Лекция 3

- Критерии качества требований
- 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 (Приоритизированное)
Understandable (Понятное)

BABOK – руководство по своду знаний по бизнес-анализу

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.

Не смотря на различия формулировок можно проследить сходство.

https://habr.com/ru/articles/842296/

1. ПОЛНОТА (или ЗАВЕРШЕННОСТЬ).
Каждое требование должно содержать всю информацию, необходимую для его понимания, не оставлять пробелов или недомолвок. Этот критерий относится и к бизнес- и к техническим требованиям.

Пример 1: *Пользователь может загрузить данные в формате PDF, CSV и т.д.*
Неясно, какие ещё форматы поддерживаются помимо указанных. Что подразумевается под «и т.д.»?

Полное требование: "Пользователь может загрузить данные в форматах PDF, CSV и JSON."

Пример 2: *Приложение должно работать на всех популярных мобильных платформах.*
Неясно, какие платформы считаются «популярными».

Полное требование: "Приложение должно поддерживаться на платформах Android версии 10.0 и выше, iOS версии 13.0 и выше."

Пример 3: *Система должна обрабатывать запросы с высокой производительностью.*
Не указано, что конкретно подразумевается под «высокой производительностью» (какие показатели важны).

Полное требование: "Система должна обрабатывать до 1000 запросов в секунду с задержкой не более 200 миллисекунд."

2. НЕПРОТИВОРЕЧИВОСТЬ (или СОГЛАСОВАННОСТЬ).
Требования не должны противоречить друг другу. Обнаружение несоответствий может быть крайне затруднительным, если требования к одной и той же функциональности продукта находятся в разных местах.

Пример 1:

- Пользователь должен быть автоматически разлогинен после 15 минут бездействия.

- Пользователь должен оставаться в системе до тех пор, пока не завершит редактирование документа.

Противоречие: Первое требование подразумевает автоматический выход по истечении времени, а другое не допускает выхода во время работы над документом.

Непротиворечивые требования: Пользователь должен быть выведен из системы через 15 минут бездействия, за исключением времени активного редактирования документа.

Пример 2:

- Отчет должен генерироваться и отправляться пользователю мгновенно.

- Генерация отчета может занимать до 30 минут в зависимости от объема данных.

Непротиворечивые требования: Отчет должен генерироваться мгновенно при объеме данных до 200 килобайт и отправляться в течение 30 минут при большем объеме.

3. КОРРЕКТНОСТЬ

Требования должны полностью удовлетворять нужды заинтересованных сторон и отвечать их запросам.

Пример 1: *Пользователь должен быть в состоянии настроить двухфакторную аутентификацию для повышения безопасности*

> Это требование к пользователю, а не к программе. Мы делаем продукт и можем диктовать ему требования, но не можем контролировать состояние его пользователей.

Корректное требование: Приложение должно поддерживать двухфакторную аутентификацию и предоставлять пользователю возможность включить эту функцию в настройках безопасности.

Пример 2: *После завершения загрузки файла пользователь должен увидеть пагинацию с сообщением о том, что загрузка прошла успешно.*

> Пагинация относится к разбиению контента на страницы, что не имеет смысла в контексте завершения загрузки файла. Вместо этого должно быть указано, что пользователь должен увидеть сообщение о том, что загрузка прошла успешно.

Корректное требование: После завершения загрузки файла пользователь должен увидеть уведомление о том, что загрузка прошла успешно.

4. **НЕДВУСМЫСЛЕННОСТЬ** (или ОДНОЗНАЧНОСТЬ)

Разговорный язык может содержать двусмысленности, которые обычно проявляются в двух формах: когда одно требование можно интерпретировать по-разному и когда разные люди понимают одно и то же требование по-своему. Важно, чтобы требования были сформулированы однозначно, без жаргона, аббревиатур и неясных фраз, чтобы не возникало различных трактовок.

Пример:

Сервис должен быть интегрирован с СК.

Что значит СК? Система Криптографии? Сервер Коммуникаций? Система Качества? Служба Консультирования? А что, если в компании есть все эти сервисы?

Однозначное требование:

Сервис должен быть интегрирован с Системой Контроля (СК), предоставляющей разграничение управления доступом.

5. ВЫПОЛНИМОСТЬ (или ОСУЩЕСТВИМОСТЬ).

