонстрирует членам команды и другим заинтересованным сторонам результаты их действий (или бездействия). Таким образом, ранние тематические исследования показали, что Kanban изменяет поведение организации и способствует более активному сотрудничеству на рабочих местах. Кроме того, визуализация узких мест, потерь, отклонений и их влияния способствует обсуждению улучшений, благодаря чему команды очень быстро начинают внедрять улучшения своих процессов.

Таким образом, Kanban побуждает к постепенному развитию существующих процессов, к эволюции, которая, в общем, соотносится с ценностями Agile и Lean. Kanban не требует от вас немедленных революционных изменений способа работы, напротив – он побуждает вас к постепенным изменениям.

К таким изменениям, которые не только понятны всем, но и по которым достигнут консенсус между всеми работниками.

Kanban, как система вытягивания (pull), способствует тому, чтобы действия как по приоритезации новой работы, так и по поставке текущей работы, выполнялись как можно позже (в соответствии с принципом "точно во время (just in time)"). Обычно команды согласовывают некий распорядок встреч по приоритезации, на которых они с заинтересованной стороной, источником задач, решают, над чем работать дальше. Эти встречи могут проводиться часто, потому что они, как правило, очень короткие. Они должны дать ответ на очень простой вопрос – что-то вроде такого: "С последней встречи освободилось два слота. На сейчас оперативное время составляет 6 недель до поставки. Выберите 2 задачи, которые вы больше всего хотели бы увидеть реализованными через 6 недель?" Эффект получается двойной. Простой вопрос, как правило, позволяет быстро получить качественный ответ, и встречи не затягиваются. А характер вопроса приводит к тому, что активности по выбору следующей работы откладываются до последнего момента, когда ещё не поздно принять решение. Это повышает нашу гибкость за счет управления ожиданиями, сокращения оперативного времени (от принятия задачи в работу до поставки), и отсутствия переделок, так как минимизируется вероятность того, что приоритеты изменятся.

Последнее, что хочется добавить про Kanban – эффект ограничения незавершённой работы (НЗР) обеспечивает предсказуемость оперативного времени и делает сроки поставки более надежными. Метод "остановки конвейера" при обнаружении препятствий и дефектов, как, оказалось, способствует достижению очень высокого уровня качества и быстрому уменьшению количества переделок.

Хотя все это и станет очевидным из удивительно четких разъяснений в этой книге, но тема происхождения Kanban-а останется нераскрытой. Kanban не был придуман в один прекрасный день в результате невероятного прозрения. Наоборот – он развился в течении нескольких лет. Многие глубокие психологические и социальные эффекты, которые влияют на изменения культуры, способностей, зрелости или организации, никто себе и представить не мог. Но они были обнаружены в ходе экспериментов. Многие результаты, полученные с применением Kanban, противоречат интуиции. То, что представляется весьма механическим подходом – ограничивать НЗР и вытягивать задачи – в реальности оказывает глубокое влияние на людей и на то, как они взаимодействуют и сотрудничают друг с другом. Ни я, ни кто-либо из первых Kanban-практиков, этого не ожидали.

Я применял то, что впоследствии стало Канбан-ом, как подход к изменениям, который вызывал бы наименьшее сопротивление. Это для меня было ясно еще в 2003 году. Я также применял этот подход из-за его чисто механических преимуществ. В то время, применяя методы Lean, я обнаружил, что если управление НЗР имело смысл, то его ограничение имело ещё больше смысла. Это позволяло убрать из управления лишнее. Поэтому в 2004 году я решил попробовать реализовать систему работы, основанную на вытягивании. Я получил такую возможность, когда один из менеджеров в Microsoft подошел ко мне и попросил помочь в управлении изменениями, которые он хотел внедрить в команде по обслуживанию и модернизации внутренних приложений. Первая реализация была основана на вытягивающей системе из Теории Ограничений, известной как Барабан-Буфер-Верёвка. Успех был ошеломляющий: операционное время уменьшилось на 92%; пропускная способность увеличилась более чем в 3 раза, а предсказуемость сроков исполнения держалась на весьма приемлемой отметке в 98%.

В 2005 году Дональд Рейнертсен убедил меня реализовать полнофункциональную Kanban-систему. А возможность появилась в 2006 году, когда я возглавил департамент разработки программного обеспечения Corbis в Сиэтле. В 2007 году я начал выступать с докладами о результатах. Первая презентация прошла на Lean New Product Development Summit в Чикаго в мае 2007 года. Затем последовал доклад-обсуждение на Agile 2007 в Вашингтоне в августе этого года. Присутствовали 25 человек. Трое из них были из Yahoo! – Аарон Сандерс, Карл Скотланд и Джо Арнольд. Они вернулись домой в Калифорнию, Индию и Великобританию и внедрили Kanban в командах, которые уже

мучились со Scrum-ом. Они также создали Yahoo!-группу, которая на момент написания включает почти 800 членов. Kanban только начинал своё распространение, и первопроходцы делились своим опытом.

Теперь, в 2009 году, Kanban действительно становится популярным, и появляется всё больше и больше докладов о внедрении. За последние 5 лет мы узнали о Kanban-е много нового, и с каждым днем мы узнаём всё больше. Я сфокусировался на том, чтобы практиковать Kanban, писать о Kanban-е, рассказывать о Kanban-е и думать о Kanban-е с тем, чтобы лучше осмыслить и объяснить его другим. Я сознательно отступил от сравнения Kanban-а с другими Agile-методами, хотя всё-таки кое-что сделал в 2008 для объяснения того, почему Kanban заслужил право считаться Agile-совместимым подходом.

Отвечать на вопросы типа "Как Kanban соотносится со Scrum-ом?" я оставил тем, кто имеет более широкий опыт. Я так рад, что в этой области Хенрик Книберг и Маттиас Скарин проявились в качестве лидеров. Вам, работникам-практикам умственного труда, необходима информация, чтобы принимать обоснованные решения и двигаться вперед. Хенрик и Маттиас удовлетворят ваши потребности таким образом, каким я бы никогда не смог. Меня особенно впечатлили вдумчивый подход Хенрика к сравнению методов и его фактическая, незаангажированная и сбалансированная подача материала. Особенно интересны его комиксы и иллюстрации – они часто заменяют чтение многих страниц текста. Реальный опыт Маттиаса является важным, поскольку он демонстрирует, что Kanban – это гораздо больше, чем теория, и он показывает вам пример того, как Kanban может быть полезным для вас и вашей организации.

Я надеюсь, что вам понравится эта книга, сравнивающая Kanban и Scrum, и что она даст вам более глубокое понимание Agile в целом и Kanban-а со Scrum-ом в частности. Если вдруг вы захотите узнать больше о Kanban-е, пожалуйста, посетите наш веб-сайт Общества Ограниченной НЗР, http://www.limitedwipsociety.org/

Дэвид Дж. Андерсон

Скуим, Вашингтон, США
8 июля 2009

Вступление

Обычно мы не пишем книги. Мы предпочитаем быть на передовой, помогая клиентам оптимизировать, налаживать процессы и реорганизовывать компанию. Но в последнее время мы обнаружили определённые закономерности, и потому, хотели бы поделиться с вами кое-какими мыслями по этому поводу. Вот вам типичный случай:

- **Джим:** "Наконец-то у нас в проекте полноценный Scrum!"
- **Фред:** "Ну и как?"
- **Джим:** "Я бы сказал, гораздо лучше, чем раньше, но ..."
- **Фред:** "... Но что?"
- **Джим:** "... но ты же знаешь, мы занимаемся поддержкой".
- **Фред:** "Да, и что из этого?"
- **Джим:** "Нам в целом нравится расставлять приоритеты в Product Backlog-е, самоорганизовываться, проводить ежедневные встречи, ретроспективы и т.д."
- **Фред:** "Так в чем же проблема?"
- **Джим:** "Мы продолжаем заваливать спринты"
- **Фред:** "Почему?"
- **Джим:** "Нам тяжело запланировать свою работу на две недели вперед. Итерации для нас особого смысла не имеют, мы просто работаем над тем, что наиболее актуально в данный момент. Может нам укоротить итерацию до одной недели?"
- **Фред:** "А вы сможете запланировать работу на неделю? И хотя бы на неделю спокойно сосредоточиться на исключительно запланированной заранее работе?"
- **Джим:** "Не думаю – новые проблемы возникают КАЖДЫЙ день. Возможно, нам нужен спринт длиной в день ..."
- **Фред:** "Все ли задачи у вас решаются меньше, чем за один день?"
- **Джим:** "Нет, иногда они занимают несколько дней"
- **Фред:** "Тогда и однодневный спринт работать не будет. А вы думали вообще отказаться от спринтов?"
- **Джим:** "Ну, честно говоря, мы бы с удовольствием. Но разве это не против Scrum-а?"
- **Фред:** "Scrum – это всего лишь инструмент. Именно вы выбираете, когда и как его использовать. Не становитесь его заложником!"
- **Джим:** "Так что же нам делать?"
- **Фред:** "А вы слышали о Kanban-е?"
- **Джим:** "Нет. А что это? В чем разница между ним и Scrum-ом?"

- **Фред:** "Держи, прочитай вот эту книгу!
- **Джим:** "Но Scrum мне в целом нравится, почему я должен от него отказаться?"
- **Фред:** "Не надо, можно сочетать оба метода!"
- **Джим:** "Что!? Как?"
- **Фред:** "Просто прочитай ..."

Цель этой книги

Если вы интересуетесь методологиями гибкой разработки программного обеспечения, вы не могли не слышать о Scrum-е, и наверняка кое-что уже слышали о Kanban-е. Мы всё чаще слышим вопрос: "Что такое Kanban, и чем он отличается от Scrum-а?" В чем они дополняют друг друга? А в чем противоречат?

Цель этой книги – развеять непонимание и прояснить, чем именно Kanban и Scrum могут быть полезны в вашем проекте.

Дайте нам знать, получилось ли у нас!

Часть 1 – Сравнение

Первая часть книги – попытка объективно и прагматично сравнить между собой Scrum и Kanban. Это слегка обновленная версия моей статьи "Kanban против Scrum" (Kanban vs Scrum), написанной в апреле 2009-го. Эта статья оказалась настолько популярной, что я решил сделать из неё книгу и попросил моего коллегу Маттиаса приправить её рассказом о проекте одного нашего клиента – так сказать, "с передовой". Приправа удалась на славу! Если вы хотите начать сразу с истории – пропускайте эту часть и начинайте со второй части, я не обижусь. Ну, разве что чуть-чуть.

Хенрик Книберг

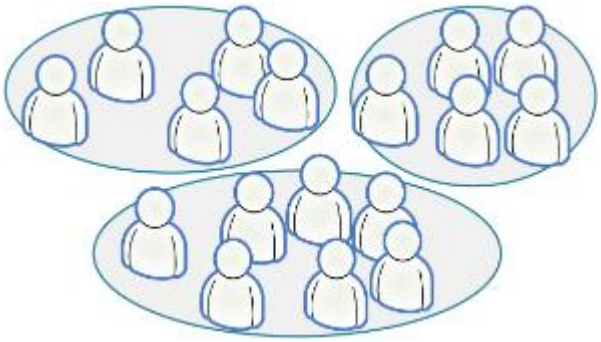
1

Так что же такое Scrum и Kanban?

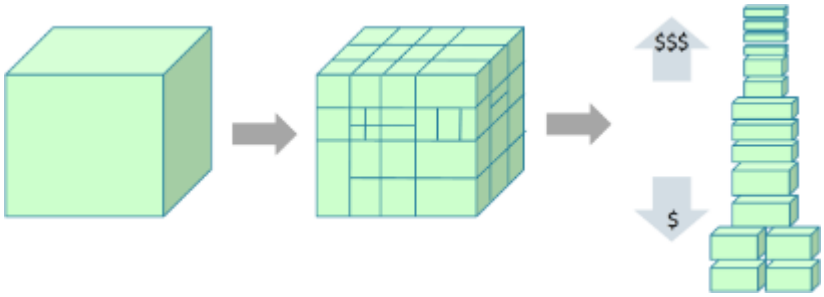
Хорошо, давайте попробуем описать Scrum и Kanban так, чтобы вложиться в сотню слов.

Коротко о Scrum-е

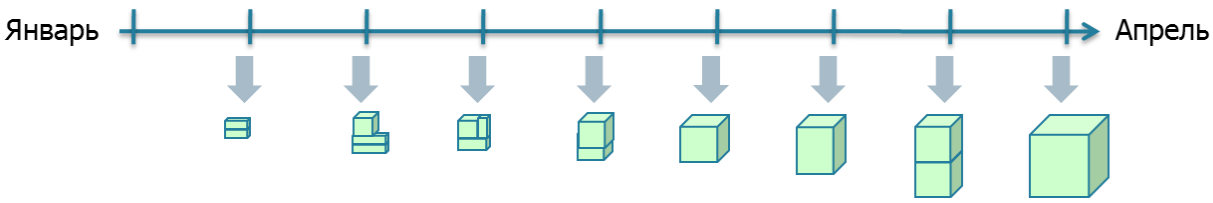
- **Разделите Вашу организацию** на маленькие, кросс-функциональные, самоорганизующиеся команды.



- **Разделите Вашу работу** на маленькие, конкретные компоненты. Отсортируйте этот список по приоритетам и оцените относительный объём работы по каждому элементу.



- **Разделите время** на короткие итерации фиксированной длины (обычно 1-4 недели) так, чтобы после каждой итерации проводилась демонстрация потенциально готового к использованию кода.



- **Оптимизируйте план релиза** и корректируйте приоритеты совместно с клиентом, основываясь на данных, получаемых при рассмотрении релиза после каждой итерации.

- **Оптимизируйте процесс** с помощью проведения ретроспективы после каждой итерации.

Итак, вместо **большой команды**, которая **долго** работает над **чем-то большим**, у нас получается **небольшая команда**, которая **короткими итерациями** работает над **небольшими кусочками**. Но с **регулярной интеграцией**, чтоб видеть целостность картины.

107 слов … почти уложились.

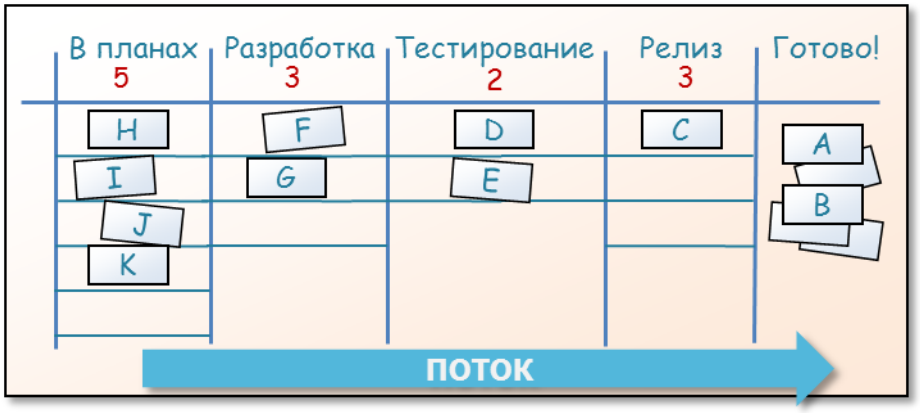
Более подробную информацию можно найти в книге "Scrum и XP: заметки с передовой" – она есть в свободном доступе в сети. Я знаком с автором, он отличный парень :o)

http://www.infoq.com/minibooks/kanban-scrum-minibook

Больше ссылок о Scrum-е на http://www.crisp.se/scrum

Коротко о Kanban-е

- **Визуализируйте поток работ**
    - Разбейте работу на части, выпишите каждый из пунктов на карточку и прикрепите на стену.
    - Подпишите столбцы, чтобы видеть на какой стадии находится каждое задание.
- **Ограничьте НЗР (WIP)** (прим. переводчиков work-in-progress – незавершённая работа) – определите возможное количество незавершённых пунктов на каждой стадии рабочего процесса.
- **Измеряйте время выполнения задачи (lead time)** (среднюю продолжительность времени для завершения одного пункта, иногда называемую "оперативным временем" (cycle time)), оптимизируйте процесс, чтобы свести время выполнения задачи к минимуму и сделать его настолько прогнозируемым, насколько это возможно.



Мы собираем полезные ссылки о Kanban-е: http://www.crisp.se/kanban

2

Так как же Scrum и Kanban связаны друг с другом?

Scrum и Kanban – инструменты процесса

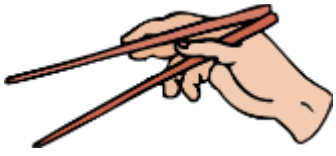
Инструмент – это то, что используется в качестве средства для выполнения задачи или достижения цели. *Процесс* – это то, как вы работаете.

Scrum и Kanban – это *инструменты процесса*, они помогают вам работать более эффективно, до определенной степени подсказывая, что делать. Java – тоже инструмент, он упрощает программирование компьютера. Зубная щетка – инструмент, который помогает вам добраться до зубов, чтобы почистить их.

Сравнивай, чтобы понять, а не осудить

Нож или вилка, – какой из инструментов лучше?

Довольно бессмысленный вопрос, не так ли? Ведь ответ зависит от контекста. Если мы хотим съесть котлетку, лучше воспользоваться вилкой. Для измельчения грибов подойдёт нож. А чем барабанить по столу – всё равно. Чтобы расправиться с отбивной вы, скорее всего, воспользуетесь и тем, и другим одновременно. Для употребления в пищу риса ... ну ... кто-то берёт вилку, а кому-то больше по душе палочки.



Поэтому сравнивать инструменты надо осторожно. Сравнивайте, чтобы понять, а не осудить.

Не бывает универсальных инструментов, не бывает и идеальных

Как и любые инструменты, Scrum и Kanban не совершенны и не универсальны. Они не расскажут вам *все*. Вместо этого они дают определенные правила, устанавливают ограничения и предписания. Например, Scrum предписывает фиксированные по времени итерации и кросс-функциональные команды, а Kanban – использование визуализации процесса на доске задач и ограничение НЗР.

Достаточно любопытно, что ценность инструмента в том, что он *ограничивает ваш выбор*. Инструмент процесса, который позволяет вам делать что угодно, не очень полезен. Мы могли бы назвать такой процесс "Делайте что хочешь", а как насчет "Делайте все правильно"? Процесс "Делайте все правильно" гарантированно будет работать, это серебряная пуля! Ведь если он не работает, то вы явно не следовали процессу :)

Правильные инструменты помогут вам достичь успеха, но не обязательно его обеспечат. Легко спутать успех или провал *проекта* с успешным или неудачным *использованием инструмента*.

- Проект может быть успешным из-за очень хорошего инструмента.
- Проект может быть успешным, несмотря на паршивый инструмент.
- Проект может провалиться из-за паршивого инструмента.
- Проект может провалиться, несмотря на очень хороший инструмент.

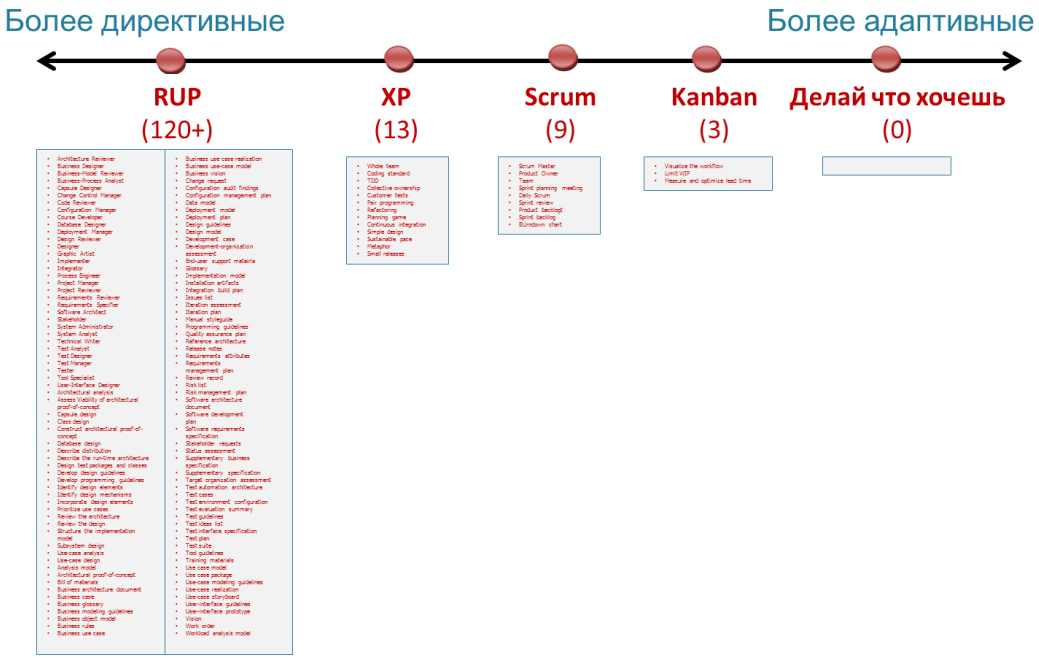
Scrum более директивный, чем Kanban

Можно сравнить инструменты по количеству предписываемых правил, которое в них содержится. *Директивный* означает "содержащий больше правил, которым надо следовать", а *адаптивный* – "содержащий меньше правил". При 100%-но директивном подходе вам не придется напрягать свой мозг, есть правила для всего. А при 100%-й адаптивности – "Делай что хочешь", тут не существует никаких правил и ограничений. Как видите, обе крайности выглядят весьма нелепо.

