Теория Тестирования

Систематизация знаний для уверенного старта.

Кто я?

Григорий Звягинцев, QA Engineer.

- Практикующий специалист в области обеспечения качества ПО.
- Помогаю начинающим специалистам войти в профессию и подготовиться к реальным задачам на проекте.
- Сегодня я ваш проводник в мир структурированных знаний о QA.

План полёта

1. 
2. 
3. 
4. 
5. 
6. 

1. **Контекст:** Роль QA в жизненном цикле разработки ПО (SDLC).
2. **Основы:** Пирамида тестирования как модель организации тестов.
3. **Классификация:** Виды и уровни тестирования.
4. **Практика:** Работа с требованиями и тестовой документацией (ТД).
5. **Документация и Инструменты:** Артефакты и системы управления.
6. **E2E**: Жизненный цикл фичи от идеи до релиза.

Для кого эта презентация?

Junior QA

Для систематизации имеющихся знаний и выявления пробелов.

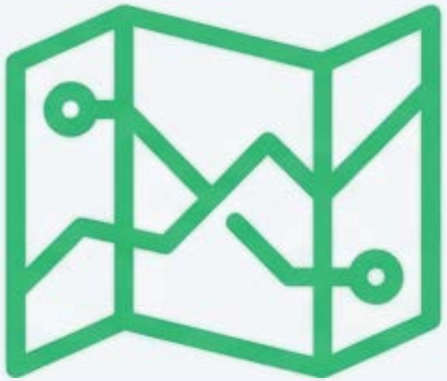
Свитчеры

Для глубокого понимания IT-процессов и роли QA в них.

Кандидаты на собеседование

Для уверенной подготовки к техническим вопросам и кейсам.

Что вы получите?



Структуру: Чёткая карта знаний о процессах QA, разложенная "по полочкам".

Словарь: Понимание ключевой профессиональной терминологии.

Уверенность: Ясное видение своего места в команде и процессах разработки.

Базу: Фундамент для дальнейшего профессионального роста и развития.

Зачем QA в разработке

Главная цель QA:
Обеспечение качества продукта и предотвращение бизнес-рисков.

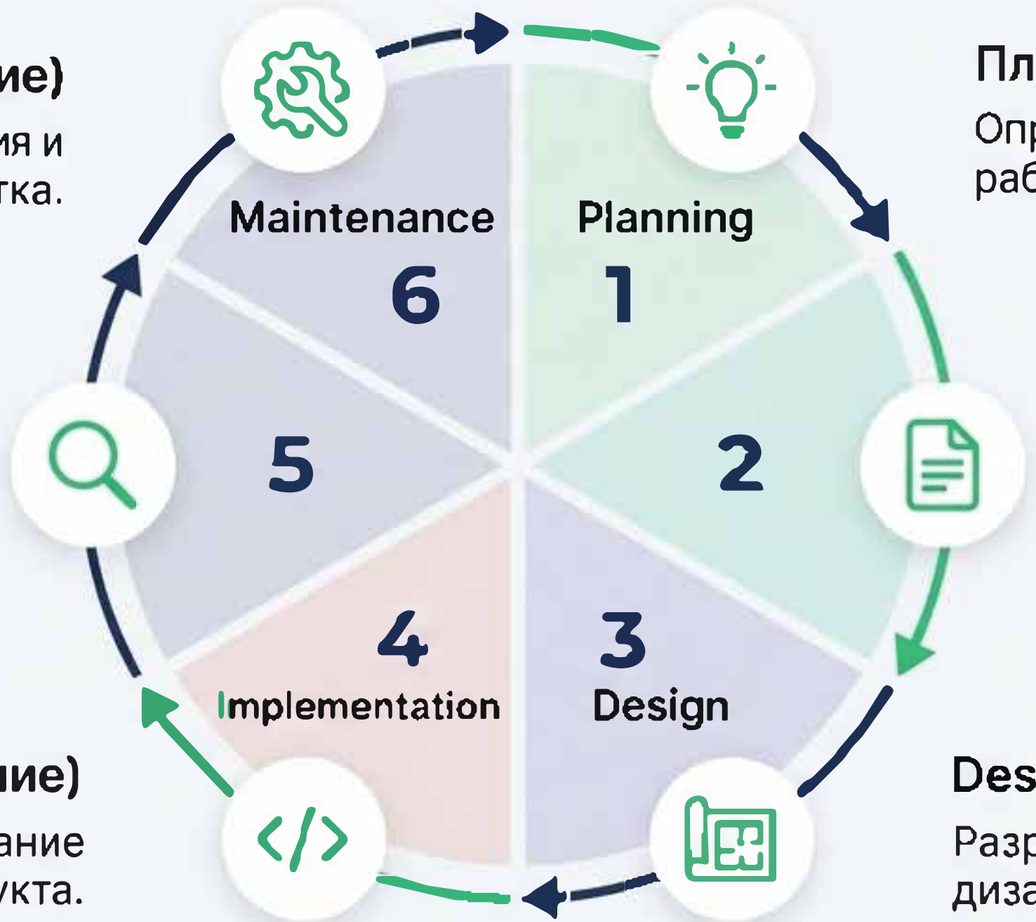
Тестирование — это процесс проверки соответствия между реальным и ожидаемым поведением программы.

Ключевой вопрос, на который отвечает QA: "Соответствует ли результат ожиданиям?".

QA-специалист помогает снизить риски, связанные с выпуском некачественного продукта.

SDLC (Жизненный цикл ПО)

SDLC (Software Development Life Cycle) — это структурированный процесс создания и поддержки ПО.



Поддержка и Внедрение)

Выпуск, эксплуатация и доработка.

Планирование

Определение целей, объёма работ и ресурсов.

Тестирование

Проверка продукта на соответствие требованиям.

Анализ требований

Сбор и формализация требований к продукту.

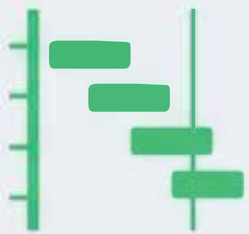
Implementation (Кодирование)

Написание кода и создание рабочей версии продукта.

Design (Проектирование)

Разработка архитектуры и дизайна системы.

Команда проекта (Часть 1)



PM (Project Manager)

Управляет проектом для достижения целей в рамках бюджета, сроков и заданного качества.

Аналитик

‘Мост’ между заказчиком и командой. Собирает, анализирует и документирует требования.

Архитектор

Проектирует техническую структуру системы, выбирает технологии и определяет взаимодействие компонентов.

Разработчик (Developer)

Пишет программный код, реализуя функциональность на основе требований и архитектуры.

Команда проекта (Часть 2)

QA Engineer

Отвечает за качество продукта и процессов. Планирует и проводит тестирование, документирует дефекты.

DevOps Engineer

Настраивает инфраструктуру проекта, автоматизирует процессы сборки, развертывания и тестирования (CI/CD).

UI/UX Дизайнер

Проектирует пользовательские интерфейсы (UI) и опыт взаимодействия (UX).

Тимлид (Team Lead)

Технический лидер. Отвечает за управление командой разработки, менторство и качество технических решений.

Взаимодействие (QA в команде)

Роль QA — это не только поиск багов в готовом продукте.

Итоги модуля

- QA — это неотъемлемая часть команды и всего жизненного цикла разработки ПО (SDLC).
- Ключевая ценность QA — минимизация продуктовых и проектных рисков на всех этапах.
- Понимание своего места в SDLC и ролей других участников — основа эффективной работы и профессионального роста.

Классические модели (Waterfall)

QA вступает в процесс только здесь. Изменения вносить дорого и сложно.

Плюсы

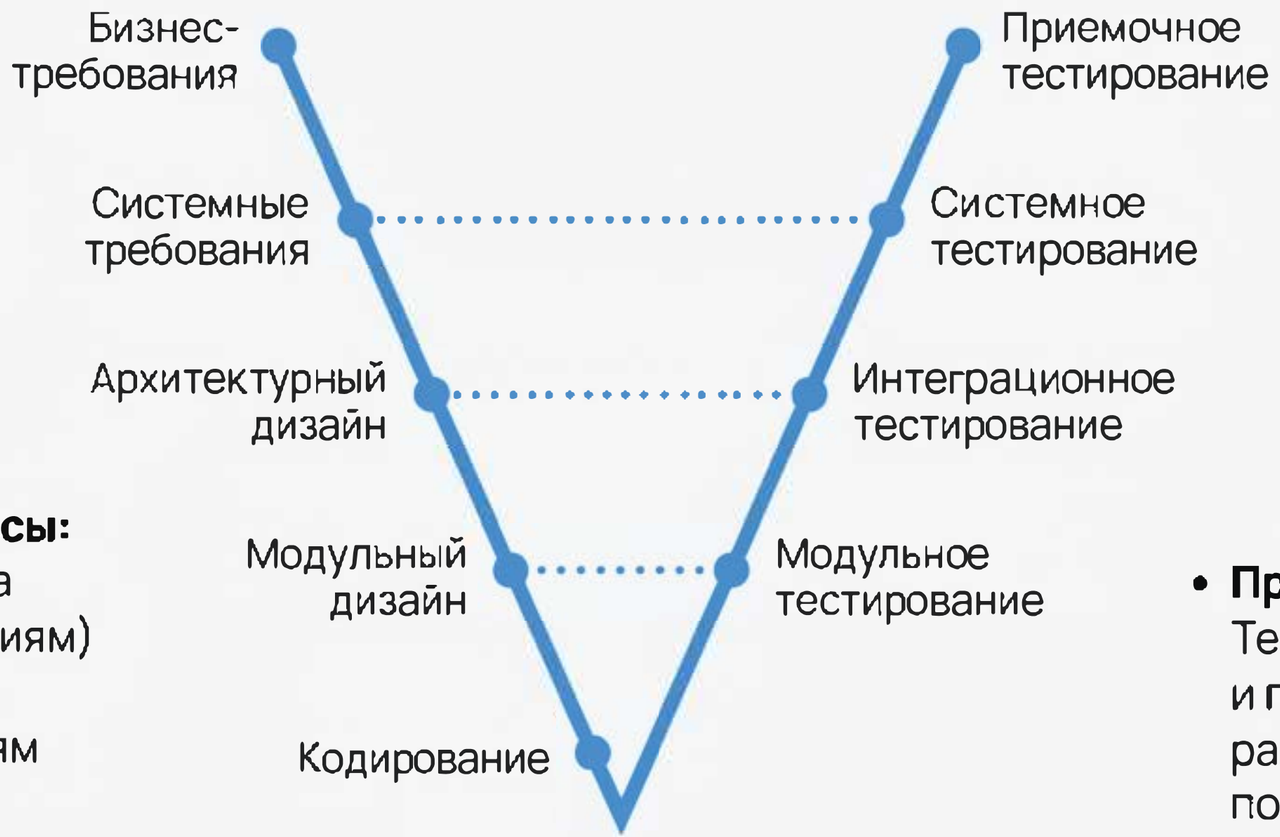
- Простота планирования и управления
- Четко определенные этапы и результаты
- Строгая документация

Минусы

- Высокая цена ошибки
- Долгий цикл обратной связи
- Низкая гибкость к изменениям

V-модель

- **Модификация Waterfall:** Усовершенствованная каскадная модель, которая подчеркивает связь между разработкой и тестированием на каждом этапе.
- **Параллельные процессы:** Верификация (проверка соответствия требованиям) и Валидация (проверка соответствия ожиданиям пользователя) планируются параллельно с разработкой.

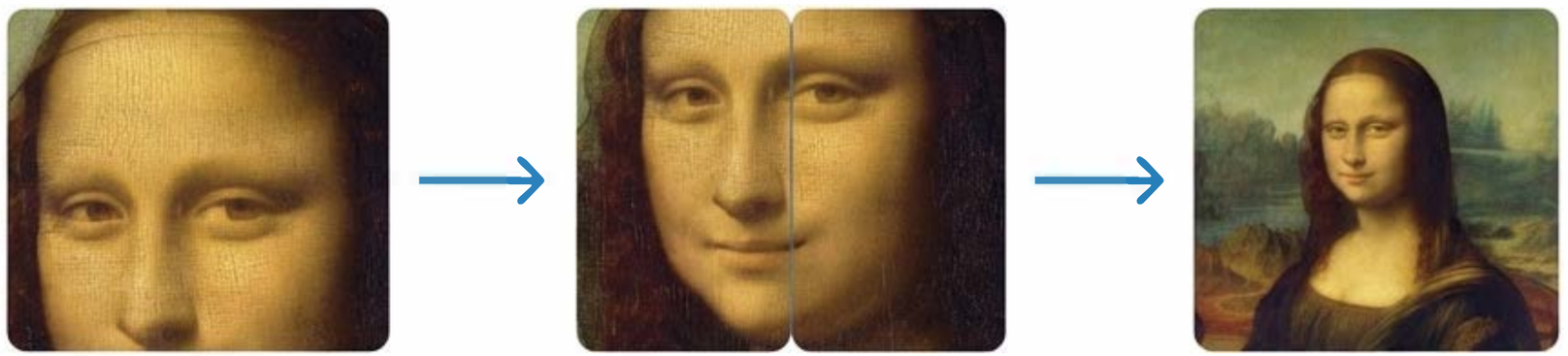


- **Преимущество для QA:** Тестирование планируется и проектируется на ранних стадиях, что повышает качество и позволяет раньше находить дефекты в логике.

