

Лекция 3.

Тестирование требований к ПО и документации

Что такое требование?

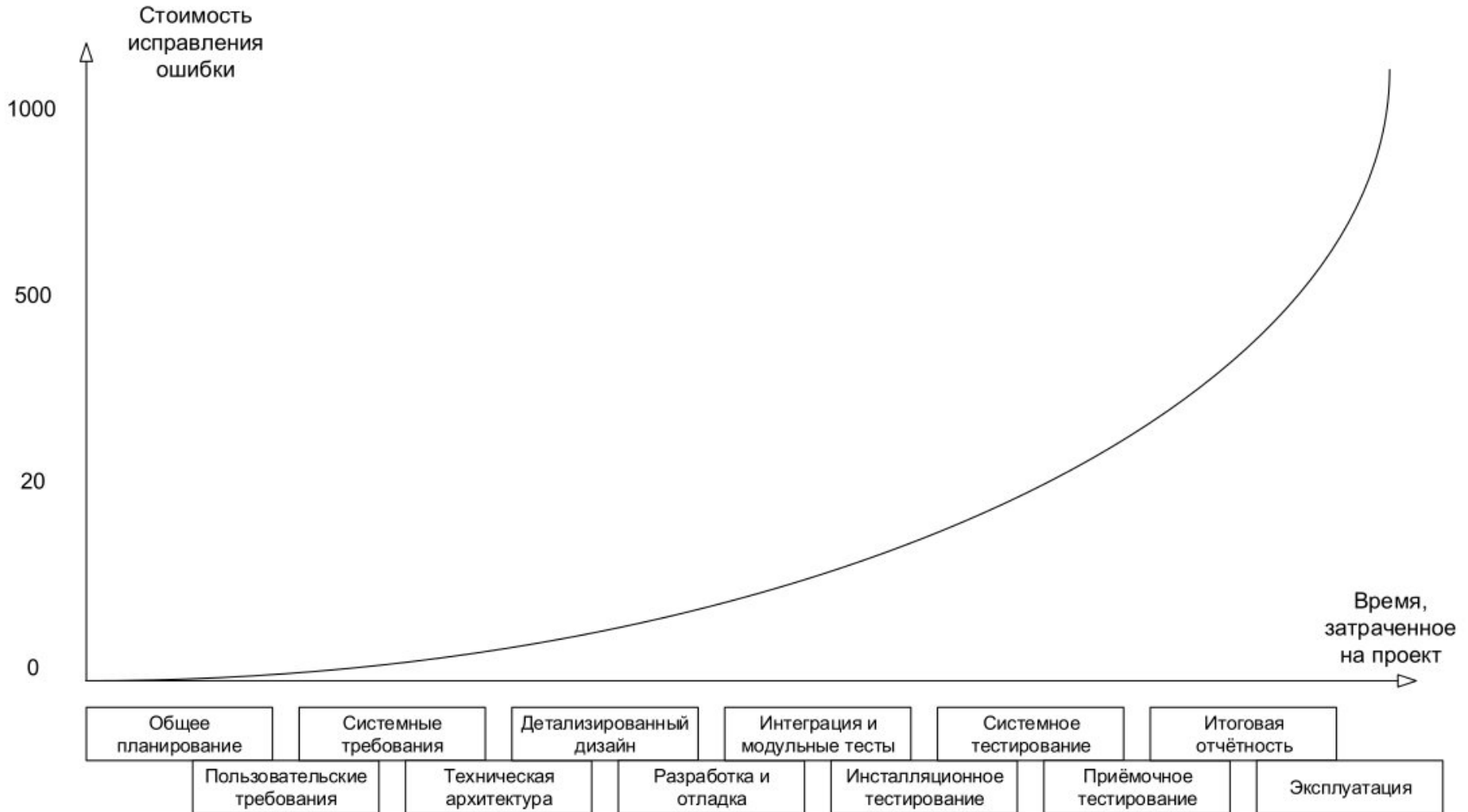
Требование (requirement) - описание того, какие функции и соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.

Почему начинаем с требований?

- До 70% дефектов ПО — следствие ошибок на этапе требований
- Чем позже найден дефект — тем дороже его исправление
- Тестирование требований = профилактика будущих проблем

Цель: убедиться, что требования понятны, полны,
непротиворечивы и проверяемы

Почему начинаем с требований?



Стоимость исправления ошибки в зависимости от момента ее обнаружения

Свойства качественных требований

Почему свойства требований важны?