Agile-методы иногда называют *легковесными*, в частности, потому что они менее директивны, чем традиционные методы. В самом деле, первый принцип Agile-Манифеста гласит "Люди и взаимодействие важнее процессов и инструментов".

Scrum и Kanban являются высоко адаптивными, но, *если их сравнить*, то Scrum более директивен, нежели Kanban. Scrum предоставляет вам больше ограничений, и, таким образом, оставляет меньше вариантов для выбора. Например, Scrum предписывает использование фиксированных по времени итераций, Kanban – нет.

Давайте сравним несколько инструментов процесса на шкале "предписание/адаптация":



RUP – очень даже директивный метод, – насчитывает более 30 ролей, более 20 мероприятий и более 70 артефактов; просто куча всего для изучения. Вы конечно не должны использовать все, предполагается выбор какого-то подмножества для конкретного проекта. К сожалению, на практике это не просто. "Хм … потребуются ли нам *результаты проверки инфраструктуры*? Потребуется ли нам роль *Менеджера по управлению изменениями*? А черт его знает! Давайте просто оставим на всякий случай." Это может быть одной из причин, почему RUP-реализации часто оказываются довольно тяжеловесными по сравнению с Agile-методами, такими как Scrum и XP.

XP (eXtreme Programming) – намного более директивен по сравнению со Scrum-ом. Эта методология включает в себя большую часть Scrum-а + кучу весьма конкретных инженерных практик, таких как TDD (разработка через тестирование) и парное программирование.

Scrum менее директивный, чем XP, так как он не навязывает использование никакой конкретной инженерной практики. Scrum более директивный, чем Kanban, поскольку он предусматривает такие вещи, как итерации и кросс-функциональные команды.

Одно из основных различий между Scrum и RUP в том, что в RUP вы получаете сразу много всего, но надо избавиться от лишнего. А в Scrum – слишком мало, и надо добавить недостающее.

Kanban оставляет почти все на ваше усмотрение. Единственные ограничения – визуализировать ваш ход работы и ограничить незавершенную работу (НЗР). Всего в нескольких сантиметрах от "Делайте что хочешь", но все равно на удивление мощно.

Не ограничивайте себя одним инструментом!

Подбирайте и сочетайте инструменты так, как вам удобно! Например, я с трудом могу себе представить успешную Scrum-команду, которая не использует большинство практик XP. Многие Kanban-команды используют ежедневные пятиминутки, а это Scrum-практика. Некоторые Scrum-команды описывают задачи из backlog-а в виде Use Case – сценариев использования (практика RUP), или ограничивают размер очередей задач (практика Kanban). Что угодно, лишь бы работало!

Об этом красиво сказал Миямото Мусаси (легендарный самурай 17-го века, прославившийся техникой фехтования двумя мечами)

Вам не следует иметь любимого оружия. Владеть каким-то одним оружием очень хорошо, тогда как другими вы владеете плохо — такой же недостаток, как и не владеть этим оружием вообще.

- Миямото Мусаси

Внимательно относитесь к ограничениям каждого из инструментов. Например, если вы используете Scrum и решили отказаться от использования ограниченных во времени итераций (или любой другой основной практики Scrum), то не говорите, что вы используете Scrum. Scrum сам по себе и так достаточно минимизирован, если вы удалите что-то и по-прежнему будете называть это Scrum, то слово станет бессмысленным и непонятным. Назовите это как-нибудь вроде "по мотивам Scrum-а" или "подмножество Scrum", или вообще "Scrum-ное" ☺

3

Роли, определённые в Scrum-е

Scrum предписывает наличие 3-х ролей: Product Owner (отвечает за видение продукта и приоритеты), Команда (отвечает за реализацию продукта) и Scrum Master (устраняет препятствия в работе и руководит Scrum-процессом).

Kanban же не предписывает вообще никаких ролей.

Но это, однако, не значит, что вам нельзя или не стоит иметь роль Product Owner-а при применении Kanban-а! Это просто означает, что *эта роль необязательна*. Применяя как Scrum, так и Kanban, вы вольны добавлять какие угодно дополнительные роли.

Однако при добавлении ролей следует сохранять осторожность – убедитесь, что дополнительные роли действительно добавляют некую ценность и не конфликтуют с другими элементами выбранного процесса. Уверены, что вам сейчас нужна эта роль менеджера продукта? Для большого проекта это может быть замечательной идеей: возможно, на эту роль подойдёт парень, который помогает нескольким командам и Product Owner-ам синхронизироваться друг с другом. Для маленьких же проектов эта роль может оказаться "потерей (waste)" или, того хуже, привести к локальной оптимизации и микроменеджменту.

Основным посылом как в Scrum-е, так и в Kanban-е является "Чем меньше, тем лучше". Поэтому в случае сомнений начинайте с меньшего.

Далее я собираюсь использовать термин "Product Owner", имея в виду любого, кто устанавливает приоритеты для команды, независимо от применяемого процесса.

4

Scrum предписывает итерации, ограниченные по времени

Scrum основан на ограниченных по времени итерациях. Длину итерации вы можете выбрать любую, однако главная идея состоит в том, чтобы сохранить эту длину неизменной на протяжении определённого периода времени, тем самым задав *ритм (cadence)*.

- **Начало итерации:** создается план итерации, т.е. команда выбирает определенное количество историй из Product Backlog-а, исходя из приоритетов Product Owner-а и количества работы, которое команда надеется выполнить за спринт.

- **В ходе итерации:** команда фокусируется на выполнении задач, которые она обязалась закончить. Объем работы на итерацию фиксирован.

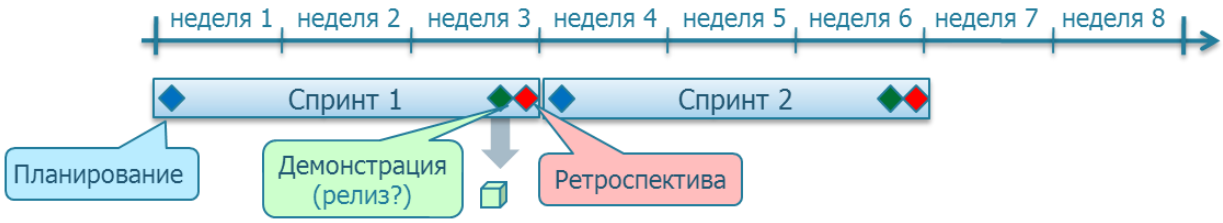
- **Конец итерации:** команда демонстрирует работающий код заинтересованным сторонам, в идеале этот код должен быть *готовым к релизу* (т.е. протестированным и готовым к использованию). После этого команда проводит ретроспективу, чтобы обсудить и улучшить свой процесс.

Таким образом, итерации в Scrum задают единый ритм, объединяющий три различных вида деятельности: планирование, улучшение процесса, и (в идеале) релиз.

В Kanban-е ограниченные по времени итерации не обязательны. Вы можете сами выбирать время, когда заниматься планированием, улучшать процесс и делать релиз. Вы можете выполнять эти действия на систематически ("релиз каждый понедельник") или по требованию ("выпускаем релиз, как только у нас будет что-то полезное, что можно выпустить").

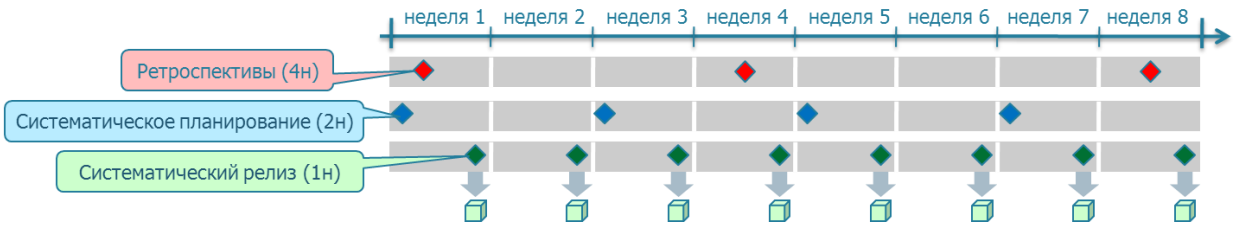
Команда №1 (один ритм)

"Мы работаем спринтами" (как в Scrum-е).



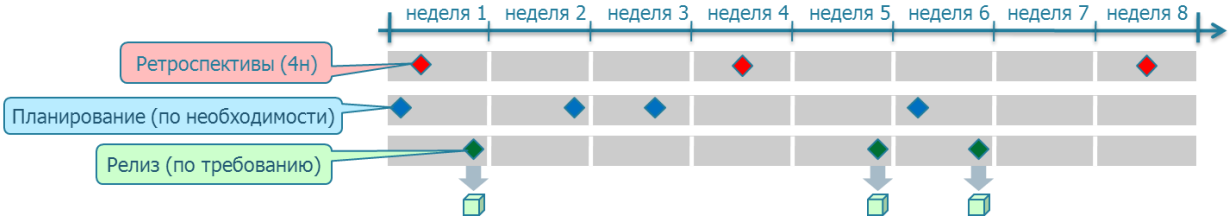
Команда №2 (три ритма)

"У нас есть три разных ритма. Каждую неделю мы выпускаем то, что готово к выпуску на этот момент. Каждую вторую неделю мы проводим планирование, обновляем приоритеты и план релиза. Каждую четвертую неделю мы проводим ретроспективу, чтобы улучшить наш процесс".



Команда №3 (управляемая событиями)

"Мы проводим планирование, когда у нас заканчивается работа. Мы делаем релиз, когда есть минимальная коммерчески ценная функциональность (minimum marketable feature set – MMF), готовая к выпуску. Мы проводим спонтанное обсуждение качества, когда сталкиваемся с одной и той же проблемой во второй раз. Мы также проводим более развернутые ретроспективы каждую четвертую неделю ".

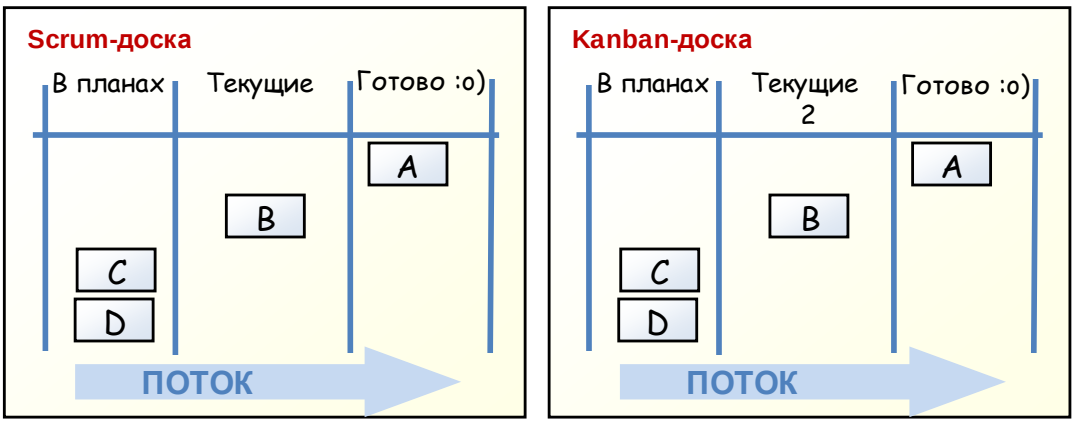


5

Kanban ограничивает НЗР по статусу задачи, Scrum – по итерациям

В Scrum-е backlog спринта отображает список задач, которые должны быть выполнены в ходе текущей итерации (т.е. в ходе "спринта" в терминах Scrum-а). Такие списки в основном представляют в виде карточек на стене, так называемой Scrum-доски или доски задач.

Так в чем же разница между досками в Scrum-е и Kanban-е? Давайте сравним эти два подхода на примере простого проекта:



В обоих случаях мы отслеживаем прогресс в ходе работы, используя список задач, представленных набором бумажек. Мы решили использовать три статуса: «В планах» (To Do), «Текущие» (Ongoing), «Готово» (Done). Вы можете использовать любые статусы, которые вам понравятся – некоторые команды вводят дополнительные статусы, такие как «Интеграция» (Integrate), «Тестирование» (Test), «Релиз» (Release) и т.д. Тем не менее, не забывайте придерживаться принципа *"Чем меньше, тем лучше"*.

Так в чем же тогда разница между этими двумя досками? Да – вот эта маленькая "2" в средней колонке на Kanban-доске. И всё. Эта цифра 2 означает что "в этой колонке может быть не более 2-х элементов одновременно".

Scrum не запрещает команде поставить все задачи в колонку «Текущие» одновременно! Тем не менее, некоторый предел все же существует, так как сама итерация имеет определенные ограничения по количеству задач. В этом случае максимальное количество задач в колонке равно 4 потому, что здесь всего четыре задачи на доске. Таким образом, Scrum ограничивает незавершённую работу косвенно, Kanban же делает это явно.

Большинство Scrum-команд, в конце концов, приходят к тому, что считают добавление большого количества задач в колонку " Текущие" плохой практикой, и вырабатывают в себе навык заканчивать одни задачи перед тем, как браться за другие. Некоторые из них даже решают явно ограничить количество задач в средней колонке, а затем – Опа! – Scrum-доска превращается в Kanban-доску!

Таким образом, и Scrum, и Kanban, ограничивают незавершенную работу, но по-разному. Scrum-команды обычно измеряют *производительность (velocity)* – сколько задач (или соответствующих единиц, таких как "story points") выполняется за одну итерацию. После того, как команда узнает свою производительность, это значение считается границей незавершенной работы (или, во всяком случае, значение, к которому стремятся). Команда, у которой средняя производительность 10, как правило, не назначает более десяти задач (или story points) в рамках одного спринта.

Таким образом, в Scrum-е *незавершенная работа ограничена за единицу времени.*

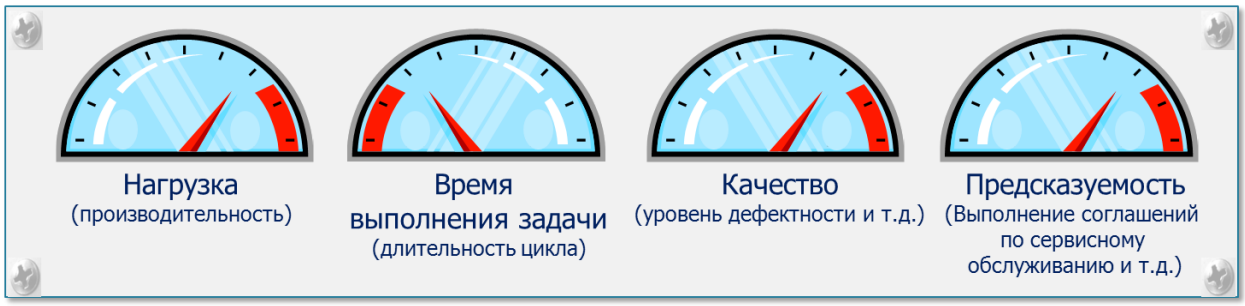
В Kanban-е *незавершённая работа ограничена по каждому из статусов.*

В приведенном выше примере Kanban, не более двух задач могут быть в состоянии «Текущие» одновременно, независимо от длины итерации (спринта). Вам надо определиться с ограничениями, которые будут применены к каждому из состояний рабочего процесса, то есть основная идея заключается в том, чтобы ограничить незавершенную работу для *всех* состояний рабочего процесса, начиная с как можно более ранних этапов потока создания ценностей (value stream), и заканчивая как можно более поздними его этапами. Таким образом, в приведенном выше примере мы должны подумать о добавлении ограничения в колонку «В планах» (или в любую другую колонку, с которой начинается незавершенная работа). После того как мы определились с ограничениями незавершенной работы, мы можем приступить к оценке и прогнозированию времени выполнения задачи, то есть среднего времени, необходимого для того, чтобы пройти все статусы на доске. Имея возможность предсказать время выполнения задачи, вы можете выполнять условия соглашений об уровне предоставления услуг (SLA) и правильно планировать релиз.

Если у вас есть очень разные по размерам задачи, вы можете установить ограничение незавершенной работы не в штуках, а в story point-ах, или в каких либо других единицах измерения, которые более всего вам подходят. Некоторые команды стараются разбивать задачи таким образом, чтобы они были примерно одинакового размера, чтобы избежать такого рода анализа и оценок (вы можете даже посчитать оценку мусором). Намного проще получить сбалансированный поток, если оперировать задачами примерно одинакового размера.

6

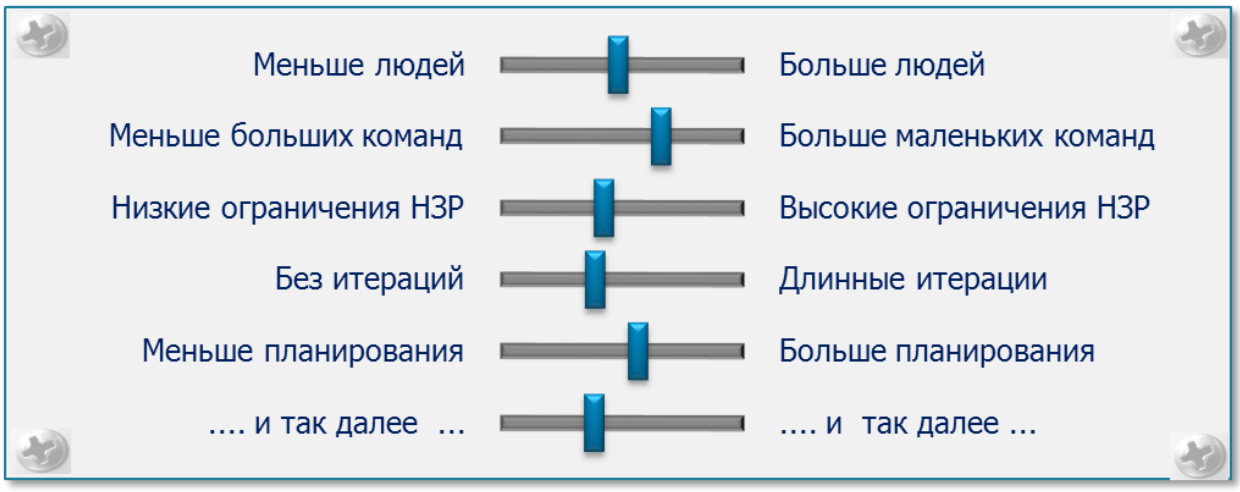
Оба процесса – эмпирические



Представьте, что к этим индикаторам были бы ещё приделаны и регуляторы, и вы бы могли настраивать процесс простым поворотом этих ручек управления. "Вот я хочу высокую производительность, малое время выполнения задачи, высокое качество и высокую предсказуемость. Ну, так я поверну ручки на 10, 1, 10 и 10 соответственно".

Ну, разве это не было бы замечательно? К сожалению, у нас нет инструментов, которые позволили бы осуществлять такое непосредственное управление. Ну, по крайней мере, я про такие не слышал. Если вы что-то такое найдёте, то сообщите мне, пожалуйста.

Вместо этого, у нас есть уйма элементов *косвенного* управления.



Scrum и Kanban являются эмпирическими в том смысле, что ожидается, что вы будете экспериментировать с процессом и заточите его под свои условия. На самом деле, вам даже *придётся* экспериментировать. Ни Scrum, ни Kanban не содержат ответов на все вопросы – они просто дают вам базовый набор ограничений, чтобы было от чего отталкиваться при улучшении вашего процесса.

- Scrum говорит, что ваши команды должны быть кросс-функциональными. А кого включать в какую команду? Не знаю – экспериментируйте.

- Scrum говорит, что именно команда определяет, какой объем работы взять на спринт. Так, а сколько же брать? Не знаю – экспериментируйте.

- Kanban говорит, что вы должны ограничивать НЗР. А какое же установить ограничение? Не знаю – экспериментируйте.

Как я уже упоминал ранее, Kanban накладывает меньше ограничений, чем Scrum. Это означает, что вам достаётся больше параметров для настройки и больше ручек, чтобы вертеть. Это может быть как недостатком, так и преимуществом, в зависимости от вашей ситуации. Что вам больше по душе при использовании окон настроек приложения – 3 параметра для настройки или 100? Скорее всего, что-то среднее. Предпочитаемое количество настроек зависит от того, насколько много необходимо донастроить и насколько хорошо вы знакомы с инструментом.

Допустим, мы уменьшаем ограничение НЗР, исходя из предположения, что это улучшит наш процесс. После этого мы наблюдаем, как изменяются такие параметры процесса как производительность, время выполнения задачи, качество и предсказуемость. Потом делаем выводы и меняем что-то ещё, таким образом, непрерывно улучшая наш процесс.

Для этой деятельности придумано много названий. Кайдзен (непрерывное улучшение, в терминах Lean-а), "изучай и улучшай" (в терминах Scrum-а), адаптивное управление процессом, или (а почему бы и нет?) научный метод.

Самое важное тут – *цепь обратной связи*. Измените что-нибудь => Посмотрите что получилось => Сделайте выводы => Измените что-нибудь снова. Вообще, чтобы адаптировать процесс быстро, цепь обратной связи должна быть как можно короче.

