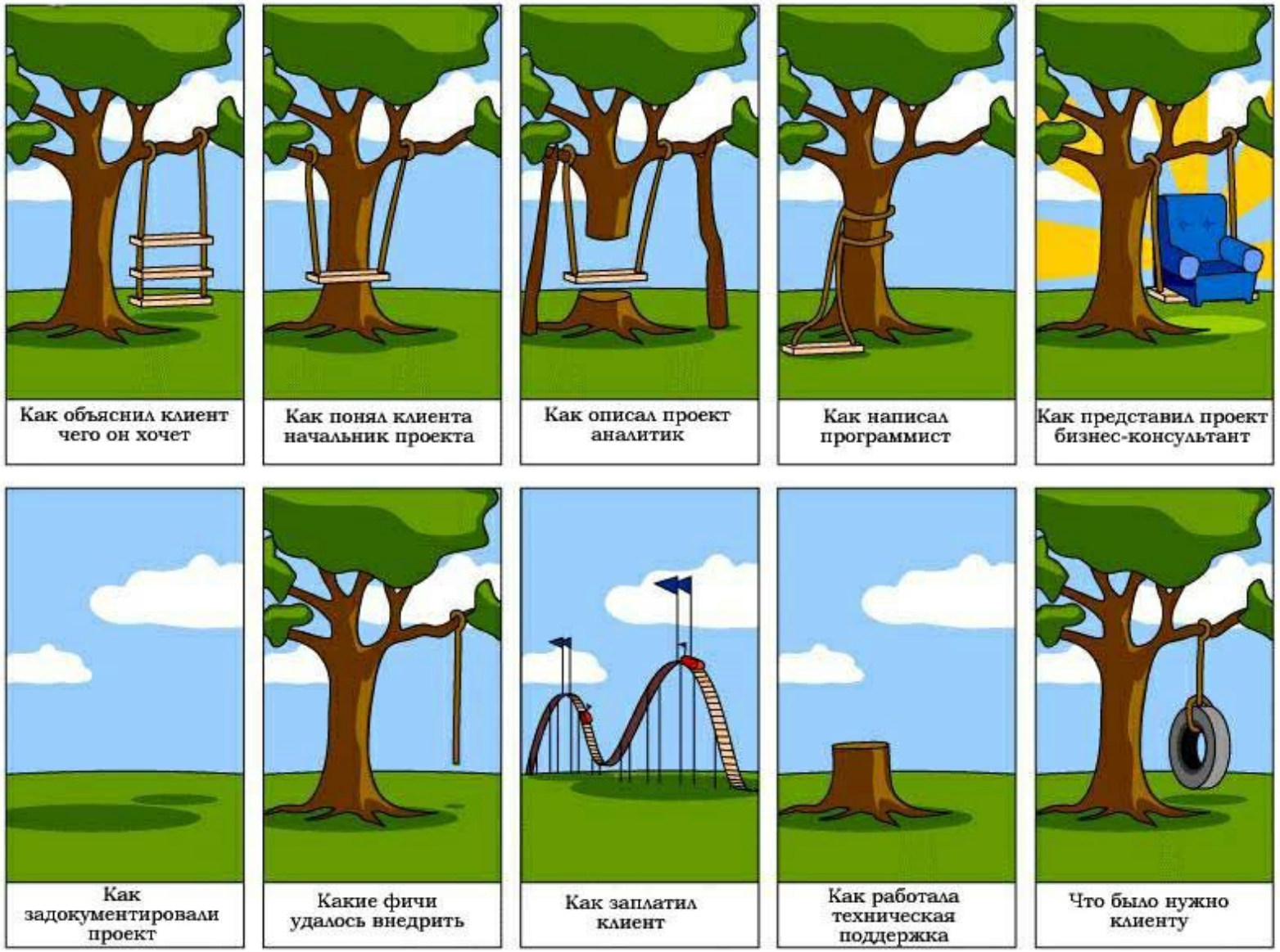
Основы тестирования ПО

Тестирование документации и требований

Важность документации

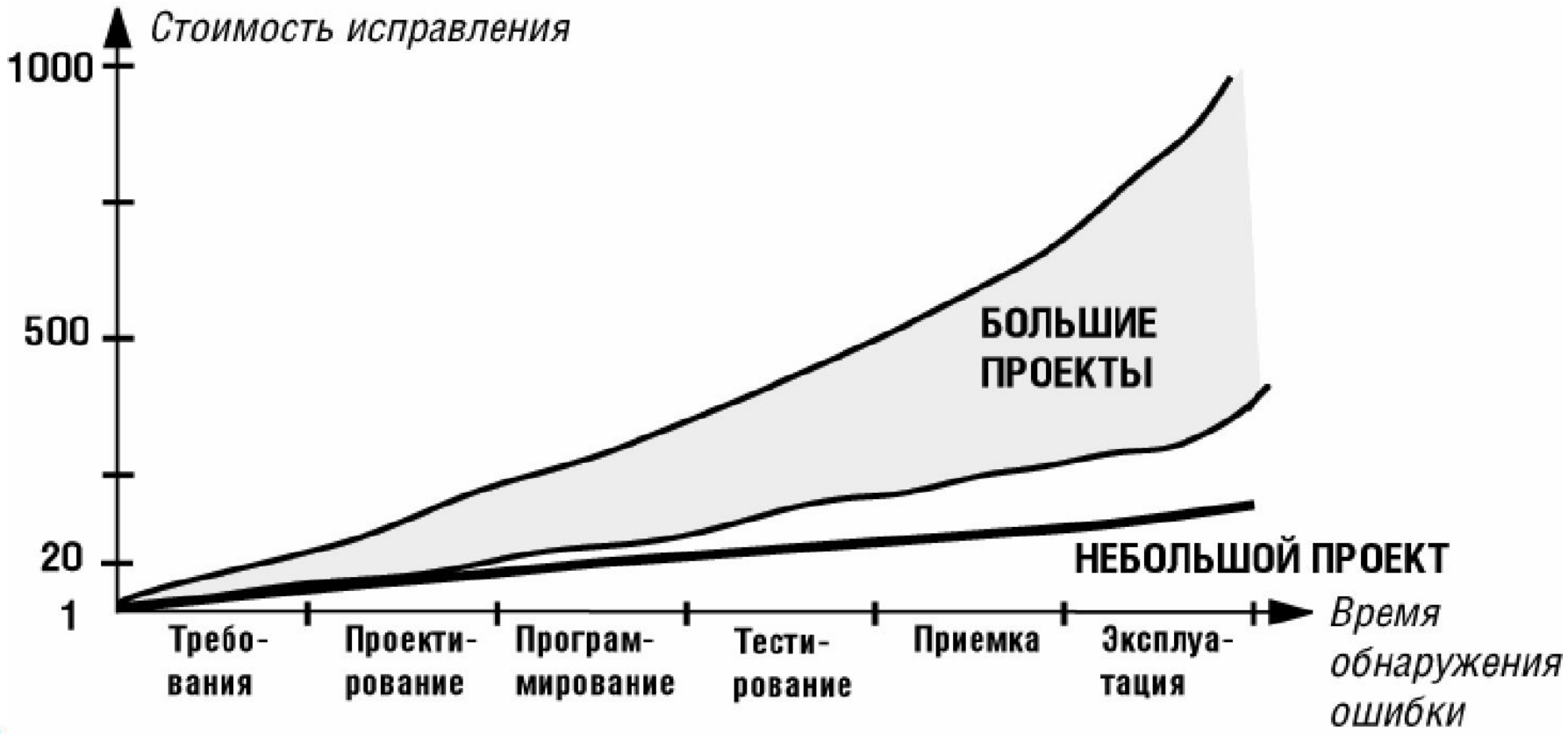


Причины возникновения смысловых конфликтов:

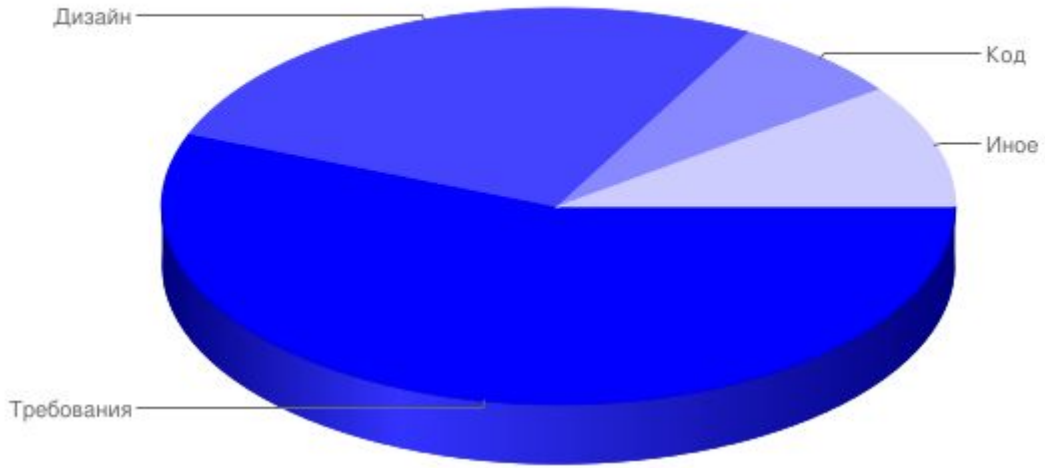
1. то, что заказчик предполагает,
2. то, что заказчик хочет сказать,
3. то, что заказчик, как ему кажется, говорит,
4. то, что заказчик говорит на самом деле,
5. то, что исполнитель хочет услышать,
6. то, что, как кажется исполнителю, он слышит,
7. то, что исполнитель слышит,
8. то, что исполнитель хочет понять,
9. то, что исполнитель понимает,
10. то, что, как исполнителю кажется, он понимаете

Все вышеперечисленное является относительно различной информацией.

Стоимость исправления дефыекта на различных стадиях развития проекта:



Именно в требованиях берёт начало больше всего багов (а не в коде, как думают многие).



Важность тестирования требований:

Хорошо проработанные требования позволяют:

- Выработать общее понимание между заказчиком и разработчиком.
- Определить рамки проекта.
- Более точно определить финансовые и временные характеристики проекта.
- Обезопасить заказчика от риска получить продукт, в котором он не сможет работать.
- Обезопасить разработчика от риска попасть в ситуацию «неконтролируемого размытия границ», которое может привести к непредвиденным затратам ресурсов сверх начальных ожиданий.

Виды тестируемой документации

Проектная документация:

- Требования к программному продукту (product requirements).
- Функциональные спецификации к программному продукту (functional specifications).
- Архитектуру (architecture) и дизайн (design).
- План проекта (project plan) и тестовый план (test plan).
- Тестовые случаи, тестовые сценарии (test cases).

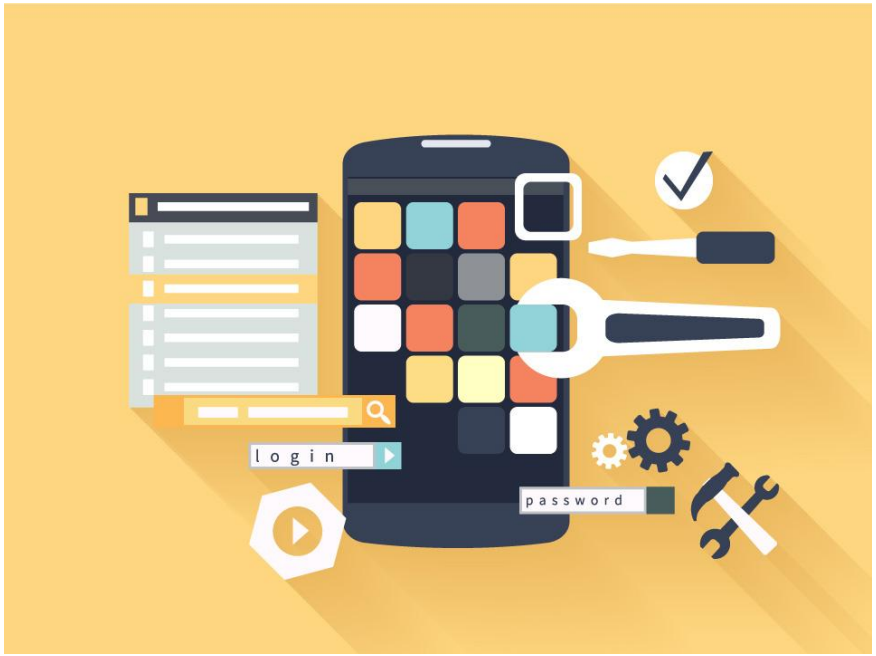
Сопроводительная документация (документация для пользователей):

- Интерактивную помощь (on-line help).
- Руководства по установке (Installation guide) и использованию программного продукта (user manual).

Основные типы требований

Функциональные требования:

Функциональные требования объясняют, что должно быть сделано. Они идентифицируют задачи или действия, которые должны быть выполнены. Функциональные требования определяют действия, которые система должна быть способной выполнить.



Нефункциональные требования:

Нефункциональные требования — требования, определяющие свойства, которые система должна демонстрировать, или ограничения, которые она должна соблюдать, не относящиеся к поведению системы. Например, производительность, удобство сопровождения, расширяемость, надежность, факторы эксплуатации

Уровни требований:

1 - Бизнес требования (общее видение и обзорная документация) - Зачем вообще нужен ПП и что с его помощью будет делаться. Например "Нам нужен инструмент, извлекающих бизнес-информацию из различных источников и представляющий её в виде диаграмм и таблиц"

2 - Пользовательские требования (Use Cases) - Зачем вообще нужен ПП и что с его помощью будет делаться. Например "Нам нужен инструмент, извлекающих бизнес-информацию из различных источников и представляющий её в виде диаграмм и таблиц"

3 - Функциональные и нефункциональные требования (Требования к ПО)

Для корректной работы с требованиями, желательно знать следующие моменты:

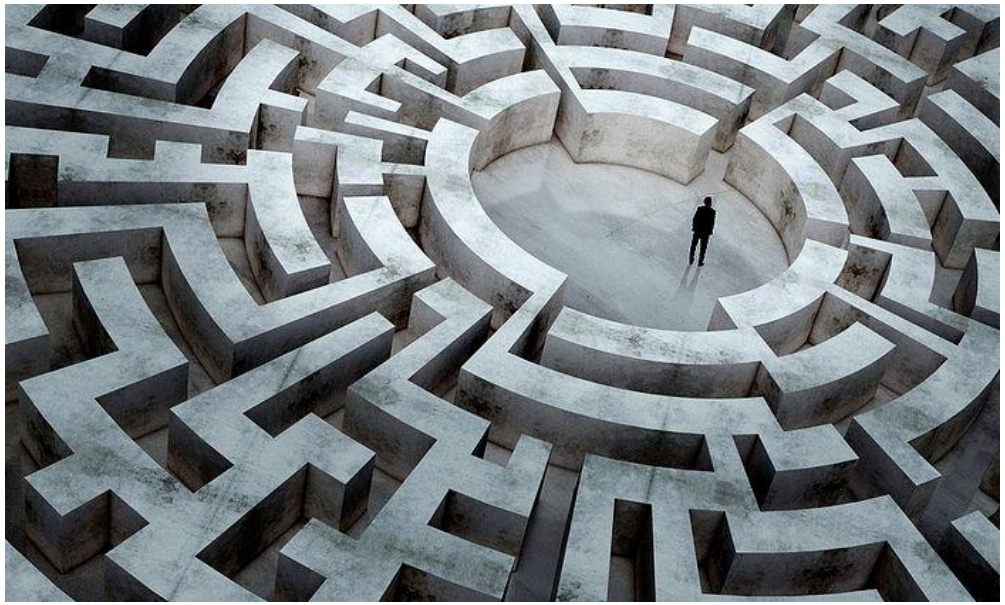
1. Где хранятся требования?
2. Какие источники требований у нас есть?
3. Кто помещает требования в официальное хранилище?
4. Как мы узнаем об изменениях в требованиях?
5. Что более правильно – изначальные спецификации, последующие письма, прототип?
6. Кто утверждает окончательные требования?
7. Как найти новые/измененные требования?

8. Как мы можем предложить внести изменения в требования?
9. Кому мы можем задавать вопросы?
10. Кому и как мы должны сообщить о проблемах с требованиями?
11. Если нам не отвечают, кого спрашивать следующим, кого в конечном итоге?
12. Кто ответственный за работу с требованиями на проекте?

Пути выявления требований:

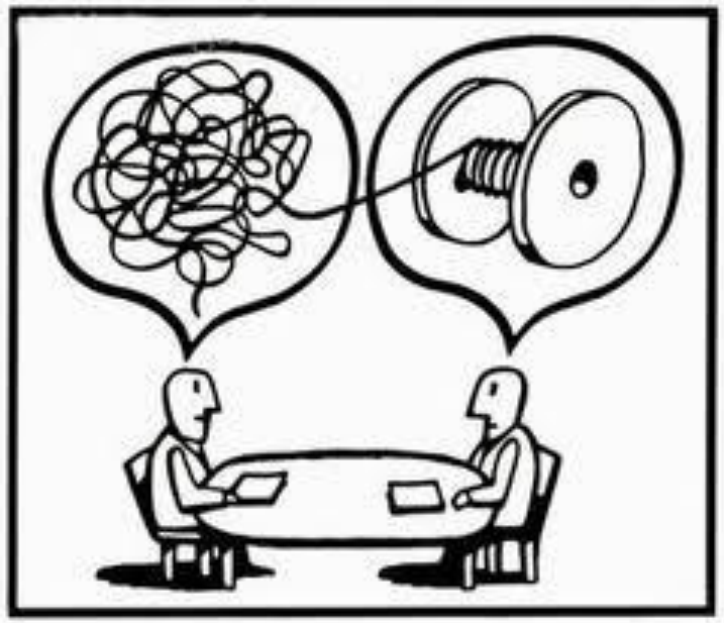
Основными путями выявления требований являются:

- Интервью.
- Наблюдение.
- Самостоятельное описание.
- Семинары.
- Прототипирование.



Свойства корректного требования:

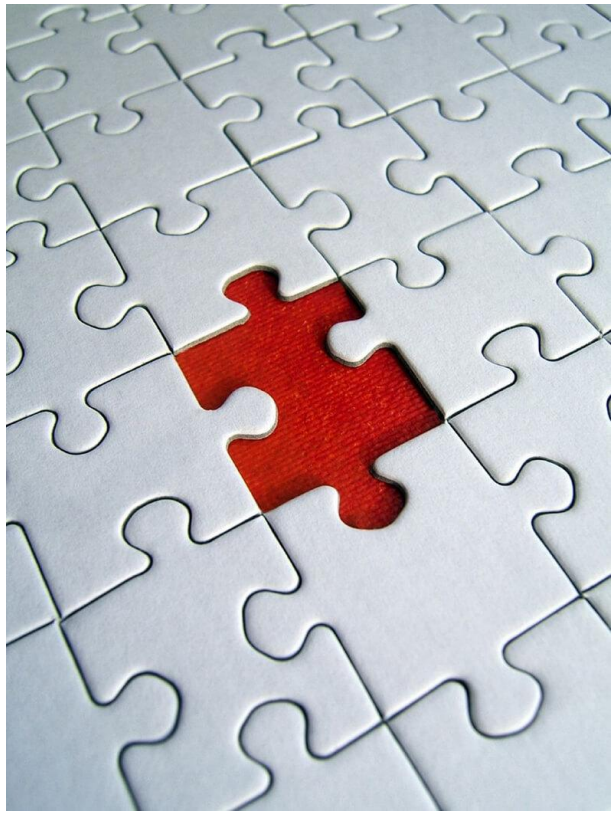
- Завершенность (complete)
- Непротиворечивость (consistent)
- Корректность (correct)
- Недвусмысленность (unambiguouse)
- Проверяемость (verefiable)



Проблемы незавершенности:

Отсутствуют нефункциональные требования или нефункциональные составляющие требования.

Мы знаем, что система должна делать. А про то, как (как быстро, как безопасно) в лучшем случае узнаём только в самом конце.



Решение:

Рекомендуется задавать общие вопросы.
Их преимущества:

- Они универсальные – большая часть их применима к любому проекту, независимо от специфики продукта.
-Они не навязывают решение – больше шансов что, заказчик случайно упомянет то, что для него очевидно (и поэтому он нам об этом раньше не говорил).
- Они не загоняют заказчика в ситуацию, когда ему приходится выбирать из имеющихся вариантов (а на самом деле всё будет по тому варианту, который вы забыли упомянуть).

Хорошая аналогия – вопросы репортера (или маленького ребёнка): «А что?», «А зачем?», «А почему?»

Проблема TBD:

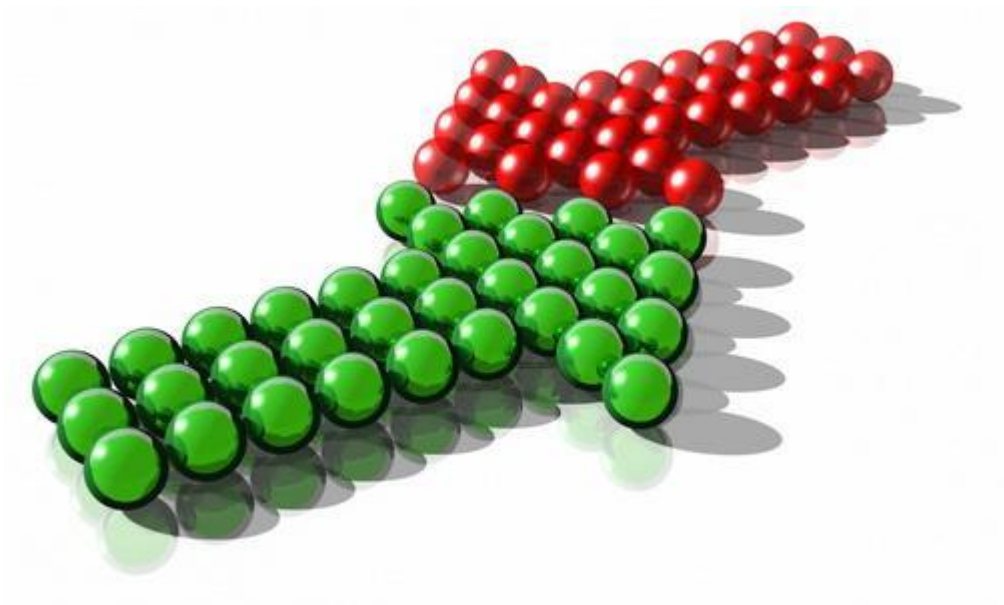
Ещё одной проблемой чрезмерной общности утверждений является т.н. TBD («to be defined», «будет определено»).