Выполнимость требований в проекте определяется их возможностью быть реализованными с учетом технических, бюджетных и временных ограничений. Для оценки выполнимости можно использовать инкрементальную разработку и прототипы. Если требование невозможно выполнить, важно учитывать его влияние на проект и принимать решения о его исключении.

Примеры:

1. *Приложение должно распознавать и выполнять мысленные команды пользователей.*
2. *Приложение должно распознавать эмоции, по тексту и автоматически подбирать цветовой фон сообщения, соответствующий настроению пользователя.*

6. ПРОВЕРЯЕМОСТЬ (или ТЕСТИРУЕМОСТЬ).

Тестируемость требований означает их способность быть проверенными через объективные тест-кейсы, которые ясно показывают правильность реализации. Неполные, несогласованные или двусмысленные требования не поддаются проверке.

Пример 1: *Скидка Премиум-пользователя в размере 10% применяется по возможности к заказам пользователя, имеющего статус Премиум.*

Неясно, что означает "по возможности" и как проверить наступление или отсутствие этой возможности.

Проверяемое требование: Скидка Премиум-пользователя в размере 10% автоматически применяется ко всем заказам Премиум-пользователей.

Пример 2: *Система должна обладать высокой доступностью.* (Как это проверить ?)

Проверяемое требование: SLA * 99,99%

* 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.

7. ПРИОРИТИЗИРОВАННОСТЬ (или УПОРЯДОЧЕННОСТЬ).

Требования должны быть упорядочены по важности, стабильности и срочности:

- **важность** определяет, насколько успех проекта зависит от выполнения требования;
- **стабильность** показывает, насколько вероятность изменений в требовании мала;
- **срочность** влияет на распределение усилий команды по времени.

Неверно расставленные приоритеты могут привести к неэффективному распределению ресурсов, выполнению ненужной работы и нарушению сроков.

Пример

1. Пользователь должен иметь возможность переключаться между светлой и темной темой интерфейса.

2. Система платежей должна быть интегрирована с новым партнером. API партнера может меняться по мере разработки.

3. Пользователь должен иметь возможность просмотреть информацию о своих заказах

Нарушена упорядоченность по важности и срочности, так как последнее требование намного полезнее для пользователя, чем наличие светлой и темной темы. Нарушена упорядоченность по стабильности, так как интеграция с системой, чья API может изменяться, создает высокую степень неопределенности и риск ненадежной работы функции, что может потребовать значительных доработок и тестирований. Реализацию этого требования можно отложить до релиза стабильной версии API партнером.

8. Атомарность (или "единичность")

Каждое требование должно быть самодостаточным и описывать только одну ситуацию. Если требование можно разбить на несколько независимых, оно перестаёт быть атомарным. Проблемы с атомарностью возникают, когда в одном требовании описываются разные элементы интерфейса или разные состояния/эффекты от действий пользователя, что затрудняет его понимание и приводит к путанице.

Пример:

На странице отчёта должно отображаться имя пользователя, а также должна быть возможность скачивания PDF-версии отчёта.

Это требование описывает два разных элемента интерфейса и две разные функции программы: отображение имени пользователя и возможность скачать отчет.

А вот пример атомарных требований:

- На странице отчёта должно отображаться имя пользователя.

- На странице отчёта должна быть кнопка для скачивания PDF-версии отчёта.

9. Необходимость (или "обязательность")

Каждое требование должно приносить реальную пользу бизнесу, выделять продукт на рынке или обеспечивать соблюдение стандартов и правил. Если какое-то требование устарело, было замещено другим или просто не обязательно для реализации, его необходимо удалить из набора требований.

Пример 1:

Первоначальное требование: *"Пользователи должны иметь возможность вернуться на предыдущую страницу с помощью кнопки «Назад»."*

Новое требование: "*Пользователи должны иметь возможность вернуться на предыдущую страницу с помощью жеста свайпа*."

Кнопка «Назад» становится ненужной после внедрения жеста. Так как жест выполняет ту же функцию, но удобнее и соответствует современным стандартам пользовательского интерфейса. После удаления кнопки "Назад" должно быть удалено и требование о ней. Иначе будет путаница.

Пример 2 : "*Приложение должно поддерживать браузер Internet Explorer 10.*" Это требование стало устаревшим и должно быть удалено, так как Internet Explorer 10 больше не используется и не поддерживается разработчиками.

10. Прослеживаемость (или "трассируемость")

Требования должны быть оформлены в структурированном виде и, в идеале, иметь уникальные идентификаторы. В контексте тестирования прослеживаемые (или "трассируемые") требования — это те, которые удобно связать с тестами. Для этого используются матрицы трассировки, которые будут рассмотрены позже.