Итеративная и Инкрементальная модели

- **Инкрементальная модель**: Продукт создается и поставляется по частям (инкрементам). Каждый инкремент — это готовый, протестированный фрагмент функциональности. Система собирается как конструктор.
- **Итерационная модель:** Разработка ведется циклами (итерациями). На каждой итерации создается новая, более совершенная версия всего продукта. Начинаем с наброска (MVP) и постепенно детализируем.

Инкрементный подход



Итерационный подход

Спиральная модель

- Объединяет итеративный подход с акцентом на анализе рисков.
- Идеальна для больших, сложных и рискованных проектов с неясными требованиями.

RAD (Rapid Application Development)

- Разновидность инкрементной модели для сверхбыстрой разработки.
- **Принципы:** Работа нескольких команд параллельно, активное прототипирование, жесткие временные рамки (60-90 дней).

Agile (Философия)

Не методология, а семейство гибких подходов, основанных на общих общих ценностях и принципах Agile Manifesto.

Люди и взаимодействие важнее процессов и инструментов.

Работающий продукт важнее исчерпывающей документации.

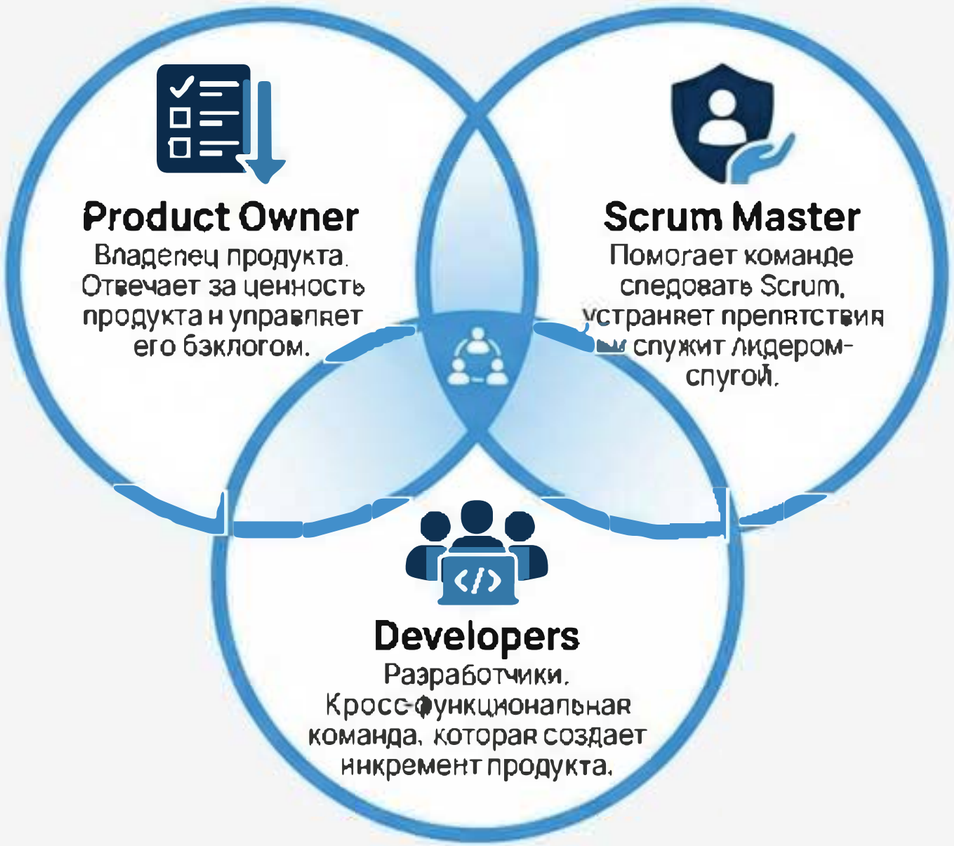
Сотрудничество с заказчиком важнее согласования условий контракта.

Готовность к изменениям важнее следования первоначальному плану.

Scrum (Фреймворк)

Самый популярный фреймворк Agile: структурированный и итеративный подход для разработки сложных продуктов.

Ключевые роли (Scrum Team)



Основные артефакты

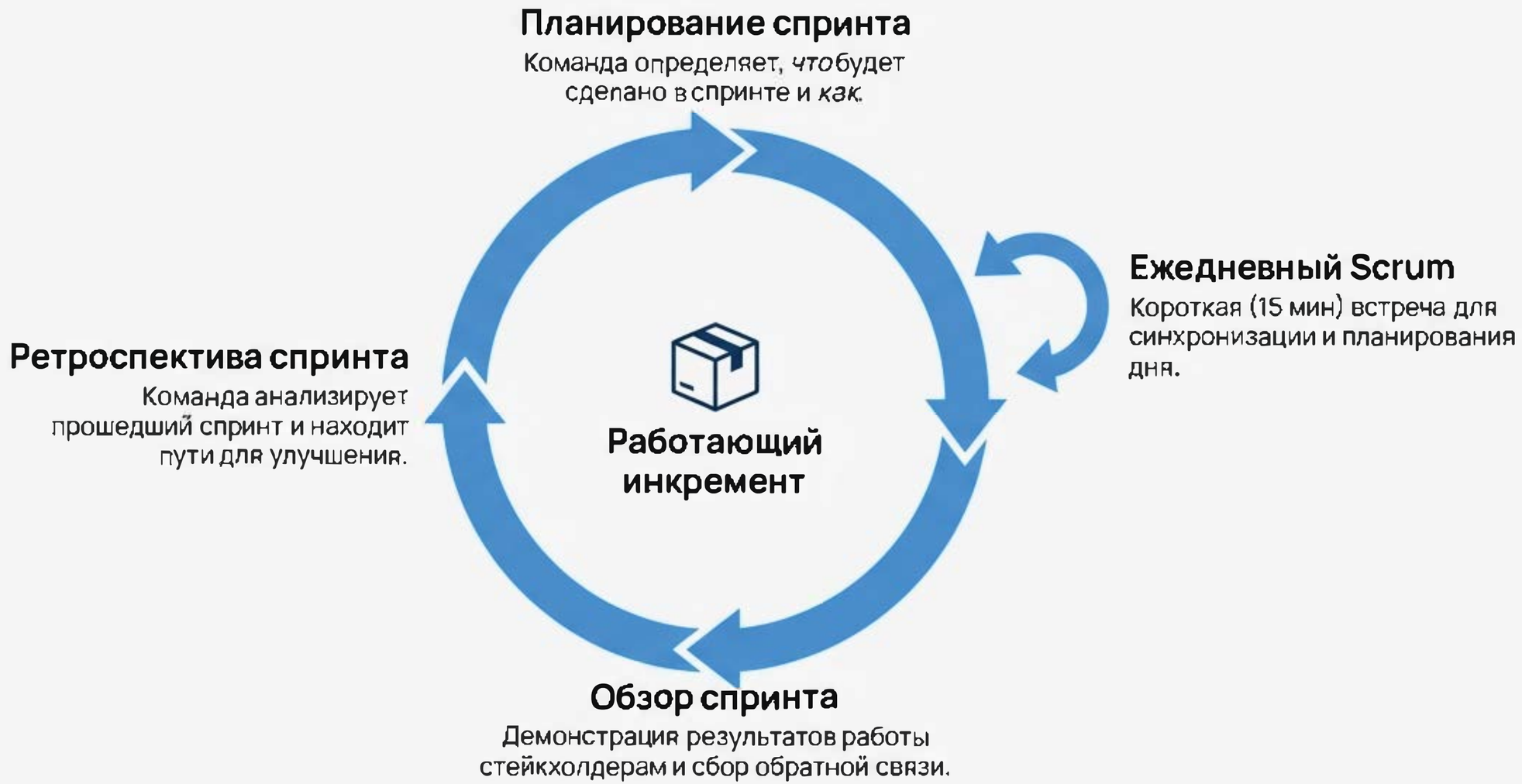
Product Backlog. Приоритезированный список всех требований к продукту.

Sprint Backlog. Набор задач, выбранных для выполнения в текущем спринте.

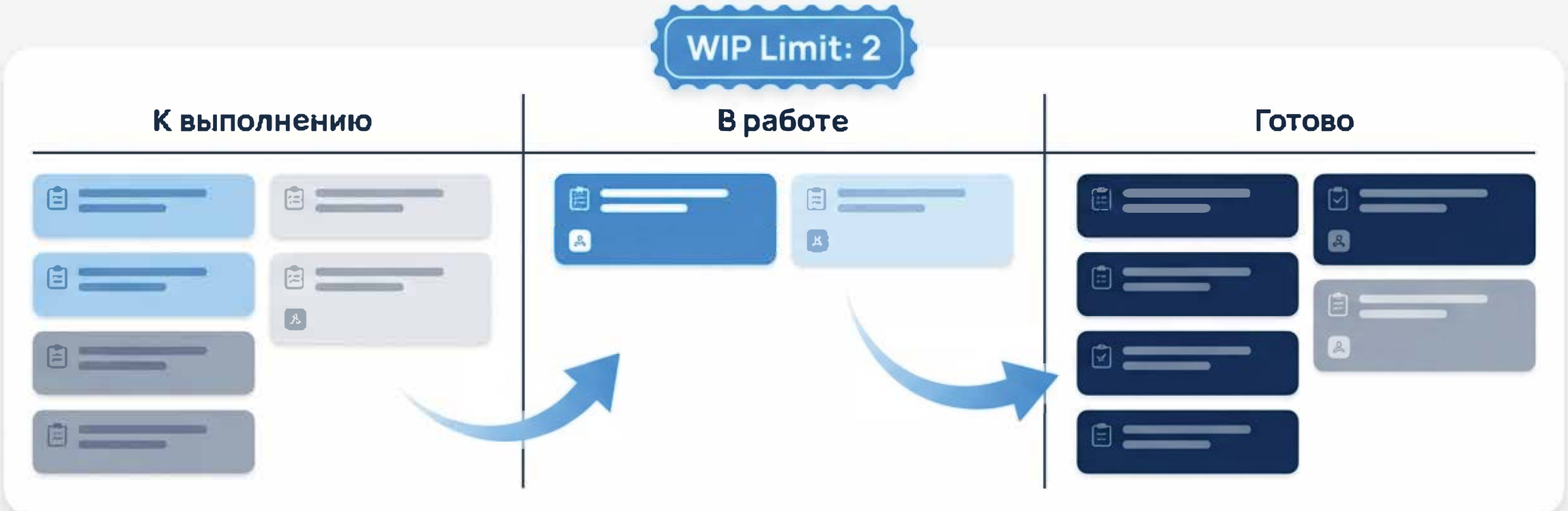
Increment. Готовый к выпуску, протестированный фрагмент продукта, созданный за спринт.

Жизненный цикл Спринта

Спринт — это фиксированный по времени отрезок (1-4 недели), сердце Scrum, в течение которого команда создает готовый инкремент продукта.



Канбан (Kanban)



Основа — визуализация и поток

- Работа визуализируется на доске (Kanban board) с колонками, отражающими этапы.
- Ключевой принцип — **WIP limits**: Ограничение количества задач, которые могут находиться "в работе" одновременно. Это помогает сфокусироваться и выявить "узкие места".