Мы можем успокоиться (на время) , только если знаем, кто и когда должен определить эти TBD. И почему они не определены сейчас.

Также бывает, что стороны, ответственные за закрытие TBD, просто забывают об этом. Вывод? Нужно напоминать.

Проблемы противоречивости:

- Противоречия внутри одного требования.
- Противоречия между двумя и более требованиями.
- Противоречия между таблицами и текстом.
- Противоречия между картинкой и текстом.
- Противоречия между требованием и прототипом.

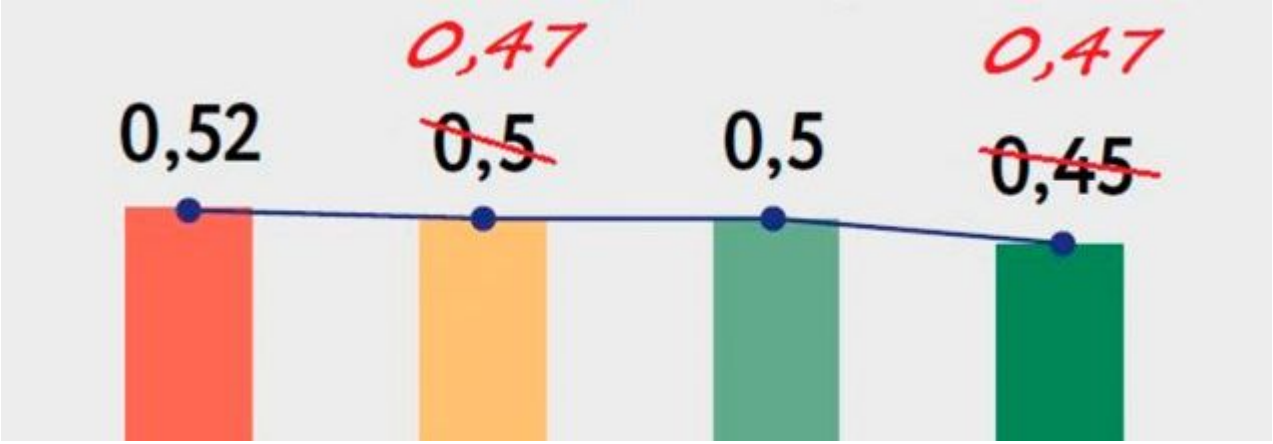


Проблемы некорректности:

Документы часто бывают большими, объёмными, сложными. Они меняются по ходу разработки. Как и любой продукт деятельности человека, документы (и требования в т.ч.) с большой вероятностью содержат ошибки.

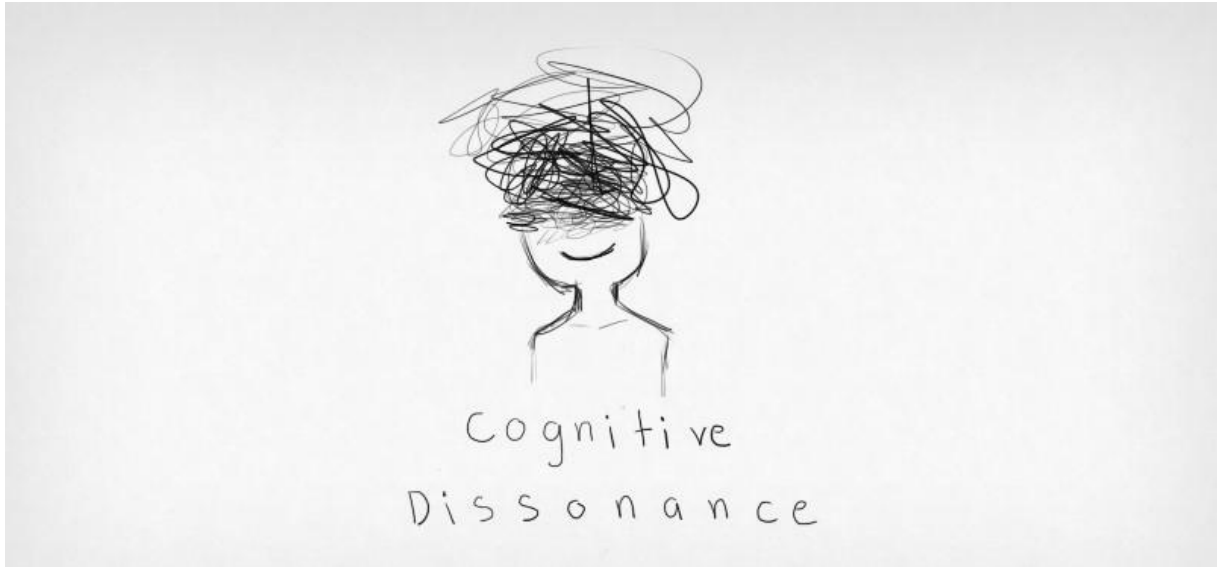
Ошибки могут быть вызваны:

- опечатками, последствиями «copy-paste»;
- остатками устаревших требований;
- наличием «озолочения» («gold plating»);
- наличием технически невыполнимых требований;
- наличием неаргументированных требований к дизайну и архитектуре.



Проблемы двусмысленности:

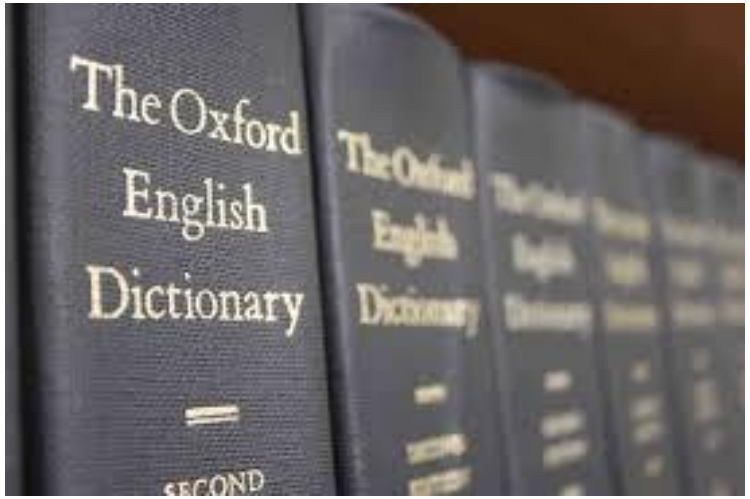
Если что-то можно понять несколькими способами, можно быть уверенным, что разные люди поймут это по-разному.



В английском языке есть много слов-индикаторов, наличие которых в требовании должно насторожить тестировщика.