«Требование может быть написано — но быть **бесполезным** для разработки и тестирования»

Цель:

Требования должны быть не просто текстом, а **рабочим инструментом** для всей команды.

Если требование не обладает ключевыми свойствами:

- Разработчик не поймёт, **что делать**
- Тестировщик не сможет **написать тест**
- Заказчик будет недоволен результатом
- Возникнут **споры, переделки, задержки**

 Качество требований = основа предсказуемости проекта

Проект с плохими требованиями



Так клиент
объяснил, чего он
хочет



Так клиента понял
менеджер проекта



Так аналитик
описал проект



Так программист
реализовал проект



Так проект был
прорекламирован
консультантами



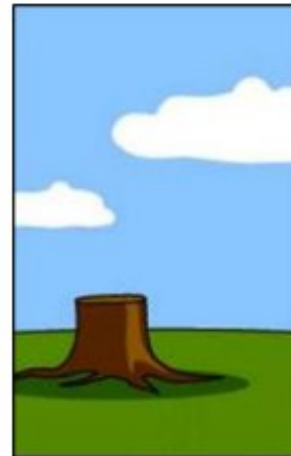
Так проект был
задокументирован



Так проект был
сдан в
эксплуатацию



В такую сумму
проект обошёлся
заказчику



Так работала
техническая
поддержка



Что на самом деле
было нужно
клиенту

Ключевые свойства качественных требований

Требование должно быть:

1. **Полным** — ничего не упущено
2. **Однозначным** — только одна интерпретация
3. **Проверяемым** — можно подтвердить тестом
4. **Согласованным** — нет противоречий
5. **Трассируемым** — можно отследить происхождение и покрытие
6. **Понятным** — ясно целевой аудитории
7. **Осуществимым** — реально реализуемо в рамках проекта

Эти свойства — критерии при ревью и инспекции

Полнота (завершенность) требований

Завершённость (completeness⁸⁷). Требование является полным и законченным с точки зрения представления в нём всей необходимой информации, ничто не пропущено по соображениям «это и так всем понятно».

Типичные проблемы с завершённостью:

- Отсутствуют нефункциональные составляющие требования или ссылки на соответствующие нефункциональные требования (например: «*пароли должны храниться в зашифрованном виде*» — каков алгоритм шифрования?).
- Указана лишь часть некоторого перечисления (например: «*экспорт осуществляется в форматы PDF, PNG и т.д.*» — что мы должны понимать под «и т.д.»?).
- Приведённые ссылки неоднозначны (например: «*см. выше*» вместо «*см. раздел 123.45.b*»).

Полнота (завершенность) требований

Способы обнаружения проблем	Способы устранения проблем
<p>Применимы почти все техники тестирования требований⁽⁵¹⁾, но лучше всего помогает задавание вопросов и использование графического представления разрабатываемой системы. Также очень помогает глубокое знание предметной области, позволяющее замечать пропущенные фрагменты информации.</p>	<p>Как только было выяснено, что чего-то не хватает, нужно получить недостающую информацию и дописать её в требования. Возможно, потребуется небольшая переработка требований.</p>

Проверяемость и однозначность требований

Проверяемость

× «Система должна быть быстрой»

✓ «Система обрабатывает запрос за ≤ 500 мс при нагрузке 100 RPS»

Однозначность

× «Пользователь может отменить действие, если захочет»

✓ «После отправки формы пользователь может отменить операцию в течение 30 секунд с помощью кнопки "Отмена"»

💡 **Правило тестировщика:**

«Если я не могу написать чёткий тест-кейс — требование недостаточно проверяемо»

Однозначность требований

Недвусмысленность (unambiguousness⁹⁰, clearness). Требование должно быть описано без использования жаргона, неочевидных аббревиатур и расплывчатых формулировок, должно допускать только однозначное объективное понимание и быть атомарным в плане невозможности различной трактовки сочетания отдельных фраз.

- Использование неочевидных или двусмысленных аббревиатур без расшифровки (например: «доступ к ФС осуществляется посредством системы прозрачного шифрования» и «ФС предоставляет возможность фиксировать сообщения в их текущем состоянии с хранением истории всех изменений» — ФС здесь обозначает файловую систему? Точно? А не какой-нибудь «Фиксатор Сообщений»?)
- Формулировка требований из соображений, что нечто должно быть всем очевидно (например: «Система конвертирует входной файл из формата PDF в выходной файл формата PNG» — и при этом автор считает совершенно очевидным, что имена файлов система получает из командной строки, а многостраничный PDF конвертируется в несколько PNG-файлов, к именам которых добавляется «page-1», «page-2» и т.д.). Эта проблема перекликается с нарушением корректности.

