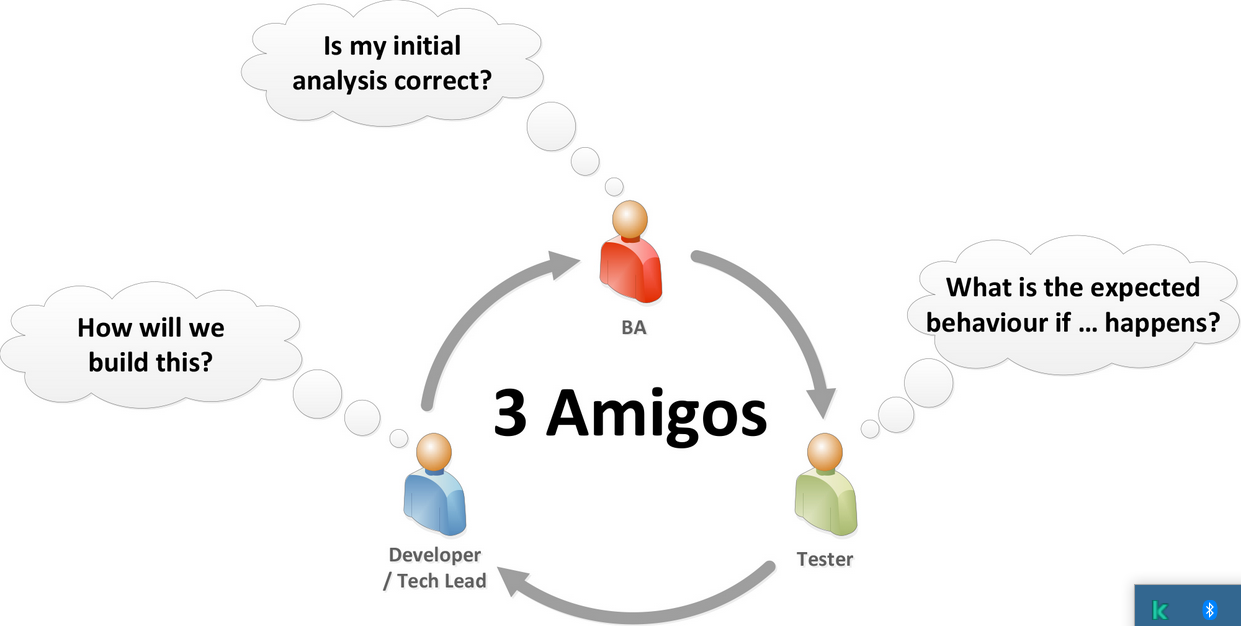
Three\_amigos

<https://habr.com/ru/company/tinkoff/blog/449424/>



3 Амиго — это практика, которая дает всеобщее понимание того, что будет доставлено клиенту. Помогает доносить голос команды, а не переплетение разрозненных мнений.

Данный способ коммуникации внутри команды помогает:

* прийти к общему соглашению относительно ожиданий до начала разработки
* сформировать соглашение о том, как сделать правильную вещь сразу

А также, способствует пониманию:

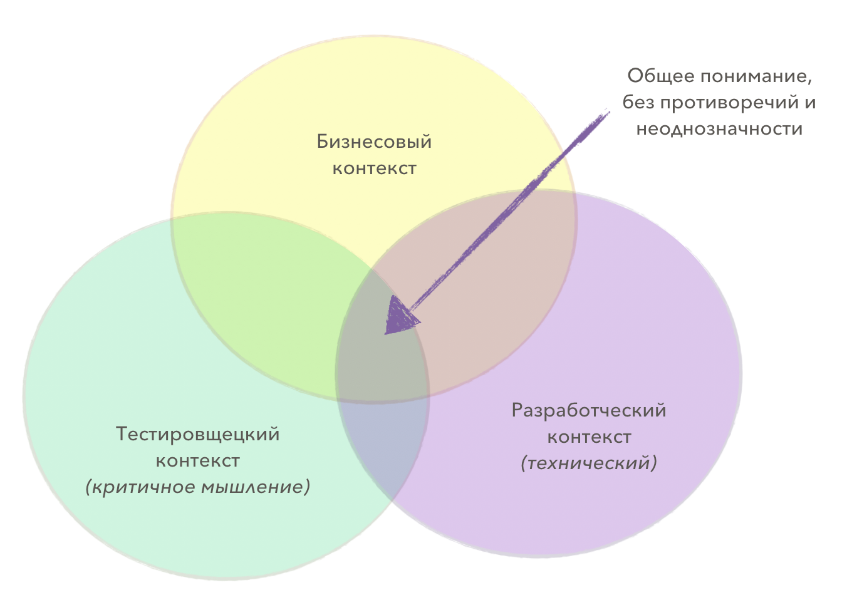
* какую проблему мы решаем
* какие есть варианты решения
* что нам нужно сделать, чтобы задача была готова