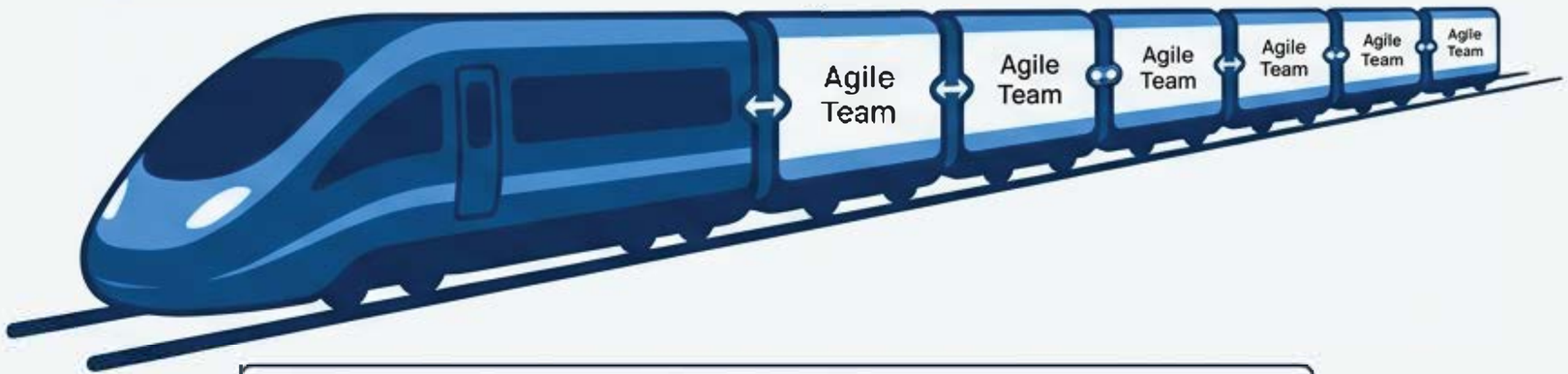
Отличия от Scrum

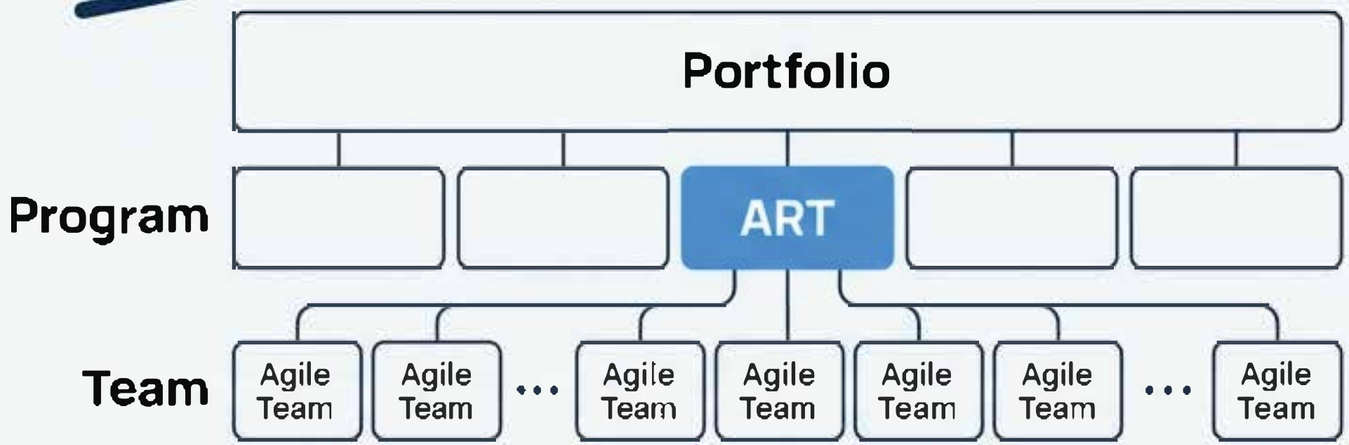
- Нет строгих спринтов; работа идет непрерывным потоком.
- Роли (PO, SM) не обязательны.
- Изменения можно вносить в любой момент.
- Идеально для команд поддержки и проектов с постоянным потоком разнородных задач.

SAFe (Масштабирование)

Scaled Agile Framework (SAFe): Фреймворк для применения Agile-принципов в очень больших организациях (50-125+ человек).

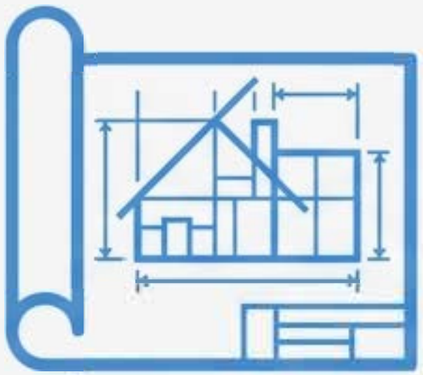
Agile Release Train (ART)





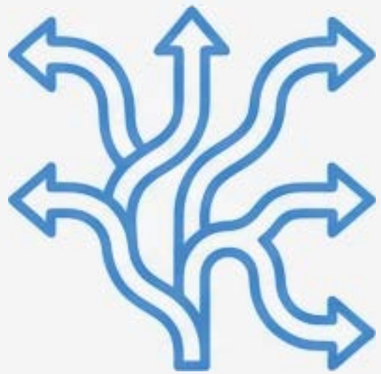
- **Цель:** Синхронизировать работу множества Agile-команд для достижения общих бизнес-целей.
- **Ключевое понятие — ART:** "Релизный поезд" — это долгоживущая команда команд (5-12 Agile-команд), которые совместно планируют, разрабатывают, тестируют и выпускают продукт.
- **Структура:** Добавляет новые уровни планирования (Program, Portfolio) поверх командного уровня Scrum для стратегического управления.

Сравнение: когда что выбирать?



Когда выбирать Waterfall / V-модель?

- Требования к проекту четкие, полные и не изменятся.
- Проект несложный, с предсказуемым результатом.
- Высокая цена ошибки, требуется строгая документация (госзаказы, медицина, авиация).



Когда выбирать Agile (Scrum / Kanban)?

- Требования неясны или быстро меняются.
- Нужно быстро вывести продукт на рынок (Time-to-Market).
- Высокая неопределенность и требуется гибкость.
- Проекты, где важна тесная обратная связь с пользователем (стартапы, продуктовая разработка).

Итоги модуля

- **Не существует 'плохой' или 'хорошей' методологии.** Есть только подходящая или неподходящая для конкретного контекста.
- **Выбор зависит от множества факторов:** целей проекта, требований, размера команды, бюджета, сроков и готовности к рискам.

- **Роль QA:** Понимание методологии помогает тестировщику эффективно выстраивать процессы, предотвращать дефекты и быть ценным участником команды.
- **Ваша задача на собеседовании:** Показать, что вы понимаете не только 'как' работает каждая модель, но и 'почему' и 'когда' ее следует применять.

Что такое тестирование

- **Определение:** Тестирование — это процесс проверки соответствия между реальным поведением программы и её ожидаемым поведением, описанным в требованиях.
- **Ключевая аналогия:** Мы сравниваем «то, что есть» с «тем, что должно быть».
- **Основная задача:** Поиск несоответствий (дефектов) до того, как их найдут пользователи.

«Тем, что должно быть» (требования)

«Тем, что есть» (реальность)

Цели тестирования

Предоставление информации о качестве

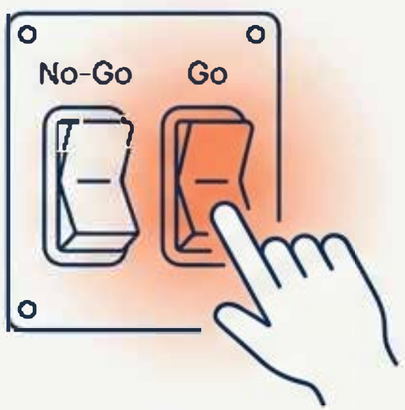
Дать команде и бизнесу объективные данные о текущем состоянии продукта.

Проверка соответствия требованиям

Убедиться, что реализованный функционал решает поставленные задачи и соответствует техническому заданию (ТЗ).

Основа для принятия решений

Помочь менеджеру проекта принять взвешенное решение о выпуске новой версии (релизе).



Обнаружение и предотвращение дефектов

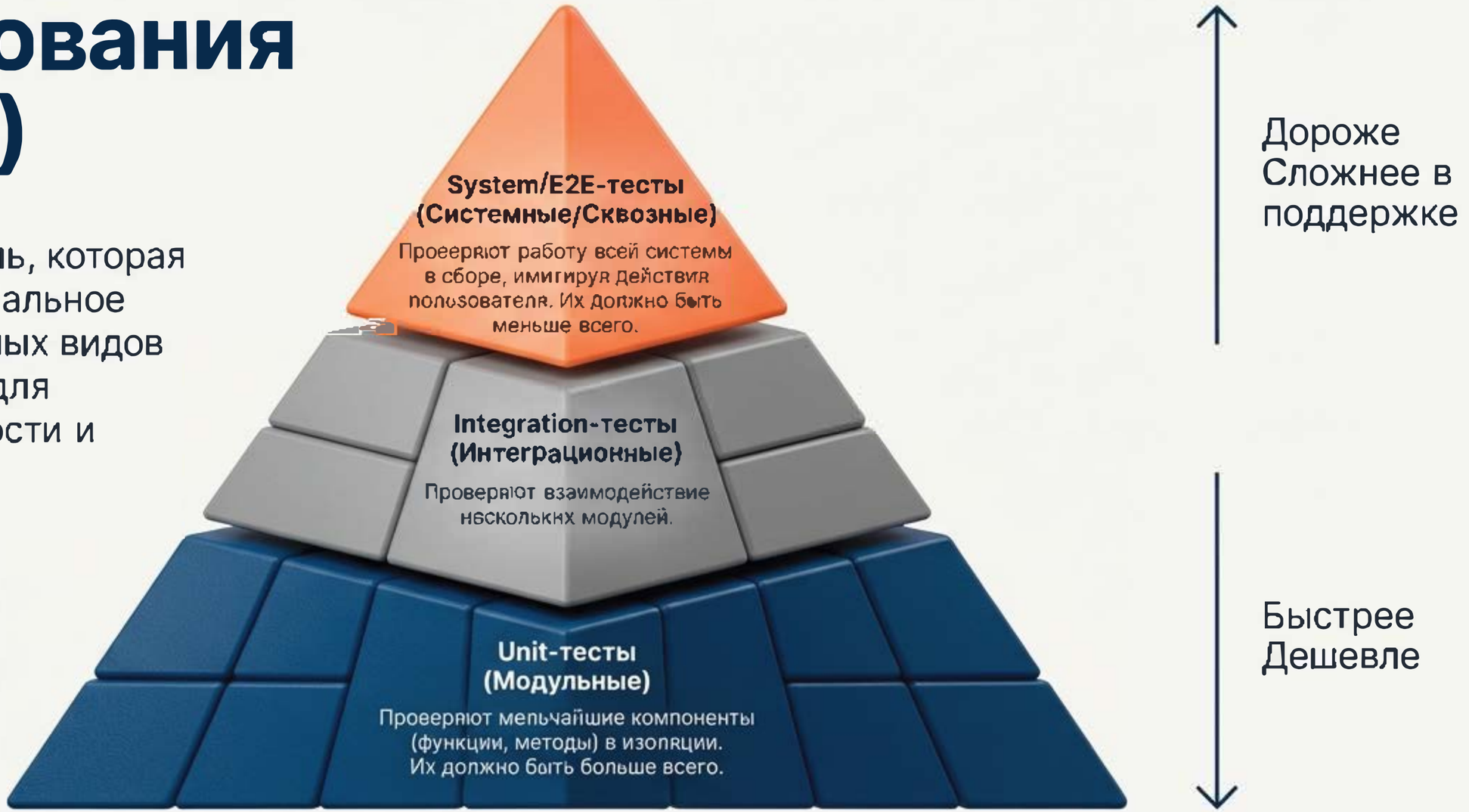
Найти ошибки как можно раньше, чтобы снизить стоимость их исправления и риски для бизнеса.

STLC (Жизненный цикл тестирования)

Важное различие: STLC (Software Testing Life Cycle) — это часть общего SDLC (Software Development Life Cycle), но с фокусом на активности по обеспечению качества.

