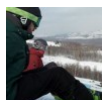


medium.com

Как писать User Story

Alexander Tvar

29-41 минута



User Story — это короткая формулировка намерения, описывающая что-то, что система должна делать для пользователя.

User Story — это не требования.

Несмотря на то, что user story играют в огромной степени роль, ранее принадлежавшую спецификациям требований, сценариям использования и т. п., они все же ощутимо отличаются рядом тонких, но критических нюансов:

- Они не являются детальным описанием требований (то-есть того, что система должна бы делать), а представляют собой скорее обсуждаемое представление намерения (нужно сделать что-то вроде этого)
- Они являются короткими и легко читаемыми, понятными разработчикам, стейкхолдерам и пользователям
- Они представляют собой небольшие инкременты ценной функциональности, которая может быть реализована в рамках нескольких дней или недель
- Они относительно легко поддаются эстимированию, таким образом, усилия, необходимые для реализации, могут быть

быстро определены

- Они не занимают огромных, громоздких документов, а скорее организованы в списки, которые легче упорядочить и переупорядочить по ходу поступления новой информации
- Они не детализированы в самом начале проекта, а уже более детально разрабатываются «точно в срок», избегая таким образом слишком ранней определенности, задержек в разработке, нагромождения требований и чрезмерно ограниченной формулировки решения
- Они требуют минимум или вовсе не требуют сопровождения и могут быть безопасно отменены после имплементации

Текст самой user story должен объяснять роль/действия юзера в системе, его потребность и профит, который юзер получит после того как история случится

К примеру: Как, <роль/персонаж юзера>, я <что-то хочу получить>, <с такой-то целью> .

«Представьте, что вы составляете „пожелание пользователя Amazon.com“. Пробный вариант выглядит так: „Мне как потребителю нужен крупнейший в мире магазин книг, где я могу купить любую книгу в любое время“. Это описание вполне отвечает характеру Amazon, но история получилась слишком расплывчатой, чтобы с ней можно было что-то сделать. Нужно фрагментировать нашу историю. Сделать ее действительно очень конкретной и функциональной.

Приведу несколько образцов пользовательских историй, которые вы можете написать, имея в виду книжный интернет-магазин:

Как потребителю мне удобно искать книги по жанрам, чтобы быстро найти те, которые я люблю читать.

Как потребитель я, отбирая книги для покупки, хочу

класть сразу каждую в корзину.

Как управляющий по выпуску новой продукции я хочу иметь возможность отслеживать покупки наших клиентов, чтобы быть в курсе, какие книги им можно предлагать.

Вот профессионально сделанные пожелания пользователя, характер которых группа должна принять во внимание».

1. Есть один actor
2. Есть одно действие
3. Есть одна ценность / value / impact. Я выделяю ее **ЗАГЛАВНЫМИ БУКВАМИ** при написании User story.

С актером все более-менее просто. Вы выделили персоны, или у вас есть роли, и вы легко их вписываете в начало истории. Есть одна проблема. Убери часть истории про актера. Если история ничего при этом не потеряла — значит эта часть бесполезна.

Вы определили роли в Системе и поняли что их не очень много — Пользователь, Оператор и Админ. И креативите по 100 историй, которые начинаются как “Как Пользователь Я ...”. У вас закрываются несколько спринтов, истории которых начинаются одинаково. Зачем вам это нужно? Да, это бесполезно.

Джеф Паттон предлагает следующее:

1. Разделите всех актеров на группы. Целевая группа, важная группа, менее важная группа и тп.
2. Дайте уникальные названия актерам в этих группах. Даже если в системе у них будет одинаковые роли “Пользователя системы”
3. Пишите истории с точки зрения этих актеров указывая их уникальные названия.

4. В результате вы сможете визуально увидеть какие истории необходимы для актеров целевой группы, какие — для каждой группы и тп. Вы не просто можете использовать это при разборе истории и выстраивания анализа вокруг указанного актера. Вы сможете более правильно выстроить приоритет, так как истории актеров целевой группы для нас более важны.

Наверное здесь сложно ошибиться — это суть истории, “что нужно сделать”. Что можно улучшить. Действие должно быть одно — основное. Нет смысла описывать “авторизуется и выполняется поиск” или “указывает параметры поиска и выполняет поиск”. Укажите то действие, что вам действительно нужно.

Важно описывать историю на уровне “ЧТО?” делает, а не “КАК?”. Это главное в истории. Опишите проблему, а не ее решение. Лучше вы потом с командой это обсудите и найдете более оптимальное “КАК”-решение.

Главная проблема с User Story. Вы всегда знаете первую часть истории, но всегда сложно указать для чего это делается. Но это Scrum, все должно быть указано как User story согласно шаблону, и потому вы пишете “чтобы ...” и какую-то чушь, в которую сами не верите.

Уберите эту часть из истории. Если ничего не потеряли — значит формализация ценности в истории была бесполезна. Что же можно сделать?

Отказаться от формулировки “чтобы”. Это корень зла. ДА, для каких-то историй можно указать ценность истории в таком формате, но не для большинства.

Перейти с понятия ценности (value) на влияние (impact).
Ваша история не обязательна должна иметь ценность, но

обязательно должна оказывать влияние на кого актера, что указан в истории. А уже это влияние ведет в конечном итоге к цели, которая имеет для вас ценность.

Представим что вы создали историю — “Как инвестиционный аналитик я получаю отчет №17 об инвестициях чтобы БЫСТРЕЕ принять решение”.