Встреча трёх амиго — это способ совместного мышления, которое устраняет пробелы в понимании бизнес-спецификаций. Она помогает в разработке крутых пользовательских историй.

Цель заключается в том, чтобы делать работу в срок, но при этом планировать не настолько заранее, чтобы детали успевали устаревать.

**Почему 3? Кем применяется?**

Название практики не ограничивает нас тремя контекстами, а лишь создаёт минимальные рамки. Чтобы убедиться в том, что в проработке требований учли все технические нюансы, явные и неявные случаи, что спецификация отражает действительную нужду клиента — требуется 3 различных мышления/контекста: бизнеса, разработчика, тестировщика. При этом встреча не ограничивается только этими людьми. В ней участвуют все, кто вовлечены в реализацию требования. Например, если в вашей задаче есть не только web, но и mobile, то нужен дополнительный контекст. То есть тот человек, который будет делать реализацию в мобильном приложении. В наших командах чаще всего во встрече участвуют 4 разработчика (1 ios, 1 android, 1 back, 1 front), тестировщик и бизнес-аналитик.



**Бизнес-аналитик или PO**

Представитель бизнеса знает (почти всегда), что он хочет получить в итоге и какое value от этого получит клиент и бизнес. Важно рассказывать об этом команде.  
Участвуя во встрече 3 Амиго, бизнес-аналитик делится информацией с участниками, чтобы у всех в команде было одинаковое понимание и ожидание от пользовательской истории. Также только он может ограничить scope приемочных критериев, по которым потом будет происходить приемка.

**Разработчики**

Разработчик знает, как реализовать требование от бизнеса, какие есть для этого возможности. Как правило, он думает о деталях, которые ему нужно знать, чтобы приступить к реализации. Задавая вопросы, исходя из своего опыта и знания системы, разработчик помогает вскрывать различные нюансы еще на этапе обсуждения требований.

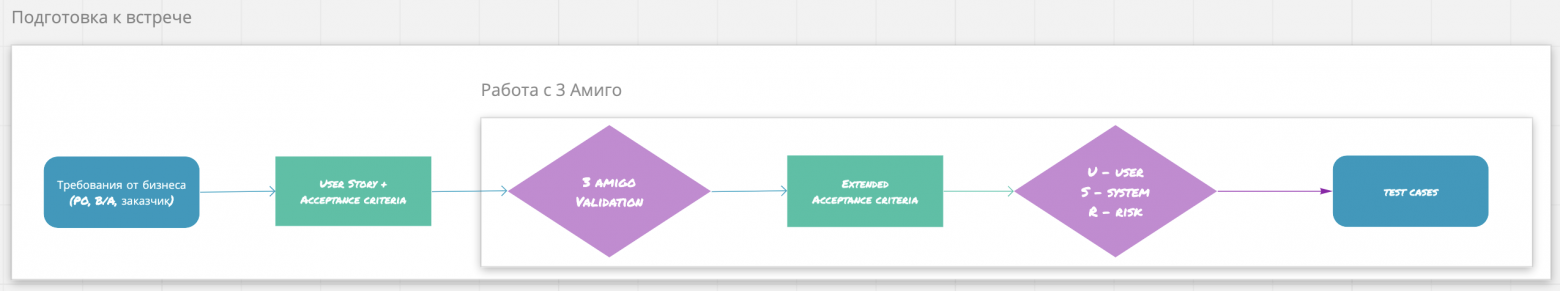
**Тестировщик**

Тестировщик, так же, как и другие члены команды, помогает обогащать требования различными тестовыми случаями. Исходя из своего опыта, он больше и чаще подвергает сомнению любые утверждения, которые озвучивает команда. Поэтому лучше находит крайние случаи, неявные сценарии, задается вопросом, что может пойти не так, чего следует остерегаться.