"Приложение должно обеспечивать высокую безопасность, поддерживать различные уровни доступа для пользователей и администраторов, иметь возможность интеграции с внешними системами, такими как системы управления проектами; также оно должно работать на мобильных устройствах и поддерживать уведомления. Пользовательский интерфейс должен быть интуитивно понятным, с возможностью настройки тем, а производительность должна быть достаточной для работы с большими объемами данных. Важно обеспечить стабильную работу системы при одновременном подключении большого количества пользователей."

У такого полотна требований отсутствует прослеживаемость, потому что:
- нет структуры, выделяющей функциональные и нефункциональные требования;
- отсутствует нумерация требований;
- несколько требований смешаны в одном абзаце, что затрудняет отслеживание и реализацию;
- нет четкого разделения по категориям (безопасность, интерфейс, производительность и т.д.).

11. Модифицируемость

Модифицируемость требований подразумевает легкость их изменения. Если требования к продукту разбросаны по разным хранилищам или одно и то же требование встречается в нескольких местах, это значительно усложняет их изменение. Хранение требований в единой базе и использование уникальных идентификаторов помогают избежать избыточности и облегчить управление изменениями.

Пример:

В разделе "Авторизация" в Notion: "Все пользователи системы должны проходить двухфакторную аутентификацию для доступа к личным данным."

В разделе "Безопасность" в Confluence: "Все пользователи системы должны проходить двухфакторную аутентификацию для доступа к личным данным."

Плохо: Одно и то же требование описано как в разделе безопасности, так и в разделе авторизации, ещё и в двух разных хранилищах. При изменении требования в одном месте (например, добавлении новых условий аутентификации) возникает большой риск появления противоречий с другими частями требований. Чтобы внести изменения, придётся прочесывать все хранилища и перечитывать требования, а затем изменять их в нескольких местах. Это увеличивает трудозатраты и время на внесение изменений.

12. Понятность

Понятность определяется тем, насколько легко требования понимает целевая аудитория. Требования должны быть написаны с использованием терминологии, знакомой всем членам команды, которая их использует в работе. Это позволит избежать недоразумений и неправильной интерпретации. Если требования описаны неясно или с использованием специального жаргона, который не является общепринятым, это усложняет их понимание, выполнение и замедляет онбординг новых сотрудников.

Пример:

"Сайт должен быть способен к улучшению взаимодействия через динамическое управление параметрами."

Фразы "улучшение взаимодействия" и "динамическое управление параметрами" слишком абстрактны и не конкретизируют, какие аспекты взаимодействия нужно улучшить и какие параметры должны быть динамически управляемыми. Требование не несёт полезной информации.

Понятное требование:

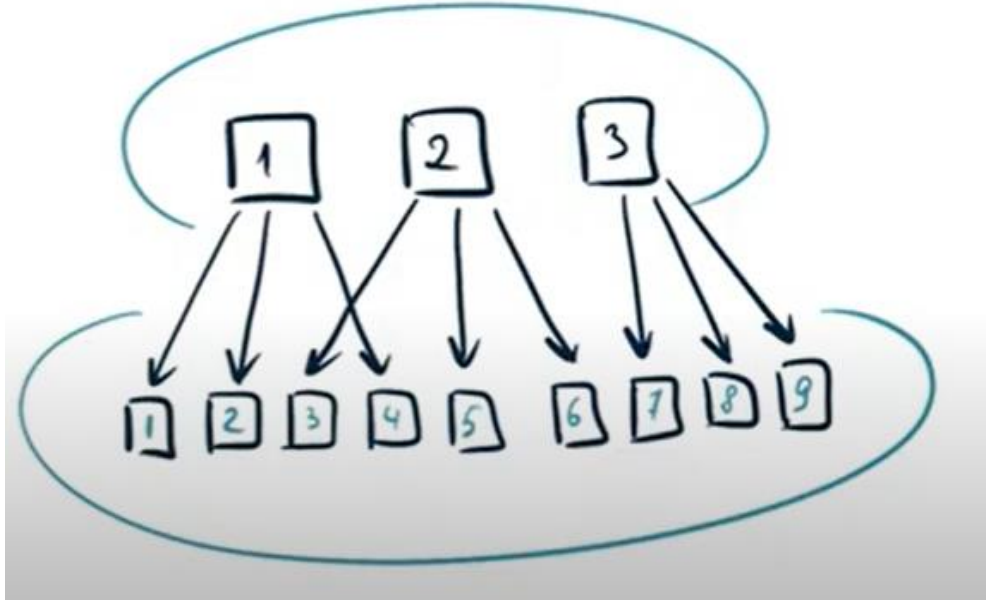
Сайт должен предоставлять возможность настройки пользовательского интерфейса в реальном времени: переключение светлой и тёмной темы, три варианта величины шрифта, переключение в режим высокой контрастности."