У меня Acceptance Criteria — это метрика на value в US. Как померить такой value? Как понять что аналитик принял решение быстрее? Как вы поймете в конце что история выполнена?

Переделаем историю на влияние — “Как инвестиционный аналитик я получаю отчет №17 об инвестициях БЫСТРЕЕ”.

То есть сейчас этот отчет формируется за 60 сек. Вы указываете в AC что отчет должен формироваться за 15 сек. В конце понятно выполнено ли AC, понятно какое влияние вы оказали на работу аналитика.

Но в чем ценность того, что аналитик стал получать отчет быстрее?

Здесь можно перейти к общей постановке Цели для продукта. Чтобы прийти к такой истории вы:

1. Вы построили Impact mapping
2. Вы определили Цель и метрику на нее. Например, “ускорение сроков согласования инвестиционных бюджетов”.
3. Вы определили что “инвестиционный аналитик” может вам помочь в достижении этой цели.
4. Вы сделали предположение что если аналитик будет получить отчет №17 быстрее, то это приведет вас к вашей цели.
5. Потому данная история — это проверка данного

предположения достижения цели.

То есть смысл Impact map — это трассировка от User story к общей Цели продукта. Если такой связи нет и вы не можете ее найти — значит вы делаете что-то бесполезное.

То же самое можно сделать в любой момент времени на любом этапе создания продукта. Вы знаете что нужно сделать и это будет полезно, но вот ценность определить однозначно не можете. Вы видите десятки вариантов то, что нужно написать в “чтобы ...”. Постройте путь до цели в Impact map. Найти то влияние что вы должны оказать на актера в результате. Не пишите всякую чушь, в которую сами не верите.

Юсторию можно оценить по критериям «INVEST»:

- Independent. Reduced dependencies = easier to plan;
- Negotiable. Details added via collaboration;
- Valuable.
- Estimable. Too big or too vague = not estimable;
- Small. Can be done in less than a week by the team;
- Testable. Good acceptance criteria;

История для юзера

Пример: «Как пользователь я хочу управлять рекламными объявлениями, чтобы удалять устаревшие или ошибочные объявления»

На первый взгляд, вы не увидите в этой истории никаких изъянов — все элементы на месте. А теперь расскажите-ка, для кого вы собираетесь сделать эту фичу и что этот юзер знает об управлении объявлениями? Он администратор

портала объявлений, которому нужно время от времени чистить базу и премодерировать объявления? Или, может, он рекламодатель, которому нужно просматривать список созданных им объявлений и иметь возможность удалять ненужные объявления прямо из этого списка?

Вы могли заметить, что у этих двух пользователей совсем разные роли, с разными ожиданиями с разными требованиями к системе. Основная ошибка этой истории — игнорирование роли и персоны пользователя.

История для продакт оунера

Пример: «Как продакт оунер, я хочу, чтобы в системе была возможность удалять объявления, чтобы юзеры могли удалять объявления»

И опять, вроде бы все на месте, но что-то не так. Для начала эта история формата «хотели историю? — получили».

Очевидно, что автор истории написал ее исключительно ради самого написания:) Конечно, ваш продакт оунер может написать все истории со своей точки зрения, а не с точки зрения нужд конечных пользователей, но это означает всего 2 вещи: вы движетесь не в ту сторону и у вас проблемы с имплементацией аджайла. Возьмите вашего продакт оунера за шкуру, потрясите его хорошенько и заставьте думать головой:)

История для девелопера

Пример: «Как девелопер, я хочу заменить виджет папок, чтобы у меня был лучший виджет папок:»

Часто такие истории становятся частью technical debt, который состоит из переход на обновленные версии

фреймворков и библиотек, рефакторинга и так далее. У них есть полное право на то, чтобы быть сделанными, но на словах они не представляют никакой ценности для юзера и вам достаточно тяжело будет заставить продакт оунера «купить» их. К тому же по хорошему любая история должна создавать пользовательскую ценность и ваша команда должна уметь рассказать на демо, зачем она все-таки заимплементила ее. Так как же поступить с этой историей?

Перепишите ее с пользовательской точки зрения: «Как рекламодатель, я хочу, чтобы система позволяла создавать мне папки, чтобы я мог быстрее работать с большими списками объявлений»

Технические задачи к этой истории могут выглядеть так:

- «Отрефакторить механизм добавления папок, чтобы позволять создавать вложенные папки вплоть до 3 уровня вложенности»
- «Проапдейтить версию SDK для использования нового механизма локального хранения данных на устройстве»

При этом к такой истории гораздо проще написать критерии приемки:

- «Пользователь может создавать папки с тремя уровнями вложенности»
- «Пользователь не может создать больше 100 папок»

Никакой бизнес ценности для пользователя

Пример: «Как рекламодатель, я хочу чтобы у меня была возможность фильтровать объявления»

У нас есть роль, есть потребность, но причина или бизнес ценность куда-то запропастились. Зачем рекламодателю

фильтровать объявления? Чего он хочет достигнуть? Не думайте, что это буквоедство, история действительно теряет смысл без нужных элементов.

Никаких критериев приемки

В любом из примеров приведенных выше, кроме истории для девелопера, нет критериев приемки. Истории могут проваливать тесты, или тест кейсы могут проверять не те критерии из-за отсутствия понимания того, как должен выглядеть конечный результат и каким требованиям он должен соответствовать. Критерии приемки нужны именно для того, чтобы рассеять ложные предположения, а иногда даже перепланировать историю или разбить ее на меньшие.