**Шаги применения практики для проработки AC**

**1. Формулируем требования как User Story**

Я рекомендую применять прорабатывать требования, основываясь на пользовательских историях. И в качестве основы брать одну историю и на встрече 3 амиго прорабатывать только ее.



Здесь очень важный момент в том, чтобы требование от бизнеса действительно было сформулировано как пользовательская история. То есть соддержала в себе 3 части, а именно:

Я как <роль>, хочу <действие>, чтобы <мотивация>

Это на самом деле самый популярный шаблон для формирования пользовательских историй и называется Connextra. Есть также и другие шаблоны, но я рекомендую использовать именно этот. А почему, сейчас объясню.

Традиционно есть 2 проблемы при формировании user story:

* При записи либо не указывают роль, оставляя действие и мотивацию
* Либо же указывют роль и действие, а мотивацию выкидывают.

Это на самом деле влечет проблемы, и я постараюсь на простых примерах объяснить какие.

1. Зная <роль>, вы как члены команды будете лучше понимать границы критериев приемки, которые вам нужно сформировать. Не до конца понимания заинтересованное лицо в этой пользовательской истории, вы можете сделать либо сверх нужного, либо наоборот недоделать какую-то функциональность.
   * **Пример**: команда плохо понимала роль в user story, и при проработке задачи решила, что сделает 3 метода для api: на добавление, редактирование и удаление. А еще на фронте добавят кнопки, которая будет вызывать эти методы. Когда они презентовали свое решение бизнесу — то получили фидбек, что на самом деле их клиент, это конкретный бизнес-аналитик, которому достаточно метода добавления. И удалять и редактировать он не будет. Да и вызвать этот метод он может через postman.
2. Команда не понимает мотивацию "роли" для кого они делают пользовательскую историю. Из-за непонимания плохо очерчиваются границы самой user story. И помимо этого, критерии приемки в итоге могут и не решать конечную клиентскую проблему, хотя будут соответствовать тому действию, что озвучено в user story.
   * **Пример**: команда взяла в работу user story, где бизнес-аналитик не мог чётко сформулировать мотивацию. С одной стороны, она была в том, чтобы оставить клиента лояльным и сделать для него улучшение. С другой стороны — сокращение расходов для банка. В этом случае способы реализации и критерии приемки были кардинально разными. Потому что в первом случае, команда фокусировалась на пользовательских критериях приемки. Во втором — команда обращала внимание на то, как сделать максимально простую и быструю реализацию.

Но формулирование в вышеописанном формате не является обязательным условием. Я знаю команды, которые проводят встречи в формате 3 Амиго и без user story. А в качестве основы используют ТЗ. В этом случае возникают свои подводные камни, но это был осознанный выбор команды.

Встреча 3 Амиго начинается с подготовки. В рамках подготовке к ней формулируются требования, так чтоб они были понятны всей команде. Эти требования валидируются на готовность к работе.

Валидация включает в себя оценку истории по критериям INVEST. А также качество формулировки самой истории. Тут исключается любая неоднозначность, многословность и при оценке по INVEST, особенное внимание обращается на размер истории. Если команда понимает, что требование представляет из себя epic, то происходит декомпозиция.

После того, как требование прошло этап трансформирования в user story и валидацию со стороны команды (3 амиго), можно приступать к проработке acceptance criteria.

**2. Дополняем acceptance criteria к User Story на встрече 3-х Амиго**

Для начала нужно определиться, а чем же все-таки являются критерии приемки.

