Лекция 4

Верификация и валидация требований

Как оценить качество требований?

Дать определение качеству требований и измерить это качество, достаточно трудно.

Требования считаются качественными, если они **отражают потребности заинтересованных сторон и предоставляют достаточно информации для создания программного обеспечения**, которое удовлетворяет целям продукта или проекта.

Важно понимать, что критерии качества требований - это полезный инструмент, который можно и нужно использовать для улучшения эффективности работы.

На любом проекте **некачественные требования приводят к нежелательным последствиям**, таким как:

- Множество раундов разъяснений и уточнений;
- Сложности в планировании;
- Частые переделки;
- Превышение времени и бюджета;
- Сложности в поддержке требований;
- Несчастная команда проекта;
- Недовольный клиент.

Развлекательное видео, которое просто и наглядно демонстрирует почему качественные требования так важны

Exact Instructions Challenge - THIS is why my kids hate me. | Josh Darnit – YouTube

Какие существуют критерии качества?

Существует масса списков критериев качества требований, составленных различными авторами, и "заточенных" под разные методологии разработки. Вот лишь некоторые из них:

По BABOK:

Atomic (Атомарное)
Complete (Полное)
Consistent (Непротиворечивое)
Concise (Краткое)
Feasible (Выполнимое)
Unambiguous (Однозначное)
Testable (Тестируемое)
Prioritized (Ранжированное)
Understandable (Понятное)

BABOK – руководство по своду знаний по бизнес-анализу

? Вигерс лк. 1
? INVEST лк. 2

По Вигерсу:

Correct (Верное)
Feasible (Выполнимое)
Necessary (Необходимое)
Prioritized (Приоритизированное)
Unambiguous (Однозначное)
Verifiable (Тестируемое)

INVEST (Bill Wake):

Independent (Независимое)
Negotiable (Обсуждаемое)
Valuable (Полезное)
Estimable (Поддающееся оценке)
Small (Компактное)
Testable (Тестируемое)

Requirements Management Using IBM Rational RequisitePro*:

Unambiguous (Однозначное)
Testable (verifiable) (Тестируемое)
Clear (Четкое)
Correct (Верное)
Understandable (Понятное)
Feasible (Выполнимое)
Independent (Независимое)
Atomic (Атомарное)
Necessary (Необходимое)
Implementation-free (abstract) (Абстрактное)
Consistent (Согласующееся)
Nonredundant (Не избыточное)
Complete (Полное)

* Инструмент IBM Rational RequisitePro предназначен для организации работы аналитиков и автоматизации их деятельности в области управления требованиями.

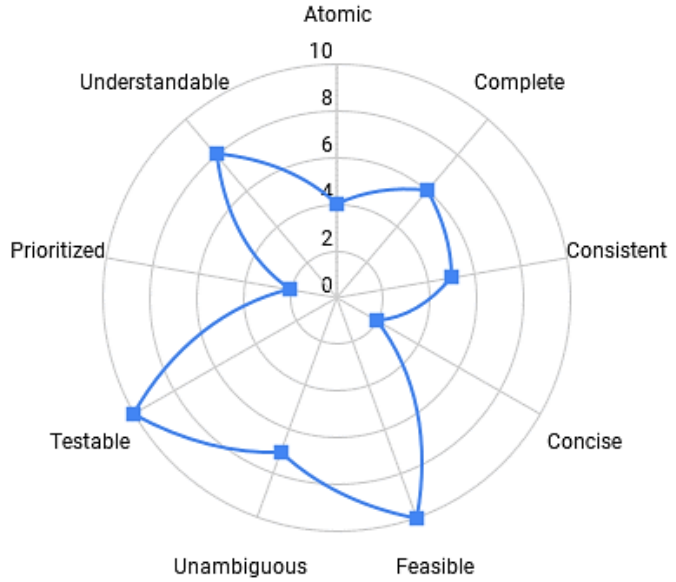
Самый простой и первый способ верифицировать требования – это проверить их на соответствие критериям качественного формулирования:

- ✓ **атомарность** – в одном требовании изложено только одно требование;
- ✓ **полнота** – достаточный уровень детализации;
- ✓ **согласованность** – непротиворечивость с другими требованиями и бизнес-правилами;
- ✓ **краткость** – отсутствие лишней информации в описании;
- ✓ **понятность** – представление в терминах, ясных для стейкхолдеров и в соответствии с глоссарием проекта;
- ✓ **однозначность** – возможность проверить, удовлетворяет ли решение исходную потребность;
- ✓ **реализуемость** – осуществимость в рамках проектных ограничений (время и деньги) с приемлемым уровнем риска;
- ✓ **тестируемость** – возможность проверить выполнение в конкретных количественных показателях, например, SLA * 99,99% вместо высокой доступности;
- ✓ **приоритезируемость** – возможность определить относительную важность и ценность.

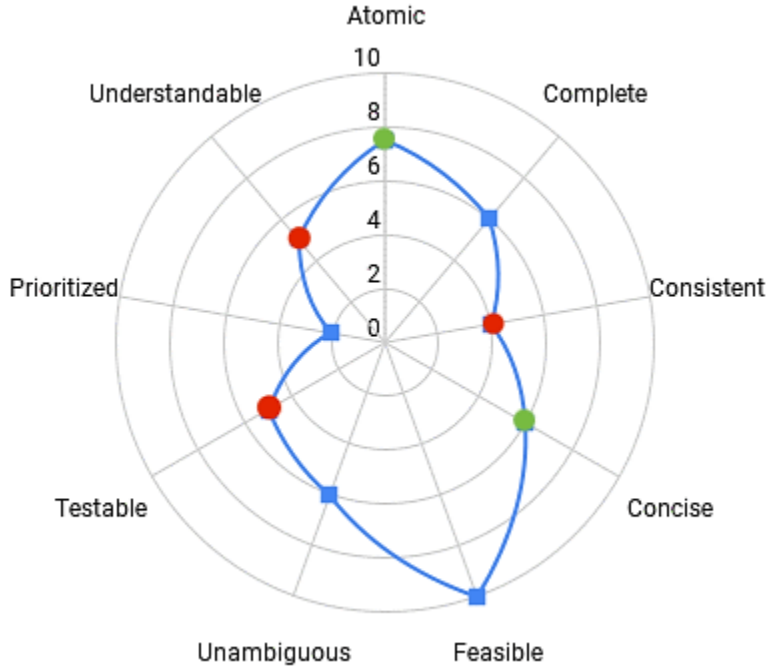
* SLA (Service Level Agreement) или соглашение об уровне предоставления услуги, выражаемый в процентах, описывает время безотказной работы и простоя системы, что относится к ее доступности. Например, SLA 99,99 означает, что система будет доступна 99,99% времени. Так, время безотказной работы составит в сутки 23 часа 59 минут 51 секунду, а допустимое время простоя – 9 секунд. В неделю время безотказной работы составит 6 дней 23 часа 59 минут, а допустимое время простоя – 1 минуту.

Как улучшить качество требований?

Предположим, мы определились с наиболее важными критериями качества. Для наглядности можно дать им некоторую условную оценку и представить их на лепестковой диаграмме.
Например, если сосредоточить свои усилия на улучшении какого-то выбранного критерия, допустим - атомарности (Atomic), то спустя время мы увидим следующее:



По некоторым критериям будут также наблюдаться улучшения, хотя усилия на них направлены не были. Другие же начнут ухудшаться. В графике выше, например, когда мы добиваемся лучшей атомарности (Atomic), параллельно улучшается оценка краткости (Concise), но ухудшается оценка понятности (Understandable).



Это может быть связано с несколькими причинами:
Некоторые критерии взаимосвязаны. Дробление требований на более мелкие элементы для улучшения атомарности скорее всего автоматически приведет к большей краткости требований (прямая зависимость). При этом такое дробление может ухудшить понятность - элементы слишком малы и сложнее проследить зависимости, и в какой-то момент может просесть, например, тестируемость (обратная зависимость).