- Лучше написать много историй поменьше, чем несколько громоздких.
- Каждая история в идеале должна быть написана избегая технического жаргона — чтобы клиент мог приоритезировать истории и включать их в итерации.
- *Истории должны быть написаны таким образом, чтобы их можно было протестировать*
- Тесты должны быть написаны до кода.
- Как можно дольше стоит избегать UI. История должна выполняться без привязки к конкретным элементам.
- Каждая история должна содержать оценку.
- История должна иметь концовку — т.е. приводить к конкретному результату.
- История должна вмещаться в итерацию.
- Слишком формальные/слишком детализированные задачи.

Иногда владельцы продукта с самыми хорошими намерениями стремятся писать слишком детальные истории. Если на встрече по планированию итерации команда видит набор историй выглядящий как многотомная спецификация, то искушение предположить, что все детали отлично освещены и пропустить обсуждение, очень велико.

- Не пропускайте обсуждения. Истории для этого и обсуждаются на планировании итерации, чтобы вскрыть неясные моменты, уточнить все детали и получить полное представление о задаче. Если вы не обсудили их всей командой, вы рискуете начать двигаться в неверном направлении во время разработки.
- Перестаньте выдавать технические задачи за истории. Если вы пытаетесь втиснуть в формат истории технические задачи, то в конце итерации у вас точно не появится готовый кусочек продукта. Если технические задачи действительно нужны и у них есть ценность для каких-либо пользователей (в том числе и внутренних, в вашей команде), то смело оформляйте их в технические истории, но не составляйте всю итерацию из подобных задач, ведь у вас, вероятнее всего, есть и бизнес-клиенты.

Рекомендации от ведущих специалистов Scrum

1. Как правильно написать User Story?

Командой. Причем команда обязательно должна включать в себя менеджера продукта/клиента/стейкхолдера или даже конечных пользователей вашего продукта. Пишите user story не для того, чтобы получить формальные «требования», а чтобы вытащить на свет все важные для вашего продукта, бизнеса и пользователей нюансы.

Обязательно формулируйте персоны вашего продукта до начала работы над user story. Это поможет вам лучше прочувствовать пользовательские нужды/боли/желания и лучше понять, для кого вы проектируете ваш продукт.

Ваша идеальная история должна быть написана по такому образцу:

Как, <роль пользователя>, я <что-то хочу получить>, <с такой-то целью>

Сейчас вы сформулировали бизнес-ценность для пользователя вашего продукта. Но прелесть пользовательской истории в том, что она формулирует не только бизнес-ценность, но и требования для разработки и тестирования. К этой простой формулировке вы можете добавить критерии приемки, технические заметки, описание обработки ошибок, которые суммируют все задачи, которые вам нужно сделать.

Вот как в укороченном виде выглядела пользовательская история в [одном из моих проектов](#):

Как водитель с загоревшейся лампочкой бензина я хочу быстро найти ближайшую хорошую заправку, чтобы заправиться качественным бензином.

Критерии приемки:

1. Как водитель с загоревшейся лампочкой я могу просмотреть все ближайшие заправки.
2. Как ... я могу выбрать заправки подходящих мне брендов АЗС.
3. Как ... я могу видеть ближайшие заправки выбранных брендов списком.

4. Как ... я могу видеть ближайшие заправки выбранных на карте.

Обработка ошибок:

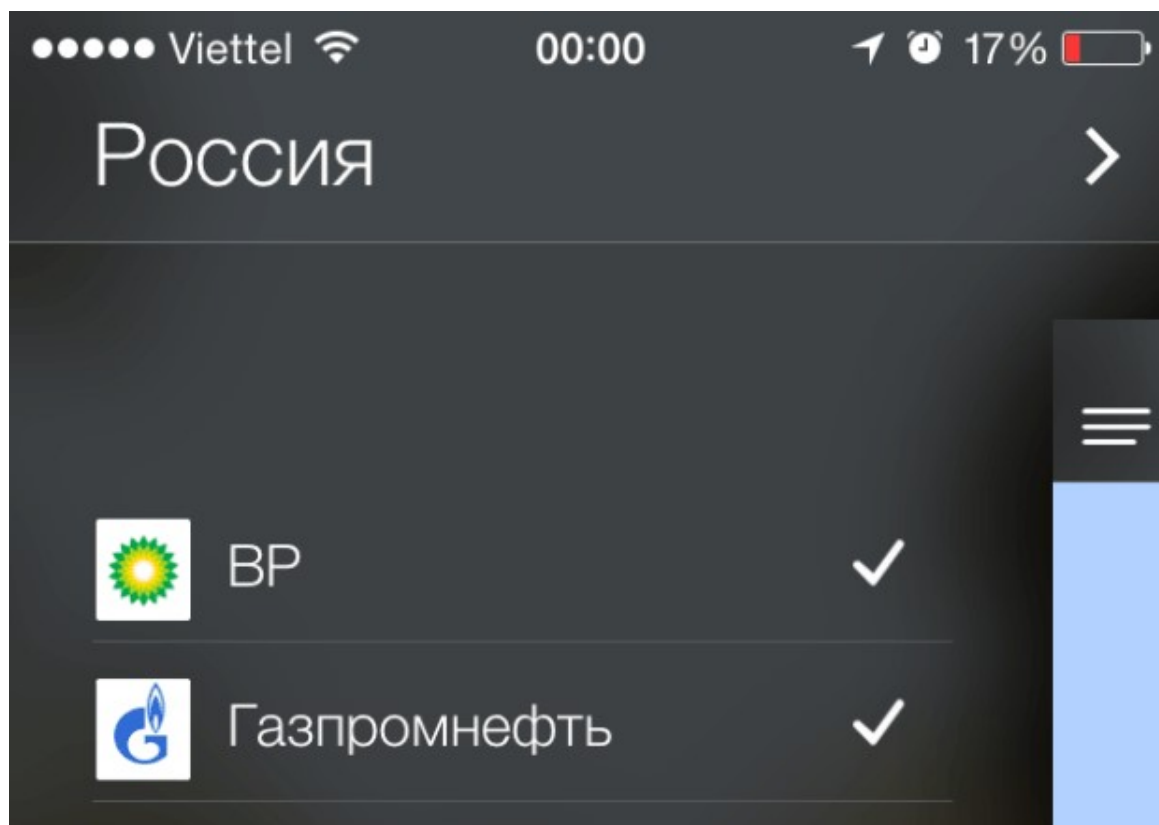
1. При выключенной геолокации пользователя необходимо дать ему информацию о том, где ее включить.