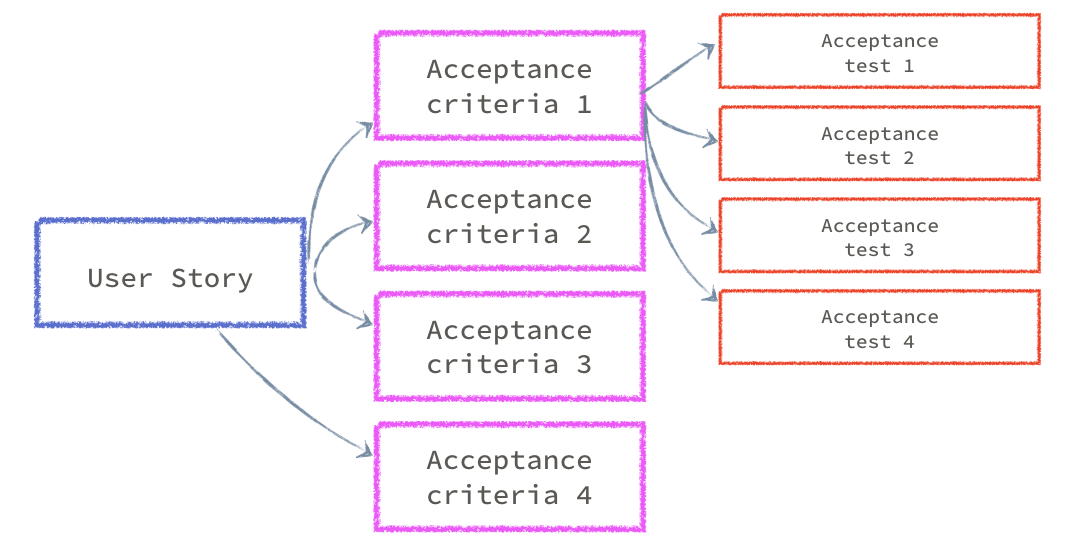
Итак, критерии приёмки (acceptance criteria) — это требования от заказчика; спецификация по которой может быть проверена система/user story.

У них есть альтернативное название, conditions of satisfaction — условия удовлетворения ожиданий (автор Майк Кон).

На встречу 3-х Амиго команда уже приходит подготовленная. У них уже имеется провалидированная user story, которая возможно уже даже обладает каким-то набором критериев приемки, которые представитель бизнеса сформулировал самостоятельно.

Во время встречи, задачи участников дополнить/обогатить историю достаточным количеством критериев приемки для ее последующей реализации.

В критериях приемки не должны находиться детали реализации. Фактически, критерии приемки — это бизнес-правила, которым подчиняется прорабатываемая пользовательская история. Детали же реализации фиксируются в приемочных тестах, но после того, как все критерии приемки были сформулированы.



Смешивать детали реализации с критериями приемки не стоит еще хотя бы потому, что при ограниченных сроках, вы как команда всегда можете "выкинуть" какой-то scope критериев, который в ближайший момент не так важны бизнесу. При наличие деталей с технической реализацией в критериях — вы рискуете потерять важную информацию и время, которое уже было потрачено на обсуждение деталей реализации. Ваши детали реализации напрямую зависят от scope критериев, которые вам надо сделать.

Также избегайте последовательного описания сценария с набором шагов (классическая система ведения тест-кейсов). Любое ваше утверждение должно быть атомарным. Лучше использовать описательный стиль ожидаемого поведения создаваемой функции.

Например:

\*Когда пользователь заходит в карточку сотрудника, то видит пустые поля ввода.\*>

**3. Выдерживаем тайминг**

Хорошо декомпозированная US как правило не содержит в себе больше 10 acceptance criteria. Если в процессе обсуждения обнаруживается большое количество acceptance criteria и все они подлежат реализации, то тогда эту историю необходимо декомпозировать на несколько историй с разным пулом acceptance criteria.

Если этого не сделать, то встреча 3 Амиго будет сильно затягиваться.