Однозначность требований

Самый страшный враг двусмысленности – числа и формулы: если что-то можно выразить в формульном или числовом виде (вместо словесного описания), обязательно стоит это сделать.

Если это невозможно, стоит хотя бы использовать максимально точные технические термины, отсылки к стандартам и т.п.

Согласованность требований

Согласованность

→ Нет ли конфликта?

Требование 1: «Данные хранятся локально»

Требование 2: «Статистика синхронизируется между устройствами» → ✗

Согласованность (непротиворечивость) требований

Типичные проблемы с непротиворечивостью:

- Противоречия внутри одного требования (например: *«после успешного входа в систему пользователя, не имеющего права входить в систему...»* — тогда как он успешно вошёл в систему, если не имел такого права?)
- Противоречия между двумя и более требованиями, между таблицей и текстом, рисунком и текстом, требованием и прототипом и т.д. (например: *«712.a Кнопка “Close” всегда должна быть красной»* и *«36452.x Кнопка “Close” всегда должна быть синей»* — так всё же красной или синей?)
- Использование неверной терминологии или использование разных терминов для обозначения одного и того же объекта или явления (например: *«в случае, если разрешение окна составляет менее 800x600...»* — разрешение есть у экрана, у окна есть размер).

Согласованность (непротиворечивость) требований

Способы обнаружения проблем	Способы устранения проблем
<p>Лучше всего обнаружить противоречивость помогает хорошая память 😊, но даже при её наличии незаменимым инструментом является графическое представление разрабатываемой системы, позволяющее представить всю ключевую информацию в виде единой согласованной схемы (на которой противоречия очень заметны).</p>	<p>После обнаружения противоречия нужно прояснить ситуацию с заказчиком и внести необходимые правки в требования.</p>

Прослеживаемость (трассируемость) требований

Трассируемость

→ Можно ли связать требование:

- с бизнес-целью?
- с тест-кейсом?
- с кодом?

→ Инструмент: **Матрица трассируемости**

Прослеживаемость (трассируемость) требований

Типичные проблемы с прослеживаемостью:

- Требования не пронумерованы, не структурированы, не имеют оглавления, не имеют работающих перекрёстных ссылок.
- При разработке требований не были использованы инструменты и техники управления требованиями.
- Набор требований неполный, носит обрывочный характер с явными «пробелами».

Способы обнаружения проблем	Способы устранения проблем
Нарушения прослеживаемости становятся заметны в процессе работы с требованиями, как только у нас возникают остающиеся без ответа вопросы вида «откуда взялось это требование?», «где описаны сопутствующие (связанные) требования?», «на что это влияет?».	Переработка требований. Возможно, придётся даже менять структуру набора требований, но всё точно начнётся с расстановки множества перекрёстных ссылок, позволяющих осуществлять быструю и прозрачную навигацию по набору требований.

Уровни и типы требований

Типы требований

Требования строятся **сверху вниз** — от бизнеса к реализации:

1. **Бизнес-требования** — «зачем?»
2. **Пользовательские требования** — «что хочет пользователь?»
3. **Системные требования:**
 - Функциональные — «что делает система?»
 - Нефункциональные — «как хорошо она это делает?»
4. **Ограничения и бизнес-правила** — «в каких рамках работает система?»

Уровни и типы требований



Бизнес-требования

Бизнес-требования

- Цели, ценность, KPI, обоснование инвестиций
- Пример:
 - | «Запустить веб-версию “Сапёра” для монетизации»
- Фиксируются в **Vision & Scope, BRD**

Вопрос тестировщика:

| «Как измерить успех? Какие метрики?»

- *Нужен инструмент, в реальном времени отображающий наиболее выгодный курс покупки и продажи валюты.*
- *Необходимо в два-три раза повысить количество заявок, обрабатываемых одним оператором за смену.*
- *Нужно автоматизировать процесс выписки товарно-транспортных накладных на основе договоров.*

Пользовательские требования

Пользовательские требования

- Описывают **задачи**, а не функции
- Форматы: **User Stories, Use Cases, User Flows**
- Пример:
 - | «Как игрок, я хочу отмечать флагами, чтобы не открыть мину»

Тестировщик спрашивает:

| «Какие сценарии? Есть ли крайние случаи?»