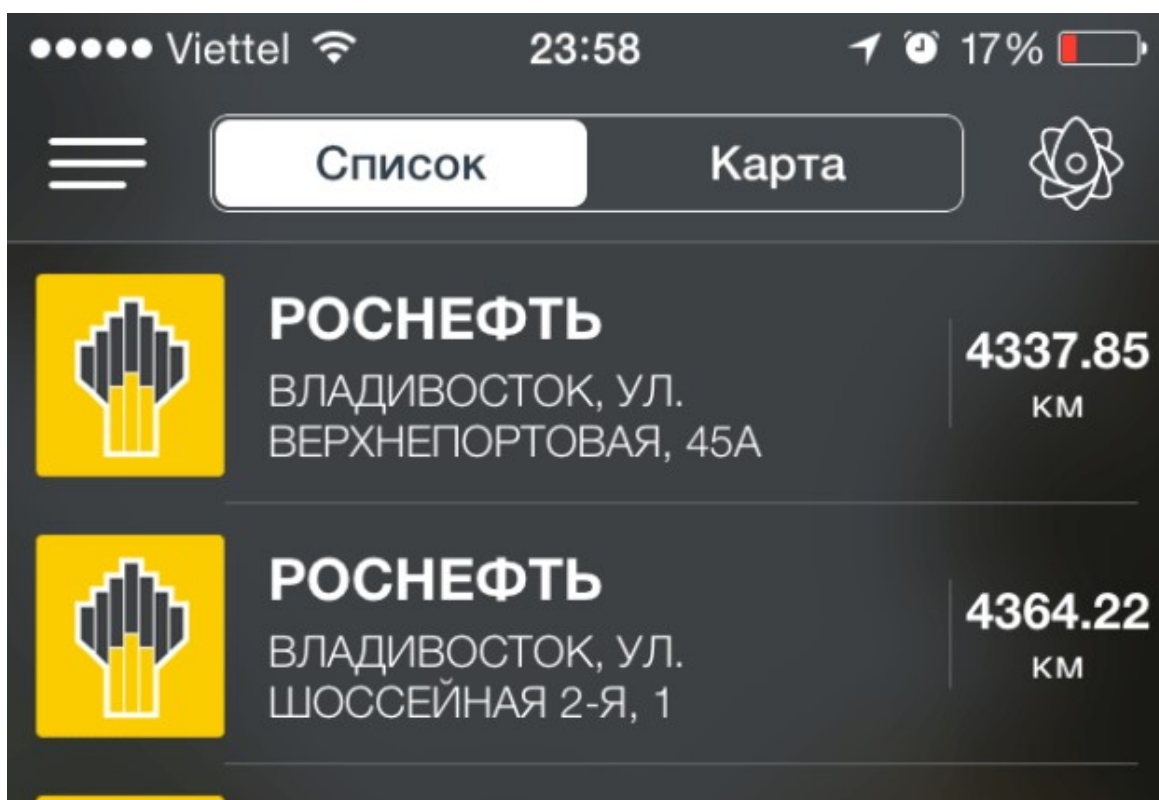
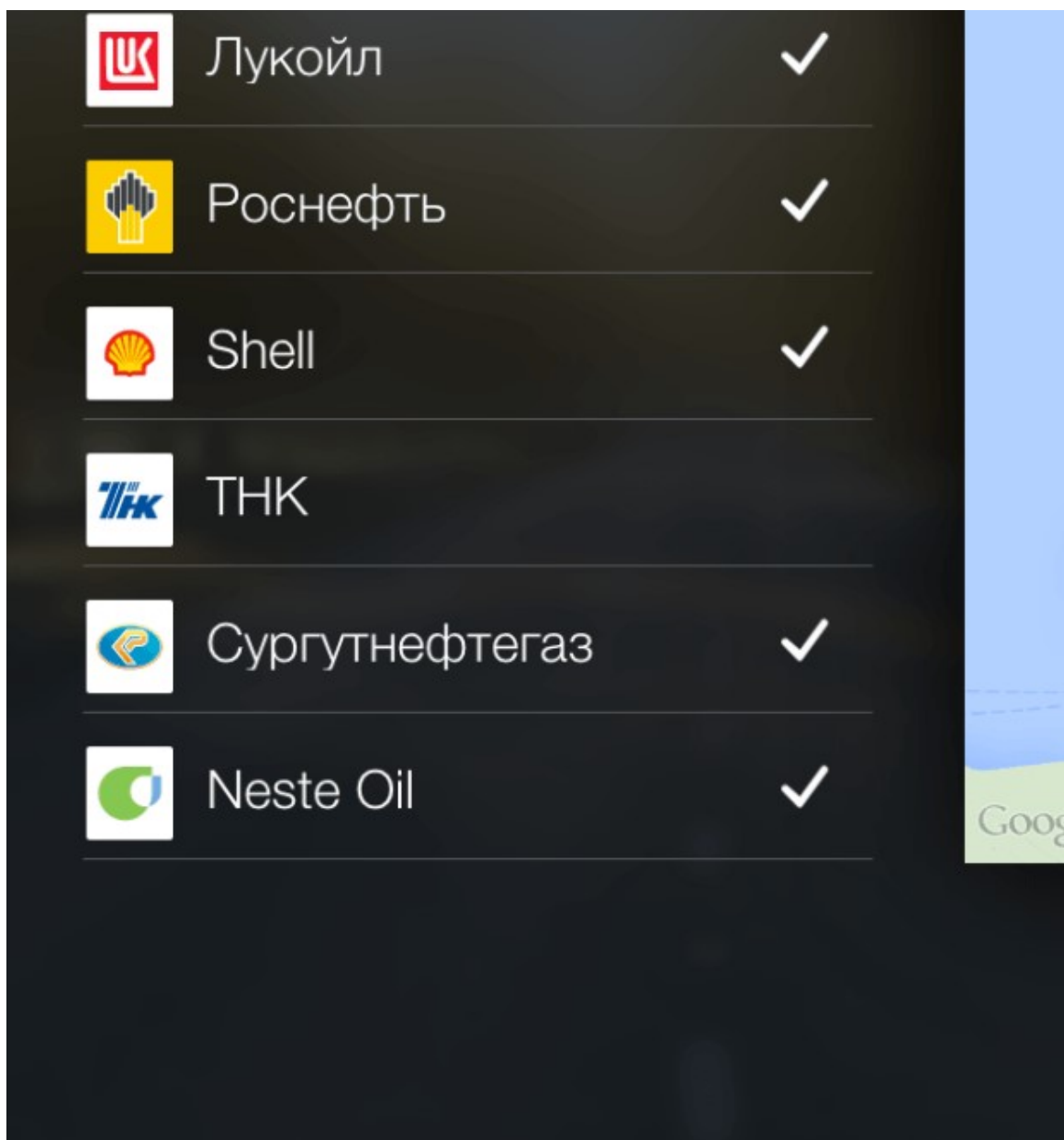
Технические заметки:




1. Заправки в списке должны обновляться при изменении местоположения пользователя на 100 метров.

Такая формулировка задачи помогает разработчикам и тестировщикам не пытаться сравнивать готовую задачу с требованиями, которые быстро устаревают, а смотреть на основные проблемы, которые должны быть решены в рамках задачи. Дополненная прототипами, такая история легко становится задачей в джире или бейскемпе, которую можно делать даже без финального дизайна.

Вот как выглядели экраны, относящиеся к этой истории, в итоговом приложении:





	РОСНЕФТЬ ПРИМОРСКИЙ КРАЙ, БОЛЬШОЙ КАМЕНЬ, УЛ. ВОР...	4367.50 КМ
	РОСНЕФТЬ ПРИМОРСКИЙ КРАЙ, НАХОДКА, УЛ. ПОГРАНИЧНА...	4369.44 КМ
	РОСНЕФТЬ ПРИМОРСКИЙ КРАЙ, НАХОДКА, УЛ. ПОГРАНИЧНА...	4370.25 КМ
	РОСНЕФТЬ ПРИМОРСКИЙ КРАЙ, АРТЕМ, УЛ. СОЛНЕЧНАЯ, 10А	4371.78 КМ
	РОСНЕФТЬ ПРИМОРСКИЙ КРАЙ, НАХОДКА, УЛ. ПЕРЕВАЛЬНА...	4375.51 КМ

2. Как объективно оценить ее полезность и востребованность?

Пользовательские истории полезны, если вы понимаете, что с написанием пользовательской истории для самого простого проекта вы ступили на тяжелый путь сомнений: «зачем мы делаем наш продукт?», «точно ли нужна эта фича в продукте?», «да пользователей с такими потребностями днем с огнем не сыщешь», «кто будет пользоваться тем, что мы делаем?». Эти вопросы не очень приятны, но честные ответы на них помогут вам спроектировать лучший продукт.

3. Чего делать не стоит при работе с User Story?

Писать их в гордом одиночестве или поручать написать пользовательские истории, к примеру, менеджеру проекта. Если, конечно, вы не являетесь конечным соге пользователем продукта, который вы разрабатываете :)

Также не очень здорово писать объемные, большие истории. Если ваша история не вмещается в стандартную итерацию вашей команды (я надеюсь, что это максимум 4 недели:), то она слишком велика и стоит задуматься, как можно ее поделить на несколько.

И самые главные грабли — писать пользовательские истории, которые пойдут в разработку, до того, как вы прошли через процесс customer development. Хорошо сделать это для общего понимания того, что пользователь, по вашему мнению, будет делать с продуктом.