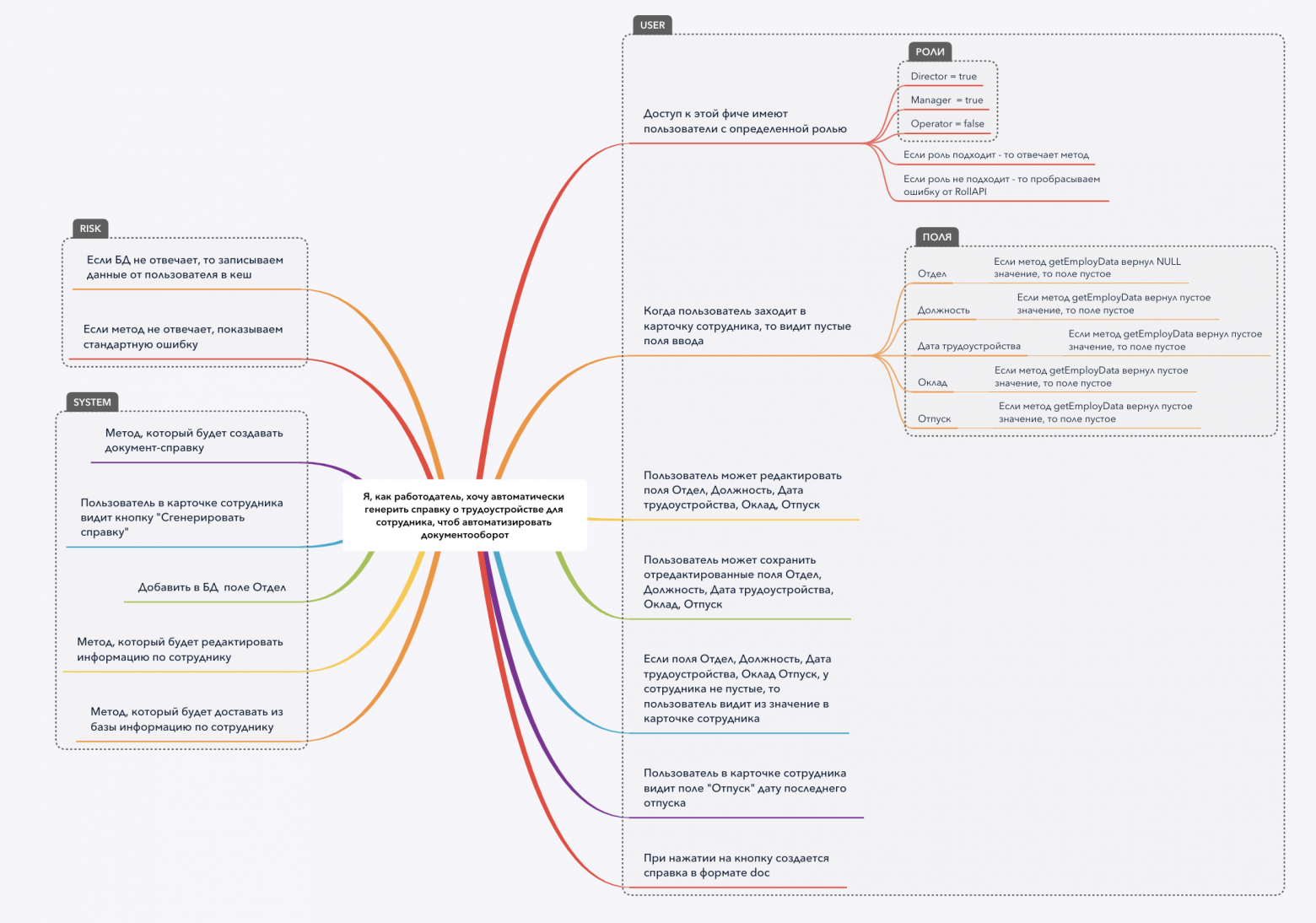
Оптимальное время для проведения встречи 3-х Амиго — от 30 мин до 1 часа.

Допустимо увеличение в самом начале пути — когда команда только учится общаться и применять эту практику.