- *При первом входе пользователя в систему должно отображаться лицензионное соглашение.*
- *Администратор должен иметь возможность просматривать список всех пользователей, работающих в данный момент в системе.*
- *При первом сохранении новой статьи система должна выдавать запрос на сохранение в виде черновика или публикацию.*

User Stories, Use Cases, User Flows

- При сборе и анализе требований используются разные форматы описания.
- **User Stories, Use Cases** и **User Flows** — три ключевых инструмента для фиксации пользовательских потребностей.
- Каждый из них имеет свою цель, уровень детализации и целевую аудиторию.
- Понимание различий помогает тестировщику:
 - Глубже понять бизнес-логику продукта.
 - Создавать более полные и качественные тест-кейсы.
 - Выявлять неоднозначности и пробелы в требованиях на ранних этапах.

User Story (Пользовательская история)

- **Что это?** Краткое, неформальное описание функциональности с точки зрения пользователя.
- **Формат:** «Как [роль], я хочу [цель], чтобы [мотивация]».
- **Фокус:** На «что» и «почему», а не на техническую реализацию.
- **Пример:** «Как покупатель, я хочу добавить товар в корзину, чтобы позже оформить заказ».
- **Роль в тестировании:** Является основой для определения акцептанс-критериев, которые напрямую трансформируются в высокоуровневые проверки.

Use Case (Прецедент / Вариант использования)

- **Что это?** Структурированное описание последовательности действий между актором (пользователем) и системой для достижения цели.
- **Фокус:** На «как» система должна работать в ответ на действия пользователя.
- **Структура:**
 - Актор
 - Основной поток событий
 - Альтернативные и исключительные потоки
- **Пример:** Прецедент «Восстановление пароля» с шагами: нажать кнопку → ввести email → получить ссылку → задать новый пароль.
- **Роль в тестировании:** Идеальный источник для проектирования подробных тест-кейсов, покрывающих как основной, так и все альтернативные сценарии.

User Flow (Пользовательский поток)

- **Что это?** Графическое или схематичное представление последовательности экранов и действий, которые выполняет пользователь для выполнения задачи.
- **Фокус:** На UX/UI и навигации по продукту.
- **Формат:** Диаграмма (flowchart), wireflow.
- **Пример:** Схема переходов: Главная страница → Каталог → Карточка товара → Корзина.
- **Роль в тестировании:** Помогает тестировщику понять контекст использования функции, выявить проблемы с навигацией и спроектировать сквозные (end-to-end) сценарии тестирования.

Сравнение подходов

Характеристика	User Story	Use Case 
Цель	Выразить ценность	Описать логику
Детализация	Низкая	Высокая
Формат	Текст (шаблон)	Структурированный текст
Аудитория	Продуктовая команда	Аналитики, разработчики
Источник для тестов	Акцептанс-критерии	Подробные сценарии

Вывод: Эти артефакты дополняют друг друга. Совместное их использование обеспечивает полное и многогранное понимание требований, что является залогом успешного тестирования.

Функциональные требования

Функциональные требования

- Поведение системы: «Если..., то...»

- Примеры:

«ПКМ по ячейке → флаг»

«После победы → сохранить статистику»

→ Каждое → минимум 1 **тест-кейс**

- *В процессе инсталляции приложение должно проверять остаток свободного места на целевом носителе.*
- *Система должна автоматически выполнять резервное копирование данных ежедневно в указанный момент времени.*
- *Электронный адрес пользователя, вводимый при регистрации, должен быть проверен на соответствие требованиям RFC822.*

Нефункциональные требования


Нефункциональные требования

- Производительность, совместимость, безопасность, юзабилити, надёжность

- Пример:

«Игра работает на экране от 320px»

«Данные не отправляются на сервер»

 Часто упускаются — но критичны для UX!