Но пользовательские истории нужно писать не только для того, чтобы выразить ваше мнение о продукте или мнение заказчика. Они должны выражать мнение тех, кто будет покупать и пользоваться продуктом (не забудьте о том, что это не только конечные пользователи, но и те, кто оказывают влияние на совершение покупки. К примеру, конечными пользователями игр часто являются дети, но покупают их родители).

Поэтому для того, чтобы написать ценную и реалистичную пользовательскую историю, вам нужно получить максимум информации о ваших будущих пользователях:

- считают ли они проблему, которую решает ваш продукт, достаточно серьезной (к примеру, все игры решают

серьезную проблему — убийство времени и побег от скуки);

- как они решают свои проблемы сейчас;
- какие заменители или конкуренты есть у вашего продукта;
- и еще массу важных моментов, которые стоит узнать до того, как вы написали гору кода :).

Это самый большой и объемный пункт, поэтому очень хочу порекомендовать к прочтению 2 книги:

[Four Steps to the Epiphany](#) — библия customer development, которая даст вам фундаментальное понимание об этапе создания продуктов, которые вам нужно пройти перед тем, как написать пользовательские истории.

[User Stories Applied](#) — самая лучшая и полная книга о том, как писать, оценивать, тестировать и принимать пользовательские истории.

Я бы сказал, что user story — это инструмент. Инструмент этот обычно используют outsourcing компании. Он позволяет начать диалог с клиентом и работать в одной карте понимания задачи. Так что чаще всего user story пишет заказчик. Сам формат user story, который выглядит так «As !WHO! I want !WHAT! so that !WHY!» предполагает, что её пишет пользователь/заказчик, который объясняет ЧТО он хочет и ЗАЧЕМ. Мы разрабатываем продукты для глобального рынка и разрабатываем продукты самостоятельно, поэтому таким инструментом, как user story мы не пользуемся. Для нас более актуальными являются сценарии использования, которые мы в том числе используем для QA продукта.

1. Как правильно написать User Story?

Хорошая User Story должна соответствовать модели [INVEST](#).

2. Как объективно оценить ее полезность и востребованность?

Объективно оценить её полезность и востребованность достаточно сложно, т.к. непосредственно в процессе разработки она не участвует, а служит отправной точкой. Это способ начать диалог без понятных маркеров его завершения. Субъективно для нас этот инструмент бесполезен.

3. Чего делать не стоит при работе с User Story?

- Использовать историю для выполнения которой требуется более одной итерации
- Писать громоздкие истории.
- Использовать жаргон.
- Сразу привязывать историю к конкретному интерфейсу.

1. Как правильно написать User Story?

Не важно то, как она будет написана и оформлена, главное — насколько правильно и точно она описывает потребности пользователя. В Touch Instinct мы проговариваем пользовательскую историю с клиентом устно, во время переговоров. Делаем заметки. Кто пользователи, чего они хотят? Мы выясняем формализованные потребности: мгновенная покупка, удобное чтение новостей, бронирование мест, заказ билетов и т.д., из которых прорабатываем детальные требования к сценариям использования будущей

программы. «Я как пользователь хочу сортировать товары по цене, чтобы выбрать лучшее из одной ценовой категории». «Я как пользователь хочу сохранять музыку в кэш, чтобы слушать без интернета». Далее на основе юз кейсов строим интерфейс, на этом этапе мы понимаем, от какого функционала стоит отказаться, например, нужны ли комментарии к фотографиям или нет.

2. Как объективно оценить ее полезность и востребованность?

Заказчик хорошо знает свой продукт и потребителя. На переговорах мы стараемся вытянуть из него максимальную информацию о том, чего хочет пользователь. Полезность юзер стори прежде всего в том, что они помогают разработчику лучше понять область, продукт, аудиторию заказчика. Мы не совершаем действий ради действий. Востребованность юзер стори оценивается обнаружением и проработкой пользовательских потребностей, на выходе продукт их должен удовлетворять.

3. Чего делать не стоит при работе с user story?

Не стоит зацикливаться и затягивать с проработкой. Зафиксировали ключевой функционал, держите его в фокусе, перед глазами, но не воспринимайте как инструкцию. Юзер стори достаточно гибкая вещь, в которую можно вносить изменения.

User Story обычно используется при гибких методологиях разработки. В нашей компании часть проектов ведется по такой методологии. Обычно мы организуем встречу с клиентом, на которой просим его описать обычным

пользовательским языком пожелания к функционалу сайта или мобильного приложения. На основе этого мы составляем конечное описание работ для итерации (беклог). Правильный пользовательский сценарий, на наш взгляд, должен быть:

- понятным для всех участников проекта (и конечного пользователя);
- коротким, чтобы можно было оценить сроки его выполнения, но при этом с достаточно точным описанием;
- сценарий должен совпадать по смыслу и идее с основным проектом (чтобы очередная итерация «не выпадала» из общей идеи проекта. Это как раз слабое место гибких методологий, потому что при большом количестве итераций порой забывается основная идея и смысл проекта, он обрастает дополнительным и не всегда нужным функционалом, превращаясь в громоздкого «Франкенштейна»)

Объективно можно оценить полезность в том случае, если по user story можно сформировать удобный и понятный конечному потребителю продукта интерфейс.

При написании user story нужно стараться придерживаться максимально простого описания (без ухода в технические детали), учитывать роли пользователей при работе с продуктом, стараться не увеличивать размер истории, т.к. это должно вписаться в одну итерацию, которая при гибких методологиях длится не более 2х недель.