Трассировка требований (прослеживаемость)

Трассировка — это способ представления отношений между требованиями различного уровня в системе, помогающий определить источник любого требования.

Трассируемость – это связь требования с уровнем выше и уровнем ниже.

Бизнес требования Заказчика



Требования нижнего уровня (спецификация, ТЗ)

Трассировка требований позволяет:

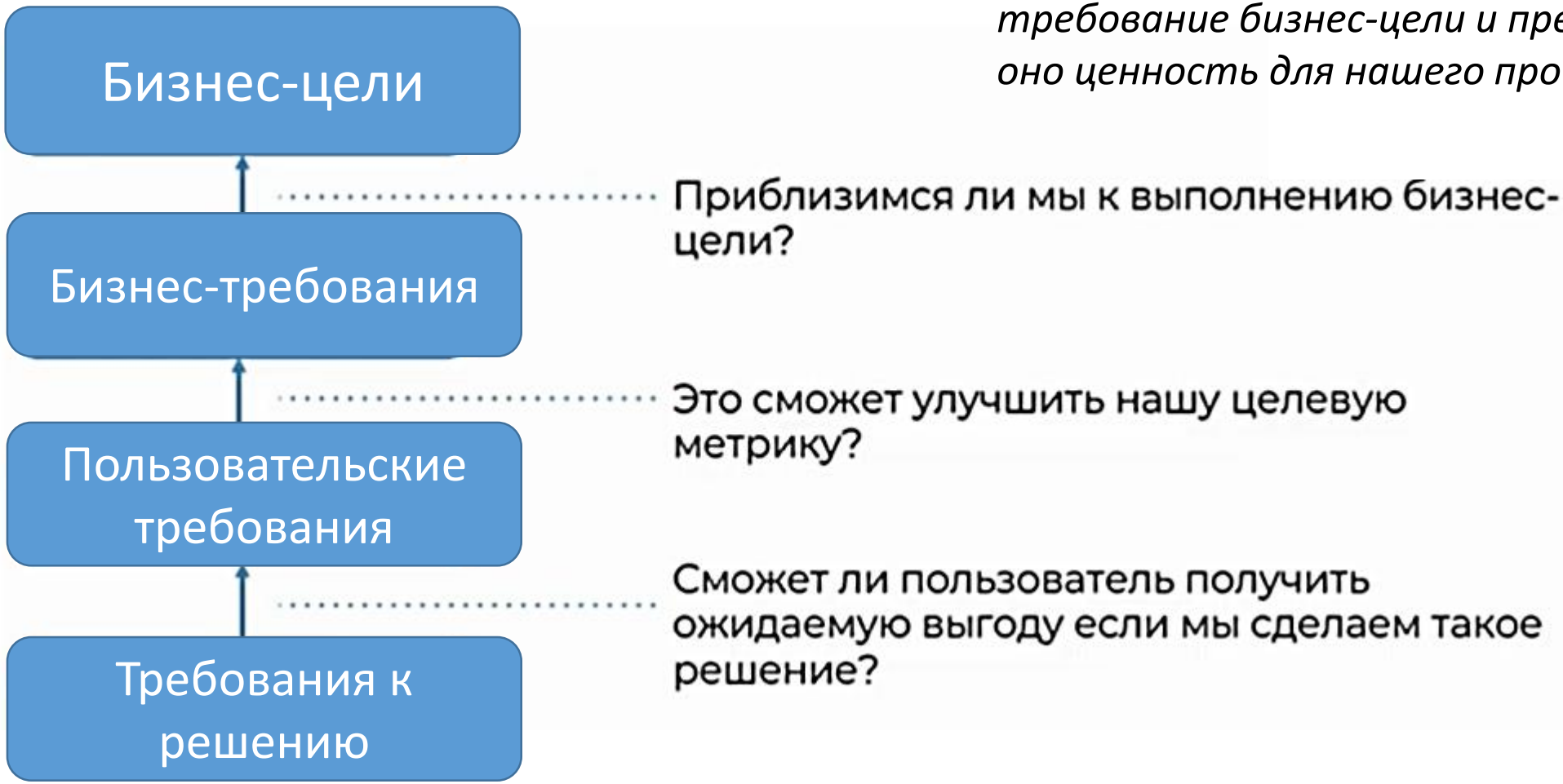
- подтвердить, что реализация удовлетворяет всем требованиям: все, что требовал заказчик, было реализовано;
- подтвердить, что приложение делает только то, что было заказано: не реализовано то, что заказчик никогда не просил;