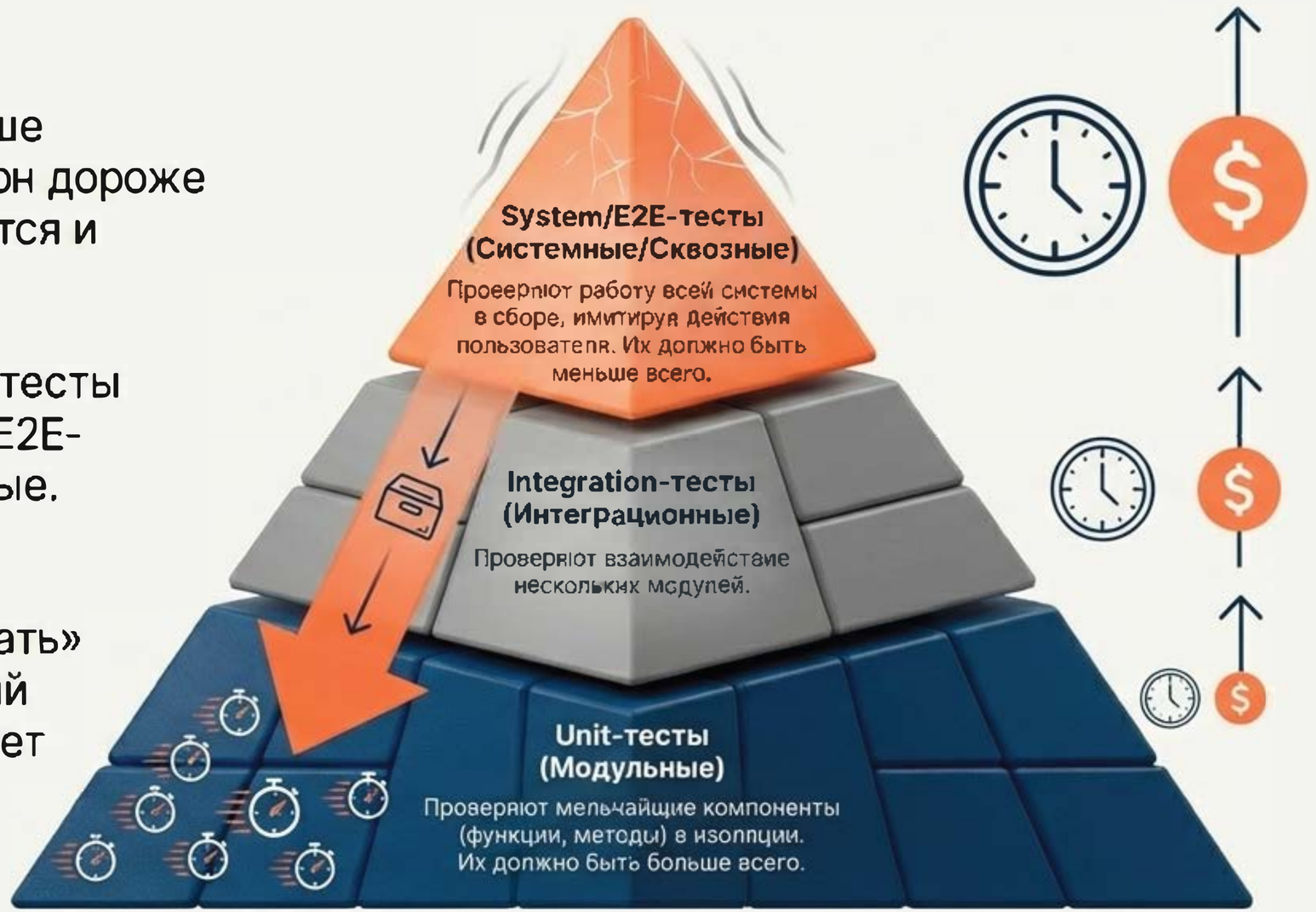
Пирамида тестирования (Схема)

Концепция: Модель, которая показывает оптимальное соотношение разных видов тестов в проекте для достижения скорости и стабильности.



Принципы пирамиды

- **Стоимость и скорость:** Чем выше уровень теста в пирамиде, тем он дороже в разработке, дольше выполняется и сложнее в поддержке.
- **Изоляция и стабильность:** Unit-тесты наиболее стабильны и быстры. E2E-тесты более хрупкие и медленные.
- **Стратегический принцип:** Необходимо стремиться «спускать» проверки на максимально низкий возможный уровень. Это ускоряет получение обратной связи (feedback) и удешевляет разработку.

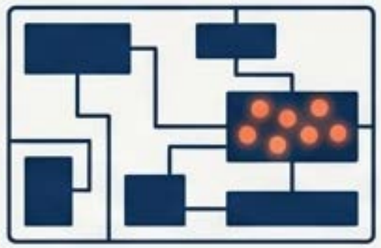


Принципы тестирования

Тестирование
демонстрирует
наличие дефектов,
а не их отсутствие.

Исчерпывающее
тестирование
недостижимо.

Раннее тестирование
экономит время
и деньги.



Скопление дефектов.

Парадокс пестицида.

Тестирование зависит
от контекста.

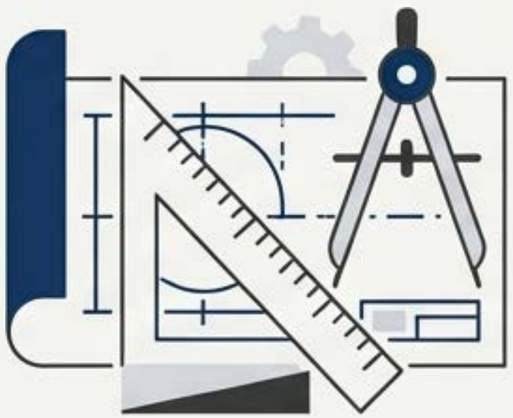
Заблуждение об
отсутствии ошибок.

«Делаем ли мы продукт правильно?»

Проверка соответствия продукта его спецификациям и требованиям.

Фокус на техническую корректность.

Пример: «Соответствует ли форма регистрации требованиям ТЗ?»



«Делаем ли мы правильный продукт?»

Проверка соответствия продукта ожиданиям и потребностям пользователя.

Фокус на ценность для конечного пользователя.

Пример: «Удобна ли эта форма регистрации для нашего пользователя?»

QA vs QC vs Testing

Качество ПО

Качество — это многогранное понятие. Оно определяется набором характеристик, важных для пользователя и бизнеса.

Функциональность (Functionality)

Выполняет ли ПО свои заявленные функции?

Удобство использования (Usability)

Насколько легко и приятно им пользоваться?

Поддержка (Maintainability)

Насколько легко вносить изменения и исправлять ошибки?

Надежность (Reliability)

Насколько стабильно ПО работает?

Производительность (Efficiency)

Насколько быстро и с какими ресурсами ПО выполняет задачи?

Безопасность (Security)

Насколько ПО защищено от угроз?

Метрики тестирования

Зачем нужны: Для объективной оценки процесса тестирования, качества продукта и эффективности команды.

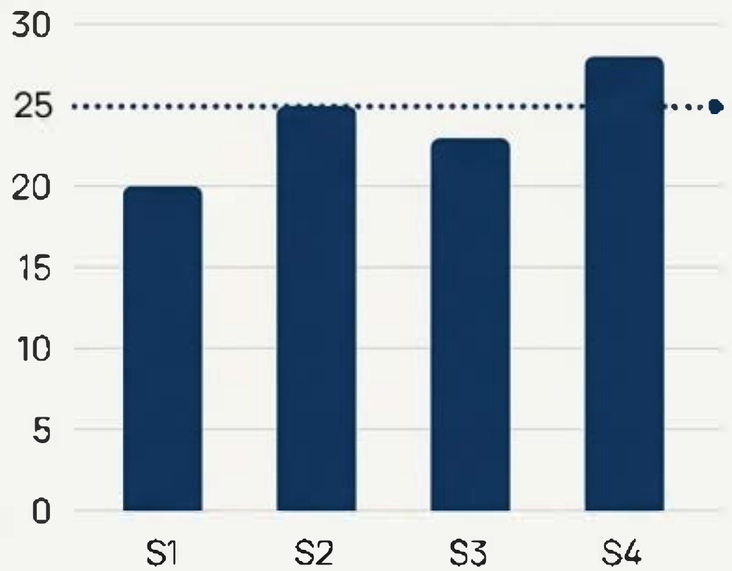
Количество дефектов

Показывает «здоровье» продукта и динамику его стабилизации.

Тестовое покрытие

Какой процент требований или кода покрыт тестами.

Скорость команды



Метрика из Scrum, которая помогает прогнозировать объем работы на следующий спринт.

Итоги модуля

Фундаментальные концепции

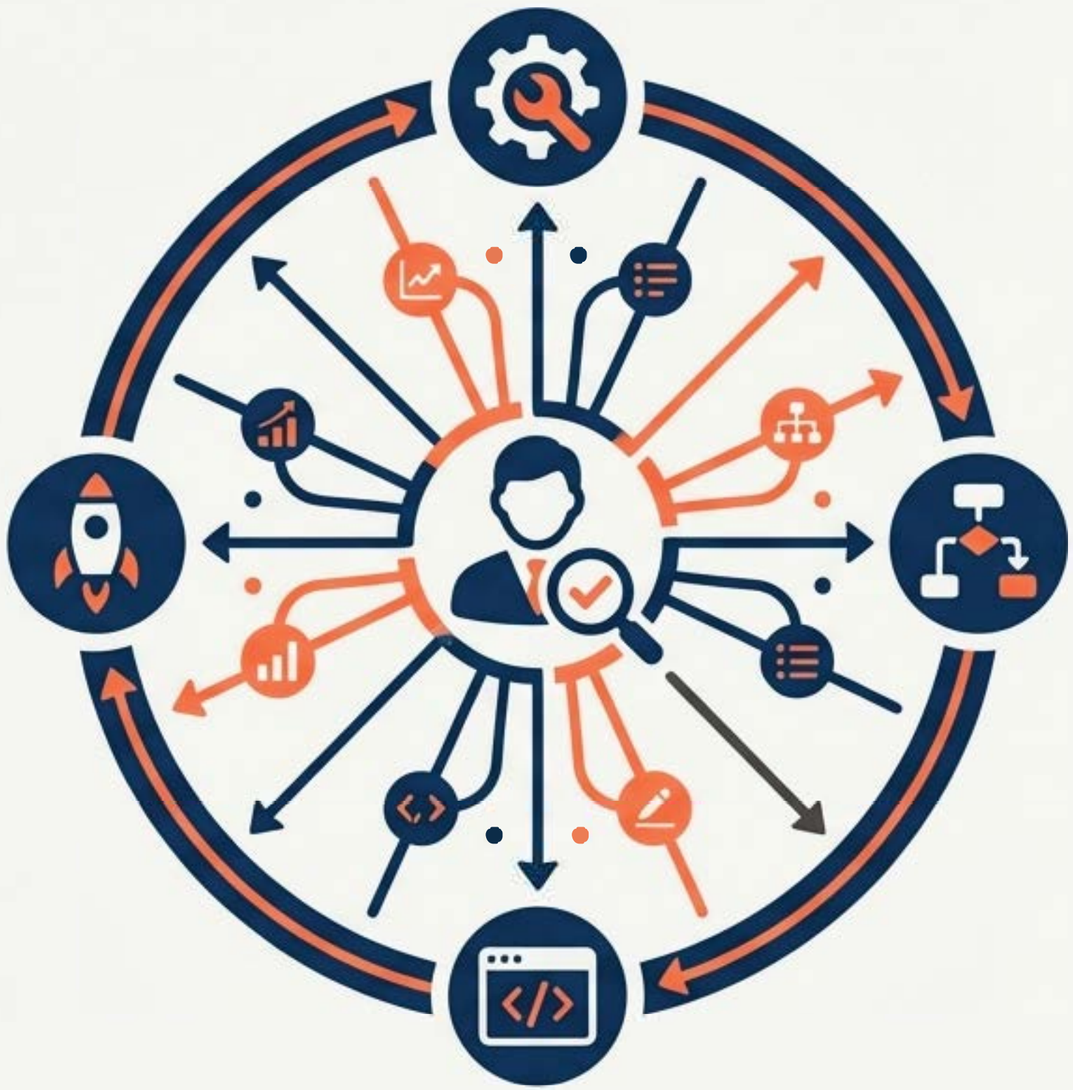
Понимание STLC и Пирамиды тестирования — это основа для построения эффективной стратегии качества.

Роль QA — сквозная

Специалист по обеспечению качества участвует на всех этапах жизненного цикла разработки (SDLC) — от анализа требований до релиза и поддержки.

Ваша цель

Не просто находить баги, а помогать команде создавать качественный продукт, улучшая процессы и предотвращая появление ошибок.



Уровни тестирования

Приемочное (Acceptance): Финальный этап. Проверка готовности продукта к выпуску и соответствия ожиданиям заказчика.

Системное: Полная проверка всей системы на соответствие требованиям. Эмуляция реального использования.

Интеграционное: Проверка корректного взаимодействия между несколькими модулями системы.

Типы тестирования (Обзор)

Функциональные

Проверяют, **ЧТО** делает система.
Соответствует ли её поведение бизнес-логике?

Нефункциональные

Оценивают, **КАК** работает система
(производительность, удобство, безопасность).

Структурные (White Box)

Анализ внутренней структуры кода без его
запуска (например, Code Review).

Связанные с изменениями

Проводятся после внесения правок в код
(Re-test, Regression).

Функциональное тестирование

Главная цель: Убедиться, что ПО выполняет все заявленные функции согласно требованиям.