Adequate, be able to, easy, provide for, as a minimum, be capable of, effective, timely, as applicable, if possible, TBD, as appropriate, if practical, at a minimum, but not limited to, capability of, capability to, normal, minimise, maximise, optimise, rapid, user-friendly, simple, often, usual, large, flexible, robust, state-of-the-art, improved, efficient.

Очевидно, их русские аналоги приводят к тем же проблемам.



Проблемы проверяемости:

Все вышеперечисленные проблемы ведут в том числе к тому, что мы не можем проверить, удовлетворяет ли продукт требованию.

Как понять, что требование непроверяемо? Попробовать придумать несколько тестов для его проверки.

Если мы не можем проверить требование по объективным причинам, мы уточняем его до тех пор, пока оно не станет проверяемым. В зависимости от ситуации, мы расспрашиваем заказчика, разработчиков, своих более опытных коллег и т.д.

Свойства корректного набора требований:

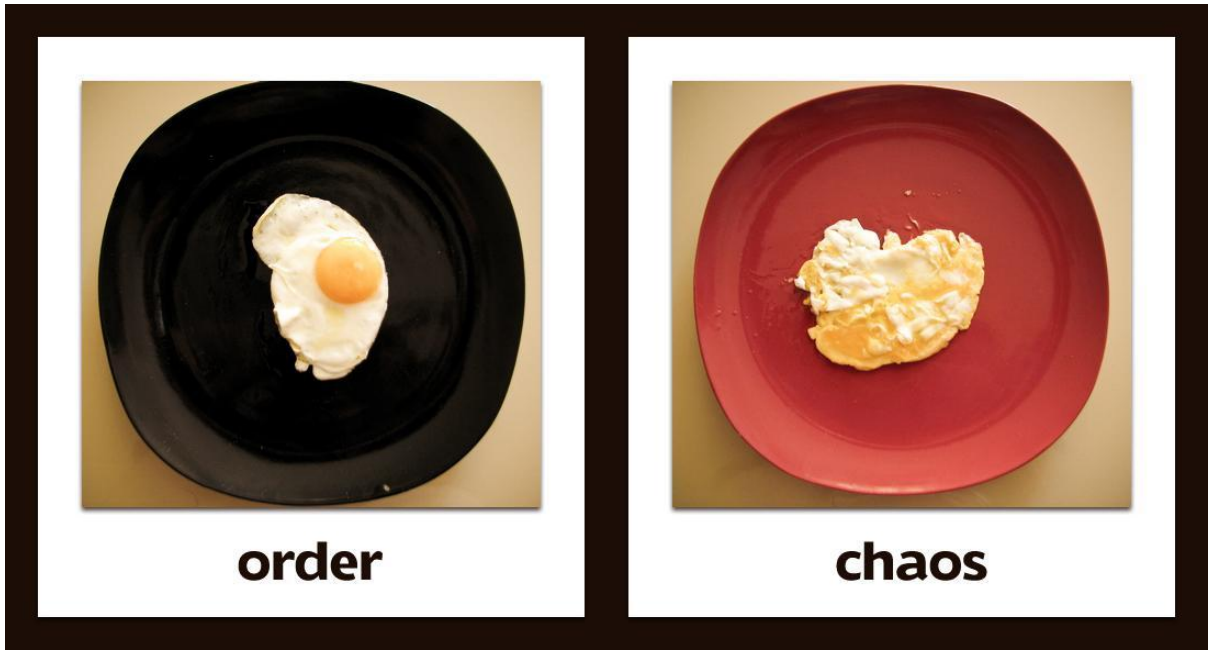
- Модифицируемость (modifiable)
- Прослеживаемость (traceable)
- Проранжированность (ranked for importance, stability and priority)

Проблемы модифицируемости:

После каждого обновления требований понадобится тратить недели чтобы выловить все появившиеся противоречия.

Проблемы непрослеживаемости:

Если требования не пронумерованы, не имеют чёткого оглавления, не имеют работающих перекрёстными ссылок – это хаос. А в хаосе ошибки плодятся с удивительной скоростью.



Проблемы непроранжированности:

Если требования не проранжированы по важности, стабильности и срочности, мы рискуем уделить основное внимание не тому, что на самом деле важно для заказчика.

Варианты :
… по важности (importance)
… по стабильности (stability)
… по срочности (priority)

Техника работы с требованиями

Вопросы

Самый простой и не требующий большого опыта способ – задавать как можно больше вопросов.

- Начинать с маленьких наиболее важных вопросов, сначала заработайте репутацию;
- Попытайтесь найти решение сами;
- 1 общий вопрос вместо 10 мелких;
- Добавьте вежливости и пишите по делу;
- Узнайте особенности проекта, у кого что спрашивать;
- Хорошие личные взаимоотношения помогают;
- Простые предложения, простой язык;
- Попросите коллег перечитать, если вопрос сложный.

Что делать с ответом:

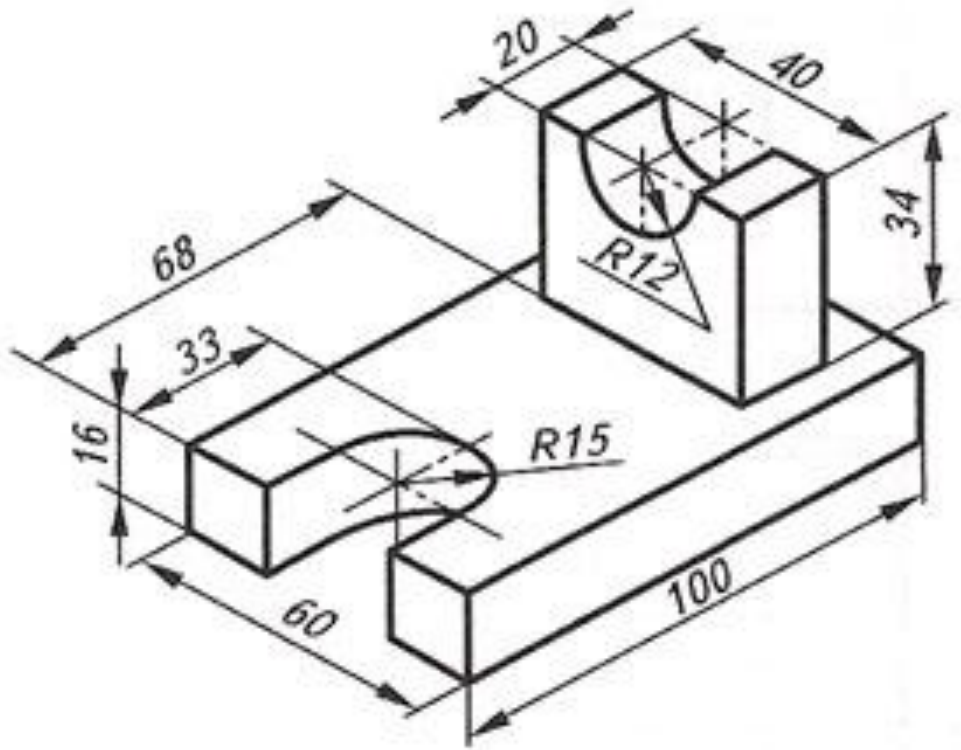
- Расценивайте ответ как новое требование;
- Есть ли расхождения с тем, что нам уже известно?
- Актуализируйте свои тесты, согласно новым данным;
- Получили ли вы нужное представление о интересующем вас вопросе?
- Нет – задайте дополнительные вопросы (иногда нужно 5+ циклов вопрос/ответ, чтобы решить проблему);
- Документируйте/сохраняйте ответы.