Выходить из положения, соответственно, можно двумя способами: увеличить ресурсы на улучшение качества требований (что не всегда в наших силах), или же осваивать новые инструменты, техники и шаблоны. Например,

- ✓ изучить нотации для моделирования (BPMN, IDEF0, UML);
- ✓ попробовать использовать различные шаблоны для документирования требований, выбрать наиболее подходящие для конкретного применения;
- ✓ детально изучить возможности используемой системы управления требованиями, при возможности - попробовать различные системы;
- ✓ освоить инструменты для прототипирования и снабжать артефакты качественными прототипами;
- ✓ поработать над синтаксисом требований, изучив различные подходы к синтаксису
- ✓ систематизируйте рекомендации по конкретным критериям, для "самодиагностики". *Например как в таблице на след слайде*

Просто мем для привлечения внимания, он не служит иллюстрацией к данному тексту, хотя….

Критерий	Описание	"Симптомы"	"Лечение"	"Плохой" пример	"Хороший" пример
Independent (Независимое)	Пользовательская история может быть взята в работу и имплементирована независимо от других историй.	Прослеживается зависимость типа: •Пересечение •Зависимость порядка выполнения •Вложенность	Разбейте пользовательскую историю на более мелкие; предложите функциональность-заглушку или "hard-code", чтобы временно исключить зависимость.	Как клиент я хочу просмотреть список доступных достопримечательностей, настроенных Администратором, чтобы я мог решить, какие из них я хочу посетить.	Как клиент, я хочу просмотреть список достопримечательностей, чтобы решить, какие из них я хочу посетить. (в критериях приемки упомянуть, что список "hard-coded") Как администратор я хочу управлять списком достопримечательностей, чтобы покупатель мог просматривать актуальный список.
Negotiable (Обсуждаемое)	Пользовательские истории должны оставлять место для обсуждения.	Содержит слишком много деталей; предписывает конкретную технологию, подход к реализации или решение.	Пользовательские истории должны оставаться высокоуровневыми, с фокусом на конечной цели, а не на решении	Содержимое отчета необходимо экспортировать в электронную таблицу Microsoft Excel.	Содержимое отчета должно быть экспортировано в формате, позволяющем передать его в органы власти для дальнейшего аудита.
Valuable (Полезное)	Пользовательская история должна приносить пользу заинтересованным лицам (IRACIS).	История сосредоточена на технологии и преимуществах для разработчиков, а не на конечной цели.	Попытайтесь четко определить и сформулировать основную ценность (IRACIS: Increase Revenue, Avoid Costs, Improve Service, + Meet Regulations, Generate Information, etc.); объедините пользовательские истории;	Токен доступа должен инвалидироваться через 12 часов.	Система должна принудительно прерывать сессию пользователя через 12 часов активности, в соответствие с политикой защиты игрока.
Estimable (Поддающееся оценке)	Историю можно оценить: сделать какое-то суждение о ее размере, стоимости или времени реализации.	Команда не может оценить историю :) Использование аббревиатур для конкретных доменов; ссылка на стандарты и процедуры предметной области; обобщения (например, *все*)	Предоставьте команде необходимые знания предметной области; укажите все недостающие детали; разделите на историю на спайк и основную историю.	Пользователь должен иметь возможность использовать все способы оплаты, доступные в Emerchantpay.	Spike для изучения API платежной системы+ Пользовательская история (или истории) для каждого конкретного способа оплаты
Small (Компактное)	Пользовательская история не должна быть настолько большой, чтобы ее невозможно было планировать / приоритизировать с определенным уровнем точности (например, вписываться в спринт из 6-10 историй).	Оценка слишком велика	Разделяйте на более мелкие истории. Методы: SPIDR, CRUD, business rule variations, и т.п.	Как администратор я хочу получить доступ к статистике и сводным отчетам.	Разделить на более мелкие пользовательские истории для каждого конкретного типа отчета.
Testable (Тестируемое)	Проверяемая история - это история, для которой, при любых возможных входных данных, известно ожидаемое поведение системы.	Слова-триггеры: обобщения (все/ни один/любой/все комбинации/никогда), субъективные характеристики (хороший/плохой/лучший/легкий в использовании/интересный/быстрый/эффективный), некоторые наречия (быстро, безопасно, вовремя); неопределенные слова или аббревиатуры (и т.д., и/или, TBD); вероятностные понятия, аппроксимация.	Избегайте обобщений; переформулируйте субъективные понятия в измеримые величины.	Пользователь никогда не должен долго ждать, пока загрузится новая страница.	Новая страница загружается в течение двух секунд в 95% случаев.