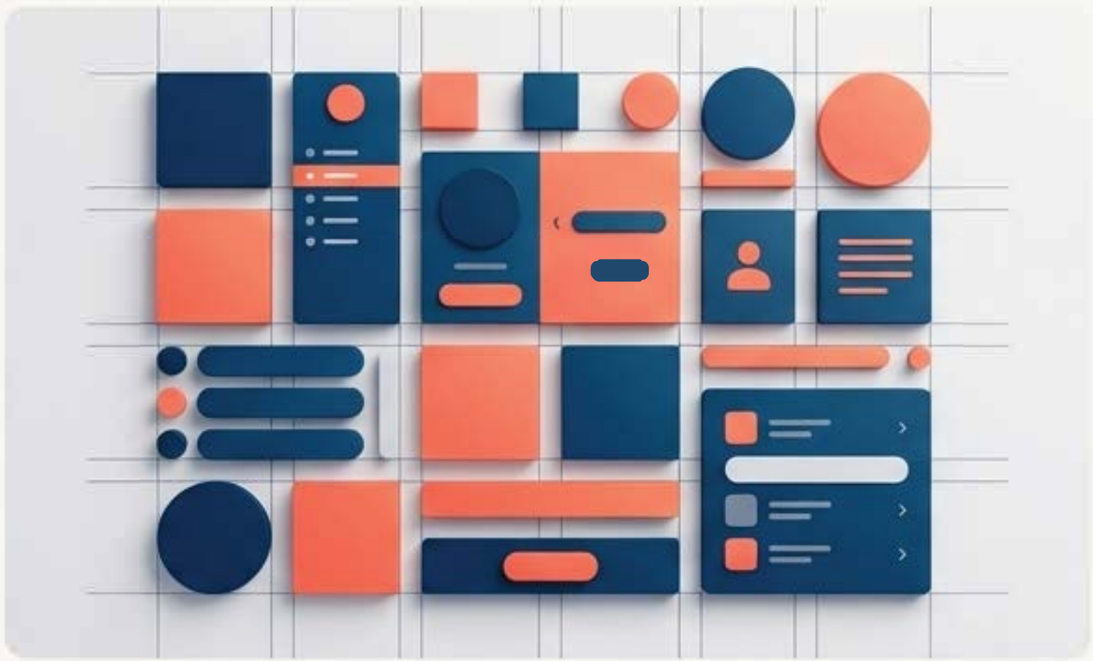
Отвечает на вопрос: «Делает ли система то, что от неё ожидается?»

Основа для тестов: Техническое задание, пользовательские истории (user stories), спецификации.

Примеры: Корректность работы форм, правильность вычислений, соответствие бизнес-процессам.

Нефункциональное: UI и Usability

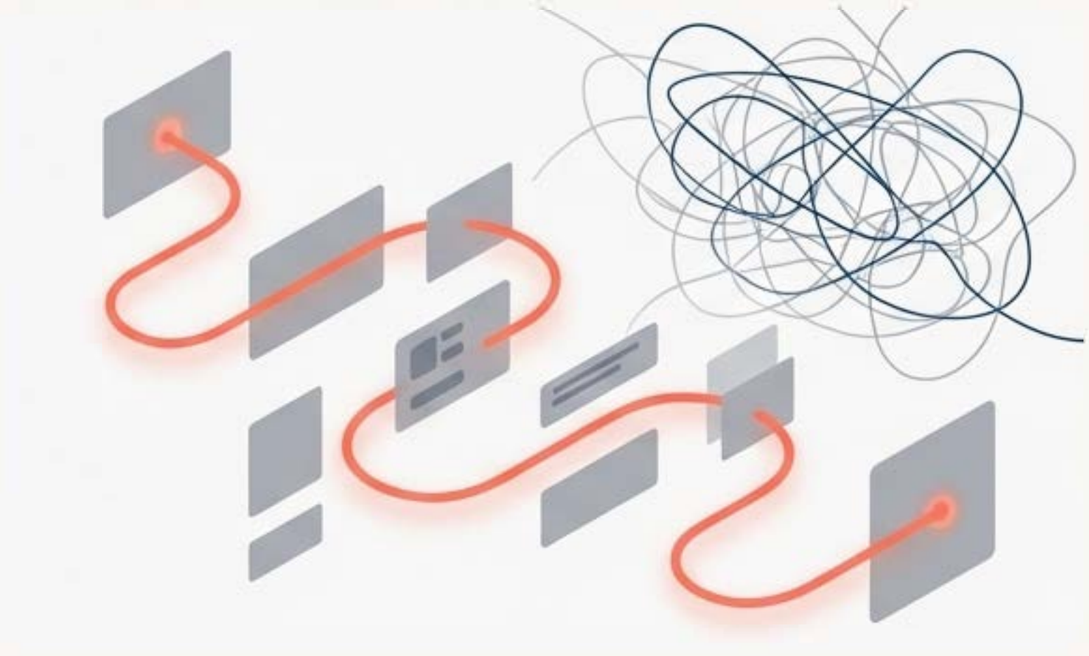
UI Testing
(Тестирование интерфейса)



Фокус на визуальной составляющей.

- Как выглядит? Проверка верстки, соответствия макетам (цвета, шрифты, отступы).

Usability Testing
(Тестирование удобства)



Фокус на пользовательском опыте.

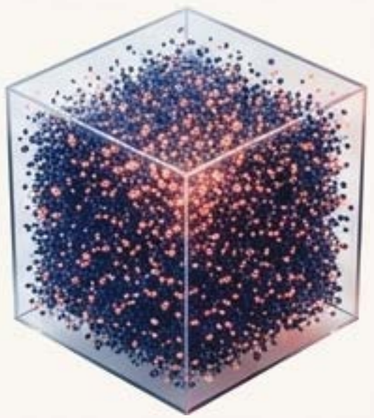
- Насколько удобно пользоваться? Интуитивность, простота навигации, логичность действий.

Нефункциональное: Производительность

Цель: Оценить скорость, стабильность и отклик системы под разной нагрузкой.

Load testing (Нагрузочное): Работа при ожидаемой, штатной нагрузке.

Stress testing (Стрессовое): Поведение на пиковых нагрузках для определения точки отказа.



Volume testing (Объемное): Работа с большими объемами данных в базе.

Stability testing (Стабильности): Длительная работа под стандартной нагрузкой.

Нефункциональное: Окружение и Безопасность

Конфигурационное/Кроссбраузерное: Проверка работы в различном окружении (ОС, браузеры, устройства).

Security Testing (Тестирование безопасности): Поиск уязвимостей в системе.

- Проверка прав доступа.

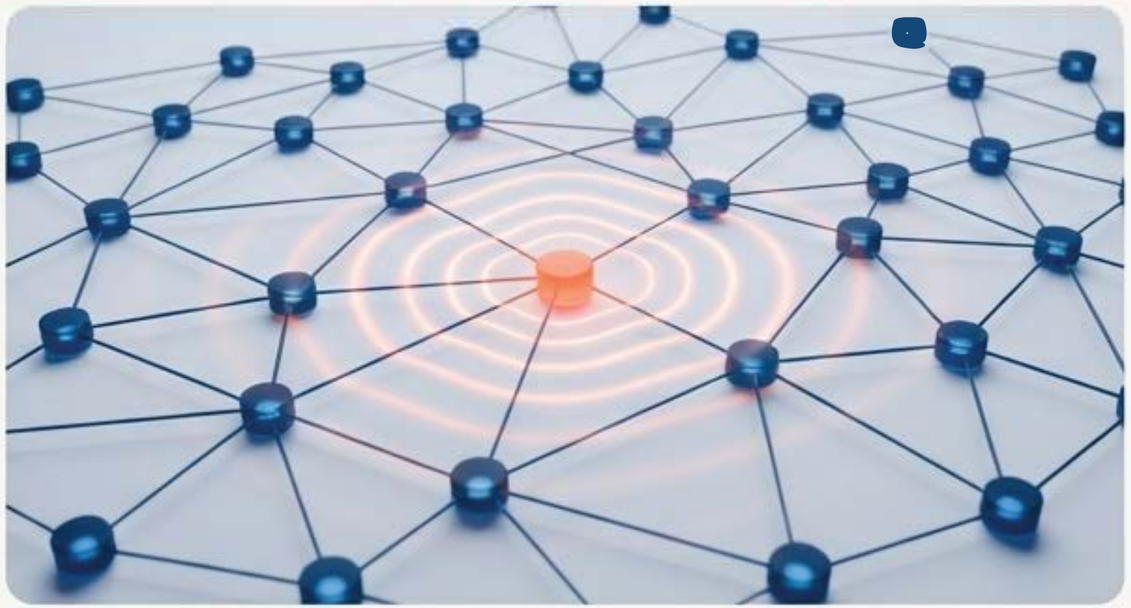
- Проверка прав доступа.
- Защита от атак (SQL-инъекции, XSS).
- Сохранность и конфиденциальность данных.

Тестирование изменений (Re-test vs Regression)

Re-test (Повторное)

Целенаправленная проверка, что конкретный баг был исправлен.

Regression (Регрессионное)



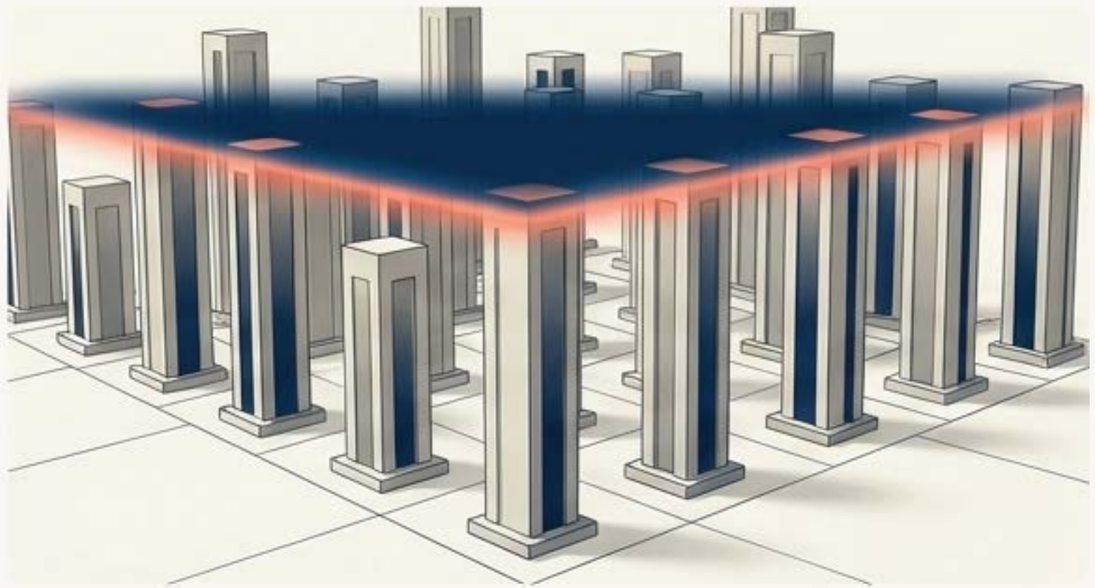
Проверка, что исправление или новая функция не сломали существующий функционал.

Ключевое отличие: Re-test проверяет **исправление**, Regression — **отсутствие новых дефектов**.

Важность: Регресс критически важен перед каждым релизом.