В Scrum-е основной цепью обратной связи является спринт. Однако есть и другие цепи, особенно, если вы сочетаете его с XP (экстремальным программированием):

Если все сделать правильно, то Scrum в сочетании с XP предоставит вам набор черезвычайно полезных цепей обратной связи.

Внутренняя цепь обратной связи – парное программирование, – обеспечивает обратную связь уже через несколько секунд. Дефекты обнаруживаются и исправляются в процессе создания кода ("Эй, а разве не эта переменная должна равняться трём?"). Это цепь типа "делаем ли мы дело правильно?".

Внешняя цепь обратной связи – спринт – обеспечивает обратную связь через несколько недель. Это цепь типа "делаем ли мы правильное дело?".

Так, а что насчет Kanban-а? Ну, прежде всего, можно (и, вероятно, нужно) интегрировать все упомянутые цепи обратной связи в ваш процесс независимо от того, используете ли вы Kanban или нет. А собственно Kanban обеспечивает вам несколько очень полезных метрик реального времени.

- Среднее время ожидания (lead time). Оно обновляется каждый раз, когда задача достигает статуса "Готово" (или как вы там называете свою самую правую колонку).

- Узкие места. Типичным признаком узкого места является то, что колонка X забита задачами, а в колонке X+1 пусто. Следите за "пробками" на вашей доске.

Прелесть метрик реального времени в том, что вы можете выбирать продолжительность цикла обратной связи, основываясь на том, как часто вы собираетесь анализировать метрики и вносить изменения в процесс. Слишком длинный цикл обратной связи означает, что ваш процесс будет улучшаться медленно. Слишком короткий цикл обратной связи означает, что вам может не хватить времени на стабилизацию процесса после внесения изменений, что может застопорить процесс.

Собственно, и с длиной цикла обратной связи тоже можно экспериментировать... вроде как рекурсия. Ну да, пора из неё выходить :)

Пример: Эксперименты с ограничениям НЗР в Kanban-е

Одной из стандартных "настроек" в Kanban-е является ограничение НЗР. Так как же нам понять, что мы установили это ограничение правильно?

Допустим, у нас команда из четырёх человек, и мы решили начать с ограничением НЗР равным "1".

Всякий раз, когда в работу попадает одна задача, мы не можем начинать новые задачи до тех пор, пока первая не будет завершена. Поэтому она будет завершена очень быстро.

Здорово! Но потом оказывается, что, как правило, нет смысла всем четырём разработчикам (как в данном примере) работать над одной и той же задачей, так что у нас появляются люди, сидящие без дела. Если такое случается только от случая к случаю – это не проблема, но если регулярно, то, как следствие, среднее время ожидания возрастёт. По сути, это означает, что при ограничении НЗР равном единице, задачи, получившие статус "Текущие", будут его просто пролетать, но при этом они будут

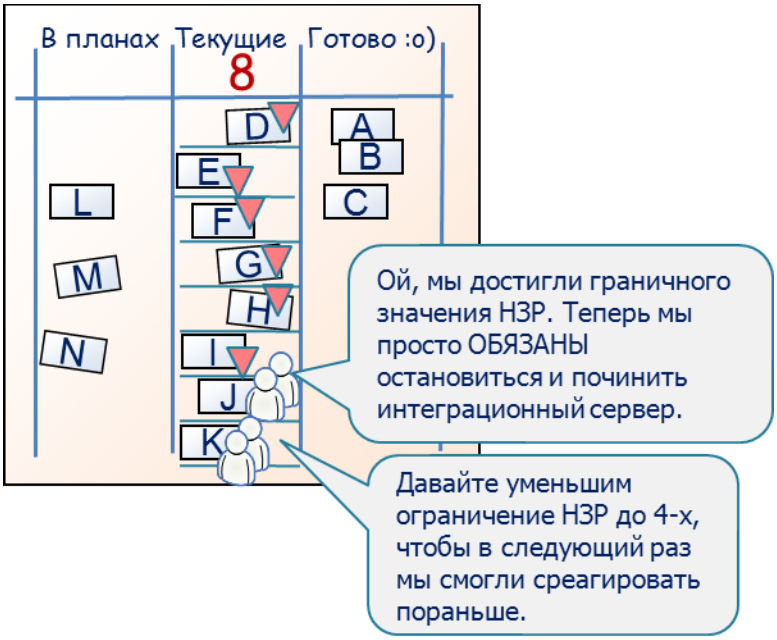
сидеть в колонке "В планах" дольше, чем нужно, и общее время ожидания будет неоправданно высоким.

Итак... Если ограничение НЗР равное единице слишком мало, то может увеличить его до 8?

Какое-то время будет лучше. В своё время мы поняли, что эффективнее всего работать в парах – так получается быстрее. Поэтому, имея команду в 4 человека, мы, как правило, работаем над двумя задачами одновременно. 8 – это верхнее ограничение, так что работать над меньшим количеством задач вполне допустимо!

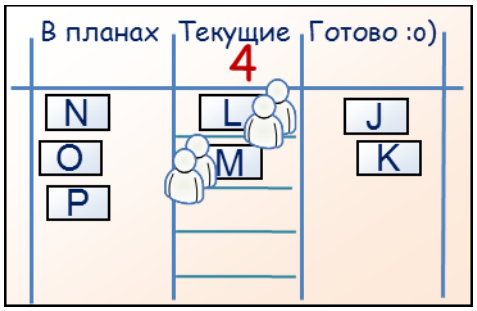
Но вот представьте теперь, что у нас возникла проблема с интеграционным сервером, и из-за этого мы не можем завершить ни одну задачу (наш критерий готовности включает в себя интеграцию). Такое ведь иногда случается, правда?

Поскольку задачи D и E мы завершить не можем, то мы начинаем работу над задачей F. Эту задачу мы также не можем проверить на сервере интеграции, так что мы вытягиваем новую задачу G. И через некоторое время мы упрёмся в наше ограничение – не более 8-ми задач в колонке "Текущие".



На этом этапе мы не можем вытягивать себе больше задач. Ага, ну значит, придётся починить этот чёртов сервер! Ограничение НЗР вынудило нас отреагировать и избавиться от узкого места, а не продолжать плодить кучи незавершёнки.

Неплохо. А вот если ограничение НЗР было бы равно четырём, то мы бы упёрлись в него намного раньше и, это обеспечило бы нам лучшее среднее время выполнения задачи. Так что это вопрос баланса. Мы постоянно измеряем среднее время выполнения задачи и подстраиваем ограничения НЗР, чтобы оптимизировать время выполнения задачи.



Через некоторое время мы, возможно, обнаружим, что задачи скапливаются в колонке "В планах". Что ж, может быть, тогда надо будет добавить ограничение НЗР и для этой колонки тоже.

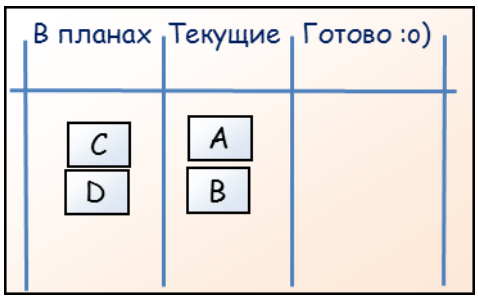
Кстати, а зачем вообще нам нужна эта колонка "В планах"? Ну, если бы клиент был всегда доступен, чтобы выдавать команде задачи, когда только они просят, то колонка "В планах" действительно была бы не нужна. Но в нашем случае клиент не всегда доступен, так что колонка "В планах" обеспечивает команде небольшой буфер, откуда она может "вытягивать" себе задачи.

Одним словом – экспериментируйте! Или, как говорят Scrum-ологи, измеряйте и улучшайте!

7

Scrum против изменений в ходе итерации

Предположим, что наша Scrum-доска выглядит так:



Что делать, если появляется некто, кто хочет добавить E на нашу доску?

Scrum-команда, как правило, говорит что-то вроде "Нет, извините, мы обещали выполнить только A + B + C + D в этом спринте. Но вы легко можете добавить E в Product Backlog. Если же Product Owner решит, что у этой задачи высокий приоритет, мы возьмем ее в следующий спринт". Спринт правильной длины дает команде возможность сфокусироваться и хоть что-то завершить, и в то же время позволяет Product Owner-у изменять и регулярно обновлять приоритеты.

А что сказала бы Kanban-команда?

Kanban может сказать "Да, пожалуйста, добавляйте E в колонку "В планах". Но в этой колонке может быть не больше двух элементов, так что в этом случае вам придется убрать C или D. Сейчас мы работаем над A и B, но как только у нас появится возможность, мы возьмем верхний элемент из колонки "В планах".

Таким образом, время реакции (время, которое необходимо, чтобы отреагировать на изменение приоритетов) Kanban-команды – это время до того момента, когда освободится место в соответствующей колонке, согласно принципу "один элемент ушел – один пришел" (ограничение незавершенного производства).

В Scrum-е время реакции в среднем составляет половину длины спринта.

В Scrum-е Product Owner не может трогать Scrum-доску, так как команда взяла на себя обязательства по конкретному набору элементов на итерацию. В Kanban-е вы должны установить собственные правила для тех, кому разрешено изменять то, что на доске. Как правило, Product Owner-у дается крайняя левая колонка вроде "В планах" или "Готово к работе (Ready)", или "Backlog", или "Предложение (Proposed)", где он может вносить какие угодно изменения.

Тем не менее, эти два подхода не взаимоисключающие. Scrum-команда *может* разрешить Product Owner-у менять приоритеты в середине спринта (хотя чаще это рассматривается как исключение). И Kanban-команда *может* решить добавить ограничения на то, когда приоритеты могут быть изменены. Kanban-команда может даже принять решение об использовании ограниченных по времени итераций с фиксированными обязательствами, точно также как в Scrum-е.

8

Scrum-доска очищается каждую итерацию

Scrum-доска на разных этапах спринта, как правило, выглядит примерно так.

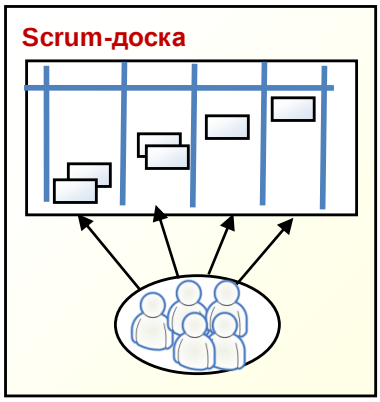
Когда спринт закончился, доска очищается – все элементы убираются. Стартует новый спринт, и после планирования спринта у нас появляется новая Scrum-доска с новыми элементами в левом столбце. Технически это потери (мусор), но для опытных Scrum-команд обычно не требуется слишком много времени, и процесс обновления доски может дать прекрасное ощущение завершенности от закрытия запланированных задач. Это как мыть посуду после обеда – делать это неприятно, зато чувствуешь себя потом прекрасно.

В Kanban-е доска, как правило, штука постоянная – вам не нужно ее обновлять, чтобы начать все сначала.

9

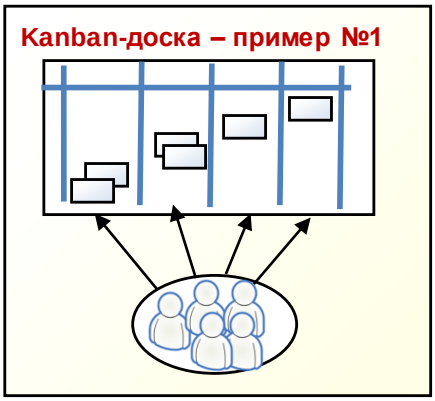
Scrum предписывает кросс-функциональные команды

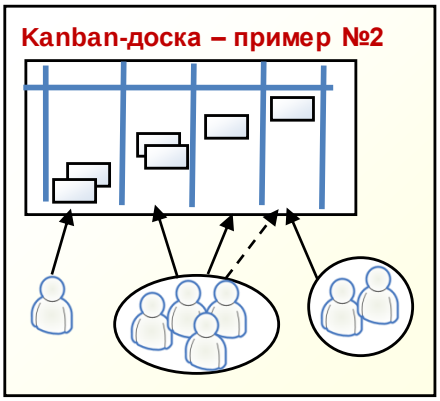
Scrum-доска используется только одной командой. Scrum-команда является кросс-функциональной – все участники обладают всеми навыками, необходимыми для успешного выполнения всех задач итерации. Scrum-доска, как правило, доступна для просмотра всем желающим, но только Scrum-команда, владеющая этой доской, может вносить изменения – это инструмент для управления их обязательствами на эту итерацию.



В Kanban кросс-функциональные команды опциональны, и доска не обязательно должна принадлежать одной конкретной команде-владельцу. Доска отображает рабочий процесс и не обязательно для одной команды.

Вот два примера:





Пример 1: Со всей доской работает одна кросс-функциональная команда. Наподобие Scrum-а.

Пример 2: Product Owner расставляет приоритеты в колонке 1. Кросс-функциональная команда выполняет разработку (колонка 2) и тестирование (колонка 3). Релиз (столбец 4) осуществляется специализированной группой. Компетентность членов команд может пересекаться, так что если команда, которая делает релиз, становится узким местом, то кто-то из разработчиков должен будет им помочь.

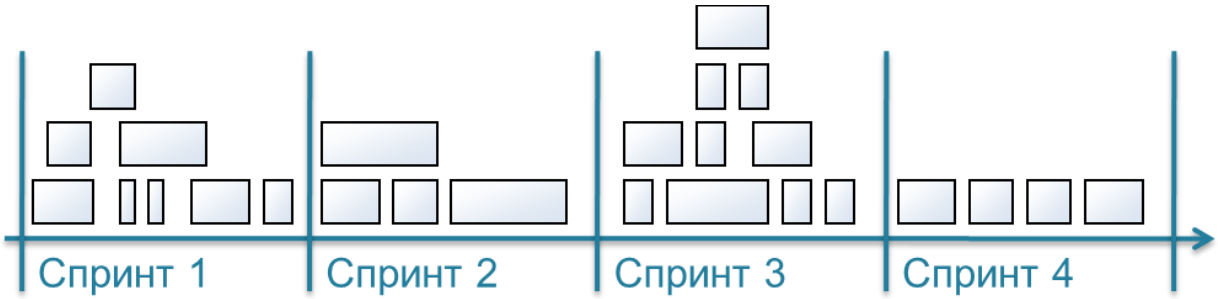
Таким образом, в Kanban-е вам необходимо установить основные правила касательно того, кто и как использует доску, а затем поэкспериментировать с ними, чтобы оптимизировать процесс.

10

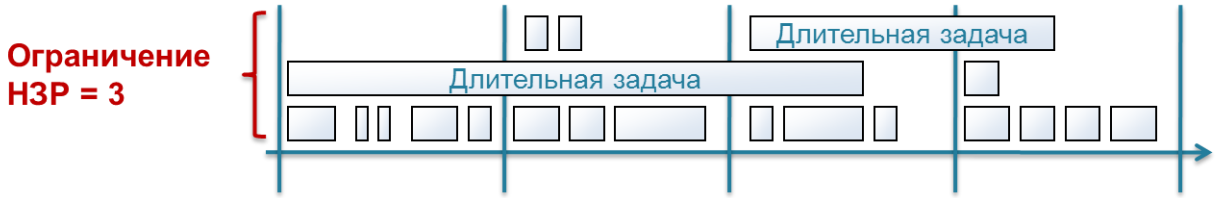
В Scrum-е элементы backlog-а должны вписываться в спринт

И Scrum и Kanban базируются на инкрементальной разработке, т.е. работа разбивается на меньшие части.

Scrum-команда возьмется только за те задачи, которые возможно закончить за одну итерацию (согласно критерию готовности). Если задача слишком велика, команда и Product Owner будут пытаться найти способ разбить ее на более мелкие куски до тех пор, пока те не впишутся в спринт. Если элементы в основном большие, увеличится длина итерации (но обычно не более четырех недель).



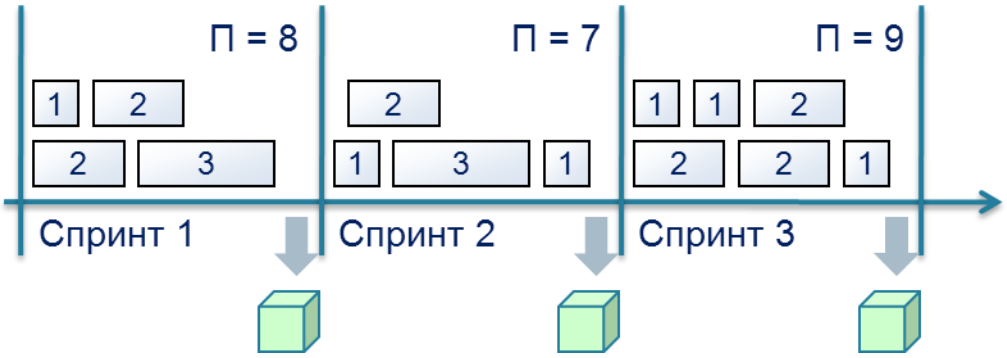
Kanban-команды пытаются минимизировать время выполнения и обеспечить равномерную загрузку, что неявно стимулирует разбивать задачи на относительно небольшие части. Но при этом не существует никакого явного правила, что элементы должны быть такого размера, чтобы вписаться в определенный временной интервал. На одной и той же доске у нас может быть один элемент, на выполнение которого уйдет один месяц, и другой, на который уйдет один день.



11

Scrum предписывает делать оценки и мерять производительность

В Scrum-е команды должны оценивать относительные размеры (т.е. объем работ) для каждого элемента, который они обязуются выполнить. Суммируя размеры каждого элемента, завершенного в конце каждого спринта, мы получаем производительность. Производительность – мера объема задач, как много мы можем реализовать за спринт. Вот пример работы команды со средней производительностью, равной 8.



Знать, что средняя производительность равна 8 – это хорошо, потому что тогда мы можем составить реалистичный прогноз о том, какие элементы мы сможем завершить в предстоящих спринтах, и, следовательно, сделать реалистичные планы релиза.

В Kanban оценки не предусмотрены. Так что, если вам необходимо принимать обязательства, вам необходимо самим решить, как сделать свою работу предсказуемой.

Некоторые команды решают проводить оценки и измерять производительность, прямо как в Scrum-е. Другие команды отказываются от оценок, но пробуют разбить каждый элемент на кусочки примерно одного размера. После этого они могут измерять производительность просто как количество элементов, которые были завершены в единицу времени (например, задач в неделю). Некоторые команды группируют элементы в МКЦФ (минимальная коммерчески ценная функциональность), измеряют среднее время создания МКЦФ, и используют его для соглашений об уровне обслуживания – например, "если мы обязуемся сделать МКЦФ, то она всегда будет завершена в течение 15 дней".

Существует множество интересных методов планирования релиза и управления обязательствами в стиле Kanban – но никаких конкретных техник не предписывается, поэтому смело ищите новые методы, экспериментируйте, пока не найдете те, которые сработают для вас. Возможно, мы увидим, как со временем выкристаллизуются какие-то "лучшие практики".

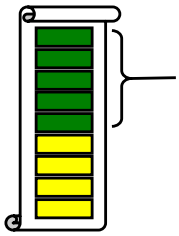
12

Оба позволяют работать над несколькими продуктами одновременно

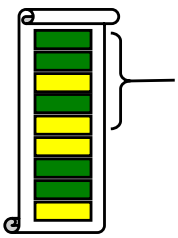
Сперва Scrum. Термин "Product Backlog" является довольно неудачным, так как неявно подразумевает, что все его элементы должны относиться к одному-единственному продукту. Вот два продукта, «Зеленый» и «Желтый», каждый со своим собственными Product Backlog-ом и своей собственной командой:

А если у вас есть только одна команда, что тогда? Без проблем – достаточно рассматривать Product Backlog скорее как Team Backlog, т.е. Backlog команды. В нем перечислены приоритетные элементы на предстоящие итерации для одной конкретной команды (или ряда команд). Таким образом, если эта команда поддерживает несколько продуктов, объедините их в один список. Это заставит нас определиться с приоритетами разных продуктов, что иногда бывает полезно. На практике есть несколько способов сделать это:

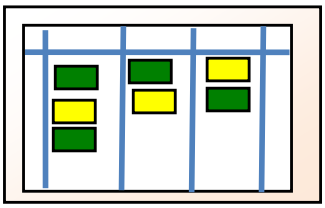
Одной из стратегий может быть концентрация усилий команды на одном продукте в течение всего спринта:



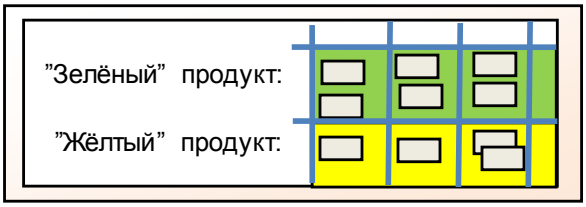
Другой стратегией может быть работа над обоими продуктами каждый спринт:



Аналогично в Kanban-е. Можно отображать на одной доске ход работ сразу над несколькими продуктами. Чтобы различать их, можно использовать либо карточки разных цветов:



либо "дорожки" (swimlanes):



13

Оба – Lean и Agile

Я не собираюсь здесь пересказывать книгу "Lean Thinking" и Agile-манифест, но в целом и Scrum, и Kanban достаточно хорошо вписываются в их ценности и принципы. Например:

- И Scrum, и Kanban используют "вытягивающие" системы планирования, которые соответствуют принципу управления запасами "Точно в срок" (TBC, JIT -Just In Time).Это означает, что именно команда выбирает, когда и сколько работы взять на себя, они "вытягивают" работу, как только они готовы, а не работа "проталкивается" к ним извне. Так же, как принтер вытягивает следующую страницу только тогда, когда он готов к печати на ней (хотя и существует небольшой, ограниченных размеров лоток, откуда он эту бумагу вытаскивает)
- И Scrum, и Kanban основываются на непрерывной и эмпирической оптимизации процесса, что соответствует принципу Kaizen в Lean.
- И Scrum, и Kanban считают, что важнее реагировать на изменения, а не следовать плану (хотя Kanban обычно позволяет отреагировать быстрее, чем Scrum), что соответствует одной из четырех ценностей Agile-манифеста.