Итак, истории:

1 Как пользователь я могу хранить свои фотографии в системе, чтобы иметь возможность показать или продать их другим пользователям.

2 Как рекламодатель я могу помещать свою рекламу в системе, ориентированную на пользователей.

3 Как администратор я могу управлять фотографиями пользователей, так чтобы контент сайта был легальным.

Во время обсуждения первой истории, заказчик и команда приходят к тому, что пользователи системы должны быть авторизованны системой перед выполнением каких-либо действий с фотографиями. Это приводит к появлению новой пользовательской роли «гостя» — группе людей, которые неавторизованны системой или вообще пока не имеют пользовательской учетной записи.

4 Как гость я могу зарегистрироваться в системе для получения пользовательской учетной записи и последующей работы.

5 Как гость я могу войти в систему под ранее созданной учетной записью, для последующей работы.

Пользуясь принципом симметричности требований, команда и заказчик принимают решение, что пользователь должен иметь возможность удалить свою учетную запись в случае необходимости:

6 Как пользователь я могу удалить свою учетную запись и перестать быть пользователем системы.

Обсуждая концепцию учетных записей, рождаются также следующие истории:

7 Как пользователь я могу изменить данные своей учетной записи.

8 Как пользователь я могу сделать некоторые поля своей учетной записи видимыми для других пользователей.

Просто? Достаточно. По крайней мере, не сложнее, чем писать спецификации. Но дальше — интереснее.

ВОПРОСЫ?

К этому моменту, я надеюсь, у вас появилось много интригующих вопросов, даже, может быть, их у вас стало больше, чем до начала чтения этой статьи... Я попробую вкратце разъяснить как же все-таки это все может работать.

КУДА ДЕЛИСЬ ДЕТАЛИ?

Первый вопрос, который задает человек, который привык работать с более тяжеловесным подходом к требованиям (основанным к примеру на подходе Software Requirements Specifications из RUP), это «куда подевались детали?»

Это вопрос затрагивает ключевой аспект использования историй. Попробую коротко объяснить.

Конечно, детали есть, и их никто не отменял — как без понимания деталей программист может написать адекватный код, а тестировщик его принять? Детали необходимы. Но использование историй смещает суть и время выработки деталей.

Детали историй — это больше не неизменная часть требований, которые продумываются заказчиками во время написания требований и предъявляются команде в готовом виде. Вместо этого заказчик и команда во время обсуждений историй совместно приходят к понимаю уровня детализации, который необходим на текущей фазе, и принимают совместные решения, пополняя истории все большим количеством информации.

ПРИМЕР ДЕТАЛИЗАЦИИ.

Рассмотрим одну из историй, идентифицированную выше:

4 Как гость я могу зарегистрироваться в системе для получения пользовательской учетной записи и последующей работы.

Во время обсуждения этой истории с командой заказчику задают вопрос о том какая информация нужна для создания пользовательской учетной записи. Обсуждая различные варианты, заказчик и команда приходят к тому, что для первой версии системы достаточно будет проверенного электронного адреса плюс имени пользователя и его пароля.

К истории дописывается этой комментарий. Теперь история выглядит так:

4 Как гость я могу зарегистрироваться в системе для получения пользовательской учетной записи и последующей работы.

Нужен проверенный email и выбранные пользователем имя и пароль.

В ходе дальнейших высказываний кто-то из тестировщиков задает резонный вопрос о минимальной длине пароля и проверке на уникальности имени. Продолжая дискуссию, команда и заказчики приходят к мнению, что необходимо описать основные критерии готовности истории, чтобы команда понимала ожидания и знала, когда объявлять историю готовой:

4 Как гость я могу зарегистрироваться в системе для получения пользовательской учетной записи и последующей работы.

Нужен проверенный email и выбранные пользователем имя и пароль.

Тест 1: пользователь не может ввести пароль меньше 6 символов

Тест 2: пользователь не может ввести имя меньше 3 и больше 20 символов

Тест 3: пользователь должен иметь уникальное имя в системе

Тест 4: после регистрации пользователь должен получить емейл для активизации своей учетной записи

Тест 5: пользователь не может войти в систему, если учетная запись не была активизирована

Тест 6: при успешном входе система приветствует пользователя текстом «Добро пожаловать, <имя пользователя>»

Возможно во время реализации, тестирования и приема истории возникнут ещё какие-то дополнительные моменты. В этом случае они могут быть описаны в виде уточняющих тестов или как комментарии. Возможно из этих дополнения появятся новые истории.

Таким образом истории пополняются деталями по мере необходимости, эволюционируя от коротких высказываний до детализированных и согласованных требований со встроенными критериями готовности.

МОЩНЫЕ ИНСТРУМЕНТЫ РАБОТЫ С ИСТОРИЯМИ: УПОРЯДОЧИВАНИЕ, РАЗБИЕНИЕ И ГРУППИРОВКА

Как видно, описанные выше истории являются более-менее автономными сущностями, и, как следствие, могут быть перечислены в другом порядке. Конечно между историями существуют связи и логические цепочки — нельзя, к примеру, удалять пользовательские записи, не умея создавать их. Но все таки можно научиться составлять истории таким образом, чтоб обеспечить некоторую свободу в выборе порядка их реализации. Свободы будет, естественно, тем больше, чем больше самих историй и чем независимее они друг от друга.