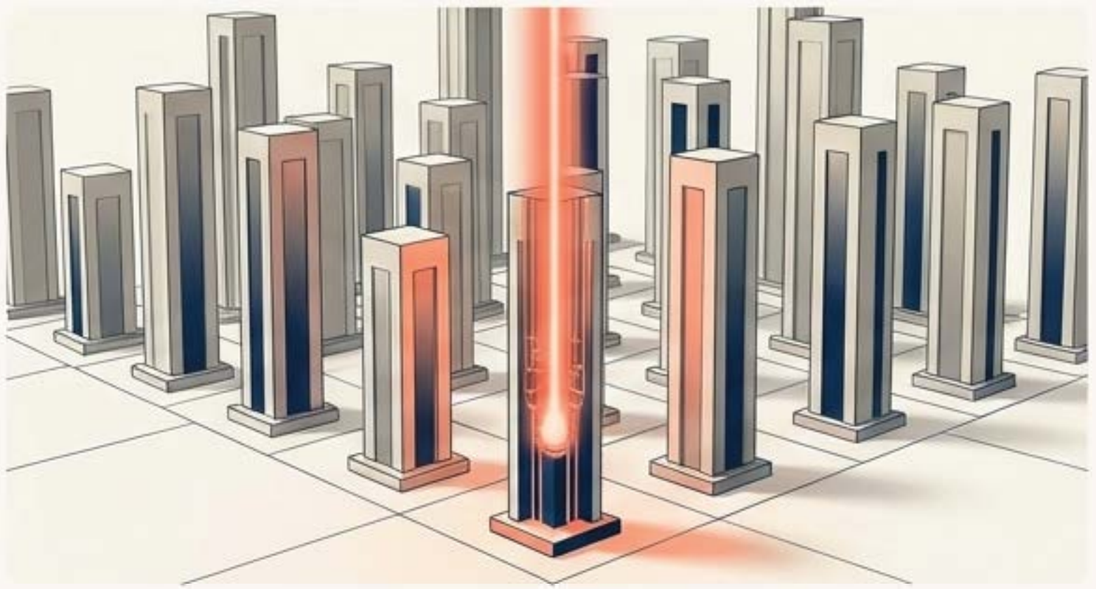
Smoke и Sanity

Smoke (Дымовое): “Вширь”



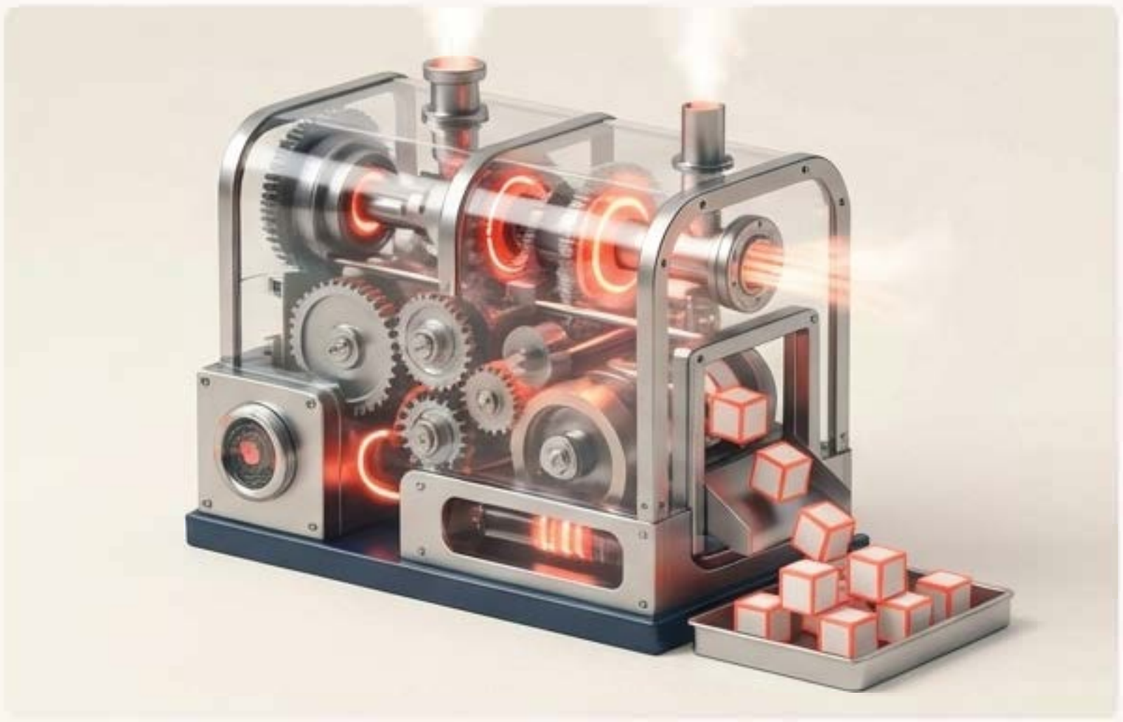
- Быстрая, поверхностная проверка самого критичного функционала.
- Ответ на вопрос: «Система вообще запускается и жизнеспособна?»

Sanity (Санитарное): “Вглубь”



- Узконаправленная проверка конкретной области после небольших изменений.
- Ответ на вопрос: «Исправленный модуль и связанные с ним функции работают?»

Статическое vs Динамическое



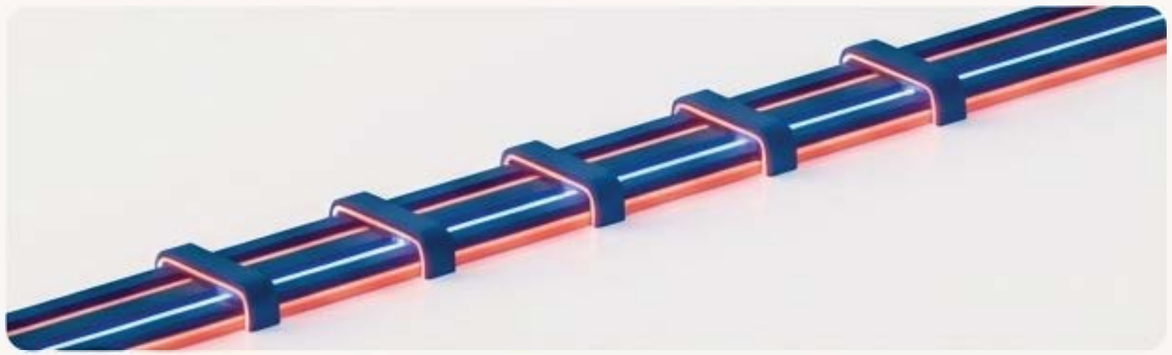
Статическое тестирование

- Проверка **без запуска кода**.
- Цель — найти дефекты на самых ранних этапах.
- Примеры: вычитка требований, ревью кода (Code Review), проверка макетов.

Динамическое тестирование

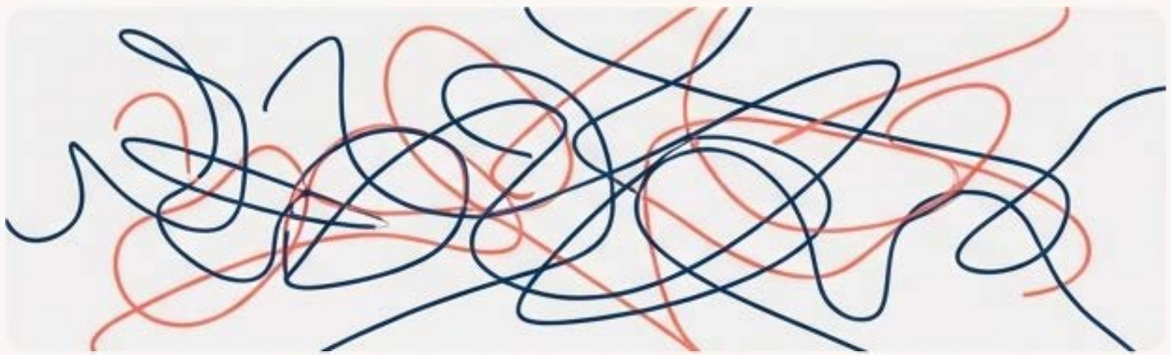
- Проверка **путем запуска кода** и взаимодействия с приложением.
- Примеры: любое функциональное или нефункциональное тестирование («черный», «белый» ящик).

Подходы к тестированию



Сценарное (Scripted):
Строгое следование заранее подготовленным тест-кейсам.
Предсказуемость и хорошее покрытие.

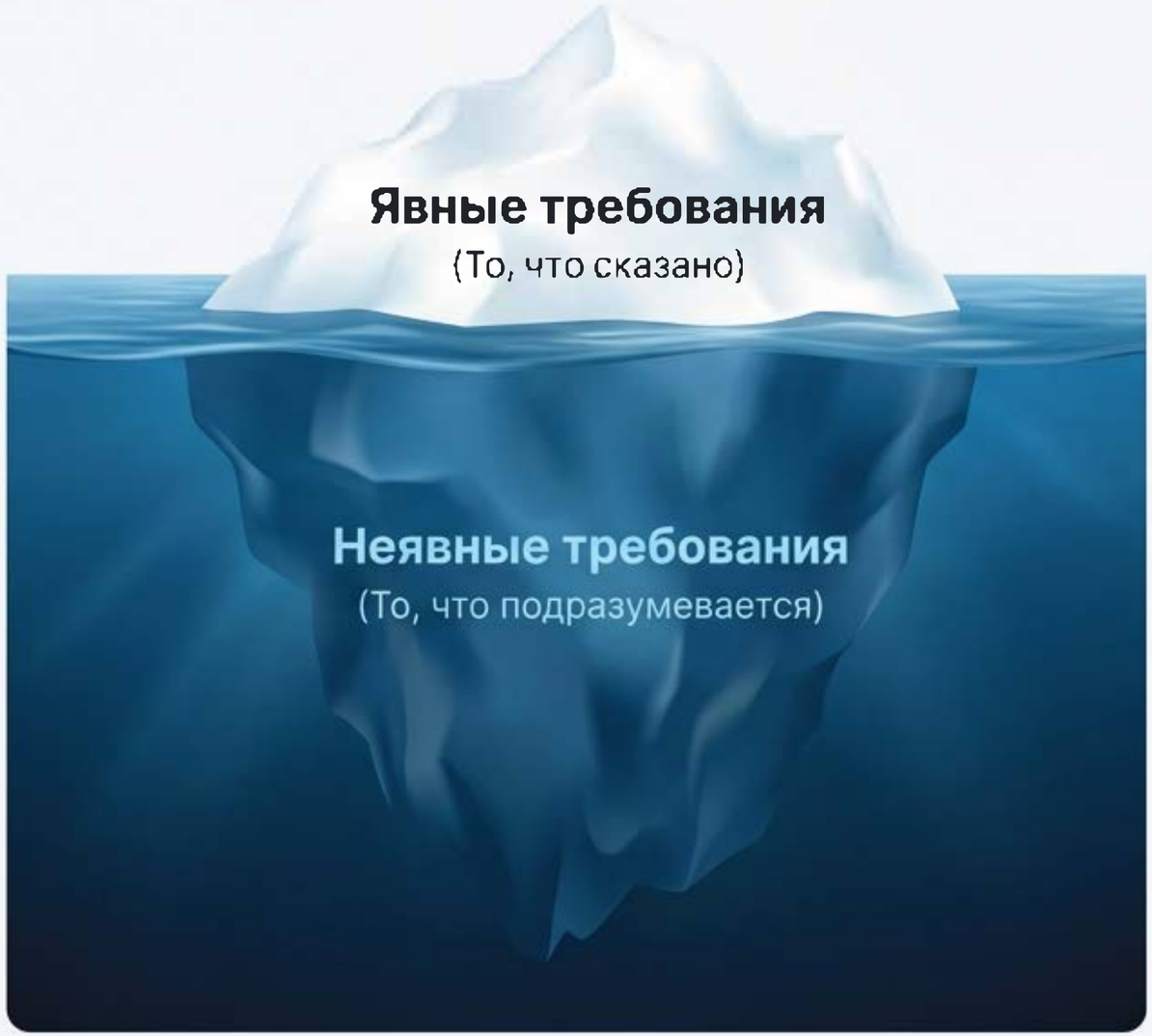
Исследовательское (Exploratory):
Одновременное изучение системы, проектирование тестов и их выполнение.
Поиск неочевидных багов.



Интуитивное (Ad-hoc): Тестирование без плана, где без плана, основанное на импровизации («метод случайного тыка»).

Итоговая карта

Требования: Фундамент качества



Функциональные
(Что система ДЕЛАЕТ?)

Пользователь может войти в систему по логину и паролю.

Нефункциональные
(Как система РАБОТАЕТ?)

Вход в систему занимает не более 2 секунд.

Анатомия идеального требования

Атомарность

Одно требование = одна мысль.

Полнота

Вся необходимая информация на месте.

Непротиворечивость

Не конфликтует с другими требованиями.

Тестируемость

Можно доказать, что оно выполнено.

Практика: Найди ошибку в требовании

Система должна позволять пользователю загружать **аватар**. Изображение должно быть в формате JPG. После загрузки аватар должен быстро отображаться в профиле.

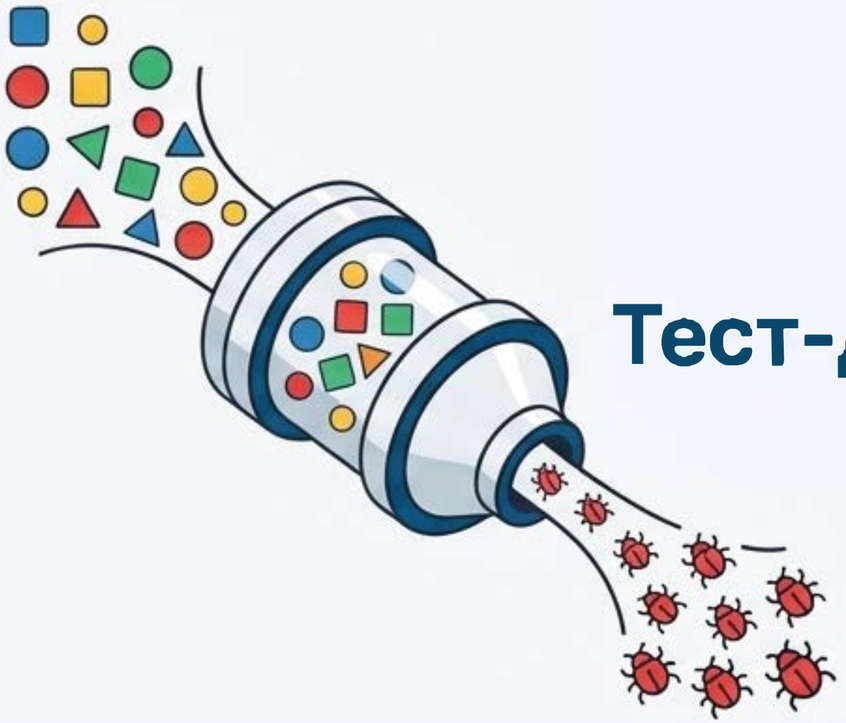
Какой максимальный размер файла? Какое разрешение?