- *При одновременной непрерывной работе с системой 1000 пользователей, минимальное время между возникновением сбоев должно быть более или равно 100 часов.*
- *Ни при каких условиях общий объём используемой приложением памяти не может превышать 2 ГБ.*
- *Размер шрифта для любой надписи на экране должен поддерживать настройку в диапазоне от 5 до 15 пунктов.*

Ограничения и бизнес-правила

Ограничения и бизнес-правила

Ограничения: технологии, законодательство, сроки

Бизнес-правила: логика предметной области

Примеры:

«Мины не генерируются рядом с первым кликом»

«Без регистрации — максимум 10 игр/день»

Тестировщик: проверяет реакцию на нарушение правил

- *Никакой документ, просмотренный посетителями сайта хотя бы один раз, не может быть отредактирован или удалён.*
- *Публикация статьи возможна только после утверждения главным редактором.*
- *Подключение к системе извне офиса запрещено в нерабочее время.*

Источники требований

Источники требований

«Требования не падают с неба — их нужно **искать, слушать, анализировать**»

Почему важно понимать источники?

- Чтобы **не пропустить** критичные требования
- Чтобы различать **формальные обязательства** и желания
- Чтобы правильно **оценить приоритеты и риски**
- Чтобы понимать, **кого спрашивать**, когда возникают вопросы

 Тестировщик должен знать:

«Кто сказал? Почему это важно? Где это зафиксировано?»

Явные (формальные) источники

Это источники, где требования **чётко сформулированы и задокументированы**:

- **Договоры и контракты**
 - Юридически обязывающие условия (сроки, функции, SLA)
- **Регуляторные документы**
 - GDPR, HIPAA, ГОСТ Р, ISO, отраслевые стандарты
- **Технические спецификации (SRS, API docs)**
 - Уже согласованные требования
- **BRD / Product Backlog**
 - Официальный список задач от Product Owner
- **Legacy-документация**
 - Требования к совместимости или миграции

✓ Преимущество: можно **ссылаться, тестировать, спорить на их основе**

Неявные (неформальные) источники

Это источники, где требования **подразумеваются**, но не записаны:

- **Разговоры на встречах**
 - «Ага, ну понятно, что так должно быть...»
- **Ожидания пользователей по умолчанию**
 - «Кнопка “Назад” должна работать», «Форма должна сохраняться при перезагрузке»
- **Поведение конкурентов**
 - Если все делают так — пользователь ожидает того же
- **Культурные/когнитивные нормы**
 - Порядок полей в форме (имя → фамилия), цвет ошибки (красный), иконки
- **Архитектурные решения**
 - Если используется REST API — подразумевается stateless, JSON и т.д.

 **Главная опасность:**

неявное требование = **невидимый для разработки дефект** = негодование пользователя

Проблемы и риски при работе с источниками

Проблема	Последствие	Как избежать 
Конфликт источников (заказчик vs регулятор)	Противоречивые требования	Раннее выявление + согласование
Недоступность источника (пользователь не опрошен)	Упущенные сценарии	Использовать personas, юзабилити-исследования
Зависимость от одного источника (только Product Owner)	Смещённый фокус (бизнес ≠ UX)	Вовлекать разные роли
Не задокументированные устные договорённости	«Я так не говорил!»	Фиксировать всё в письменной форме
Игнорирование неявных ожиданий	«Всё работает, но неудобно»	Анализ конкурентов, UX-ревью

Роль тестировщика:

- быть «антенной» для неявных требований
- задавать: «А что, если пользователь ожидает...?»

Методы выявления требований

- Интервью
- Опросы
- Наблюдение
- Workshop
- Анализ документации
- Прототипирование
- Story mapping

Тестировщик участвует — чтобы думать о тестах с первого дня

Роль тестировщика в выявлении требований

- Задаёт уточняющие вопросы
- Выявляет пробелы и неявные требования
- Проверяет **проверяемость**
- Предлагает **граничные и негативные сценарии**
- Участвует в ревью

✓ Раннее вовлечение = меньше дефектов

Статическое тестирование: обзор техник

Что такое статическое тестирование?

Тестирование без выполнения кода, направленное на анализ документации, требований, спецификаций и дизайна с целью раннего выявления дефектов.

Почему важно?

- До **70% дефектов** можно найти **до написания кода**
- Снижает стоимость исправлений в **10–100 раз**
- Повышает **ясность, согласованность и тестируемость** требований

Основные техники статического тестирования

1. **Неформальное ревью** — коллеги читают и дают замечания
2. **Ход-по-ходу (Walkthrough)** — автор ведёт обзор для команды
3. **Техническое ревью** — эксперты оценивают архитектурные решения
4. **Инспекция** — формальный процесс с ролями, чек-листами, логированием дефектов
5. **Статический анализ (для документов)** — проверка на соответствие шаблонам, стандартам, свойствам качества

Инспекция требований

Формальный процесс с ролями: автор, модератор, инспекторы

Этапы: планирование → обзор → митинг → исправление → верификация

Эффективность: до **90%** дефектов можно найти

Контрольный список

Примеры вопросов:

- Однозначно? Проверяемо? Есть ли границы?
- Есть ли NFR? Противоречия?