Тест-кейсы:

Когда вы видите требование, спросите себя: «Как я буду его тестировать? Какие тесты очевидно покажут, что это требование реализовано в ПС правильно?» Если с придумыванием таких тестов вы испытываете сложность – это сигнал: скорее всего, в требовании есть проблемы.

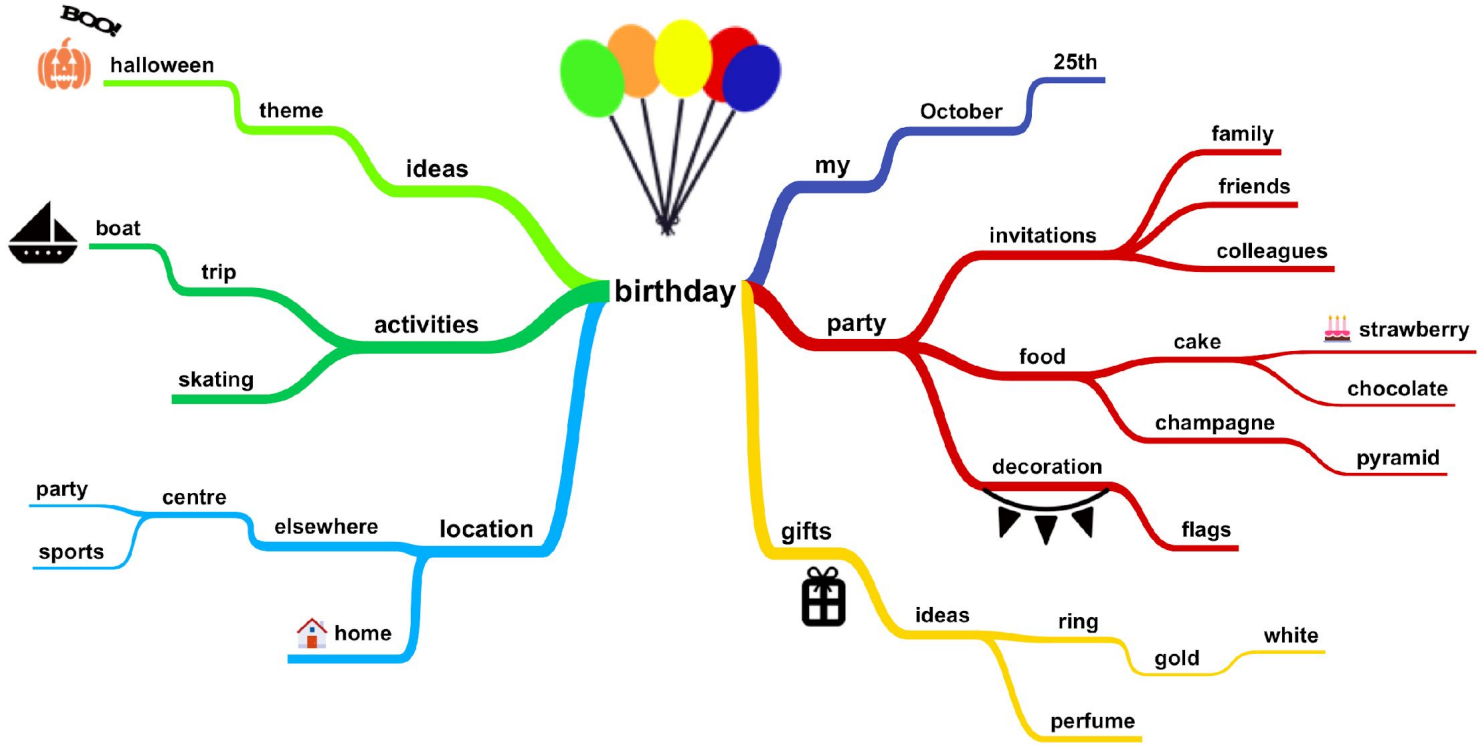
Визуальное отображение:

Чтобы увидеть общую картину требований целиком, очень удобно использовать рисунки, схемы, диаграммы и т.д.



Интеллект-карты:

Метод структуризации с использованием графической записи в виде дерева



Варианты использования (Use Case):

Use Case описывает сценарий взаимодействия участников (как правило — пользователя и системы)

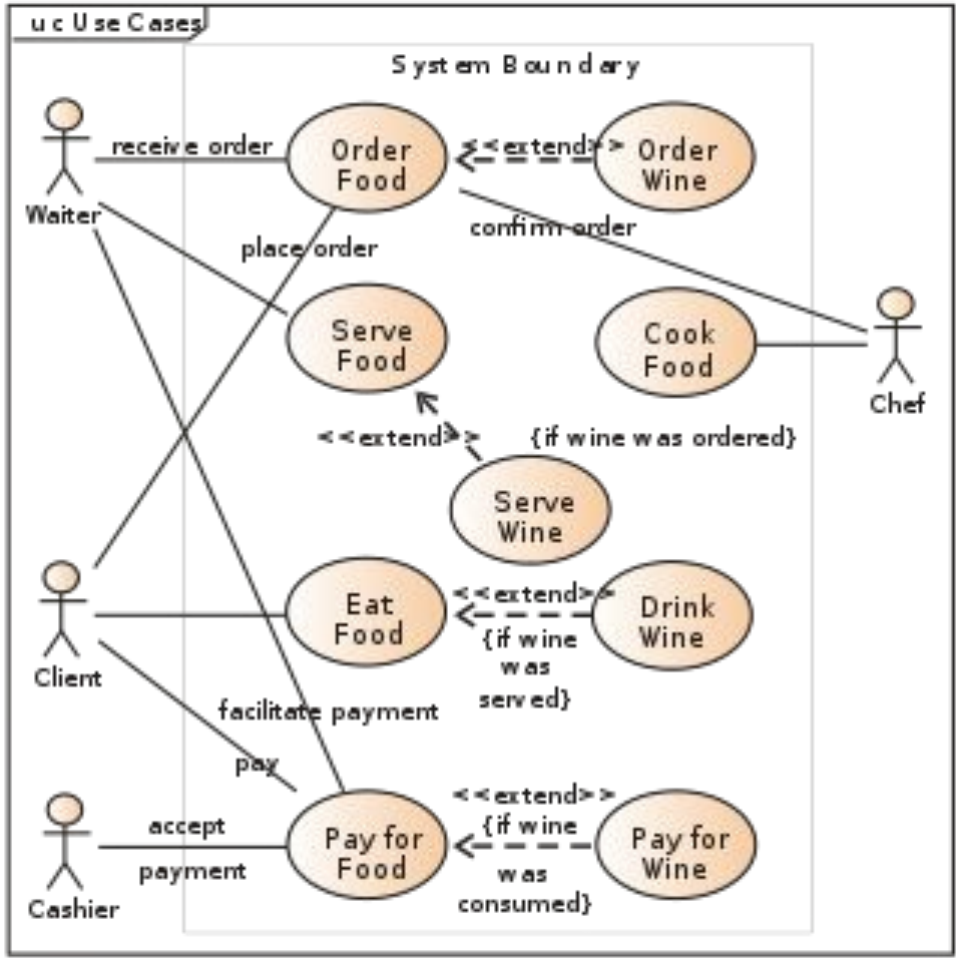
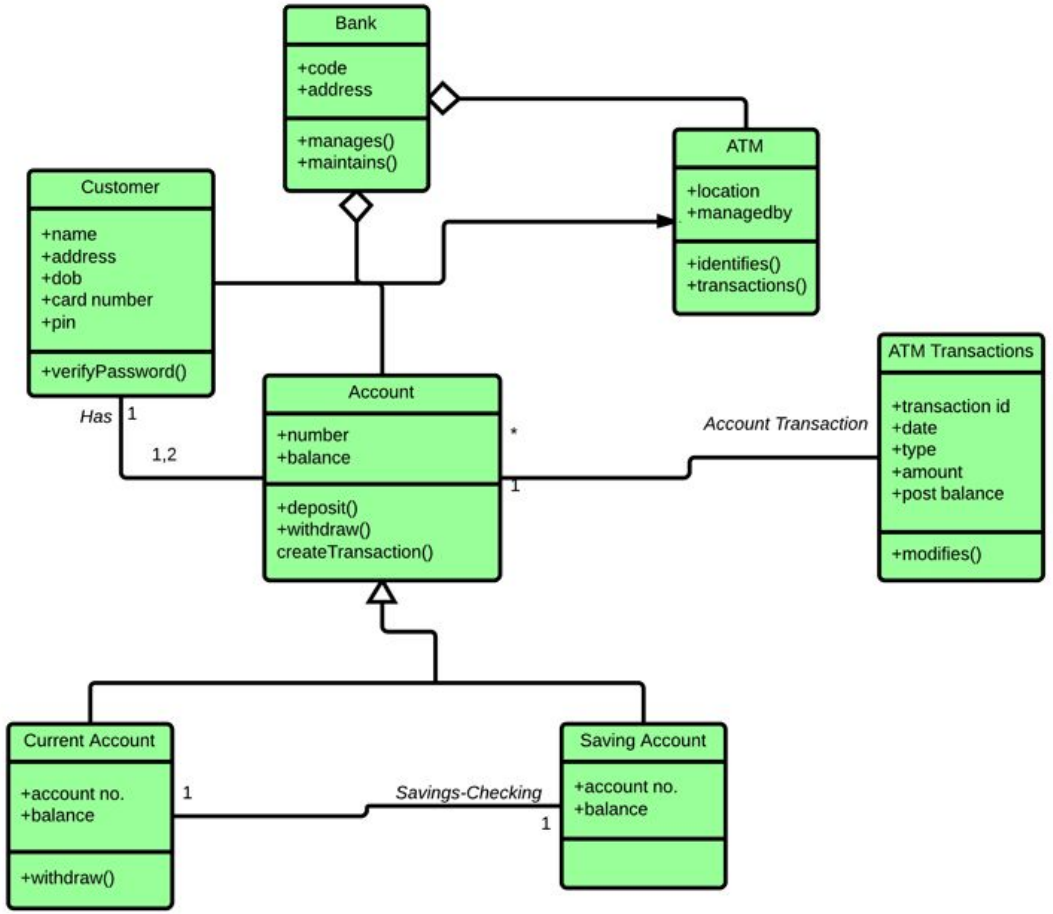