A jpeg? PNG? GIF? Что будет, если загрузить другой формат?

«Быстро» — это сколько в секундах? 1? 5?

Что такое Тест-Дизайн? Работаем умнее, а не больше

Проверить ВСЁ



Тест-дизайн

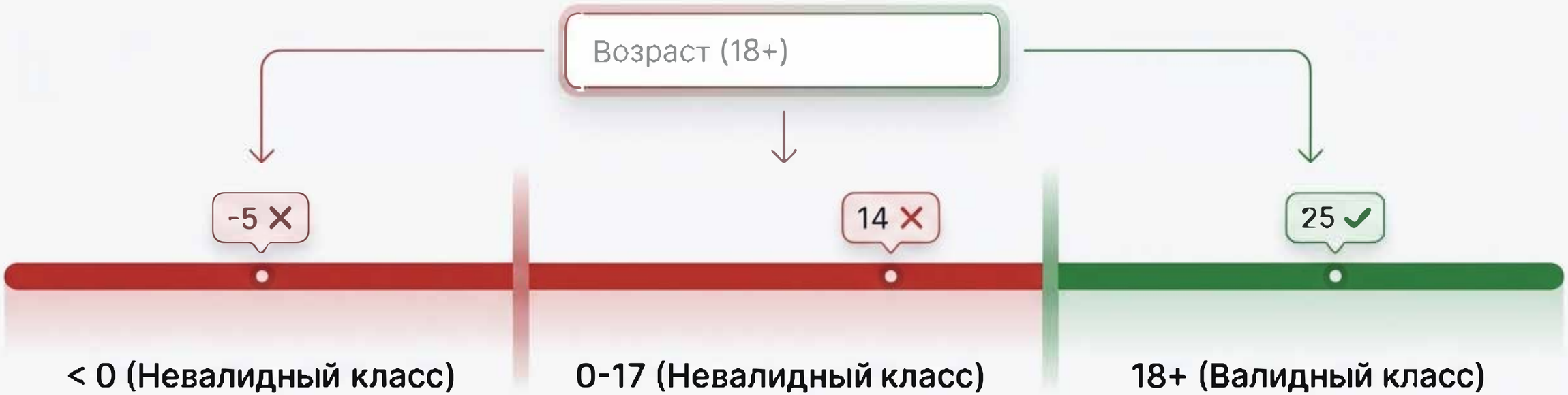
Минимум тестов = Максимум покрытия

ЧТО тестировать?
(Приоритеты и области риска)

КАК тестировать?
(Выбор оптимальной техники)

Техника №1: Классы Эквивалентности

Суть метода: Разбить все возможные входные данные на группы (классы), в которых программа ведет себя одинаково. Достаточно взять по одному представителю из каждого класса.



Итог: Вместо сотен проверок — всего три.

Техника №2: Анализ Граничных Значений

Ключевая идея: Ошибки обожают прятаться на границах диапазонов.

Техника №3: Таблица Принятия Решений

Когда использовать: Когда результат зависит от комбинации нескольких условий (сложная бизнес-логика).

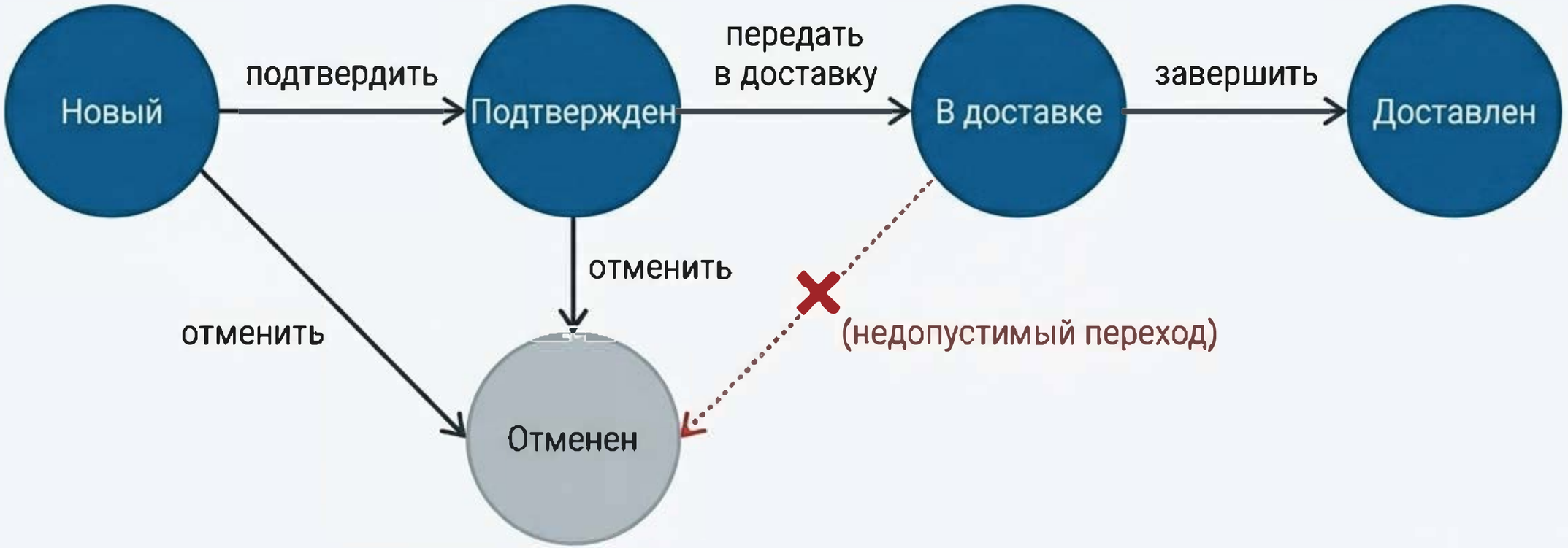
Решение о выдаче кредита

Один тестовый сценарий

Условия \ Правила	**П1**	**П2**	**П3**	**П4**
Возраст > 25 лет	Да	Да	Нет	...
Стаж > 2 лет	Да	Нет	Да	...
Доход > 100 000	Да	Да	Да	...
ДЕЙСТВИЕ: Выдать кредит	Да	Нет	Нет	...

Техника №4: Состояния и Переходы

Когда использовать: Для тестирования объектов, которые имеют жизненный цикл (заказ, документ, задача).



Техника №5: Попарное тестирование (Pairwise)

- **Проблема:** Комбинаторный взрыв. Проверить все сочетания параметров невозможно.
- **Идея:** Большинство багов возникает при взаимодействии **пары** параметров.

Полный перебор (8 тестов)

ОС	Браузер	Язык
Win	Chrome	RU
Win	Chrome	EN
Win	Firefox	RU
Win	Firefox	EN
Mac	Chrome	RU
Mac	Chrome	EN
Mac	Firefox	RU
Mac	Firefox	EN

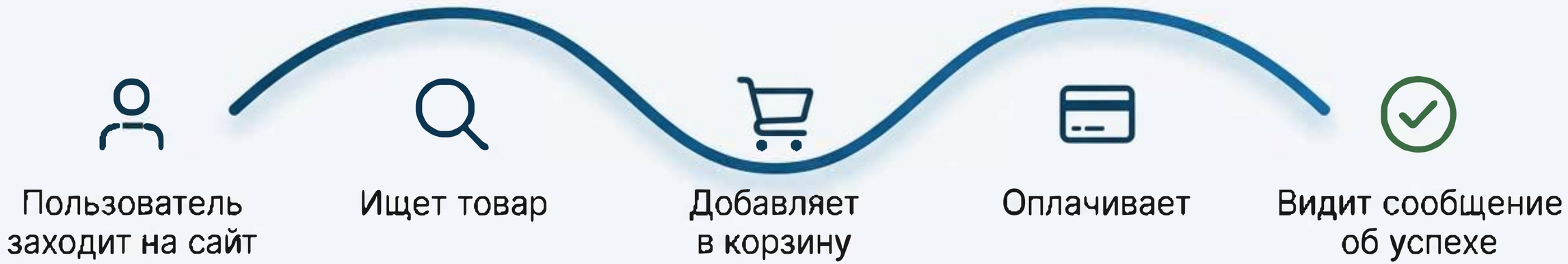
Pairwise (4 теста)

ОС	Браузер	Язык
Win	Chrome	RU
Win	Firefox	EN
Mac	Chrome	EN
Mac	Firefox	RU

- **Инструменты:** PICT, Allpairs

Техника №6: Use Case Testing

Фокус: Не на отдельных функциях, а на целостных пользовательских сценариях.
Идея: Пройти путь пользователя от начала до конца для достижения его цели.



Ключевые слова: Сквозные сценарии, бизнес-логика, интеграция.

Заглянем под капот: White Box и Покрытие Кода

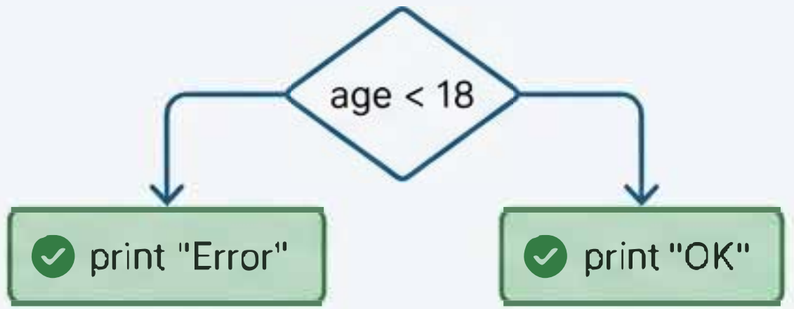
Идея: Оценить, насколько хорошо наши тесты «прошлись» по коду.

```
if (age < 18) {
  print "Error"
} else {
  print "OK"
}
```

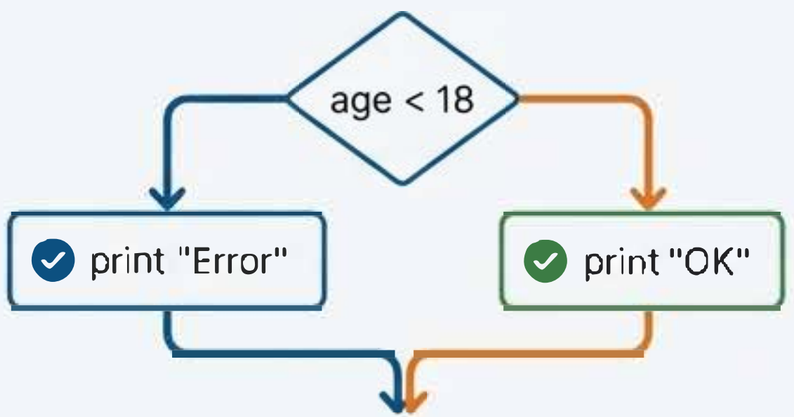
Зачем это QA?

- Чтобы измерить качество нашего тестового покрытия.
- Чтобы найти «темные углы» в коде, которые мы не протестировали.
- Чтобы говорить с разработчиками на одном языке.

Statement Coverage (Покрытие операторов)



Branch Coverage (Покрытие ветвлений)



Итоги модуля: От требований к уверенности

Ключевые выводы

- Качество начинается с анализа требований.
- Тест-дизайн — это не дополнительные затраты, а способ сэкономить время и ресурсы.
- Правильно подобранные техники находят глубокие и нетривиальные баги.
- Ваша цель — не «сломать» программу, а обеспечить уверенность в ее работе.

Workflow и Инструменты (TMS)

Зачем нужны системы управления задачами?

- Обеспечение полной прозрачности процесса для всей команды.
- Централизованное отслеживание статуса каждой задачи: от идеи до релиза.
- Фиксация затраченного времени для корректной оценки будущих работ.