… и так далее.

С одной стороны может показаться, что Scrum, который предписывает объединять задачи в ограниченные по времени итераций, не очень-то и Lean. Но это зависит от выбранной длины итерации, и от того, с чем сравнивать.
Если сравнивать с более традиционным процессом, когда мы, возможно, собираем и выпускаем что-то 2-4 раза в год, то Scrum-команда, которая выпускает рабочий код каждые 2 недели – это уже самый настоящий Lean.
А когда вы делаете итерации все короче и короче, вы, по сути, приближаетесь к Kanban-у. Когда начинаются разговоры о том, чтобы сделать итерации меньше одной недели, тогда можно вообще отказаться от ограниченных по времени итераций.
Я говорил это прежде, и я повторю еще раз: экспериментируйте, пока не найдете то, что подойдет вам! А потом продолжайте экспериментировать :o)

14

Незначительные различия

Есть ещё несколько отличий, о которых хорошо бы знать, не смотря на то, что они кажутся менее важными, чем упомянутые выше.

Scrum предписывает приоритезированный Product Backlog

В Scrum-е расстановка приоритетов всегда производится путем сортировки элементов Product Backlog-а, и изменения приоритетов вступают в силу со следующего (а не текущего) спринта. В Kanban-е же вы можете выбрать любой способ расстановки приоритетов (или даже вообще никакого), и он заработает, как только будет закончена какая-то задача (а не в определенное время). У команды может быть, а может и не быть Product Backlog-а, и он, в свою очередь, может быть приоритезированным или нет.

На практике это мало что меняет. На Kanban-доске самая левая колонка фактически выполняет те же функции, что и Product Backlog в Scrum-е. Независимо от того, отсортирован этот список по приоритетам или нет, команде необходимо какое-то правило выбора первого элемента. Примеры таких правил:

- Всегда выбирать верхний элемент
- Всегда выбирать самый старый элемент (соответственно, у каждого элемента есть метка времени)
- Выбирать любой элемент
- Тратить примерно 20% времени на поддержку, а 80% – на новую функциональность
- Распределять время команды между продуктами A и B по возможности равномерно
- Всегда сначала выбирать "красные" элементы, если таковые имеются

В Scrum-е также для Product Backlog-а можно применять Kanban-подход. Мы можем ограничить его размер и создать правила, по которым он должен быть приоритезирован.

В Scrum-е обязательны ежедневные собрания

Scrum-команда проводит короткие встречи (не более 15 минут) каждый день, в одно и то же время в одном и том же месте. Цель этого собрания – обмен информацией о происходящем, планирование работы на текущий день и определение существенных проблем. Иногда его называют ежедневным standup-ом, так как он обычно проходит стоя (чтобы он проходил быстрее и энергичнее).

Ежедневные встречи не обязательны в Kanban-е, но, тем не менее, большинство Kanban-команд все равно их устраивают. Это отличная практика, независимо от того, какой процесс вы используете.

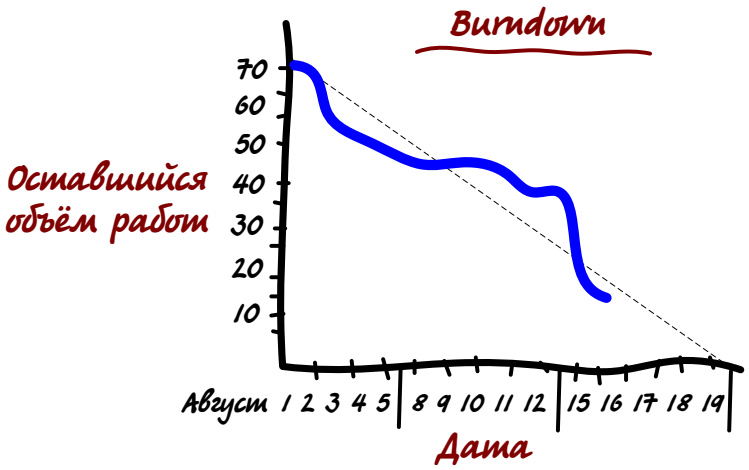
В Scrum-е формат встречи ориентирован на людей – все отчитываются один за другим. Большинство Kanban-команд используют формат более ориентированный на доску, фокусируясь на узких местах и других видимых проблемах. Этот подход более масштабируем. Если у вас 4 команды с общей доской,

и они вместе проводят ежедневные собрания, может не быть необходимости слушать каждого, так как можно сфокусироваться на проблемных частях доски.

В Scrum-е обязательны burndown диаграммы

Burndown диаграмма спринта ежедневно показывает, сколько работы осталось в текущей итерации.

Единица измерения Y-оси та же, что используется для оценки задач спринта. Обычно это часы или дни (если команда разбивает элементы backlog-а на задачи) или story points (если нет). Хотя, вообще-то вариантов масса.



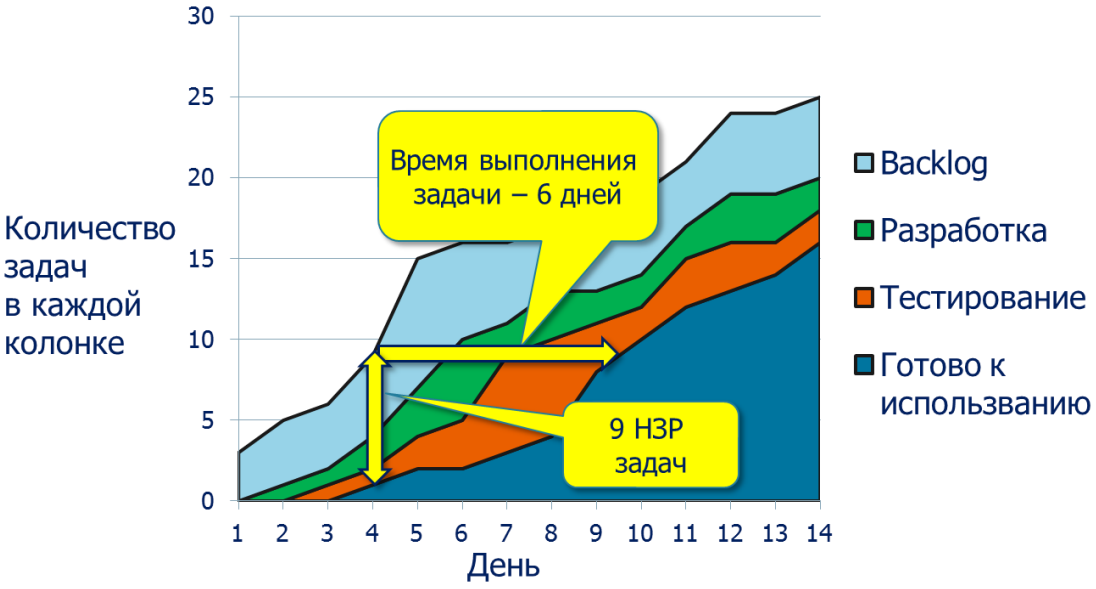
В Scrum-е burndown диаграммы спринта являются одним из основных инструментов отслеживания состояния итерации.

Некоторые команды также используют burndown диаграммы такого же формата, но на уровне релиза – обычно они показывают, сколько story points остается в Product Backlog-е после каждого спринта.

Основное назначение burndown диаграммы – как можно раньше определить, отстаем мы от графика или опережаем его, чтобы иметь возможность отреагировать.

В Kanban-е burndown диаграммы не обязательны. Да и вообще никаких обязательных диаграмм нет. Но, конечно же, можно использовать любые из них (в том числе и burndown).

Вот пример диаграммы суммарного потока. Этот тип диаграмм хорошо иллюстрирует насколько гладко идет процесс и как количество незавершенной работы влияет на время выполнения задач.



Давайте посмотрим, как это работает. Каждый день подсчитывается количество элементов в каждой колонке Kanban-доски и отмечается на оси Y. Так, на 4-й день на доске было 9 элементов. Начиная с

самой правой колонки: 1 элемент – готов к использованию, 1 – в тестировании, 2 – в разработке, и 5 элементов в backlog-е. Если мы будем ставить такие точки каждый день, а потом соединим их, то мы получим славную диаграмму, наподобие приведенной ниже. Вертикальная и горизонтальная стрелки показывают взаимосвязь между количеством НЗР и временем выполнения задач.

Горизонтальная стрелка показывает нам, что элементам, добавленным в backlog на 4-й день, понадобилось в среднем 6 дней, чтобы стать готовыми к использованию. Примерно половину этого времени заняло тестирование. Мы можем отметить, что если ограничить количество НЗР в тестировании и backlog-е, то можно существенно сократить общее время выполнения задач.

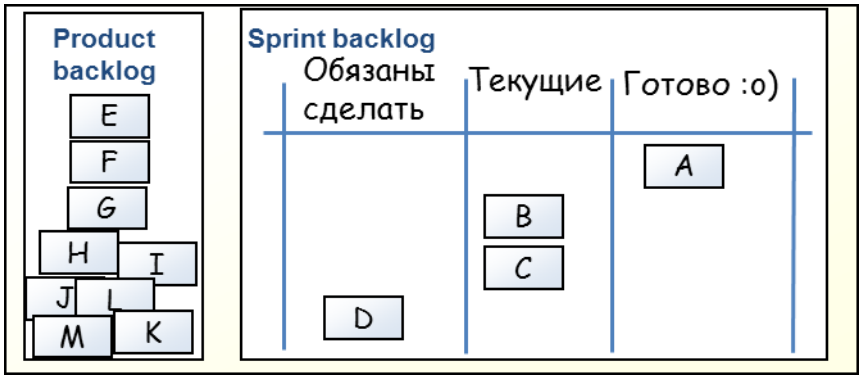
Наклон темно-синей области отображает производительность (т.е. количество завершенных элементов в день). Со временем мы сможем увидеть, как повышение производительности уменьшает время выполнения задачи, в то время как увеличение количества НЗР увеличивает это время.

15

Сравнение Scrum и Kanban досок – менее тривиальный пример

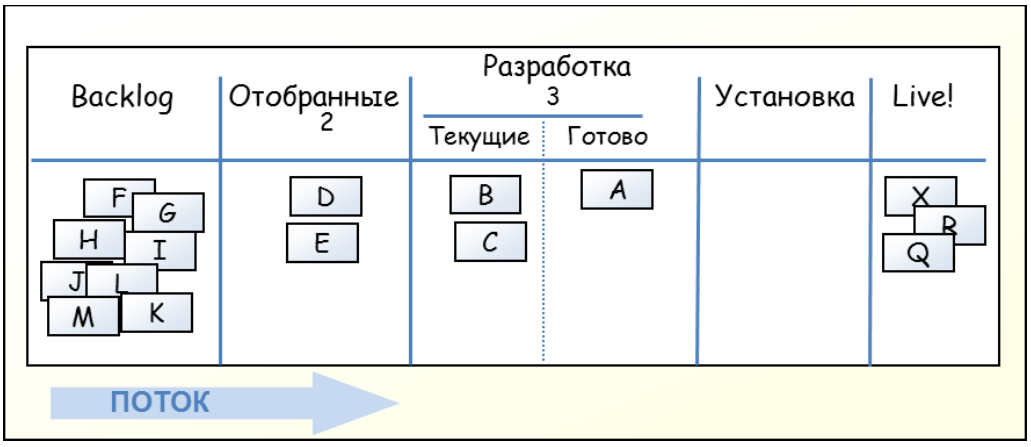
Sprint backlog в Scrum-е – это просто часть общей картины, отображающая то, чем команда занимается в рамках текущего спринта. Другая часть – это product backlog, список задач, которые Product Owner хочет видеть в будущих спринтах.

Product Owner может наблюдать за sprint backlog-ом, но, при этом, не может его менять. А вот product backlog он может менять когда угодно, но изменения вступят в силу только со следующего спринта (т.е. на текущую работу эти изменения не влияют).



По окончанию спринта команда предоставляет Product Owner-у потенциально готовый к использованию код. Итак, команда заканчивает спринт, проводит ретроспективу и с гордостью демонстрирует Product Owner-у функциональность A, B, C и D. Теперь Product Owner может принять решение, выпускать продукт или нет. Эта завершающая стадия, выпуск продукта, обычно не входит в рамки спринта и поэтому не отображается в sprint backlog-е.

А Kanban доска в этой ситуации могла бы выглядеть так:



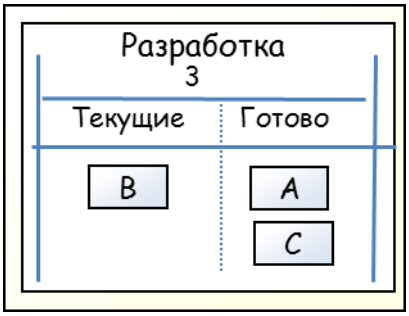
Здесь весь процесс разработки отображен на одной доске – а не просто задачи одной Scrum команды в рамках одной итерации.

В примере, приведенном выше, колонка "Backlog" – это просто общий неупорядоченный список требований. Колонка "Отобранные (Selected)" содержит наиболее приоритетные задачи и имеет Kanban ограничение в два элемента. Иными словами, здесь могут одновременно находиться только две задачи с высоким приоритетом. Когда команда готова приступить к работе над новой задачей, они берут верхнюю задачу из "Отобранные". Product Owner может менять вносить изменения в колонки "Backlog" и "Отобранные" по своему усмотрению, однако он не может поступать так с другими колонками.

Колонка "Разработка" (разделенная на две под-колонки) показывает текущие задачи и имеет Kanban ограничение в три элемента. В сетевой терминологии, Kanban ограничение – это "пропускная способность", а время выполнения задачи – "пинг" (или время отклика).

Зачем мы разделили колонку "Разработка" на две под-колонки "Текущие" и "Готово"? Чтобы команда, отвечающая за рабочую систему, понимала, что уже можно забирать для внедрения.

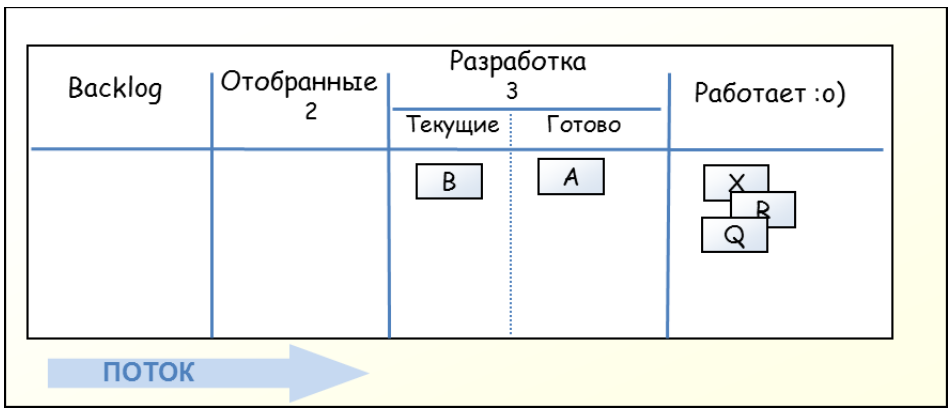
Ограничение "Разработка" в 3 элемента общее для двух под-колонок. Почему так? Допустим, есть две задачи в "Готово":



Это означает, что в "Текущие" может быть только один элемент. Что, в свою очередь, означает, что появятся избыточные ресурсы – разработчики, которые *могли бы* начать работу над новой задачей, но не смогут этого сделать из-за Kanban предела. Это даст им сильный стимул сосредоточить свои усилия и помочь выпустить эти задачи, чтобы очистить колонку "Готово", и максимизировать поток. Этот эффект является удачным и последовательным – чем больше задач в "Готово", тем меньше допускается в "Текущие", что позволяет команде сосредоточиться на правильных вещах.

Непрерывный поток

Непрерывный поток является своего рода сценарием "идеального потока", когда задачи проходят через всю доску, нигде не задерживаясь. Это значит, что в каждый момент времени над этой задачей кто-то работает. Вот как доска могла бы выглядеть в таком случае:



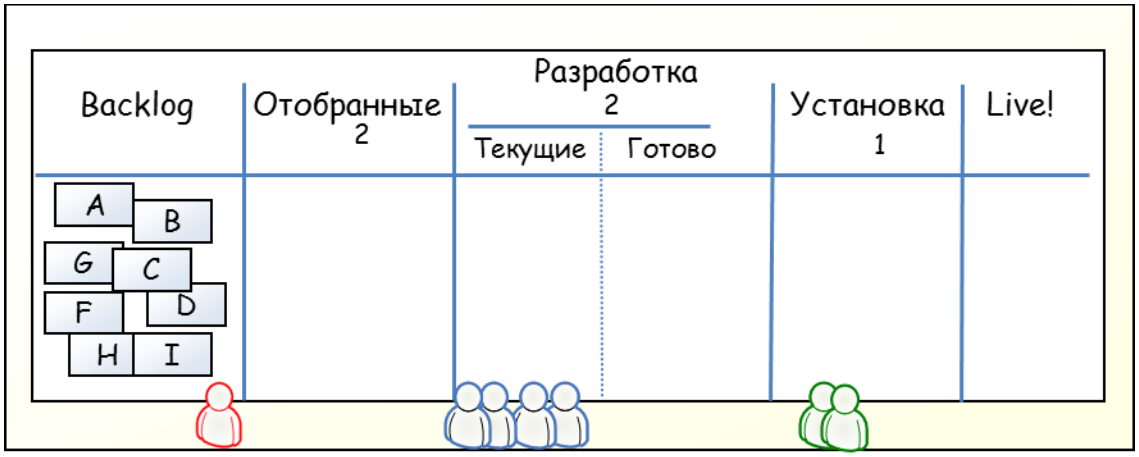
«B» сейчас в разработке, а «A» – уже работает. Когда команда готова к следующей задаче, они спрашивают у Product Owner-а, что является наиболее важным в настоящий момент и сразу получают

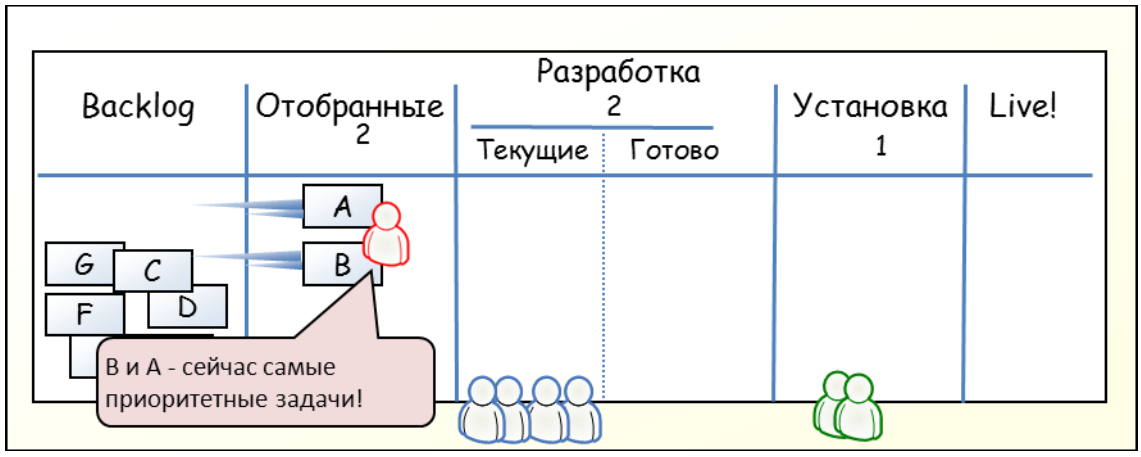
ответ. При таком идеальном сценарии мы можем избавиться от колонок "Backlog" и "Отобранные" и получить *действительно* быстрое время выполнения задачи!