Диаграмма классов:

Структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их кооперацией, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними



Рецензирование:

Суть: после того, как один человек создал требование, другой человек это требование проверяет.

Типы:

- Неформальное рецензирование (informal review);
- Разбор/сквозной контроль (Walkthrough);
- Технический (равноправный) анализ (Technical (Peer) Review);
- Инспекции (Inspections).

Типы рецензирования:

Неформальное рецензирование (informal review) – полезно просить коллег просмотреть на документ, чтобы:

Устранить недочеты, которые не видны автору;

Определить места, где знание предмета автором предполагает, что читатель знает больше, чем есть на самом деле;

Самый простой способ получения рецензии.

Разбор/сквозной контроль (walkthrough):

Обычно проводится автором (возможен проектный менеджер или руководитель команды);

Рецензентов просят комментировать и уточнять концепции и процессы, которые должны быть доставлены.

Технический анализ (Technical Rewiev):

- Ваши коллеги могут также выполнять более формальное техническое рецензирование в больших группах, без участия менеджмента;
  - Формат определен, согласован и использует документированные процессы и результаты;
  - Требует технической экспертизы;
- Главная цель – обсуждение, решение технических проблем.

Инспекции:

- Формальный процесс, основанный на правилах;
- Проводится модератором;
- Все присутствующие имеют определенные роли;
- Включает метрики и критерии входа и выхода;
- Требуется обучение.

Требования могут быть в виде:

- Общая концепция (Vision)
- Варианты использования (Use cases)
- Истории использования (User Stories)
- Функциональные спецификации (Functional specs)
- UI mockup
- Готовое приложение

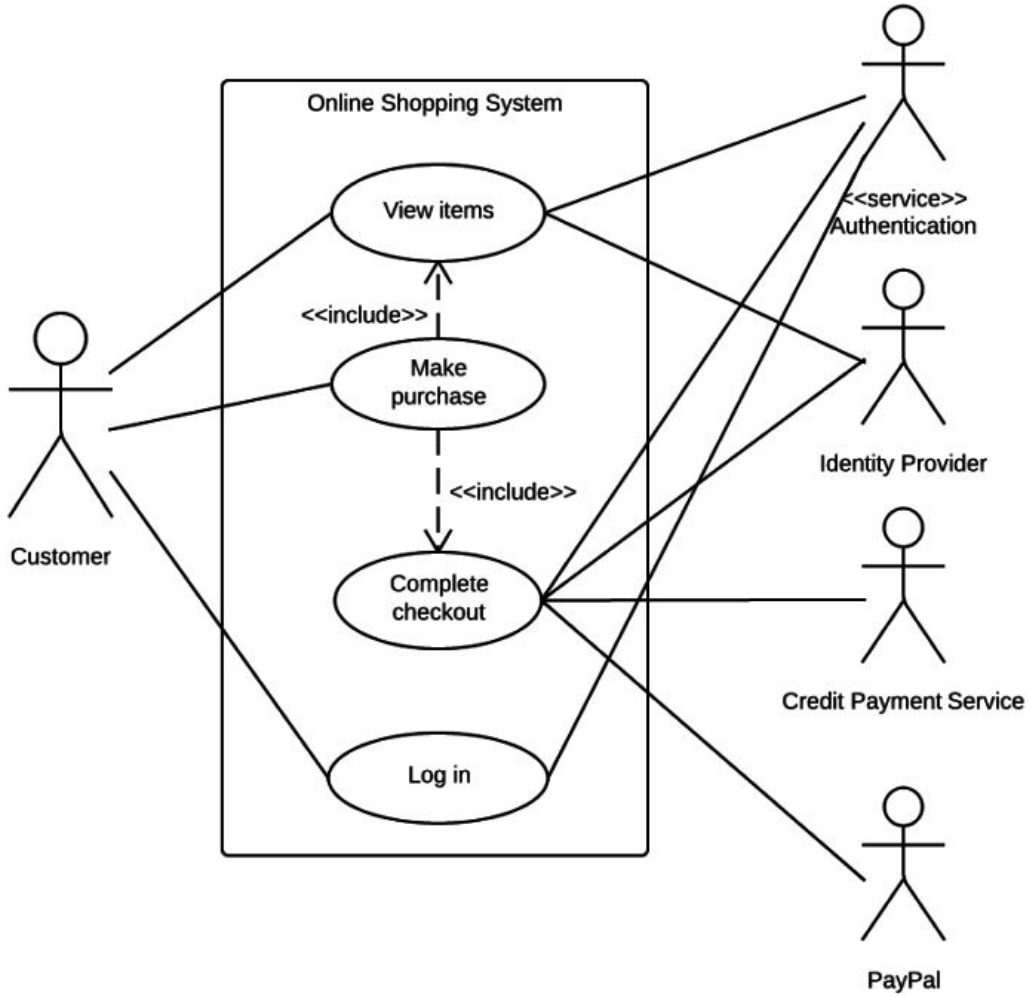
Концепция:

What was stated	What was intended
“The application shall provide cost estimates.”	“The application shall predict size for: specifications, source code, and user manuals; resources for technical employees and management; software cost estimates for all software development activities.”

