Three_amigos

3 Амиго — это практика, которая дает всеобщее понимание того, что будет доставлено клиенту. Помогает формировать голос команды, а не переплетение разрозненных мнений. Бизнес-аналитики определяют, какие фичи нужно создать. Разработчики создают их. Тестировщики находят дефекты в работе обоих. Встреча трёх амиго — это способ совместного мышления, которое устраняет пробелы в понимании бизнес-спецификаций. Цель заключается в том, чтобы делать работу в срок, но при этом планировать не настолько заранее, чтобы детали успевали устаревать.

Данный способ коммуникации внутри команды помогает:

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

Способствует пониманию:

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

Кем и когда применяется практика

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 задаче есть не только *web, но и mobile,* то нужен дополнительный контекст. То есть тот человек, который будет делать реализацию в мобильном приложении.

Непротестированная спецификация — это с большой вероятностью неоднозначные, противоречивые, неполные, продублированные и иногда даже устаревшие требования. Это происходит потому, что каждый понимает прочтенное и услышанное по-своему. Отсюда и возникает расхождения в интерпретации.

А если документация большая и подробная, то ее чтение может занять больше времени, чем сам процесс тестирования или разработки.

Если задачи стали задерживаться в тестировани, вероятно присутствуют 3 основные проблемы:

1. На этапе тестирования возникало много возвратов из-за уточнения требований. Это были ощутимые паузы в тестировании и разработке, когда бизнес-аналитик определялся с тем, как должно работать.
2. Тестирование задачи задерживалось из-за слишком подробной спецификации. Частыми были кейсы, когда тестировщик мог потратить несколько часов только на изучение требований и проработку тест-кейсов, а само тестирование происходило минут за 30. Либо, к тестированию приступали долгое время спустя после изучения спецификации, настолько она была сложная и подробная.

3. Самая распространенная проблема - в том, что при приемочном тестировании находилось много багов. Тех багов, которых можно было бы избежать, если бы о них подумали до разработки.

Выходит, что несмотря на agile разработку, все-равно присутствует этап работы с технической спецификацией. И если не учли какие-то случаи, потому что о них не сказал заказчик, или же сделали не то, что на самом деле он имел в виду, после выхода на прод, в backlog обнаруживается множество задач по доработке или даже переделке.

Проблема с оценкой:
Сложно оценить задачу, когда ты на самом деле не понятен весь объем работы, который предстоит по ней проделать. Какое количество тестов написать, сколько будет возвратов из-за багов или переделок из-за неточностей спецификации? Даже если разработчик даст точную оценку по самому времени разработки, практически никогда вы не угадать, сколько времени займет непредсказуемый этап тестирования.

РОЛИ

Возьмем пример, когда во встрече участвуют 4 разработчика (1 ios, 1 android, 1 back, 1 front), тестировщик и бизнес-аналитик.

Задачи и вклад в проект каждого из участников команды

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.

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». 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

Применяя эту практику можно прийти к уменьшению возвратов задачи из-за неточной спецификации. Задачи бэкенда получится разрабатывать «с первого раза": без багов и сразу на продакшн.

Подводные камни

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

Смысл и назначение Acceptance criteria, User Story, DoD (Definition of Done)

Шаги применения практики для проработки AC:

AC - Acceptance Criteria (**Критерии приёмки) -** список требований к конкретному PBI(Product Backlog Item), критерий приемки конечным потребителем результата. Условия, которые позволяют понять, реализована история или нет. Причём, AC может быть одним из обязательных элементов как DoR так и DoD.

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.

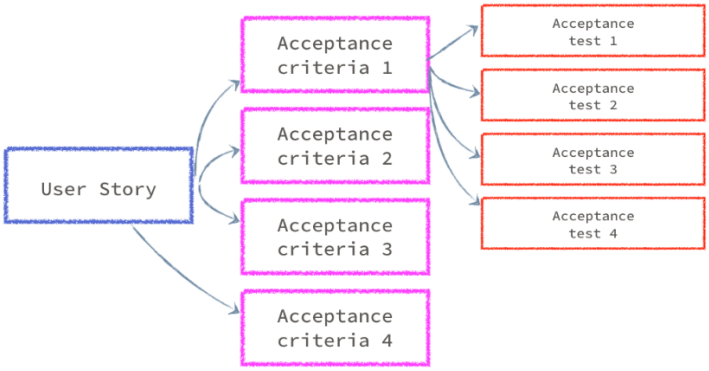
Дополняем acceptance criteria к User Story на встрече 3-х Амиго

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

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 наличии 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, следует избегать 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. *>*