Источник: О критериях качества требований (artofba.com)

Верификация и валидация требований

Стандарт ИСО 9000:2000 определяет эти термины следующим образом:

«**Верификация** — подтверждение на основе представления объективных свидетельств того, что установленные требования были выполнены».

«**Валидация** — подтверждение на основе представления объективных свидетельств того, что требования, предназначенные для конкретного использования или применения, выполнены».

Как видно, определения чуть ли не совпадают и уж если не полностью, то в значительной части. И, тем не менее, **верификация и валидация — принципиально разные действия.**

Перевод с английского этих терминов позволяет понять разницу:

verification — проверка,

validation — придание законной силы.

Верификация требований

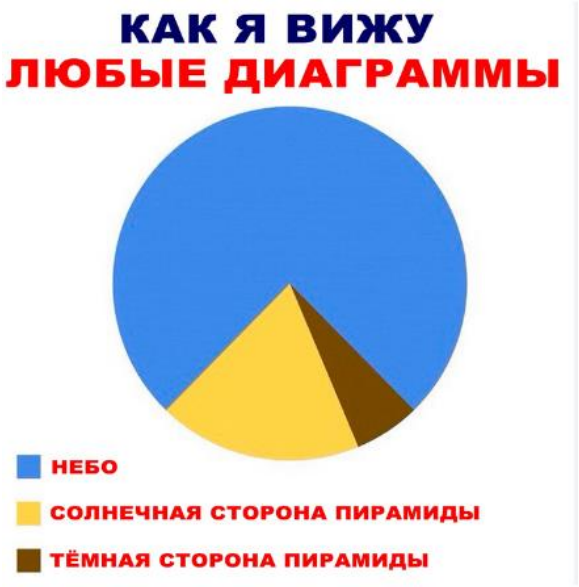
Верификация требований – это проверка, что их описание в виде спецификации (SRS) или ТЗ соответствуют отраслевым и организационным стандартам качества, а также пригодны для дальнейшего практического использования. Последнее означает, *что на основании составленных аналитиком требований ИТ-архитектор сможет выполнить архитектурный проект, разработчик – написать код, UI/UX-дизайнер – реализовать пользовательский интерфейс, а тестировщик – протестировать, как все это работает*.

Способы верифицировать требование:

- ✓ проверка требования на соответствие критериям качества.
- ✓ проверка правильности описания бизнес-процессов в формальных нотациях моделирования (IDEF0, BPMN, EPC или UML).
- ✓ проверка, насколько разумно использовать выбранные методы и инструменты с учетом дальнейшего применения этих диаграмм и способности стейкхолдеров их понимать.

Верификация требований – это не однократное мероприятие, а целый набор действий, которые аналитик периодически выполняет в рамках разработки и анализа требований.

это шутка



Валидация требований

Валидация требований – это тоже проверка. Однако, в отличие от верификации, она направлена не на форму, а на содержание. Цель валидации требований – убедиться, что все требования стейкхолдеров и требования к решению соответствуют бизнес-требованиям и смогут удовлетворить бизнес-потребности.