Если команда берет в работу большую историю, то маловероятно, что за один сеанс они смогут проработать все критерии и приемочные тесты. Потому что теряется фокус и внимательность. В этом случае нужно разбить сессию на несколько встреч. Но в данном случае есть риск потери контекста, и может понадобиться дополнительное погружение при повторной встрече.

**4. Для полноты проработки критериев применяем эвристику USR**

Когда я обучаю команды данной практике, то рекомендую в самом начале их пути пользоваться эвристиками, чтобы сгладить отсутствие опыта. С опытом у команды появляются свои эвристики и чек-листы на что обращать внимание при проработке критериев.



Эвристика USR позволяет рассмотреть критерии на всех уровнях, которые необходимы, чтоб получить максимум информации о том, что хочет заказчик.

Итак,

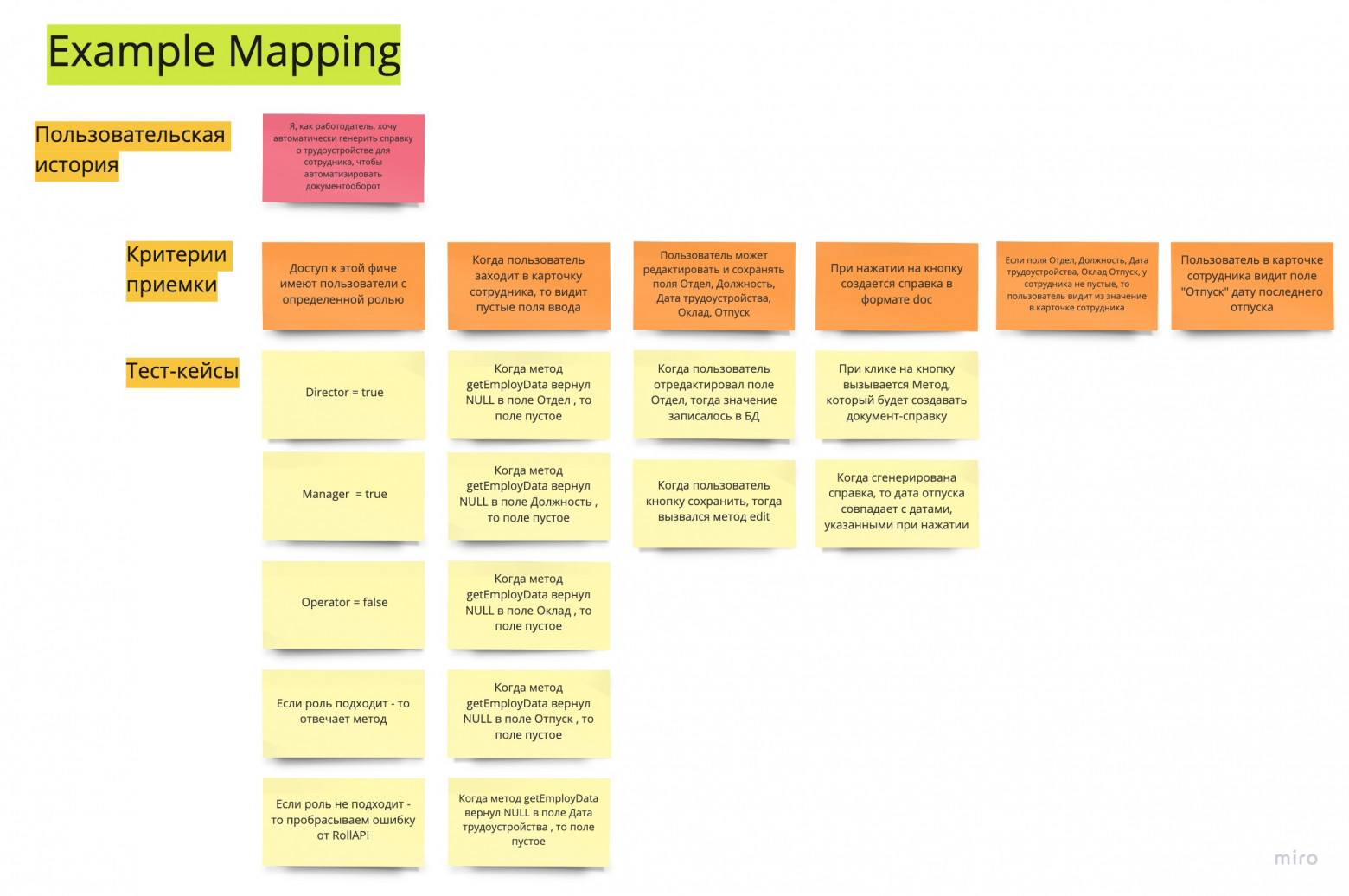
1. **USER** — Что пользователь хочет сделать с системой?
2. **SYSTEM** — Что должна делать система?
3. **RISKS** — Какие проблемы могут возникнуть?