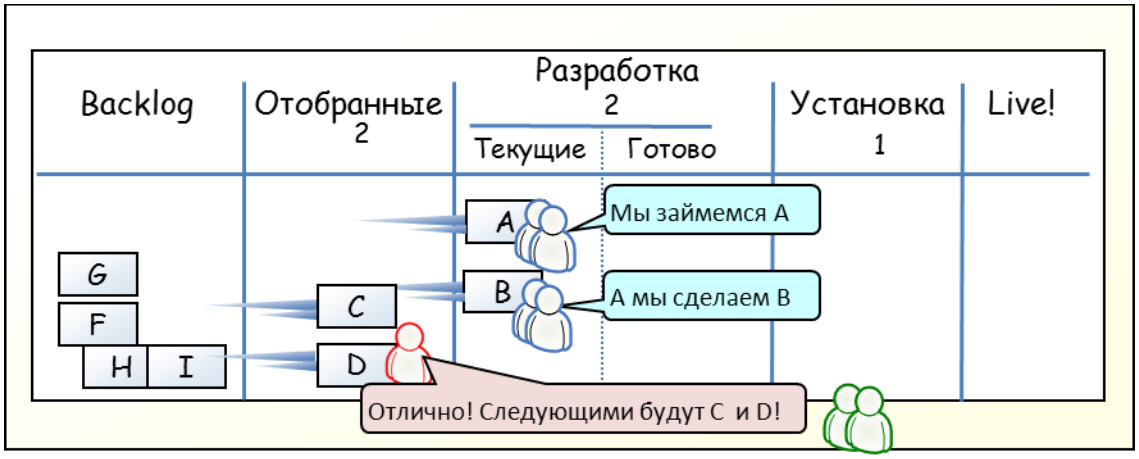
Кори Ладас сформулировал это красиво: "Идеальный процесс планирования должен всегда предоставлять команде лучшие задачи для работы, ни больше, ни меньше".

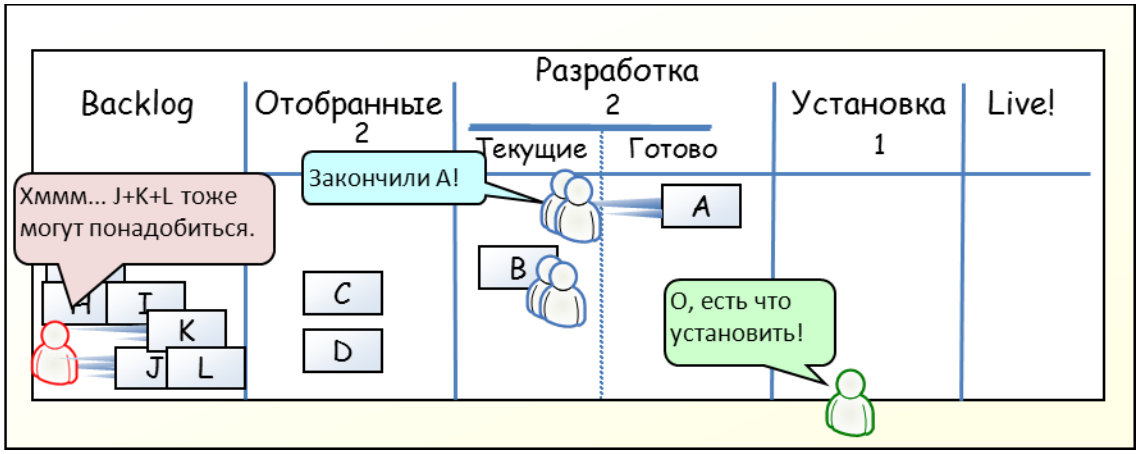
Ограничения НЗР существуют, чтобы не дать проблемам выйти из-под контроля, поэтому, если все идет гладко, ограничения НЗР реально не используются.

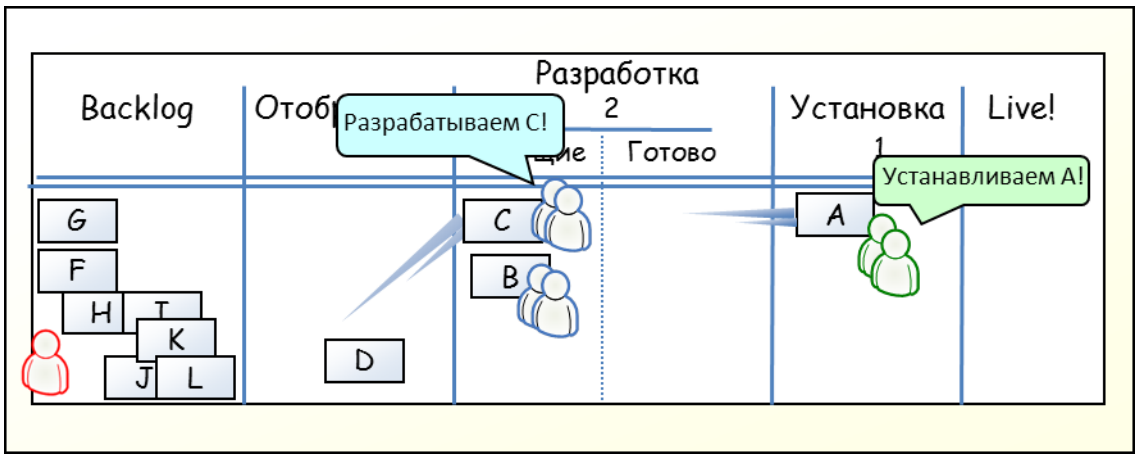
Один день в стране Канбании

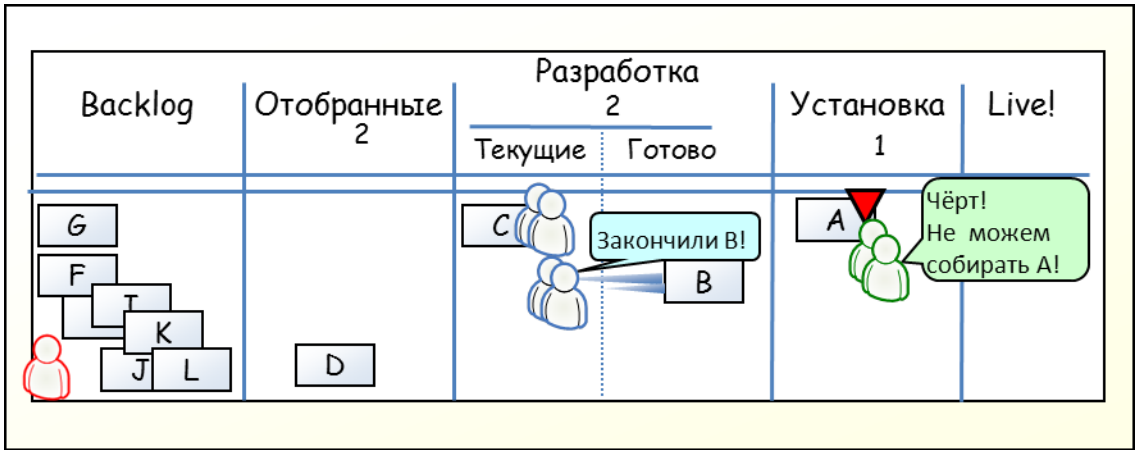


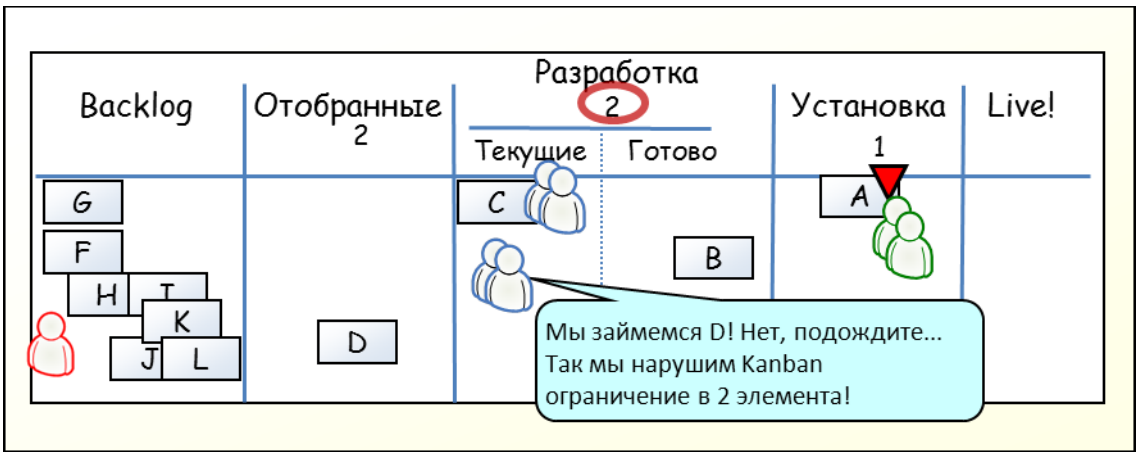


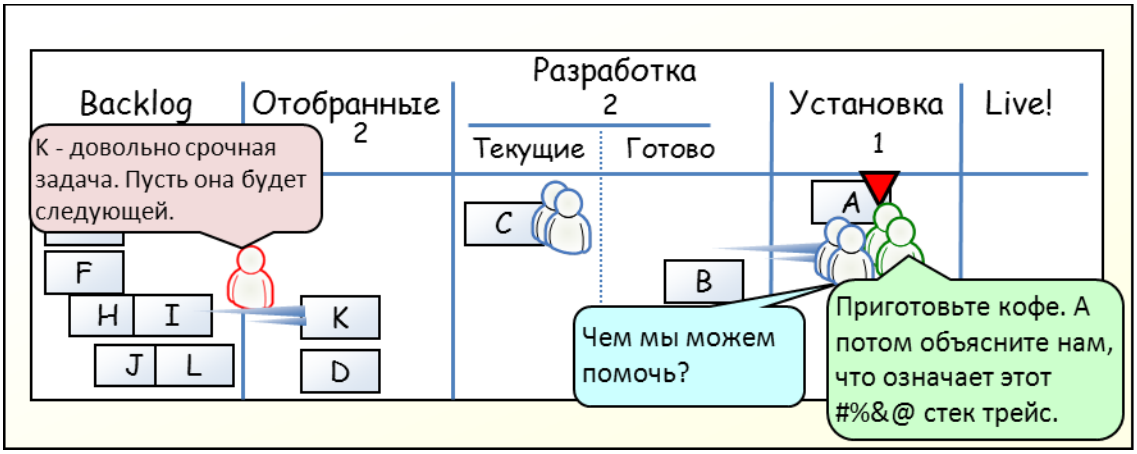


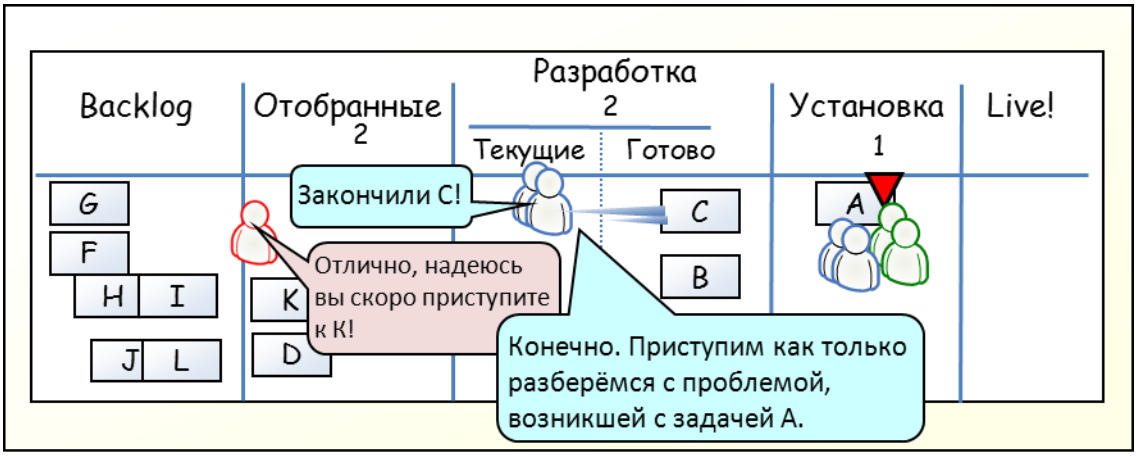

Backlog
Отобранные
2
Разработка
2
Текущие
Готово
Установка
1
Live!
Закончили A!
Хммм... J+K+L тоже могут понадобиться.
A
B
C
D
H
I
K
J
L
О, есть что установить!

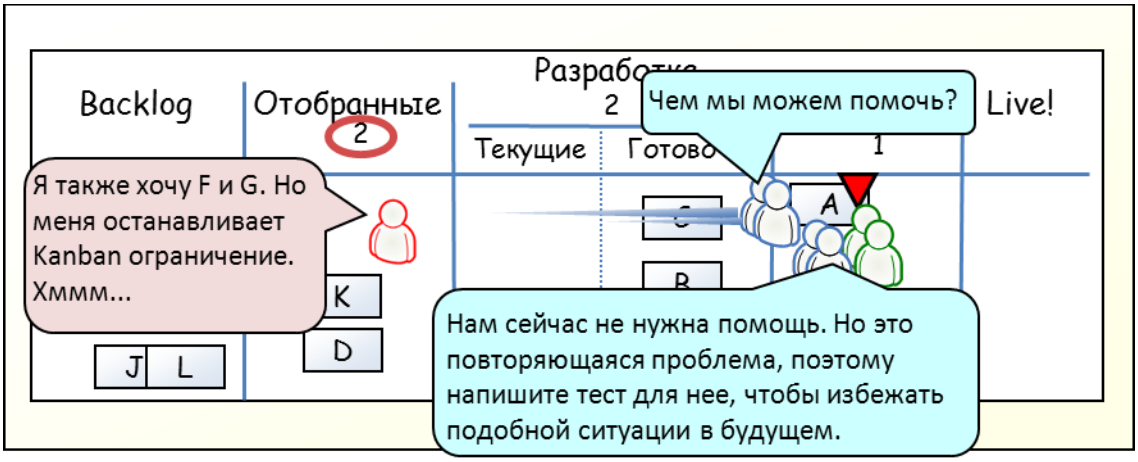

Backlog
Разработка
2
Установка
1
Live!
Готово
Разрабатываем C!
Устанавливаем A!
G
F
H
I
K
J
L
C
B
A
D

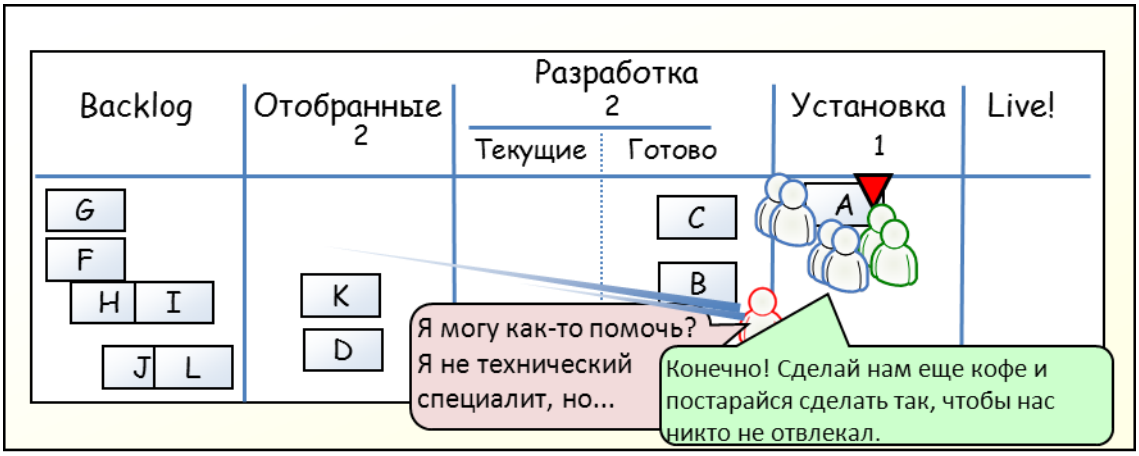

Backlog
Отобранные
2
Разработка
2
Текущие
Готово
Установка
1
Live!
G
F
H
I
K
J
L
C
Закончили B!
B
A
Чёрт!
Не можем собирать A!
D

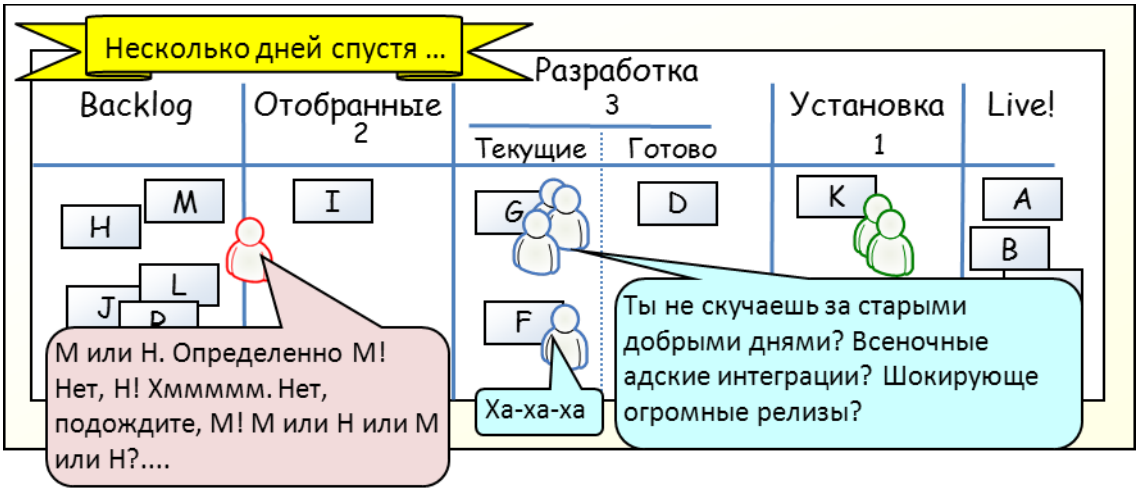

Backlog
Отобранные
2
Разработка
2
Текущие
Готово
Установка
1
Live!
G
F
H
I
K
J
L
C
B
A
D
Мы займемся D! Нет, подождите...
Так мы нарушим Kanban
ограничение в 2 элемента!


Backlog
Отобранные
2
Разработка
2
Текущие
Готово
Установка
1
Live!
K - довольно срочная задача. Пусть она будет следующей.
F
H
I
J
L
K
D
C
A
B
Чем мы можем помочь?
Приготовьте кофе. А потом объясните нам, что означает этот #%&@ стек трейс.


Backlog
Отобранные
2
Разработка
2
Текущие
Готово
Установка
1
Live!
G
F
H
I
J
L
Закончили C!
Отлично, надеюсь вы скоро приступите к K!
K
D
C
B
A
Конечно. Приступим как только разберёмся с проблемой, возникшей с задачей A.


Backlog
Отобранные
2
Разработка
2
Текущие
Готово
Установка
1
Live!
Чем мы можем помочь?
Я также хочу F и G. Но меня останавливает Kanban ограничение. Хммм...
K
D
J
L
A
Нам сейчас не нужна помощь. Но это повторяющаяся проблема, поэтому напишите тест для нее, чтобы избежать подобной ситуации в будущем.


Backlog
Отобранные
2
Разработка
2
Текущие
Готово
Установка
1
Live!
G
F
H
I
J
L
K
D
C
B
A
Я могу как-то помочь? Я не технический специалит, но...
Конечно! Сделай нам еще кофе и постарайся сделать так, чтобы нас никто не отвлекал.



Должна ли Kanban-доска выглядеть именно так?

Нет, доска, о которой шла речь выше, – всего лишь пример!

Единственное, что Kanban предписывает, это то, что поток должен быть наглядным, и незавершенная работа должна быть ограничена. Главной целью является создание плавного рабочего процесса и уменьшение времени, потраченного на выполнение задач. Поэтому вам необходимо регулярно задаваться таким вопросами:

Какие колонки нам нужны?

Каждая колонка представляет собой одно рабочее состояние, или очередь (буфер) между двумя состояниями рабочего процесса. Начните с простого и добавляйте колонки по мере необходимости.

Какими должны быть Kanban ограничения?

Когда для "вашей" колонки достигнуто Kanban ограничение, и вам нечего делать, начните искать "узкое место" дальше по потоку (т.е. задачи, которые скопились с правой стороны доски) и помогите их устранить. Если же "узких мест" нет, то это может указывать на то, что Kanban ограничение слишком мало, так как ограничения служат для минимизации риска возникновения "узких мест" дальше по потоку.

А вот если вы заметили, что много элементов находится на доске и не обрабатывается, то возможно Kanban ограничение слишком велико.

- Слишком низкое Kanban ограничение => люди простаивают => плохая производительность
- Слишком высокое Kanban ограничение => задачи простаивают => плохое время выполнения

Насколько строгими являются Kanban ограничения?

Некоторые команды рассматривают их как жесткие правила (т.е. команда не может превышать предел), некоторые – как руководство и повод для обсуждения (т.е. нарушение предела допустимо, но должно быть осознанным решением с конкретными причинами). Итак, еще раз, все зависит от вас. Я ведь говорил, что Kanban имеет мало предписаний?

16

Scrum против Kanban: резюме

Сходства

- Оба – и Lean, и Agile.
- Оба используют вытягивающие системы планирования.
- Оба ограничивают НЗР.
- Оба используют прозрачность для обеспечения улучшения процесса.
- Оба ориентированы на ранние и частые поставки продукта.
- Оба полагаются на самоорганизующиеся команды.
- Оба требуют деления задач на более мелкие.
- В обоих случаях план релиза постоянно оптимизируется на основе эмпирических данных (производительности/ времени выполнения задачи).