Этот непрерывный процесс включает следующие действия:

- ✓ ***выявление предположений***, что реализация требования поможет бизнесу и стейкхолдерам получить ожидаемую ценность. Сформулировав такие предположения, аналитик и стейкхолдеры подтверждают или опровергают их.
- ✓ ***определение измеримых критериев оценки***, целевых показателей и метрик, а также их значений, чтобы понять успех изменения после реализации решения, т.е. после перехода от текущего состояния (as is) к желаемому (as to be).
- ✓ ***оценка соответствия границам и содержанию решения*** – входит ли требование в *scope*. Хотя требование может быть полезно конкретному стейкхолдеру, оно может противоречить бизнес-требованиям и/или выходить за рамки решения. В этом случае следует пересмотреть будущее состояние и изменить *scope* решения или исключить требование. Если вариант решения нельзя проверить на соответствие требованию, то оно отсутствует или неправильно понято. Также возможно, следует изменить сам дизайн.

Чем валидация отличается от верификации?

Верификация и валидация являются проверками, это две разных задачи из одной области знаний BABOK Guide «Анализ требований и определение дизайна» (Requirements Analysis and Design Definition).

Верификация — проводится практически всегда, выполняется методом проверки (сличения) характеристик продукции с заданными требованиями, результатом является вывод о соответствии (или несоответствии) продукции.

Валидация — проводится при необходимости, выполняется методом анализа заданных условий применения и оценки соответствия характеристик продукции этим требованиям, результатом является вывод о возможности применения продукции для конкретных условий.

Верификация отвечает на вопрос «Правильно ли записаны требования?», а валидация – на вопрос «А правильные ли требования записаны?».

или

Верификация отвечает на вопрос "Делаем ли мы продукт правильно?", а валидация- на вопрос "Делаем ли мы правильный продукт?"

Что не прошло это требование валидацию или верификацию?

Рецензирование требований

Всякое исследование продукта ПО на предмет выявления проблем любым другим лицом, кроме его автора, называется рецензированием (peer review).

Неформальное рецензирование полезно при знакомстве людей с продуктом и сборе неструктурированных отзывов о нем. Есть несколько видов:

- ✓ проверка «за столом» (peer deskcheck), когда вы просите одного коллегу исследовать ваш продукт;
- ✓ коллективная проверка (passaround), когда вы приглашаете несколько коллег для параллельной проверки продукта;
- ✓ сквозной разбор (walkthrough), когда автор описывает создаваемый продукт и просит его прокомментировать.

Формальное рецензирование представляет собой строго регламентированный процесс. По его завершении формируется отчет, в котором указаны материал, рецензенты и мнение команды рецензентов о приемлемости продукта.
Наиболее хорошо себя зарекомендовавшая себя форма официального рецензирования **экспертиза** (inspection). Экспертиза это четко определенный многоэтапный процесс. В ней участвует небольшая команда участников, которые тщательно проверяют продукт на наличие дефектов и изучают возможности улучшения.

Пример, несколько компаний сэкономили ни много ни мало — целых 10 часов труда на каждый час, потраченный на экспертизу документации требований и других готовых к поставке продуктов (Grady и Van Slack, 1994).

Процесс экспертизы

Участники экспертизы должны отражать четыре точки зрения на продукт (Wiegers):

- ✓ **Автор продукта или группа авторов.** Это точка зрения бизнесаналитика, составившего документацию требований.
- ✓ **Люди, послужившие источником информации для элемента, который следует проверять**. Это представители пользователей или автор предыдущей спецификации.
- ✓ **Люди, которые будут выполнять работу, основанную на проверяемом элементе** Можно привлечь разработчика, тестировщика, менеджера проекта, а также составителя пользовательской документации, потому что они смогут выявить проблемы различных типов. Тестировщик выявит требования, не поддающиеся проверке, а разработчик сумеет определить требования, которые технически невыполнимы.
- ✓ **Люди, отвечающие за работу продуктов, взаимодействующих с проверяемым элементом.** Они выявят проблемы с требованиями к внешнему интерфейсу. Они также могут выявить требования, изменение которых в проверяемой спецификации повлияет на другие системы (эффект волны).

Все участники экспертизы, включая автора, ищут недостатки и возможности улучшения.