Зачем нужна трассировка требований?

https://www.youtube.com/watch?v=lU-2r89pgvA

- Анализ важности новых требований

При появлении новых требований можно проанализировать удовлетворяет ли новое требование бизнес-цели и представляет ли оно ценность для нашего продукта



Зачем нужна трассировка требований?

- Анализ изменений

При изменении какого-либо требования верхнего уровня по трассировке можно проанализировать какие требования нижнего уровня изменятся и какие изменения нужно в них внести, что бы обеспечить изменение требования верхнего уровня

- Управление приоритетом

Пример:

У проекта 3 бизнес-цели

У каждой бизнес-цели по 4 бизнес требования

У каждого бизнес-требования по 8 пользовательских историй

96 пользовательских историй

Как приоритизировать такое количество требований ?

Пользовательские истории можно приоритизировать по приоритету связанных с ними бизнес-целей

Матрица трассировки требований (Traceability matrix)

Является документом, зачастую в форме таблицы, в котором визуально показано соотношения между двумя утвержденными базовыми документами, для определения полноты взаимосвязей между их отдельными элементами.

Пример составления TRACEABILITY MATRIX (для части функционала Калькулятора)

ЭТАП 1: Составляем нумерованный список требований к приложению (для примера возьмём несколько существующих функций калькулятора)

ID требования	*Формулировка требования*
Rq1	Ввод цифр и действий. Можно производить вычисления, нажимая на кнопки калькулятора или вводя символы с клавиатуры. Кроме того, доступен ввод цифр и действий с цифровой клавиатуры, когда нажата клавиша NUM LOCK. Также можно вставлять математические выражения из буфера обмена и получать результат (например, набрать в Блокноте «5*3=», скопировать и вставить в Калькулятор, на «экране» которого появится ответ «15»).
Rq2	Очистка экрана / Удаление символов. Полная очистка экрана осуществляется при нажатии на кнопку «C» калькулятора или клавишу Delete на клавиатуре. Последний введённый числовой символ можно удалить, используя клавишу Backspace на клавиатуре.
Rq3	Калькулятор можно использовать для выполнения операций сложения.

ЭТАП 2: Формируем список тестов – это может быть список названий детально расписанных пошаговых тест-кейсов или же чек-лист проверок

ID теста	***Проверка***
Th1	Ввод значений и операций при нажатии на кнопки калькулятора
Th2	Ввод цифр и действий с цифровой клавиатуры, когда нажата клавиша NUM LOCK
Th3	Ввод цифровых значений с клавиатуры
Th4	Вставка математических выражений из буфера обмена (на одно, два, максимально возможное количество действий)
Th5	Очистка экрана нажатием на кнопку «C» калькулятора
Th6	Очистка экрана нажатием на клавишу Delete калькулятора
Th7	Удаление последнего введённого символа нажатием на клавишу Backspace на клавиатуре
Th8	Сложение однозначных, двузначных, многозначных чисел
Th9	Сложение десятичных чисел
Th10	Сложение отрицательных чисел

ЭТАП 3: Составляем TRACEABILITY MATRIX, которая может быть представлена разными способами

TRACEABILITY MATRIX	
ID требования	*ID теста*
Rq1	Th1
	Th2
	Th3
	Th4
Rq2	Th5
	Th6
	Th7
Rq3	Th8
	Th9
	Th10

TRACEABILITY MATRIX												
	ID теста	Th1	Th2	Th3	Th4	Th5	Th6	Th7	Th8	Th9	Th10	*Кол-во тестов, которые покрывают требование*
ID требования												
Rq1		X	X	X	X							4
Rq2						X	X	X				3
Rq3									X	X	X	3

Таким образом матрица трассируемости:

- позволяет контролировать реализацию требований, отслеживать, что все требования разработаны и протестированы и ничего не пропущено.
- помогает отслеживать какие именно требования еще не покрыты тест-кейсами.
- используется для контроля измененных требований.
- могут использоваться и заказчиком, что позволяет сделать процесс разработки и тестирования более прозрачным.