Отличия

Scrum	**Kanban**
Обязательны ограниченные по времени итерации.	**Ограниченные по времени итерации необязательны.** Могут быть отдельные ритмы для планирования, выпуска и усовершенствования процессов. Также могут быть событийно-управляемые итерации вместо ограниченных по времени.
Команда обязуется выполнить конкретный объем работы за эту итерацию.	**Обязательства опциональны.**
Как основная метрика для планирования и улучшения процессов используется **производительность.**	Как основная метрика для планирования и улучшения процессов используется **время выполнения задачи.**
Кросс-функциональные команды обязательны.	Кросс-функциональные команды, опциональны. **Допустимы узкопрофильные команды.**
Задачи должны быть разбиты на более мелкие так, чтобы они были завершены в течение одного спринта.	Нет каких-либо определенных размеров задач.

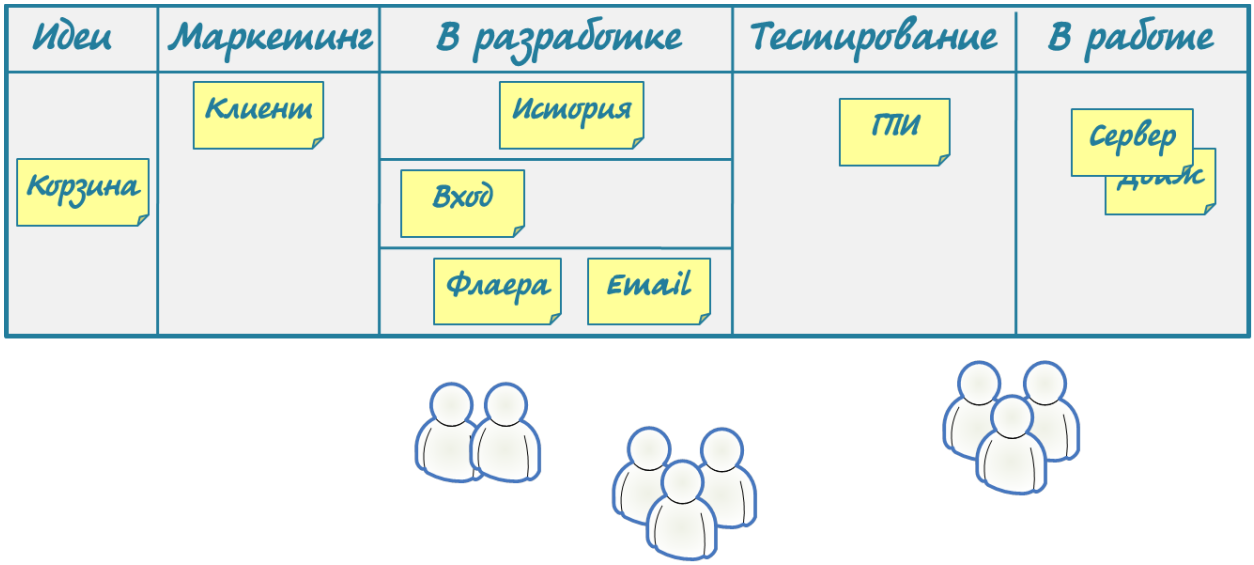
Наличие burndown-диаграммы обязательно.	Наличие каких-либо обязательных диаграмм не требуется.
НЗР ограничивается косвенно (за спринт).	**НЗР ограничивается явно** (по статусам).
Оценки задач обязательны	**Оценки задач опциональны**
Нельзя добавлять задачи в текущую итерацию.	**Можно добавлять новые задачи, когда это возможно.**
За backlog спринта отвечает только одна конкретная команда.	**Kanban-доска может совместно использоваться несколькими группами** или отдельными лицами.
Предписаны 3 роли (Product Owner / Scrum Master / Команда).	**Нет предписанных ролей.**
Scrum-доска очищается между спринтами.	**Kanban-доска является неизменной.**
Приоритезированный Product Backlog обязателен.	**Приоритезация не является обязательной.**

Вот так! Теперь вы знаете различия.

Но это еще не все, теперь пришло время для самого интересного! Наденьте ботинки, пора прыгать в окопы с Маттиасом и посмотреть, как это выглядит на практике!

Часть II – разбор конкретного случая

Kanban в реальной жизни



Это история о том, как мы научились быть лучше, используя Kanban. Когда мы начинали, не так уж много информации было доступно, и даже "доктор Google" не мог нам ничего порекомендовать. Сегодня же Kanban успешно развивается, и число источников знаний постоянно растет. Я настоятельно рекомендую ознакомиться с работами Дэвида Андерсона, например, с "классами обслуживания" ("classes of service"). Еще мы хотим первый (и, обещаем, последний) раз предостеречь. Какие бы решения вы не применяли, убедитесь, что они относится к Вашей конкретной задаче. Все. Идем дальше. Вот наша история.

/ Маттиас Скарин

17

Суть технической поддержки

Если вы когда-нибудь круглосуточно дежурили на телефоне, вам хорошо знакомы чувства, возникающие при поддержке реальной системы, с которой работают клиенты. От вас ожидают, что вы сможете разрулить любую ситуацию в любое время дня и ночи, независимо от того, по вашей вине возникли проблемы или нет. Спрашивали у всех – никто ничего не знает, поэтому они и звонят вам. Ничего себе задачка, особенно если учесть, что это не вы создавали аппаратное обеспечение, не вы писали драйвера, ОС и специализированное ПО. Часто все ваши возможные действия сводятся к локализации проблемы, уменьшению ущерба, записи информации, необходимой для воспроизведения проблемы, и ожиданию, когда же некто, ответственный за эту проблему, соизволит её воспроизвести и устранить.

Ключевые факторы успеха в этом случае – быстрота реакции на запрос и умение решать проблемы, в сочетании со скоростью и тщательностью.

18

С какой стати вообще что-то менять?

В 2008 году одним из наших клиентов была скандинавская компания, занимающаяся разработкой компьютерных игр. Они воплотили целый ряд процессных улучшений. Одно из нововведений состояло в масштабировании Scrum-а на уровень всей компании и постепенном устранении препятствий, мешающих командам разработчиков поставлять ПО. Как только поставки программного обеспечения наладились, вся нагрузка сместилась дальше по цепочке – на отдел техподдержки. Ранее команда технической поддержки по большей части наблюдала за происходящим со стороны, теперь же они все больше и больше вовлекались в процесс разработки на правах полноценных участников.

Рисунок 1. В составе технической поддержки находилось три команды: инженеры баз данных (DBAs), системные администраторы и вторая линия поддержки.

Как оказалось, наша помощь разработчикам не решила всех проблем. Если бы мы продолжали уделять внимание лишь командам разработчиков, это привело бы к задержкам не менее важных улучшений инфраструктуры, находящейся под контролем команд техподдержки. Улучшения были необходимы в обеих областях.

Вдобавок, прогресс в командах разработчиков привел к тому, что менеджеров буквально завалили просьбами проанализировать те или иные идеи и предоставить по ним отзывы. Это означало, что у менеджеров оставалось все меньше времени на приоритизацию задач и решение различных проблем. Управленческая команда поняла, что им необходимо действовать, пока ситуация не вышла из-под контроля.

19

С чего мы обычно начинаем?

Хорошо, что мы начали с опроса разработчиков о том, кто является заказчиком у техподдержки.

Техническая поддержка с точки зрения разработчиков

Я спросил: "Какие три основные мысли приходят к вам на ум, когда вы думаете о "техподдержке"? Наиболее распространенными ответами были:

"Все постоянно меняется"

"Их схема работы – полный отстой"

"Очень компетентны, когда речь идет об инфраструктуре"

"Чем эти типы вообще занимаются?"

"Они хотят помочь, но реальной помощи добиться трудно"

"Нужно отправить кучу писем, чтобы добиться от них простых вещей"

"Проекты слишком длинные"

"С ними очень сложно общаться"

Так вкратце выглядят технические службы с точки зрения разработчиков. Теперь, давайте сравним это с точкой зрения техподдержки о разработчиках.

Разработчики с точки зрения технической поддержки

"Почему вы не используете преимущества текущей платформы"?

"Давайте делать релизы с меньшим количеством проблем!"

"Мы страдаем из-за вашего ужасного качества!"

"Они должны измениться" – вот общий лейтмотив аргументов с обеих сторон. Очевидно, что для того, чтобы стать на путь решения общих проблем, надо было менять такие настроения. Но был и положительный момент: фраза "очень компетентны, когда речь идет об инфраструктуре" (указывает на доверие к основной компетенции), которая заставила меня поверить, что образ мышления "мы-они" можно исправить, если бы мы создали благоприятные условия работы. Убрать переработки и сосредоточить внимание на качестве, было единственно возможным жизнеспособным вариантом.

20

Поехали ...

Итак, нам было пора оправляться, но откуда же начать путь? Единственное, в чём мы были уверены, это в том, что закончим мы совсем не там, где начнём.

Я сам из разработчиков, так что о сути технической поддержки я знал мало. Но я не собирался "ворваться и всё поменять". Мне нужен был менее конфликтный подход, который бы, тем не менее, научил нас нужному, убрал бы ненужное, и при этом был бы лёгок в изучении.

Были рассмотрены следующие кандидаты:

1. Scrum – хорошо отработанный и проверенный подход для команд разработчиков.

2. Kanban – новый и непроверенный подход, который зато хорошо ложился на принципы Lean, которых нам как раз и не хватало.

После бесед с менеджерами, принципы Kanban-а и Lean показались хорошими ответами на вопросы, которые нас мучили. С их точки зрения спринты нам не очень подходили, так как приоритеты менялись каждый день. Таким образом, логичной отправной точкой был выбран Kanban, несмотря даже на то, что никто из нас не знал, что же это за зверь.

21

Запуск команд

Как мы обычно запускаем команды? У нас не было руководства по тому, как начать работу. Ошибиться в этом деле было бы слишком рискованно. Помимо того, что мы бы упустили возможность привнести улучшения, мы имели дело с живой системой, обслуживаемой узкими специалистами очень высокой квалификации, которых было бы очень трудно заменить. Оттолкнуть их с самого начала было бы крайне плохой идеей.

- Так что – просто начать? И разгребать проблемы по мере их поступления?
- Или же для начала провести тренинг?

Для нас было очевидно – надо провести тренинг, не так ли? Но как именно? Было проблематично собрать всю команду технической поддержки для участия в тренинге (а кто будет отвечать на звонки клиентов?). В конце концов, мы решили провести тренинг на полдня, и сделать его простым и основанным на практических упражнениях.

Тренинг

Одним из преимуществ тренинга было то, что он помог бы нам выявить проблемы уже на начальном этапе. Он также обеспечил бы нам атмосферу высокого доверия, что позволило бы обсуждать с членами команды проблемы напрямую, без намёков. Я имею в виду – давайте посмотрим правде в глаза – не все были прямо в восторге от необходимости менять текущие методы работы. Но большинство членов команды были не против попробовать. Поэтому мы провели тренинг, продемонстрировав наиболее важные принципы, и упрощённую симуляцию Kanban-а.

Изучение некоторых основных принципов

- Ограничение объёмов работ
- Загрузка и время обработки
- Незавершённая работа и пропускная способность
- Теория ограничений

Демонстрация Kanban

- 3 "типа работ":
  - ответы на вопросы,
  - создание автомобиля из кубиков Lego,
  - проектирование и построение дома.
- 3 итерации
  - Измерьте производительность для каждого вида работ
  - Поэксперементуйте, подберите значение НЗР.
- Обсуждение результатов

В конце тренинга мы провели голосование методом "кулак или пять пальцев", чтобы проверить, готовы ли команды попробовать это по-настоящему. Каких-либо возражений на этом этапе не возникло, так что у нас был карт-бланш на внесение изменений.

22

Обсуждение с заинтересованными сторонами

Было, похоже, что внедрение Kanban затронет много разных отделов. Изменения, скорее всего, будут к лучшему: команды начнут отказываться от задач, которые не в состоянии завершить, будут бороться за качество и способствовать удалению низкоприоритетных задач из своих backlog-ов. И тем не менее, заранее все обсудить – никогда не помешает.

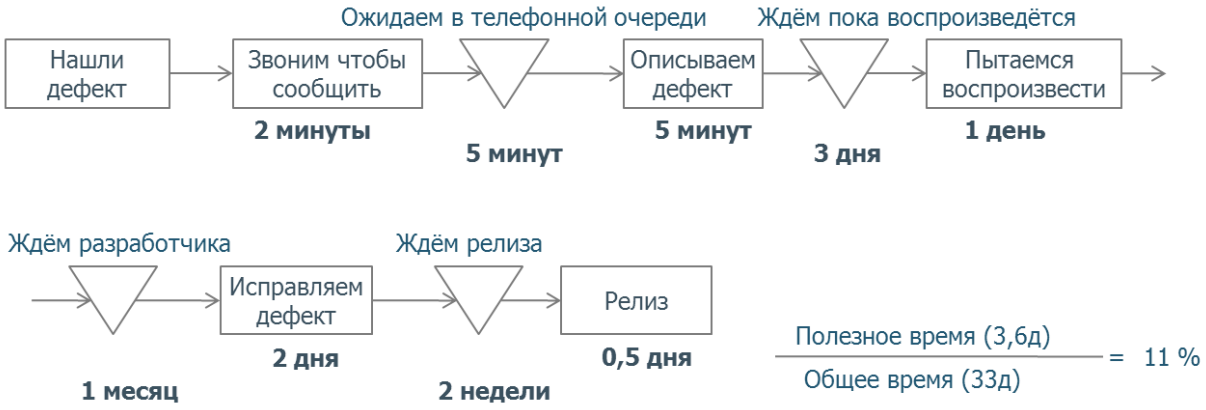
Перемены должны были в первую очередь затронуть горячую линию и начальников отделов. Но поскольку они участвовали в тренинге, то уже были настроены положительно. Аналогичная ситуация была и с командами разработки, которые ожидали тех или иных улучшений и были к ним готовы. Но с одной командой – командой поддержки – дело обстояло иначе. Их основной проблемой было то, что они были перегружены работой. Кроме того, они обрабатывали запросы клиентов, поскольку компания дала обязательство реагировать на все клиентские обращения. Скорее всего, с внедрением Kanban и жестким ограничением НЗР это должно было измениться.

Итак, мы обошли всех основных заинтересованных участников и рассказали им о наших намерениях, ожидаемых выгодах и возможных последствиях. К моему облегчению, наши идеи были в целом приняты хорошо, иногда даже с комментариями вроде: "Было бы здорово, если бы мы, наконец, смогли решить эти вопросы раз и навсегда".

23

Создание первой Kanban-доски

Хороший способ начать строительство Kanban-а – создать карту создания стоимости (value stream map). Визуализация цепочки создания стоимости (value chain) дает представление о текущем состоянии проекта, а также о потоке и времени прохождения через систему (время цикла).



Но мы начали гораздо проще – Kanban-доска, начерченная на бумаге вместе с менеджером. Пару раз пересмотрели и вперёд. Сразу же возникли следующие вопросы:

- Какие виды работ у нас есть?
- Кто ими занимается?
- Нужно ли вводить коллективную ответственность по различным видам работ?
- Как работать в условиях коллективной ответственности, но узкой специализации?

Поскольку разные виды работ имели различные соглашения об уровне услуг (SLA), вполне естественно, если каждая команда чертила бы свою собственную доску. И они придумали свои собственные столбцы и строки.

Следующим судьбоносным решением стало решение использовать коллективную ответственность при различных видах работ или нет. "Должны ли мы выделить часть команды на работу с прямыми запросами (с текучкой), а остальным дать сосредоточиться на проектной работе?" Сначала мы решили попробовать коллективную ответственность. Основной причиной было то, что самоорганизация и развитие членов команды – ключевой фактор поддержания общего роста. Недостатком такого решения было то, что, возможно, люди могли бы отвлекать друг друга, но мы подумали с этого стоит начать. Небольшое отступление: когда мы проводили семинар, команды фактически самоорганизовывались ради решения этой проблемы. Они дали одному человеку работать с текучкой, в то время как остальные работали с более крупными проблемами.

Первая модель Kanban-доски

Ниже приведена исходная доска, которую мы использовали для Kanban-а. Обратите внимание, что группа решила двигать задачи снизу вверх (как пузырьки в воде), а не использовать традиционную модель слева направо.

Рисунок 2. Это первая Kanban-доска. Приоритеты расположены слева направо, направление потока – снизу вверх. Незавершенная работа учитывается как общее число задач в строке "незавершенная работа" (обведена красным). Модель создана на основе опыта Линды Кук.

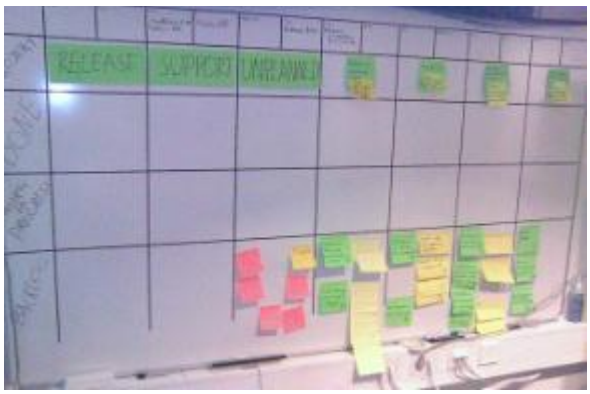


Рисунок 3. Первая доска Kanban для команды системных администраторов.

Строки

Состояние задач	Как мы это понимаем
Backlog	Истории, про которые известно, что они нужны.
Готово к разработке	Трудозатраты определены, история разбита на задачи с максимальным размером 8 часов.
Незавершенная работа	Строка, к которой применено ограничение незавершенной работы. Мы начали с ограничения 2 * n-1 (где n – размер команды, а -1 – накладные расходы на взаимодействие). Таким образом, для 4 человек ограничение незавершенной работы равно семи.
Готово	Может быть запущено пользователем.

Столбцы

Тип работы	Как мы это понимаем
Релиз	Помощь разработчикам в подготовке продукта к выпуску.
Поддержка	Менее важные запросы от других команд.
Незапланированная	Непредвиденная работа, которую необходимо сделать, но она не имеет определённого заказчика. Например, незначительные улучшения инфраструктуры.
Проект А	Большой проект группы технической поддержки, например, изменение аппаратного обеспечения демонстрационной среды.
Проект Б	Другой большой проект.

Не все Kanban-доски выглядят одинаково. Все начинались с простого эскиза и развивались шаг за шагом.

24

Первое ограничение незавершенной работы

Наше первое ограничение незавершенной работы (НЗР) было достаточно большим. Мы полагали, что благодаря визуализации потока, мы увидим и поймем, что происходит, и с первого раза мы вряд ли смогли бы угадать наилучшее ограничение НЗР. Со временем мы бы подгоняли ограничения незавершенной работы (НЗР) при каждом удобном случае (все, что нам нужно было делать – указать это на доске).

Первым ограничением НЗР, которое мы использовали, стало 2n-1. (n = количество членов команды, -1 для поощрения сотрудничества). Почему? Просто ничего лучше мы не придумали ☺. Да и споров это значение не вызывало. Формула обеспечила простое и логичное объяснение для любого, кто пытался добавить работу команде: "... учитывая, что каждый член команды может работать максимум с двумя задачами в один момент времени, одна активная и одна в ожидании, почему вы полагаете, что они возьмут на себя *еще больше?*" Оглядываясь назад можно сказать, что любое большое ограничение сработает для начинающих. Отслеживая состояние Kanban-доски легко выяснить правильное значение ограничения по ходу дела.

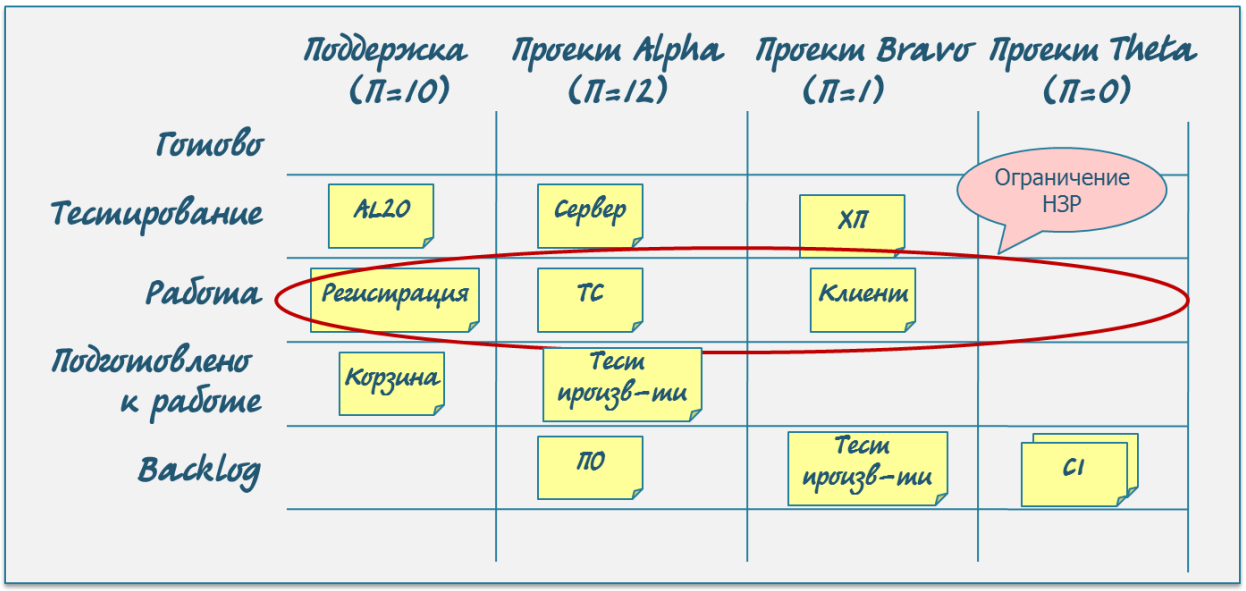


Рисунок 4. Как мы использовали ограничение незавершенной работы для команды администраторов баз данных и системных администраторов – единое ограничение для каждого типа работ.