Пример:

«Быстро» → X

«Отклик ≤ 300 мс» → 

Given-When-Then (GWT)

Что это?

Формат описания поведения системы в виде **сценария**, понятного как бизнесу, так и технической команде.

Структура:

- **Given** — начальное состояние (предусловие)
- **When** — действие пользователя или события
- **Then** — ожидаемый результат

Пример (игра «Сапёр»):

Given игрок открыл 5 ячеек и не наткнулся на мину

When он нажимает правой кнопкой мыши на пустую ячейку

Then в ячейке отображается флаг, и счётчик мин уменьшается на 1

Given-When-Then (GWT)

Зачем это нужно?

- Чётко фиксирует **сценарий использования**
- Устраняет неоднозначность
- Легко преобразуется в **ручной или автоматизированный тест**
- Используется в **BDD** (Behaviour-Driven Development)

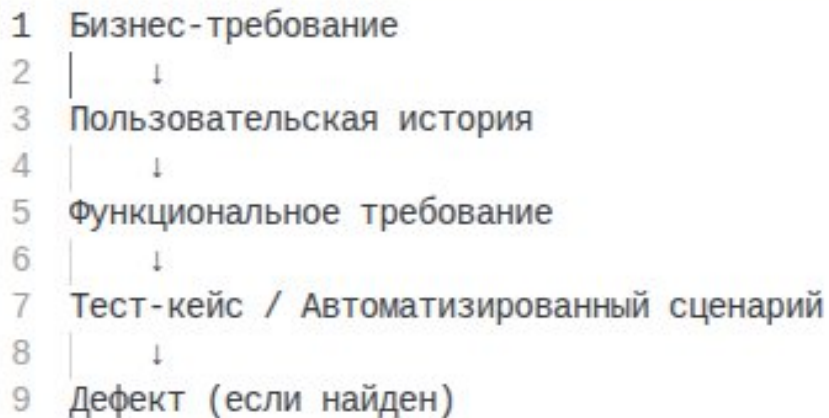
Роль тестировщика:

- формулировать GWT-сценарии на этапе анализа
- участвовать в «три amigos»-встречах (PO + Dev + QA)

Трассируемость требований

Что это?

Способность **отслеживать связь** между элементами проекта на разных уровнях:



Зачем нужна трассируемость?

- ✓ Убедиться, что **все требования покрыты тестами**
- ✓ Не писать **лишние тесты** (избыточность)
- ✓ Быстро оценить **влияние изменений** (impact analysis)
- ✓ Подтвердить соответствие **регуляторным/контрактным обязательствам**

Трассируемость требований

Инструмент:

→ **Матрица трассируемости** (Traceability Matrix)

- Строки: требования
- Столбцы: тест-кейсы
- Ячейка: / x / частично

Пример:

Требование ID	Тест-кейс ID
REQ-101	TC-205
REQ-102	TC-206, TC-207

Роль тестировщика:

- поддерживать актуальность матрицы
- выявлять непокрытые требования
- использовать трассировку при регрессионном тестировании

**Какая ещё документация
создаётся на начальном этапе?**

Документация

Помимо требований:

- Vision & Scope
- План проекта
- План обеспечения качества
- Стратегия тестирования
- Архитектурная документация
- План управления рисками


| Все влияют на тестирование!

**Кто создаёт документацию и
какова роль тестировщика?**

Документ	Автор	Роль тестировщика	↓
Vision & Scope	Product Owner	Участие, вопросы по измеримости	
План проекта	PM	Оценка трудозатрат на тестирование	
План качества	QA Lead	Соавтор	
Стратегия тестирования	Тест-менеджер	Основной автор	
Архитектура	Архитектор	Ревью на тестируемость	
План рисков	PM + команда	Выявление рисков качества	

💡 Даже если не пишешь — **ревью обязательно!**

Роль тестировщика

- Стратегия → как тестировать
 - Архитектура → где искать сбои
 - Риски → куда направить фокус
 - Vision → не потерять бизнес-цель
- |  Тестировщик — участник формирования качества на всех уровнях