Важно сначала проработать все критерии группы USER, и затем только переходить на уровень системы. Тут включаются разработчики фронта и бэкенда, и на уровне своих систем могут сформулировать критерии приемки к тому, что должно получиться, чтобы реализовать критерии из группы USER.

Ну и наконец группа RISK. Про нее чаще всего забывают про проработке требований, хотя она не менее важная. Тут рассматриваются различные сложные кейсы, которые возможно вы не будете реализовывать. Но проговорить и принять риски как команда — обязаны.

**Способы применения практики для проработки тест-кейсов до разработки**

Рекомендуемый способ формализации требований — примеры использования, в российском IT мы это называем тест-кейсами.  
Есть удобный инструмент проработки тест-кейсов на встрече 3-х Амиго, называется example mapping. В этой статье я вкратце затронула его. В следующей статье мы опубликуем подробную информацию по этому инструменту и как он применяется в рамках встречи триады.



Тест-кейсы могут быть реализованны как автоматизированные приемочные тесты к истории. Также, когда вы прорабатываете тест-кейсы, обязательно группируйте их. Разделение тест-кейсов на группы — это способ поделить историю на несколько более маленьких.  
В тест-кейсах декомпозиция бизнес-правила происходит ровно на том системном уровне, на котором они и будут реализовываться, а не со стороны пользователя.  
Все детали реализации должны находиться в ваших тест-кейсах, чтобы на них происходила ориентация разработчиков в процессе реализации.

**Как это может выглядеть в общем процессе (когда нужно проводить эту встречу)**

Рекомендуется в рамках одной встречи прорабатывать требования только на одну user story. Допускается больше, при условии, это это совсем маленькие истории или они между собой связаны по смыслу.

Так как в спринт вы берете уже готовые к работе истории, то встречу 3 Амиго по истории нужно провести до планирования. Иначе вы рискуете сильно ошибиться с предварительной оценкой. На практике, оценки истории после встречи 3-х Амиго сильно отличаются от изначальных.

Также, имеет смысл добавить в DOD новый пункт как индикатор готовности, что история готова к работе над ней в спринте.

**Definition of Done («Определение готовности», DoD)** — это список условий к процессу и инкременту при выполнении которых элемент бэклога может считаться готовым, сделанным (Done). DoD применяется ко всем элементам бэклога продукта.

Не путать с **Acceptance Criteria (Критериями приёмки, AC)** — списком требований к конкретному PBI(Product Backlog Item). Это условия, которые позволяют понять, реализована история или нет. Причём AC может быть одним из обязательных элементов как DoR так и DoD.

**Definition of Ready(Критерии готовности к взятию в работу, DoR)**— Список условий к элементу бэклога при котором ее можно взять в работу. То есть, если PBI(Product Backlog Item), отвечает критериям DoR, команда может взять её на планировании спринта в работу.

# Зачем нужен DoD

DoD существуют, чтобы обеспечить прозрачность и сфокусировать на прогрессе по каждому артефакту — сказано в Scrum Guide.