Кстати, мы заметили, что бесполезно определять ограничение незавершенной работы в story point-ах. Его было слишком тяжело отслеживать. Единственное ограничение, которое отслеживать было достаточно легко – простой подсчет количества элементов (= параллельные задачи).

Для команды поддержки мы использовали ограничение незавершенной работы (НЗР), которое определялось для колонки. Это из-за того, что нам была нужна более быстрая реакция при достижении ограничения.

25

Уважение к ограничению незавершенной работы

Легко придерживаться ограничения незавершенной работы в теории, но довольно тяжело на практике. Это значит, что на определенном этапе надо говорить "нет". Мы опробовали различные подходы к этому вопросу.

Обсуждение возле доски

При нарушении ограничения мы подводили заинтересованные стороны к доске и спрашивали, чего они хотят достичь. Поначалу наиболее частой причиной нарушений была простая неопытность. Иногда причиной оказывались разные подходы в приоритезации – чаще всего это были люди из узкоспециализированной команды, которые брали задачи их специфической области. Было всего пару разногласий, а большая часть вопросов разрешалась очень быстро после обсуждения возле доски.

Выделение колонки "переполнение"

Если говорить "нет" было неприемлемо, и удалять задачи тоже было накладно, мы передвигали низкоприоритеные элементы в колонку "переполнение", когда превышались пределы НЗР. К таким задачам применялось два правила:

1. Мы о них помним, и займемся ими, как только у нас появится время.
2. Вы будете знать, если мы решим не делать их вообще.

Всего через пару недель стало ясно, что задачами в колонке "переполнение" никто заниматься никогда не будет, так что с одобрения менеджера команды мы смогли окончательно от них избавиться.

Рисунок 5. Эскиз Kanban-доски для команды технической поддержки; колонка "переполнение" – справа.

26

Какие задачи разместить на доске?

Ранее мы решили размещать *не всю* работу, выполняемую командой. Мониторинг таких вещей как телефонный звонок или кофепитие может превратить Kanban в административного монстра. Нашей задачей было *решать* проблемы, а не создавать их ☺. Поэтому мы договорились размещать на доске задачи продолжительностью более одного часа, а всё остальное рассматривать как "белый шум". Ограничение в один час хорошо себя зарекомендовало, и было одним из того немногого, что не поменялось. (Нам пришлось пересмотреть предположение о том, какое влияние имеет фоновый шум, но об этом немного позже).

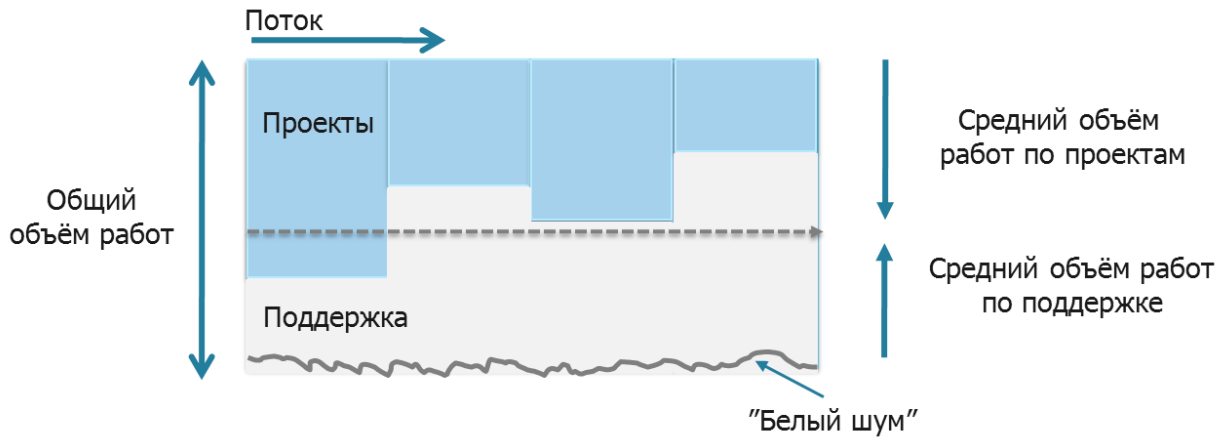


Рисунок 6. Мы начали с предположения, что общая нагрузка (capacity) направлена на выполнение работы двух типов: побольше (проект) и поменьше (поддержка). Отслеживание производительности по отдельным проектам могло бы дать нам возможность определять дату поставки, если в этом была бы необходимость. "Белый шум" (небольшая поддержка, т.е. не более одного часа, встречи, кофепитие, помощь коллегам) всегда был неотъемлемой частью всего процесса.

27

Как оценивать?

Это вечная тема для обсуждения, и здесь, конечно, нет однозначного ответа:

- Оценивайте регулярно
- Оценивайте, когда это необходимо
- Используйте идеальные дни / story points
- Точных оценок все равно не бывает, так что используйте относительные оценки (например, размеры футболок – S, M, L)
- Избегайте оценок, или оценивайте только тогда, когда стоимость задержки оправдывает это

Под влиянием Scrum-а (все-таки именно с него начинали) мы решили начать со story point-ов. Но на практике команды относились к story point-ам, как к человеко-часам (что было для них более естественно). Сначала мы оценивали все истории. Со временем, менеджеры поняли, что если количество одновременных проектов в работе невысокое, то мы не заставляем заказчиков ждать выполнения их запросов. Кроме того, стало ясно, что в случае резкого изменения ситуации всегда можно сместить приоритеты и решить новую проблему.

Дата окончания проекта перестала быть такой важной, как раньше. Это привело к тому, что менеджеры перестали просить предварительно оценить задачи. Они просили сделать оценки только тогда, когда боялись, что заказчикам придется ждать.

Когда-то один менеджер под давлением пообещал сдать проект "к концу этой недели". Так как на проекте применяли Kanban-доску, было легко оценить прогресс (количество завершенных историй) и понять, что только около 25%, скорее всего, будет выполнено после первой недели. Следовательно, понадобились бы три дополнительные недели. Столкнувшись с этим фактом, менеджер изменил приоритет проекта, остановил параллельные работы, и сделал возможной сдачу проекта в срок. Всегда сверяйтесь с доской ☺

Что означает оцениваемый объем? Время выполнения с учетом задержек или время работы?

Наши story points отражали время работы, т.е. сколько часов непрерывной работы мы ожидали потратить на эту историю; это не общее время выполнения с учетом задержек (не календарное время и не количество часов ожидания). Измеряя каждую неделю количество story point-ов завершенных задач (производительность), мы смогли просчитать время выполнения с учетом задержек.

Мы оценивали каждую новую историю только один раз, мы не пересматривали оценку истории во время выполнения. Это позволило свести к минимуму время, которое команда тратила на оценку.

28

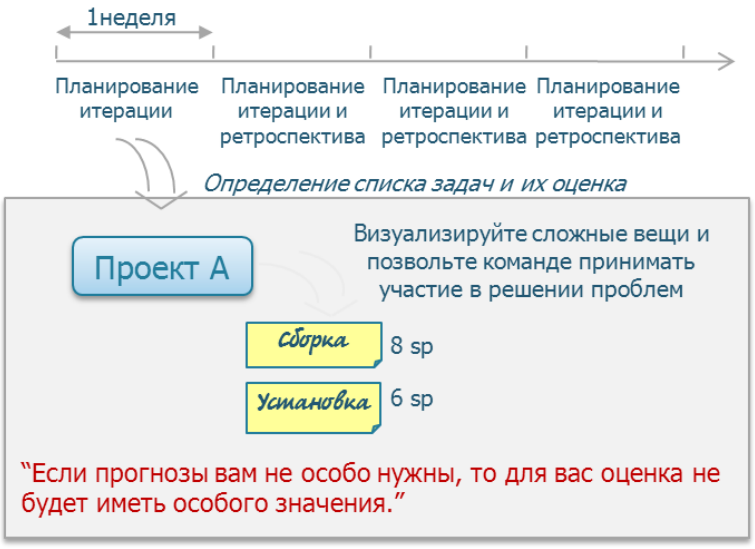
Так как же мы работали на самом деле?

Kanban действительно может быть очень ненавязчивым, вы можете работать, используя самые различные способы. Вы можете разрешить команде использовать для своих церемоний регулярное расписание, а можете провести ту или иную встречу в тот момент, когда это наиболее оправдано.

Рисунок 7. Когда три новых задачи поступают в backlog, это инициирует сессию планирования/оценки.

Мы решили запланировать две регулярные встречи:

- Ежедневный Standup – перед доской для того, чтобы выявить проблемы и помочь в формировании "общего понимания" того, что делают другие члены команды.
- Еженедельное планирование итерации, для планирования и постоянного улучшения процесса.



Нам было этого достаточно.

Ежедневный Standup

Ежедневные Standup встречи были похожи на ежедневный Scrum. Они проходили после ежедневной встречи "Scrum of Scrums" с представителями всех команд (разработка, тестирование, поддержка). Встреча "Scrum of Scrums" давала важную информацию Kanban-группам, например, какие проблемы должны решаться в первую очередь, или какая из команд разработчиков на текущий момент оказалась в самой трудной ситуации. Сначала менеджеры часто посещали эти ежедневные встречи, чтобы предложить решения и определить приоритеты. Со временем, когда команды стали более самоорганизованными, менеджеры присутствовали всё реже (и находились недалеко на случай необходимости).

Планирование итераций

Раз в неделю мы проводили планирование итерации. Мы сохранили его еженедельным в определенное время, потому что выяснили, что если не запланируем его, то это время понадобится для других дел ☺. И нам нужно было больше командного общения. Обычно на повестке дня было:

- Обновление графиков и доски.(Завершенные проекты передвигались в "Готово".)
- Обзор прошлой недели. Что именно произошло? Почему так? Что можно сделать, чтобы улучшить ситуацию?
- Изменение ограничения незавершенной работы (если необходимо).
- Разбивка задач и оценка нового проекта (если в этом есть необходимость).

В общем, планирование итерации объединяло в себе одновременно и процесс оценки и непрерывного улучшения. Небольшие и средние проблемы решались на месте при непосредственной помощи руководителей. Но решение сложных вопросов, вроде инфраструктурных, было трудным испытанием. Чтобы справиться с этой проблемой, мы дали возможность командам назначать своим менеджерам задачи по устранению "командных препятствий" (до двух штук).

Правила были такие:

1. Менеджер может одновременно решать не более двух проблем.

2. Если менеджер уже занят решением двух проблем, то добавить ему новую проблему, можно, только предварительно убрав менее важную.

3. Только команда может принять решение о том, устранено «препятствие» или нет.

Это было позитивным изменением. Внезапно, команды смогли увидеть менеджеров, помогающих им, и работающих над решением критических вопросов. Они могли указать на препятствие и спросить "Как продвигается решение вопроса?" О препятствиях больше не забывали и не заменяли чем-то новым, с более высоким приоритетом.

Одним из примеров серьезного препятствия было то, что команда поддержки не получала необходимой помощи со стороны разработчиков, когда подозревала дефект. Им нужна была помощь, чтобы понять, в какой части системы возникла проблема, но, поскольку в течение спринта разработчики были заняты разработкой нового функционала, проблемы накапливались. Не удивительно, что команда поддержки считала, что разработчикам безразлично качество.

Когда это препятствие обнаружилось, это было объяснено сначала непосредственному руководителю, а потом и руководителю отдела разработки. Руководитель поддержки запланировал общую встречу вместе с главой разработки. В ходе обсуждения руководители согласились, что качество должно быть превыше всего и предложили решение с циклической помощью команде поддержки – каждый спринт одна команда разработчиков будет "дежурной" и будет мгновенно доступна, когда поддержке понадобиться помощь. Добившись одобрения от своих руководителей, начальник отдела разработки передал список контактных лиц команде поддержки. Сразу же после этого эту идею решили протестировать, подозревая, что план не будет работать. Но "домашнее задание" было сделано в этот раз и препятствие было устранено. Что сильно облегчило жизнь команде поддержки.

29

Поиск концепции планирования, которая работает

Предыстория

Как сейчас помню поворотный момент для одной команды. Мы сидели на их второй встрече по оценке задач. Команда застряла с проектом, который они не знали, как оценить. В проекте было слишком много непонятного, поэтому весь процесс оценки застопорился. Вместо того чтобы предложить им решение и взять ответственность на себя, я попросил их самих подправить процесс оценивания. Со своим менеджером во главе команда приняла этот вызов и начала придумывать свой собственный процесс планирования. Это событие стало поворотным пунктом, важной победой, с которой началось их превращение в уверенную в себе команду. После этого команда начала развиваться так быстро, что нам оставалось лишь отойти с дороги, чтобы не мешать.

Два месяца спустя их менеджер подошел ко мне после ретроспективы. "У меня проблема" сказал он, показывая на Kanban-доску своей команды. "У нас не осталось настоящих проблем. Что нам с этим делать?"

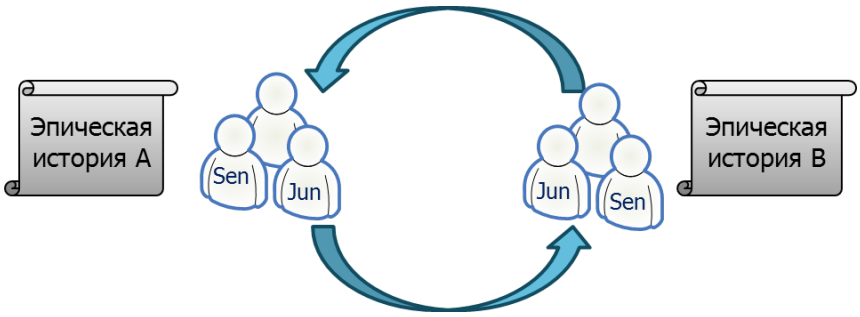
Планируем по-новому

Оценивание задач с помощью planning poker не работало ни для одной из команд техподдержки, в частности, вот по каким причинам:

1. Знания в команде были распределены слишком неравномерно.
2. Чаще всего говорил только один человек.
3. Члены группы спешили заняться другими срочными делами.

Эмпирическим путем команды пришли к двум различным процедурам оценивания, каждая из которых хорошо работала для соответствующей команды.

Подход 1: Ротация и взаимная проверка



- Для каждой истории/проекта назначается пара программистов "ведущий + начинающий» (то есть один человек, который хорошо знает эту историю, и один, который с ней не знаком), которая даёт ей оценку. Это способствует распространению знаний.

- Остальные члены команды выбирают историю, в оценке которой они хотят помочь (но не больше четырех человек на историю, чтобы обсуждение оставалось эффективным).

- Каждая "команда оценки" разбивает свою историю на задачи и, если нужно, оценивает их.

- Затем команды меняются историями и проверяют работу друг друга (один человек из команды остается возле истории, чтобы объяснить "рецензентам" позицию своей команды).

- Конец.

Обычно все планирование итерации занимало около 45 минут, и при этом никто не засыпал от скуки. Как правило, в ходе пересмотров в истории вносилось одно-два уточнения.

Подход 2 – сначала архитектурные решения, потом оценка

Два ведущих программиста просматривали истории/проект до начала планирования. Они анализировали возможные архитектурные решения и выбирали наиболее подходящее. После этого команда начинала разбивать историю на задачи, используя предложенное решение в качестве отправной точки.

Рисунок 8. Разбитие истории на задачи и "рецензирование" в ходе планирования итерации.

30

Что же мерять?

Померять можно много чего – время ожидания (время с момента выявления потребности до её удовлетворения), производительность, длины очередей, отклонения от плана работ... Важнее понимать, какие метрики *окажутся полезными* для совершенствования процесса. Мой совет: экспериментируйте и смотрите, что подходит именно вам. Мы, например, обнаружили, что burndown-диаграммы излишни для любых проектов короче четырёх недель. При этом прогресс всё равно можно отслеживать, просто взглянув на Kanban-доску (сколько историй было в backlog-е, сколько уже завершено).

Метрика	**За**	**Против**
Время ожидания	Легко измерить. Не требуется оценка. Начинается с заказчика и заканчивается заказчиком.	Не учитывается размер функциональности.
Общая производительность (в совокупности по всем типам работ)	Грубый, но легко отслеживаемый показатель, чтобы понять, что требует улучшения и изменения.	Не в состоянии предсказывать сроки поставки по конкретным видам работ.
Производительность по типам работ	Более точный показатель, чем общая производительность.	Чтобы быть полезной, должна учитывать всё от сбора требований клиента до поставки готовой функциональности. Требует больше усилий для отслеживания, по сравнению с общей производительностью.
Длины очередей	Оперативный показатель неравномерности потока задач. Легко визуализируется.	Не отвечает на вопрос, что является причиной – неравномерные запросы или неравномерная производительность. Очередь нулевой длины может фактически указывать на переизбыток ресурсов в команде.

Мы начали с отслеживания "производительности по типам работ" (Velocity per work type) и "длин очередей" (Queues lengths). Производительность по типам работ легко отслеживать, а своё дело эта метрика делает исправно. Что до длины очередей, то это отличный оперативный показатель, поскольку виден в любой момент времени (если знать, куда смотреть).

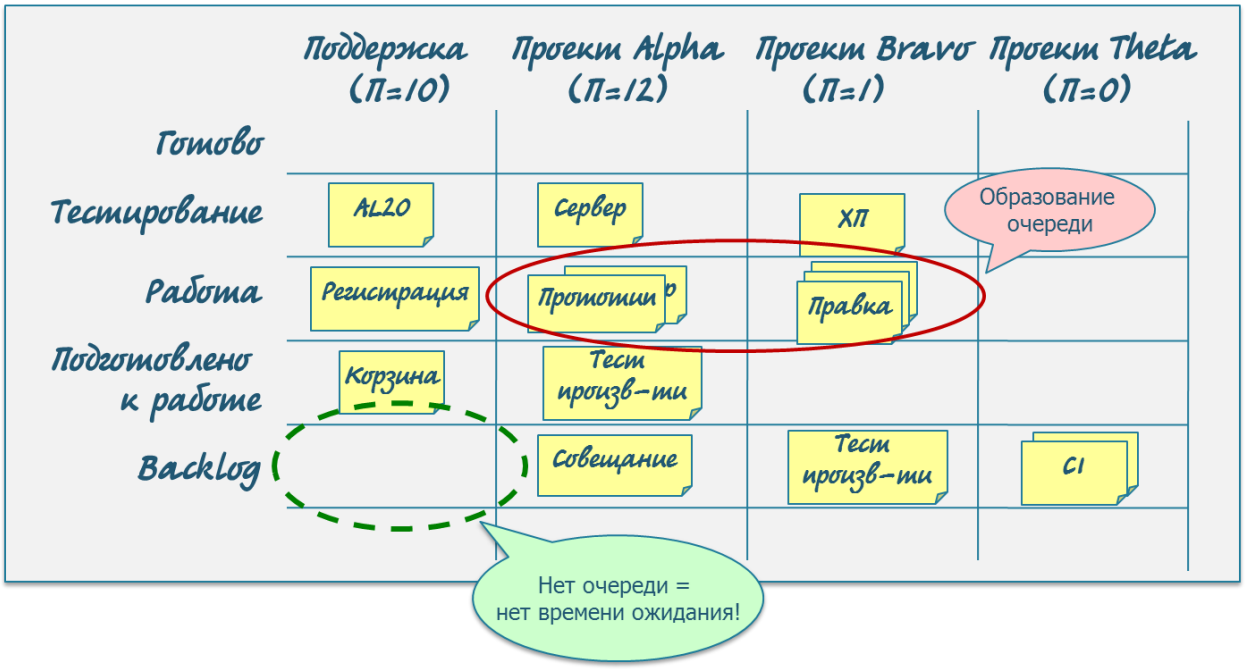
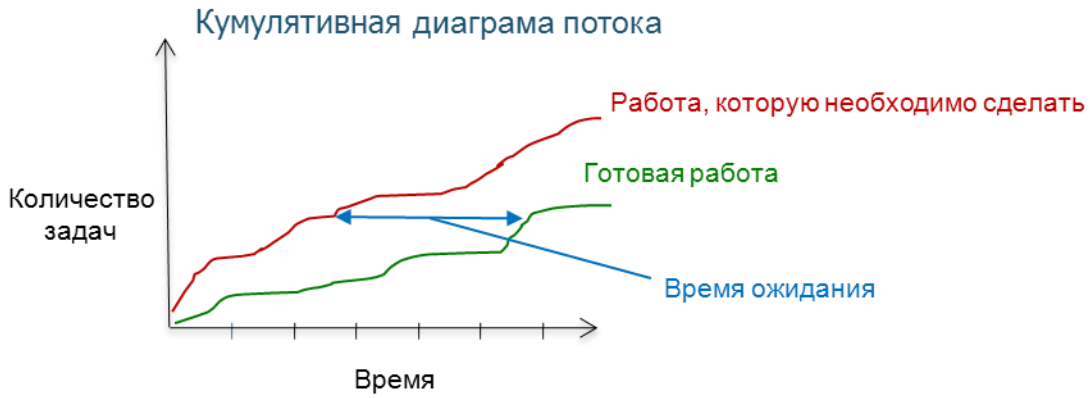


Рисунок 9. Узкие места и потенциальные возможности. Красная область показывает, как очереди собираются перед узким местом, в данном случае – тестированием. *Отсутствие* очередей в колонке "поддержка" свидетельствует о том, что работы по поддержке поступают на обработку сразу же. Это хорошо для обеспечения высокого уровня обслуживания.