Пример из книги Вегерса: в Chemical Tracking System представители пользователей проводили неофициальное рецензирование последней версии спецификации требований после каждого семинара, посвященного выявлению требований. Таким образом удавалось выявить множество ошибок. После завершения выявления требований, один аналитик свел всю информацию, полученную от всех классов пользователей в одну спецификацию требований к ПО — 50 страниц плюс несколько приложений. Затем **два бизнес-аналитика, один разработчик, три сторонника продукта, менеджер проекта и один тестировщик** обследовали этот документ за **три двухчасовых совещания**, проведенных в течение недели. Они дополнительно обнаружили **233 ошибки, в том числе десятки серьезных дефектов**.

Предотвращение неопределенности

Качество требований определяет читатель, а не автор. Бизнес-аналитик может считать, что написанное им требование кристально-ясное, свободно от неоднозначностей и других проблем. Но если у читателя возникают вопросы, требование нуждается в дополнительном совершенствовании. **Рецензирование — лучший способ поиска мест, где требования непонятны всем целевым аудиториям**.

Одной из причин возникновения неопределенности в требованиях является то, что требования пишутся на **естественном языке** (natural language), в отличии от **формального языка** (formal language), в котором двусмысленность исключена.

Формальный язык – это язык, который характеризуется точными правилами построения выражений и их понимания. К формальным языкам относятся формулы, блок-схемы, алгоритмы и т.д.

Естественный язык — это язык, который используется в повседневной жизни прежде всего для общения, и имеет целый ряд особенностей, к примеру, слова могут иметь более одного значения, иметь синонимы или быть омонимами, а иногда значения отдельных слов и выражений зависят не только от них самих, но и от контекста, в котором они употребляются.

Все это в совокупности приводит к тому, что **причиной возникновения неопределенности как раз и является естественный язык.**

Предотвращение неопределенности

Требования, изложенные неясным языком, не поддаются проверке, поэтому нужно избегать двусмысленных и субъективных терминов. В таблице (*табл. 11-2 стр. 250 в книге Разработка требований к программному обеспечению Карла Вигерса*) перечислены многие из них, а также приводятся рекомендации, как исправить такие неясности.

Ниже приведены отдельные части из этой таблицы:

Неоднозначные термины	Способы улучшения
Приемлемый, адекватный	Определите, что понимается под приемлемостью и как система это может оценить
И/или	Укажите точно, что имеется в виду — «и» или «или», чтобы не заставлять читателя гадать
Между, от X до Y	Укажите, входят ли конечные точки в диапазон
Зависит от	Определите природу зависимости. Обеспечивает ли другая система ввод данных в вашу систему, надо ли установить другое ПО до запуска вашей системы и зависит ли ваша система от другой при выполнении определенных расчетов или служб?

Неоднозначные термины	Способы улучшения
Эффективный	Определите, насколько эффективно система использует ресурсы, насколько быстро она выполняет определенные операции и как быстро пользователи с ее помощью могут выполнять определенные задачи
Быстрый, скорый, моментальный	Укажите минимальное приемлемое время, за которое система выполняет определенное действие
Гибкий, универсальный	Опишите способы адаптации системы в ответ на изменения условий работы, платформ или бизнес-потребностей
Улучшенный, лучший, более быстрый, превосходный, более качественный	Определите количественно, насколько лучше или быстрее должны стать показатели в определенной функциональной области или аспект качества
Обычно, в идеальном варианте	Опишите нештатные или неидеальные условия и как система должна вести себя в таких ситуациях
Необязательно	Укажите, кто делает выбор: система, пользователь или разработчик
Возможно, желательно, должно	Должно или не должно?
Разумный, при необходимости, когда уместно, по возможности	Объясните четко, как разработчик или пользователь должен оценивать разумность и уместность