Если же истории независимы, да к тому же их достаточно много, то можно смело предположить, что их ценность с точки зрения вклада в систему различна. А значит, варьируя порядком историй, можно выставить их в таком порядке, что первые «n» историй будут играть ключевую роль в полезности системы, в то время как другие истории будут скорее необязательными добавками, привлекающими пользователей или облегчающими их работу.

Пользуясь знанием рынка, а также здравым смыслом (к сожалению на сегодняшний день оба этих критерия не поддаются численной оценке), заказчик выстраивает список историй таким образом, чтобы максимизировать возврат вложений от проекта.

Вот пример, как могли бы быть отсортированы истории вышеописанного проекта (это всего лишь один из вариантов, конечно, есть и другие):

4 Как гость я могу зарегистрироваться в системе для получения пользовательской учётной записи и последующей работы.

5 Как гость я могу войти в систему, имперсонализируясь

с ранее созданной учётной записью, для последующей работы.

1 Как пользователь я могу хранить свои фотографии в системе, чтобы иметь возможность показать или продать их другим пользователям.

3 Как администратор я могу управлять фотографиями пользователей, так чтобы контент сайта был легальным.

7 Как пользователь я могу изменить данные своей учетной записи для корректировки измененных или неверных данных.

2 Как рекламодатель я могу помещать свою рекламу в системе, ориентированную на пользователей.

8 Как пользователь я могу сделать некоторые поля своей учетной записи видимыми для других пользователей.

6 Как пользователь я могу удалить свою учетную запись и перестать быть пользователем системы.

Как вы видите, истории выстроены в порядке, который, во-первых, логичен с точки зрения заказчика и команды, а во-вторых ценность историй уменьшается сверху вниз. Таким образом, если, к примеру, на половине проекта наступает нехватка ресурсов (скажем, после реализации истории для администратора системы), заказчики смогут получить выгоду от продукта, так как наиболее важные истории уже будут реализованы. Это ни что иное как минимизация рисков от вложений.

Конечно, порой не так легко и очевидно принять правильное решение о порядке историй, но в этом и состоит мастерство быть заказчиком (это отдельная, неисчерпаемая тема...)

Кроме инструментария ранжирования историй, в руках у заказчика есть и другие мощные средства, позволяющие повысить эффективность своих финансовых вложений. К примеру, одна из описанных на ранней фазе проекта историй в какой-то момент может показаться слишком большой в сравнении с другими, что усложняет понимание её приоритета:

1 Как пользователь я могу хранить свои фотографии в системе, чтобы иметь возможность показать или продать их другим пользователям.

В этом случае заказчик и команда могут попробовать разбить ее на несколько более мелких историй, каждая из которых может получить свой приоритет:

9 Как пользователь я могу хранить свои фотографии в системе, чтобы иметь возможность показать их другим пользователям.

10 Как пользователь я могу хранить свои фотографии в системе, чтобы иметь возможность продать их другим пользователям.

При этом нужно учесть, что начальная история не разбивается на две «под-истории», а замещается двумя новыми. Это не разбиение историй на подзадачи для постановки их программистам, это всего лишь переформулировка требований для более эффективного управления ими.

Подобный процесс разбиения сложных и больших истории на более простые может осуществляться в теории довольно долго. На практике же, заказчики и команда в скором времени вырабатывают совместное понимание адекватного размера историй и следуют ему при написании новых и

разбиении существующих историй. Этот размер зависит от количества историй, реализуемых за итерацию. Но об этом поговорим подробнее, обсуждая планирование.

Механизмом, обратным разбиению, служит группировка историй. Иногда бывает полезно склеить мелкие истории в одну побольше для улучшения понимания связности историй.

Я уверен, вы неоднократно будете пользоваться этими простыми но мощными средствами управления требованиями, когда начнете использовать истории в своих проектах.

Как вы до сих пор справлялись с подобными задачами?

ДИНАМИКА ЗНАНИЙ

Программный продукт — это не только код и документация. Это также знания о пользователях, рынке, особенностях продукта, технологиях и прочее. Если же проект — это развитие продукта, то тогда в нем должны гармонично изменяться все его составные части: код, документация и знания. А, следовательно, знания динамичны. Таким образом, что вчера казалось фактом, сегодня в свете новоприобретенных знаний таковым может уже не быть. Для историй это значит, что порядок приоритезации историй, сделанный вчера, уже сегодня, возможно, должен быть изменен.

И в этом нет ничего плохого: меняется мир, также меняются наши знания о мире. И это такой же факт для индустрии программного обеспечения, как и для всего остального.

Вот ещё один плюс хранения требований в виде списка относительно независимых историй. Его в любой момент

можно пересортировать, добавить новые или удалить ненужные истории. Хранение требований в виде историй не препятствует динамичности знаний, а наоборот, базируется на том, что наши знания будут и должны меняться, иначе продукт устареет, ещё не начав использоваться.

ЧТО ДАЛЬШЕ?

Когда начальный набор историй готов, все истории обговорены и детализированы до нужной степени, ничего больше не остается, как перейти к их реализации, выпуская программный продукт, в соответствии с приоритетами заказчиков и желаниями пользователей.

Про оценивание размера историй, планирование историй по итерациям, предсказание времени готовности историй и прочие важные аспекты речь пойдет во второй части статьи.

Откуда:

<http://agilevision.blogspot.ru/2013/07/user-stories.html>

<http://www.ozon.ru/context/detail/id/34376940/>

<http://2tickets2dublin.com/how-to-write-good-user-stories-part-1/>

<http://apptractor.ru/develop/user-story-plan-deystviy-dlya-razrabotchika.html>