Мы не использовали кумулятивную диаграмму потока, хотя она могла бы быть интересной.



Мы не использовали кумулятивные диаграммы потока, т.к. Kanban-доска и график производительности предоставляли нам достаточно информации, по крайней мере, на ранних стадиях становления процесса. Эти инструменты легко позволяли выявлять узкие места, неравномерности потока и переработки, и первые шесть месяцев мы были заняты решением этих проблем.

31

Как все начало меняться

Через три месяца после введения Kanban-а команда системного администрирования завоевала приз "Самая эффективная команда" в ИТ-департаменте. В то же время, команда сисадминов стала одним из трех "положительных пунктов", которые были отмечены на ретроспективе компании. Ретроспектива компании – это событие в рамках всей компании, проходящее каждые 6 недель, и это был первый случай, когда *команда* попала в тройку самых позитивных событий! А всего 3 месяца назад эта команда была узким местом, на которое большинство людей жаловалось.

Качество обслуживания явно выросло. Как же так получилось?

Перелом наступил, когда все стали "тянуть в одну сторону". Менеджеры четко сформулировали, на чем нужно сосредоточиться, и оберегали команды от посторонней работы, а команды взяли на себя ответственность за качество и сроки выполнения. На то, чтобы процесс устоялся, ушло три-четыре месяца, но после этого все пошло как по маслу. Не то, чтобы решились все проблемы в мире (это бы оставило нас без работы, правда? ☺) Но трудности приобрели иной характер, например "как сохранить мотивацию команды к росту и развитию, когда они перестали быть узким местом?".

Одной из важных деталей, составляющих самоорганизацию, стало введение концепции "один контакт из техподдержки на команду". Другими словам, у каждой команды разработки появился "свой человек" в отделе техподдержки. Это стало возможным благодаря Kanban-у, который позволил членам команд техподдержки самоорганизовываться вокруг задач, предотвращая переработки и способствуя постоянному совершенствованию. До этого случайный человек взял бы задание из очереди, выполнил бы его в меру своих возможностей, а затем взял бы следующее. Любое недопонимание в ходе выполнения задачи означало повторение всего цикла, начиная с нового запроса на техподдержку (который мог попасть другому человеку). Когда же была внедрена концепция "один-к-одному", у команды техподдержки вдруг появилась возможность быстро реагировать на ситуации, угрожающие работе системы, например на плохие входящие данные или на проблемы с качеством.

Забавно – через некоторое время между разработкой и техподдержкой появились настоящие "коммуникационные протоколы": со знакомыми программистами сотрудники техподдержки общались по интернет-пейджерам; с теми, кто пишет лучше, чем говорит, они переписывались по электронной почте; и наконец, если нужно было срочно решить проблему, они звонили по телефону ☺.

Раньше

Рисунок 10. Раньше: Менеджер первой линии техподдержки является основным контактным лицом для команды. Все важные задачи проходят через него. Более мелкие задачи (как правило, проблемы отдельных программистов) берутся из системы управления задачами. Программисты и техподдержка мало общаются лично.

Сейчас

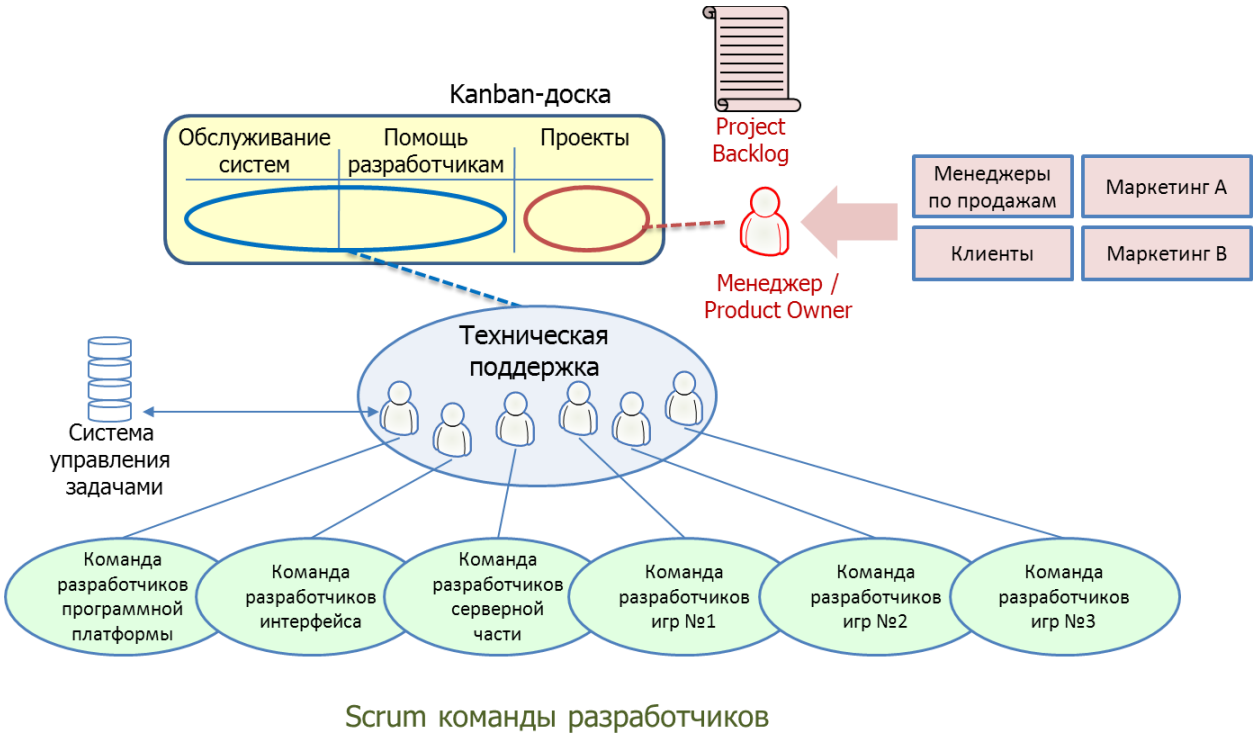


Рисунок 11. Сейчас: одно контактное лицо из техподдержки на команду Команды разработчиков напрямую общаются с определенным человеком в техподдержке. Много личного общения. Члены команды техподдержки самостоятельно организуют свою работу, используя kanban-доску. Менеджер направляет усилия на приоритезацию больших проектов и решение возникающих проблем.

А как насчет влияния всего этого на производительность команды?

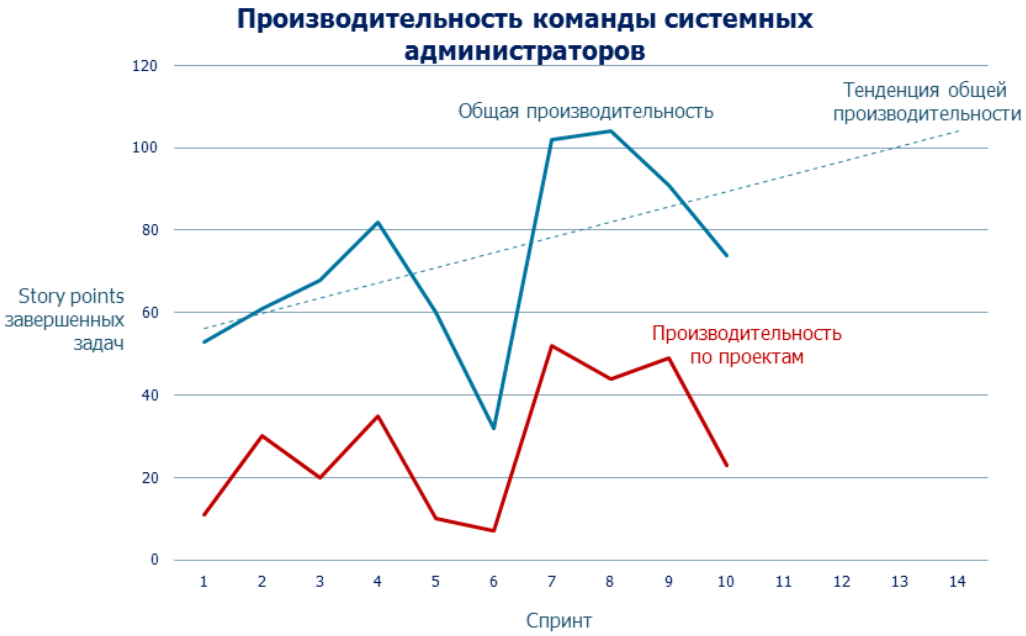


Рисунок 12. Общая производительность и производительность по проектам, подсчитанная как количество Story Points, завершенных за неделю. Общая производительность – это сумма по всем столбцам [kanban-доски]. Производительность по проектам представляет собой часть, посвященную "проектам" (большим кускам работы, например, обновлению аппаратной платформы). Два спада связаны с 1) неделей, когда почти все члены команды разъехались по отпускам, и 2) выпуском новой версии от команды разработки.

Как видим, у команды есть тенденция к росту производительности. При этом группа много времени и сил потратила на обмен знаниями (используя для этого парное программирование).

Давайте заодно посмотрим на производительность команды администраторов БД (DBA):

Рисунок 13. Общая производительность и маленькие задачи, которые не включаются в общую производительность. Спад в середине графика связан с Рождеством.

Общая производительность имеет тенденцию к росту, однако её колебания достаточно значительны. Настолько большие отклонения подтолкнули команду к отслеживанию количества небольших технических задач (слишком мелких, чтобы вешать их на Kanban-доску). Как видно на графике, между количеством таких задач и общей производительностью есть четкая обратная зависимость.

Команда техподдержки начала работать по Kanban позже двух других команды, и по ней у нас пока что недостаточно достоверных данных.

Взросление

Когда мы только начинали, было очень легко определить проблемные области. А вот что было сложно, так это определить самую большую возможность для улучшения. Kanban-доска дала нам совершенно новый уровень прозрачности. Мало того, что стало легче определить проблемы, но также всплыли важные вопросы, касающиеся потока работ, отклонений и очередей. Мы начали использовать очереди как инструмент обнаружения проблем. Через четыре месяца после внедрения Kanban-а менеджеры устроили настоящую охоту на источники отклонений, которые мешали их командам.

В процессе превращения команд из кучек специалистов в самоуправляемые боевые подразделения менеджеры осознали, что теперь они сталкиваются с совсем другим набором управленческих проблем. Им пришлось больше заниматься "человеческим фактором": разбирать жалобы, находить общие цели, разрешать конфликты и проводить переговоры. Переход не был безболезненным – менеджеры откровенно отмечали, что изучение всего этого требовало усилий и квалификации. Но они все-таки приняли этот вызов, и в результате улучшили свои лидерские навыки.

32

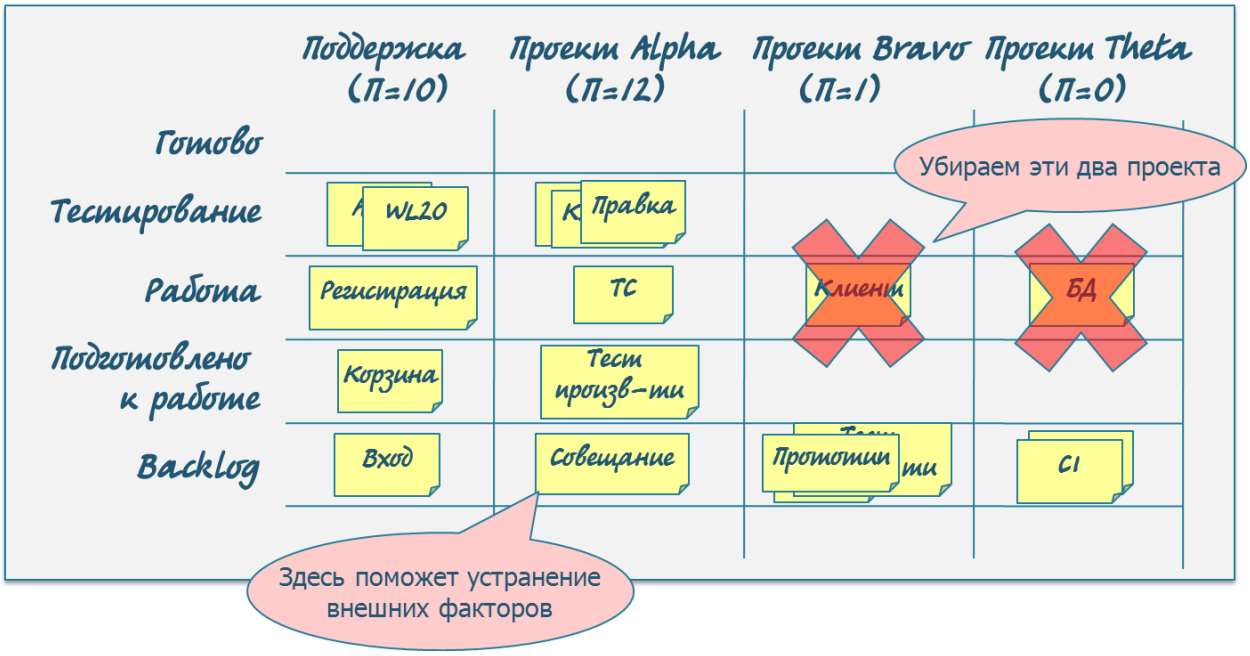
Усвоенные уроки

С уменьшением НЗР появляются ограничения

Все команды начали с достаточно больших ограничений НЗР. На тот момент большая часть усилий шла на создание и стабилизацию потока ценностей, а также на то, чтобы организация получала всю необходимую техническую поддержку.

Сначала менеджеры хотели, чтобы параллельно выполнялось несколько проектов, но через несколько недель стало очевидно, что для менее приоритетных проектов не хватает ресурсов. Одного взгляда на доску было достаточно, чтобы убедиться: ни одна из задач с низким приоритетом так и не была начата. Это подтолкнуло менеджеров к сокращению количества проектов на команду.

Со временем, когда поток для высокоприоритетных задач более-менее стабилизировался, мы начали ужесточать ограничения НЗР. Это было сделано за счет сокращения числа текущих проектов (столбцов) с трех до двух, а потом и до одного. Когда это произошло, стали всплывать ограничения, которые находились вне команды. Члены команды начали жаловаться, что вовремя не получают помощи от других команд, поэтому менеджеры обратили свое внимание на решение этой проблемы.



Кроме того, стало видно, как некачественное части, выполненные другими командами, влияли на производительность этой команды. Трудно было поддерживать быстрый и стабильный поток, когда результаты работы других команд, которые мы получали, постоянно нуждались в исправлениях.

Это все было понятно и до начала нашей работы по Kanban. А вот что было непонятно, так это с чего начать, и как об этом договориться со всеми заинтересованными. Когда появилась Kanban-доска,

каждый смог увидеть, как конкретная проблема влияет на поток. Это помогло скоординировать усилия по её решению между разными отделами организации.

Доска по ходу работы будет меняться, поэтому не отливайте её в бронзе!

Все наши Kanban-доски изменялись по ходу работы. Как правило, внешний вид доски менялся два-три раза, пока команды не находили удобный для себя формат. Поэтому тратить много времени на первый дизайн доски, скорее всего, не имеет смысла. Убедитесь только, что вы можете легко менять её внешний вид. Мы, например, обозначали столбцы и строки черной изолентой. Изолента легко переклеивалась, к тому же её можно было клеить и на доску, и на стену. Ещё я видел, что линии рисовали толстым маркером (только убедитесь, что он стирается :-).

Ниже показан типичный пример оптимизации доски. В начале проекта приоритеты часто менялись, поэтому, чтобы не двигать вперед-назад целый столбец с листиками, команда клеила приоритеты над каждой колонкой.

Рисунок 14. Ранняя Kanban-доска с наклейками для текущих приоритетов

Не бойтесь экспериментировать и ошибаться

Урок, который я извлек из этого приключения, состоит в том, что конца-то у него и нет. Как только мы решим, что закончили внедрение Kanban, мы автоматически совершим большую ошибку. Это путь постоянных экспериментов и непрекращающегося обучения. Но невозможно учиться, не делая ошибок. По ходу работы у нас было достаточно ошибок (плохой дизайн доски, оценивание задач, никому не нужный burndown ..) – но каждый раз мы учились чему-то новому и важному. Если бы мы перестали пробовать, как бы мы тогда могли научиться?

Успехи Kanban вдохновляют менеджеров и разработчиков, работающих по Scrum, начать экспериментировать с Kanban-досками. Надеюсь, эта книга им поможет!

Заключительное слово

Начните с ретроспектив!

При внедрении Kanban-а приходится осмысливать кучу вещей и мнений, да? Надеюсь, что эта книга хоть немного помогла развеять туман. Ну, нам, по крайней мере, помогла :o)

Если вы намерены изменять и совершенствовать свой процесс, позвольте нам принять за вас одно решение прямо сейчас. Если вы не проводите ретроспективы на регулярной основе, то срочно начинайте! И добейтесь того, чтобы они приводили к реальным изменениям. Наймите фасилитатора со стороны, если надо.

Только после того как вы начнёте проводить эффективные ретроспективы, вы сможете начать свой путь по созданию процесса, который лучше всего подходит для вас – будь он основан на Scrum-е, XP, Kanban-е, их сочетании, или на чём угодно ещё.

И никогда не прекращайте экспериментировать!

Ни Kanban, ни Scrum не являются целью, а вот непрерывное обучение (в том числе на ошибках) – да. Одним из замечательных аспектов разработки программного обеспечения является короткий цикл обратной связи, что и есть ключом к обучению. Так используйте его! Задавайте вопросы, экспериментируйте, терпите неудачи, извлекайте уроки, и снова экспериментируйте. Не пытайтесь всё сделать правильно с самого начала – всё равно не получится! Просто начните с чего-то и развивайтесь.

Единственная *реальная* неудача – это неспособность извлечь из неудачи урок.

Хотя нет! Ведь из этого тоже можно извлечь урок…

Удачи и приятного пути!

/ Хенрик и Маттиас, Стокгольм 2009-06-24

Н: Ну что – это и всё?

М: Думаю, да. Давай на этом закончим.

Н: Может, стоит рассказать читателям, кто мы такие?

М: Отличная идея. Если удастся выставить нас хорошими парнями, то можно получить от них заказы на консультации.

Н: И это будет справедливо. Так и сделаем!

М: Ну ладно, у нас есть ещё работа, да и у читателей тоже.

Н: У кого как. Я, например, прямо сейчас ухожу в отпуск :о)

М: Эй, не сыпь мне соль на рану!

Об авторах

Хенрик Книберг (Henrik Kniberg) и Маттиас Скарин (Mattias Skarin) являются консультантами в Crisp (Стокгольм). Они с удовольствием помогают компаниям добиться успеха в техническом направлении и во взаимоотношениях людей, разрабатывающих программное обеспечение, помогли десяткам компаний с использованием принципов Lean и Agile на практике.

Хенрик Книберг

За последнее десятилетие Хенрик был техническим директором в 3-х шведских ИТ компаниях и помог многим другим улучшить свой процесс. Он является Certified Scrum Trainer-ом и работает на регулярной основе с пионерами Lean и Agile, такими как Джефф Сазерленд (Jeff Sutherland), Мэри Поппендрик (Mary Poppendieck), и Дэвид Андерсон (David Anderson).

Предыдущая книга Хенрика "Scrum и XP: заметки с передовой" ("Scrum and XP from the Trenches") имеет более чем 150 000 читателей и является одной из самых популярных книг по этой теме. На международных конференциях неоднократно получал награду "за лучший доклад".

Хенрик вырос в Токио, а сейчас живет в Стокгольме с женой Софией и тремя детьми. В свободное время он активно музицирует, сочиняя музыку и играя на бас-гитаре и клавишных с местными группами.

henrik.kniberg <at> crisp.se

http://blog.crisp.se/henrikkniberg

http://www.crisp.se/henrik.kniberg

Маттиас Скарин

Маттиас работает Lean-тренером, помогая компаниям, разрабатывающим программное обеспечение, пользоваться преимуществами Lean и Agile. Он наставляет работников всех уровней, от разработчиков до управленцев. Он помог компании разработчиков игр сократить время разработки игры с 24 до 4 месяцев, восстановил доверие ко всему отделу разработки и был одним из пионеров Kanban.

Как предприниматель, он основал и руководит двумя компаниями.

Маттиас имеет степень магистра в области управления качеством, и работал разработчиком в течение 10 лет с критически важными системами.

Он живет в Стокгольме, танцует рок-н-ролл, увлекается автогонками и лыжами.

mattias.skarin <at> crisp.se

http://blog.crisp.se/mattiasskarin

http://www.crisp.se/mattias.skarin