Неоднозначные термины	Способы улучшения
Устойчивый к сбоям	Определите, как система должны обрабатывать исключения и реагировать на неожиданные условия работы
Цельный, прозрачный, корректный	Что означает «цельный» или «корректный» для пользователя? Выразите ожидания пользователя, применяя характеристики продукта, которые можно наблюдать
Несколько, некоторые, много, немного, множественный	Укажите сколько или задайте минимальную и максимальную границы диапазона
Не следует	Старайтесь формулировать требования в позитивной форме, описывая, что именно система будет делать
Современный	Поясните этот термин для заинтересованного лица
Достаточный	Укажите, какая степень чего-либо свидетельствует о достаточности
Поддерживает, позволяет	Дайте точное определение, из каких действий системы состоит «выполнение» конкретной возможности
Дружественный, простой, легкий	Опишите системные характеристики, которые будут отвечать потребностям пользователей и его ожиданиям, касающимся легкости и простоты использования продукта

Пограничные значения

Много неоднозначности возникает на границах числовых диапазонов как в требованиях, так и бизнес-правилах. Пример:

> *Отпуск длительностью до 5 дней не требует одобрения. Запросы на отпуск длительностью от 5 до 10 дней требуют одобрения непосредственного начальника. Отпуска длительностью 10 дней и более требуют одобрения директора.*

При такой формулировке непонятно, в какую категорию попадают отпуска длительностью точно 5 и 10 дней. Слова «от и до», «включительно» и «свыше» вносят четкость и ясность насчет пограничных значений:

> *Отпуск длительностью 5 дней и меньше не требует одобрения. Запросы на отпуск длительностью более 5 и до 10 дней включительно требуют одобрения непосредственного начальника. Отпуска длительностью свыше 10 дней требуют одобрения директора.*

Негативные требования

Иногда люди пишут в требованиях не то, система должна, а то, что она не должна делать. Как реализовать такие «негативные» требования? Особенно сложны в расшифровке двойные и тройные отрицания. Попытайтесь переформулировать негативные требования в позитивном стиле, которые описывает ограничение на поведение. Вот пример:

> *Пользователь не должен иметь возможность активизировать договор, если он не сбалансирован.*

Лучше перефразировать это двойное отрицание («не должен» и «не сбалансирован») как позитивное утверждение:

> *Система должна позволять пользователю активировать договор, только если этот договор сбалансирован*

Слова, которые приводят к двусмысленности

К словам, которые могут приводить к двусмысленности, относятся:

- соединительный союз «*и*»;
- разделительный союз «*или*»;
- пояснительные союзы «*также*» и «*только*»;
- слова «*по возможности*», «*при необходимости*».

Это далеко не весь список. Однако мне бы хотелось рассмотреть именно его, поскольку без этих слов иногда не обойтись, но применять их нужно очень осторожно.

Соединительный союз «и»

Этот союз может означать, что:

- одно действие или объект в хронологическом порядке следует за другим действием или объектом;
- одно действие является результатом другого действия;
- действие или объект зависит от предыдущего действия или объекта.

На одном из проектов аналитиком было прописано требование:ы «*Все пользователи для входа в систему используют логин и пароль*». При этом, подразумевалось, что «У каждого пользователя для входа в систему есть свой уникальный логин и пароль». Но при прочтении требований разработчик интерпретировал это требование как «Все пользователи используют один логин и пароль для входа в систему».

Пояснительные союзы «также» и «только»

Могут вносить неоднозначность в сочетании с «*и*» и «*или*», образуя сочетания:

только…или;

также…и.

Например, «*Только администратор или координатор могут заблокировать пользователя*». Возможные трактовки данного требования:

- один администратор и любой координатор вместе могут заблокировать пользователя;
- каждый администратор может заблокировать пользователя;
- каждый координатор может заблокировать пользователя.

Слова «по возможности», «при необходимости»

Использование этих слов вносит не только двусмысленность в написанное, но еще делают требования не тестируемым, поскольку неясно, в какой момент эта «возможность» или «необходимость» должна появиться.

«*При необходимости пользователь может распечатать счет*». Данное требования не дает точной информации, когда должна быть доступна эта функция.

Еще больше примеров: Как избежать двусмысленности в требованиях (analyst.by)