Проблемы концепции:

- Не понятно, как это будет реализовано на практике – отображаться, в точности функционировать;
- Опираясь на концепцию, разработчики могут принимать совершенно разные решения;
- 1 фраза может подразумевать очень большое количество функциональности;
- Бизнес-определения часто не понятны, для команды разработки.

Use Case :


Online Shopping System
View items
<<include>>
Make purchase
<<include>>
Complete checkout
Log in
Customer
<<service>>
Authentication
Identity Provider
Credit Payment Service
PayPal

Use Case (проблемы):

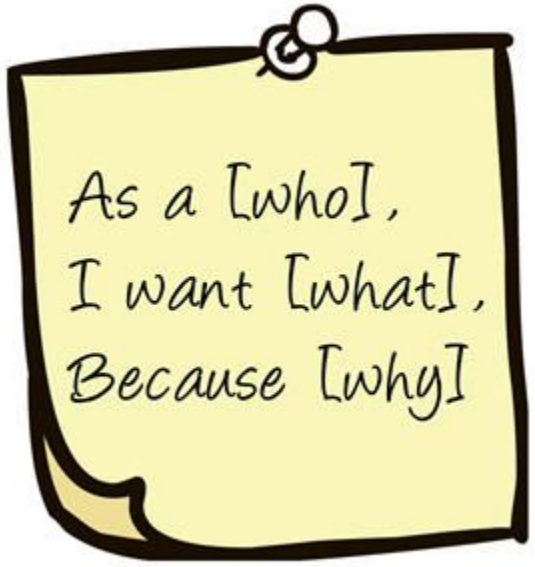
- Могут отсутствовать сценарии с альтернативным поведением;
- Не все возможные сценарии описаны;
- Предлагаемое поведение не удобно для пользователя;
- Предлагаемое очень тяжело реализовать;
- Противоречия между разными процессами использования;
- Непонятный GUI;
- Точный перечень функций не ясен.

User Story (История пользователя):

История пользователя представляет собой краткое изложение функциональных возможностей, реализация которых необходима для получения конкретным заинтересованным лицом пользы от программного продукта.

Пример использования:

Как <роль>, мне необходимо <поведение>,
чтобы получить <ценность бизнеса>

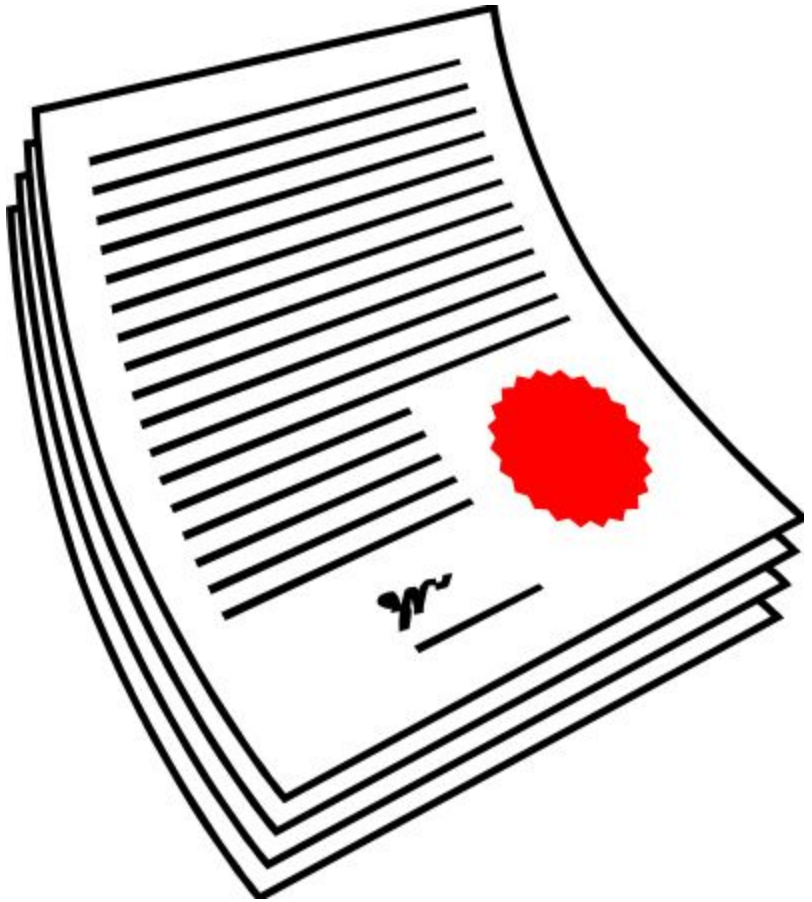


Функциональная спецификация:

Документ, описывающий требуемые характеристики системы (функциональность). Документация описывает необходимые для пользователя системы входные и выходные параметры (например, программная система).

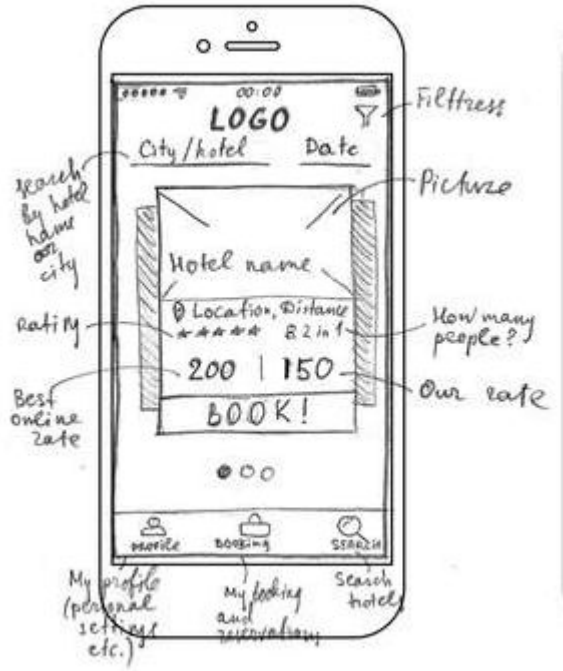
Проблемы спецификации:

- Длинные спецификации затруднительно редактировать, отслеживать изменения;
- Много страниц, трудно читать и работать с документом.



UI mockup:

Неработающая модель, выполненная в натуральную величину и выглядящая так, как будет выглядеть работающий экземпляр. То есть, сделанная в фотошопе веб-страница, отданная на верстку — это мокап



Проблемы UI mockup:

- Противоречия между картинкой и текстом;
- Противоречия между новой картинкой и старой;
- Противоречия внутри одной картинки;
- Логика работы функции непонятна (кнопка есть, но непонятно для чего);

Готовое приложение:

Проблема готового приложения:

- Нет уверенности, что мы определили все имеющиеся функции;
- Поведение которое мы видим – это баг или фича?
- А что должно случиться, если нажать эту кнопку?
- Для чего нужна эта функция?
- Какие входные данные для этой функции являются валидными, а какие нет?