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

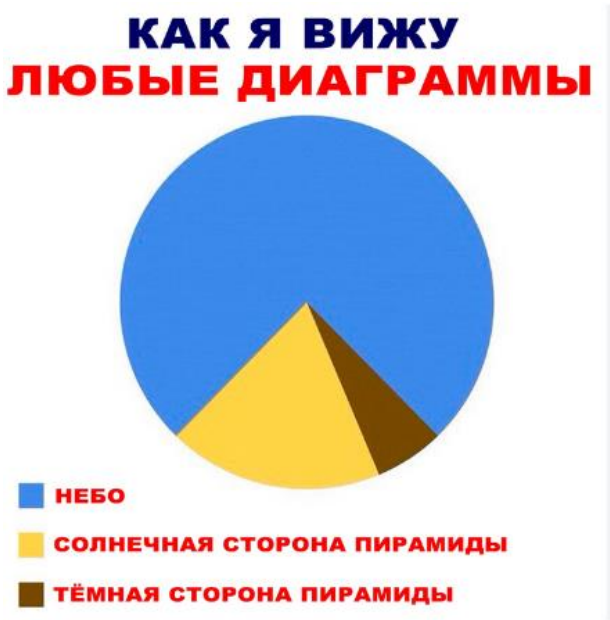
Верификация требований – это проверка, что их описание в виде спецификации (SRS) или ТЗ соответствуют отраслевым и организационным стандартам качества, а также пригодны для дальнейшего практического использования. Последнее означает, *что на основании составленных аналитиком требований ИТ-архитектор 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.

Чем валидация отличается от верификации?

Верификация	Валидация
Делают ли разработчики продукт правильно	Правильный ли получился проект
Все ли функции реализованы	Насколько грамотно реализована функциональность
Предшествует валидации. Включает в себя полноценную проверку правильности написания.	Проводится после верификации. Это – оценка качества итогового проекта.
Испытания организовываются разработчиками	Испытания организованы тестировщиками
Тип анализа – статистический. Проводится сравнение с установленными требованиями к итоговому проекту.	Тип анализа – динамический. Проект проходит испытания по эксплуатации. Это помогает понять, насколько продукт соответствует действующим нормам.
Оценка объективна. Она базируется на соответствии определенным стандартам.	Оценка субъективна. Она является личной. Это – оценка, которую ставит каждый тестировщик.

Простыми словами

Верификация отвечает на вопрос

"Делаем ли мы продукт правильно?",

а валидация- на вопрос

"Делаем ли мы правильный продукт?"

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?

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*

Пример двойного отрицания:

Be positive & avoid double negatives

Don't	Do
IF the User has typed-in **at least one character** into the field, THEN the system does **NOT show** the validation message.	IF the User has left the field **blank**, THEN the system **shows** the validation message.

Если пользователь ввел хотя бы один символ в поле, тогда система не показывает валидационное сообщение. Возникает вопрос, а что тогда должна желать система, если она не показывает? Правильно написать: Если пользователь оставил поле пустым, тогда система показывает валидационное сообщение

IF the Customer has **selected NO** Products, THEN the system does **NOT allow** to make the Order.	IF the Customer has **selected** at least one Product, THEN the system **allows** to make the Order.

Если пользователь не выбрал ни одного продукта, тогда система не позволяет сделать заказ.
Лучше: Если пользователь выделил хотя бы один продукт, тогда система позволяет сделать заказ.

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 и 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)

Gherkin - человеко-читаемый язык для описания поведения системы. Каждая строчка начинается с одного из ключевых слов и описывает один из шагов. **Такая формализация помогает исключить неопределенности.**

Don't	**Do**
The system **applies the Discount Code** IF the Customer submits one. The system does it ONLY IF the Customer has been **verified**.	GIVEN the Customer views *New Order*, GIVEN the Customer has been **verified**, IF the Customer submits the Discount Code, THEN the system **applies the Discount Code**.
IF the **Phone Number** is **new**, THEN the system **sends a Verification Message**.	GIVEN the User has **Notification Method** set to "**SMS**", IF the User updates their Profile AND saves a **new Phone Number**, THEN the system **sends a Verification Message**.