4. Для полноты проработки критериев применяем эвристику USR

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

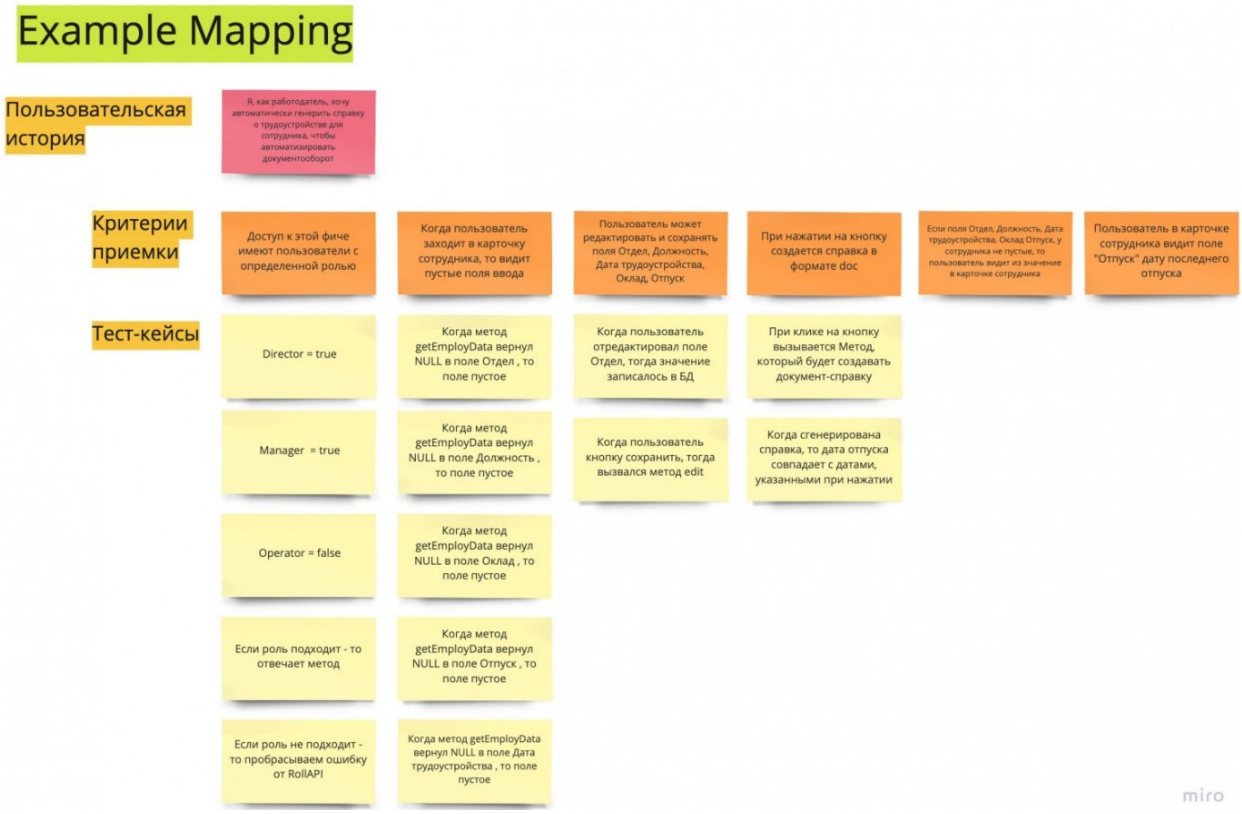
1. **USER** — Что пользователь хочет сделать с системой?
2. **SYSTEM** — Что должна делать система?
3. **RISKS** — Какие проблемы могут возникнуть?

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.

3 амиго: 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.



Тест-кейсы могут быть реализованы 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.

User Story

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.

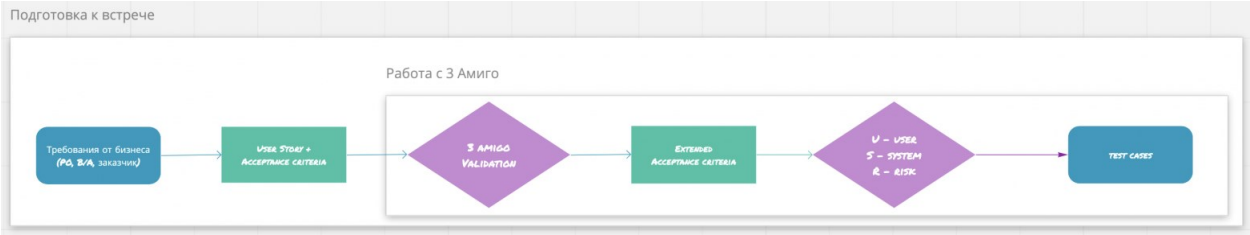
Если команда берет в работу большую историю, то маловероятно, что за один сеанс они смогут проработать все критерии и приемочные тесты. Потому что теряется фокус и внимательность. В этом случае нужно разбить сессию на несколько встреч. Но в данном случае есть риск потери контекста, и может понадобиться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

1. Формулируем требования как User Story

Примеры DoD для User Story**:**

- Юнит тесты пройдены
- Код просмотрен
- Acceptance Criteria выполнены
- Функциональные тесты пройдены
- Выполнены нефункциональные требования
- Владелец Продукта принял 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.

DoD- Definition of Done - («Критерии Готовности») — это когда все условия или Acceptance Criteria, которым должен соответствовать программный продукт, выполнены и готовы к принятию пользователем, клиентом, командой или потребляющей системой. Наиболее распространенное использование DoD — на уровне команды поставки.

Использование Definition of Done снижает количество переделок, предотвращая продвижение пользовательских историй, не соответствующих определению, в среды более высокого уровня. Это предотвращает поставку нововведений, которые не соответствуют определению, клиенту или пользователю.

Зачем нужны Definition of Done

Определение готовности обеспечивает прозрачность, предоставляя всем единое общее понимание того, какая работа была выполнена в рамках Increment

Как писать Definition of Done
Строго говоря, никаких жестких правил нет. Формат Definition of Done может быть любым, но чаще всего это простой список с перечнем активностей, которые должны быть успешно завершены, чтобы функционал мог считаться готовым. Лучше, чтобы DoD выглядели как чек-лист. В таком случае человек либо поставит "галочку" над конкретным условием, либо — нет. Пока не поставит, пользовательскую историю нельзя считать выполненной.

Пример:

- код написан;
- юнит-тесты написаны и успешно выполнены;
- код прошел ревью;
- документация обновлена;
- функциональное тестирование успешно завершено;
- регрессионное тестирование успешно завершено.

Чем больше и объемнее Definition of Done, тем более строгим оно считается. И тем больше нужно времени и усилий, чтобы функционал добрался до желаемого состояния «сделано». И тем выше вероятность, что функционал будет работать в соответствии с ожиданиями бизнеса.