Зачем нужен DoD:

* Все участники Одинаково понимали слово — Готово
* Управление качеством инкремента продукта и нефункциональными требованиями
* Взятие под контроль технического долга
* Возможность не снижать, а увеличивать скорость разработки с течением времени

Влияние DoD на скорость проекта с течением времени

# Зачем нужен DoR

* Не допустить начала работы над недостаточно детализированными PBI
* Улучшить результативность проведения Планирования спринта, и уменьшить тайминг мероприятия

# Кто ответственен

* Если нет стандарта организации за DoD ответственна Команда. Команда сама разрабатывает критерии исходя из потребностей своего продукта.
* За DoR ответственна Команда
* За AC – Владелец продукта.

Scrum-мастер или Agile Coach помогает воспитывать культуру технической дисциплины и мастерства.

# Когда DoR и DoD создаются и меняются

Важно понимать – критерии не высечены в камне – они меряются со временем в зависимости от трудностей которое стоят перед командой. Нет каких-то стандартных критериев – каждый критерий – ответ на трудность или проблему.

* Первые договорённости происходят на kickoff проекта.
* Лучшее время для пересмотра DoR и DoD — ретроспектива.

Никогда не перемещайте задачу в колонку «Сделано» преждевременно. Почти сделано, на 99 % сделано – это ещё не Done. Не спешите, даже если продемонстрировать прогресс кажется необходимым.

# Уровни и типы DoD в Scrum

Уровни и типы DoD в Scrum

DoD может быть разный для разных уровней детализации Бэклога. Причём вышестоящие уровни как бы включают все нижестоящие.

На уровне Продукта или Релиза выполняется общее DoD, например:

* Документация
* Тест ковердж >60%
* CI/CD

На уровне Эпика или User Story– это DoD инкремента.

У Бага или Инцидента может быть свой DoD.

# DoD в Kanban

DoD в Kanban

В Kanban каждая колонка Done имеет собственный DoD.

# Хорошие практики для больших компаний:

* Как и везде — введение чего-нибудь дополнительного — реакция на волне конкретный класс проблем. Нет проблемы — не надо придумывать сущности.
* При работе нескольких команд над одним продуктом необходимо использовать единый DoD.
* Включение глобальных нефункциональных требований в DoD.

**Какой результат на выходе встречи 3 Амиго?**

* сценарии/примеры (очевидные ответы)
* уточненные правила/критерии приемки
* новые/декомпозированные истории
* общее понимание проблемной области
* эмпатия (сопереживание, участие)

*Основным результатом трех амиго являются приемочные тесты, написанные в формате «* [*Дано / Когда / Тогда»*](http://jonkruger.com/blog/2011/10/24/know-what-youre-building/)*. На самом деле их написание может занять больше времени чем хотелось бы, поэтому формализовывать все требования на встрече, когда все вместе находятся, не рекомендуется. Обычно разработчик или тестировщик работают над этим вне встречи. И как только у них есть все критерии и тест-кейсы записаны, 3 Амиго кратко просматривают что получилось, чтобы убедиться, что все согласны с тем, что было записано.*

**Профит использования практики 3-х Амиго**

Стратегия 3 Amigos может оказать огромное влияние на эффективность всей команды, а также на качество и поддерживаемость ваших проектов, повышая маневренность, гибкость к изменениям.

Вот ещё несколько бонусов:

* При планировании не тратим много времени на погружение в смысл истории.
* Выявляет путаницу и недопонимание на ранней стадии, что позволяет ускорить доставку
* Гарантирует, что команда, обсудит необходимый объем работы
* Помогает найти все критерии приемки и другие атрибуты качества
* Разрабатывать по тест-кейсам значительно легче
* Провоцируем разработчиков на покрытие кода тестами
* Быстрее приходим к выводу, что эта фича не нужна

Наши команды, применяя практику, смогли прийти к уменьшению возвратов задачи из-за неточной спецификации. Задачи бэкенда ребята научились разрабатывать "с первого раза". То есть без багов и сразу на прод.

**Подводные камни**

* Не ограничивайтесь только тремя людьми. Если нужно больше — зовем.
* Не проводите встречу на всю команду. В этой встрече должно быть минимально необходимое количество людей для реализации фичи.
* Не рассматривайте как регулярное собрание — ритуал. Иначе у команды будет негатив от увеличения количества встреч.