Jira

YouTrack

Azure DevOps

Redmine

Trello

Ключевые инструменты на рынке:

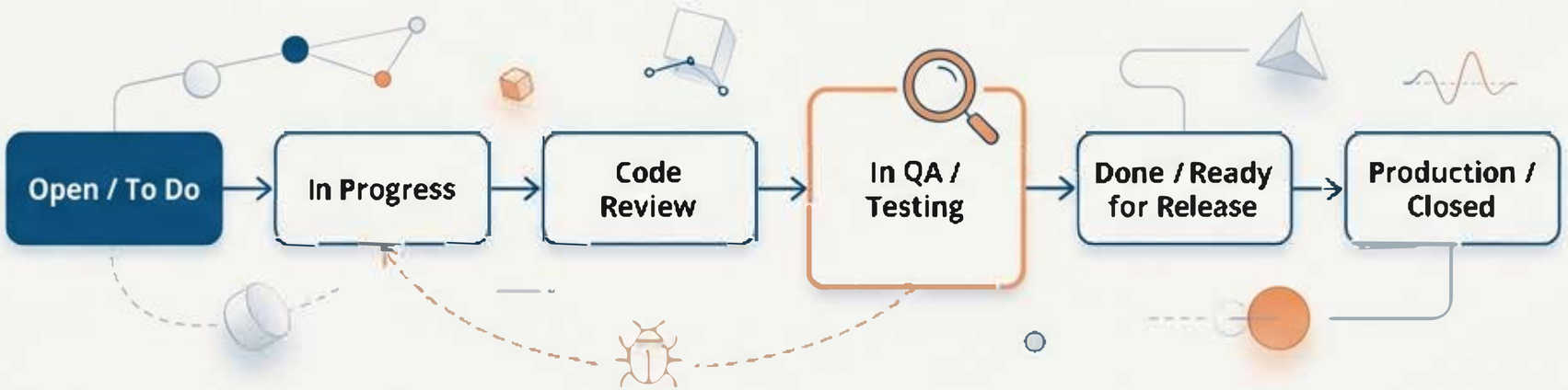
- Jira: отраслевой стандарт, особенно в гибких методологиях.
- YouTrack, Azure DevOps, Redmine, Trello: популярные альтернативы.

Что это значит для QA?

- Это ваш основной рабочий инструмент. Все задачи, как на разработку новой функции (фичи), так и на исправление дефекта (бага), живут здесь.

Жизненный цикл задачи (Lifecycle)

Жизненный цикл (Workflow) — это путь, который проходит задача от создания до полного завершения.



Типичный маршрут задачи:

- **Open / To Do**: Новая задача, ожидает взятия в работу.
- **In Progress**: Разработчик пишет код.
- **Code Review**: Другой разработчик.
- **Code Review**: Другой разработчик проверяет качество кода.
- **In QA / Testing**: **Ваш этап!** QA-специалист проверяет функциональность.
- **Done / Ready for Release**: Задача протестирована и готова к выпуску.
- **Production / Closed**: Задача выпущена и доступна пользователям.

Роль QA в этом цикле:

- На этапе **Testing** вы — главный фильтр качества.
- Если найден баг, задача возвращается разработчику (**In Progress**) с подробным описанием дефекта.

Бэклог и Груминг

Бэклог (Backlog):

- Это приоритизированный список всех задач по проекту: новые фичи, улучшения, баги, технический долг.
- За его формирование и приоритеты отвечает Владелец Продукта (Product Owner).

Груминг (Grooming / Backlog Refinement):

- Регулярная встреча команды для "причесывания" бэклога.
- **Цель:** Детально разобрать задачи из верхней части списка, уточнить требования и оценить сложность перед началом следующего спринта.
- **Результат:** Команда готова к планированию, так как все понимают, что и как нужно делать.

Тестовая документация: Обзор

Тестовая документация (артефакты) — это основа системного и прозрачного подхода к обеспечению качества.

Отвечает на вопросы "Что, как и когда мы тестируем?"

Краткий перечень того, *что* нужно проверить.

Детальное описание того, *как* именно проводить проверку.

Документ, описывающий найденный дефект.

Сводка о результатах тестирования и качестве продукта.

Тест-план

Тест-план — это главный стратегический документ, описывающий весь объем работ по тестированию для релиза или крупной фичи.

Что тестируем?
(Объект и области тестирования, функционал).

Кто тестирует?
(Ресурсы, роли и ответственность).

Когда тестируем?
(Сроки, этапы, график).

Как тестируем?
(Виды тестирования, подходы, инструменты).

Обязательно включает:

Критерии начала тестирования: (Например, билд готов и развернут на тестовом стенде).

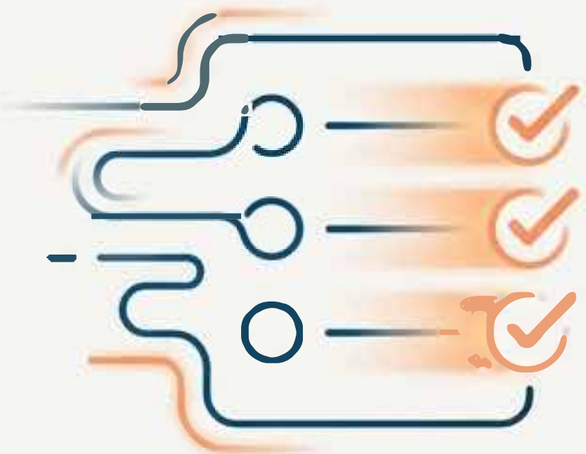
Критерии окончания тестирования: (Например, все критичные баги исправлены, 95% тест-кейсов пройдены).

Оценка рисков: (Что может пойти не так и как мы будем на это реагировать).

Чек-лист

Чек-лист — это простой и гибкий список проверок в формате "Проверить, что...".

Плюсы:



Скорость: Быстро создается и легко обновляется.

Гибкость: Позволяет опытному тестировщику применять исследовательский подход.

Покрытие: Дает общее представление о том, какие области были проверены.

Минусы:

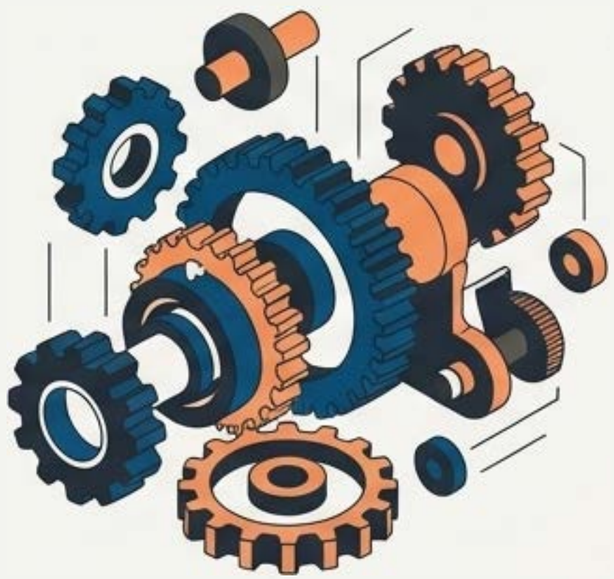
Неоднозначность: Разные специалисты могут трактовать один и тот же пункт по-разному.

Нехватка деталей: Сложно использовать новичкам без глубокого знания продукта.

Тест-кейс

Тест-кейс — это детализированная пошаговая инструкция для проверки конкретного требования.

Ключевое отличие от чек-листа: Детерминированность. Любой специалист, взяв тест-кейс, должен выполнить его одинаково и прийти к тому же результату.



Стандартная структура:

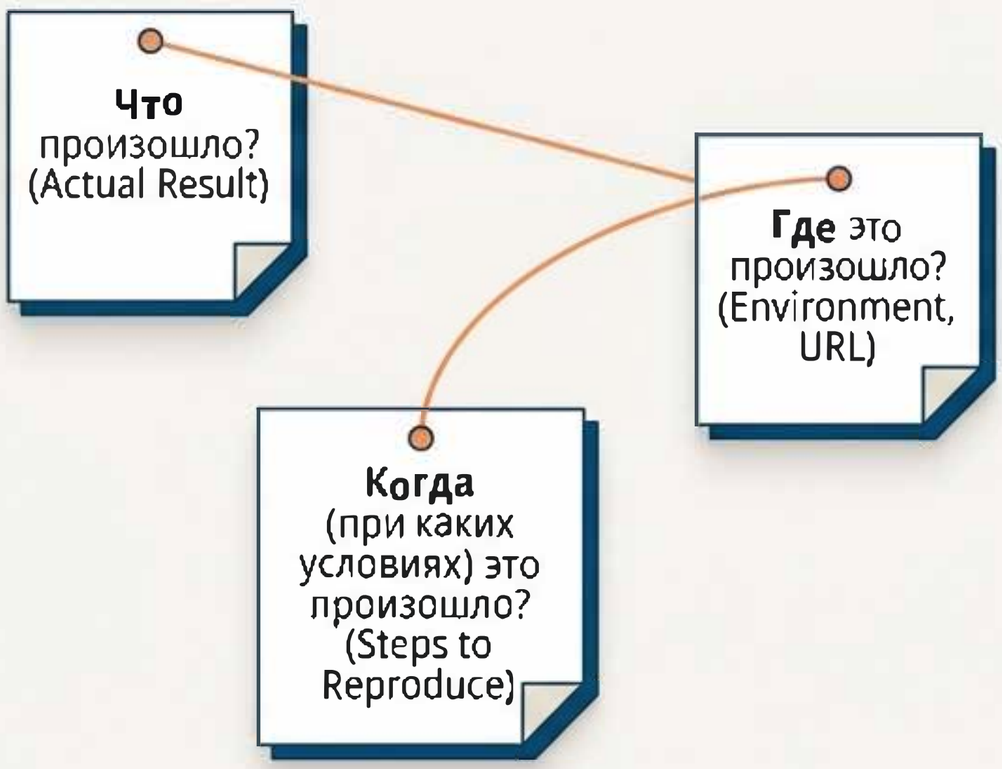
- **ID:** Уникальный идентификатор.
- **Title:** Краткое и понятное название.
- **Preconditions:** Предусловия (что должно быть выполнено до начала теста).
- **Steps:** Пошаговые действия.
- **Expected Result:** Ожидаемый результат (что должно произойти, если все работает правильно).
- **Postconditions:** Постусловия (состояние системы после теста).

Баг-репорт

Баг-репорт — это документ, который четко и лаконично описывает дефект.

Главная цель: Помочь разработчику быстро понять проблему и воспроизвести её без лишних вопросов.

Принцип "Что? Где? Когда?":



Обязательные поля:

- **Summary:** Краткое и емкое название бага.
- **Steps to Reproduce:** Четкие шаги для воспроизведения.
- **Actual Result:** Что произошло на самом деле.
- **Expected Result:** Что должно было произойти.
- **Environment:** Окружение (браузер, ОС, версия сборки).
- **Attachments:** Скриншоты, видео, логи.

Приоритезация (Severity vs Priority)

Severity (Серьезность) — это техническая оценка влияния бага на систему.

- Оценивается QA-специалистом.
- *Уровни:* Blocker, Critical, Major, Minor, Trivial.

Priority (Приоритет) — это бизнес-оценка срочности исправления бага.

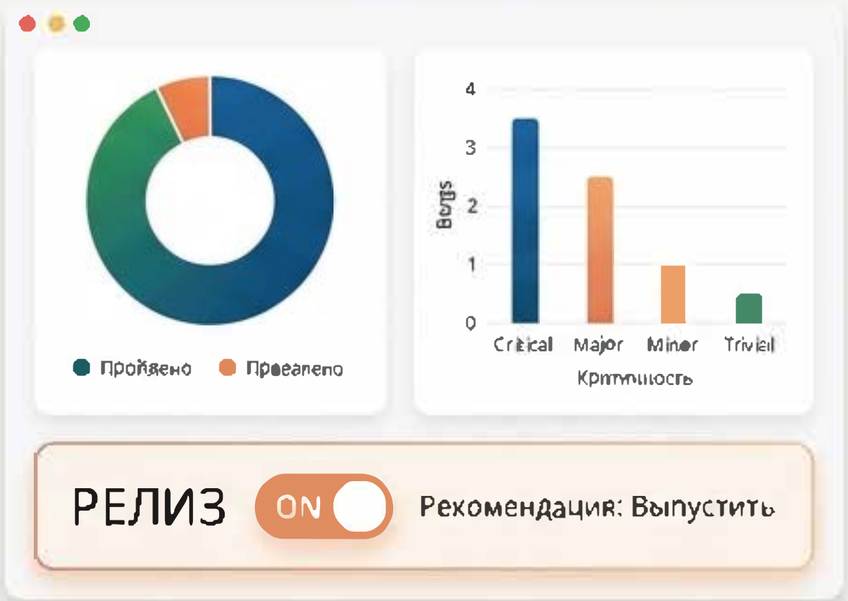
- Определяется менеджером или владельцем продукта.
- *Уровни:* High, Medium, Low.

Severity ≠ Priority! Классический пример:

- Опечатка в названии компании на главной странице.
- **Severity**: `Trivial (технически ни на что не влияет).
- **Priority**: `High (критично для имиджа, нужно исправить срочно).

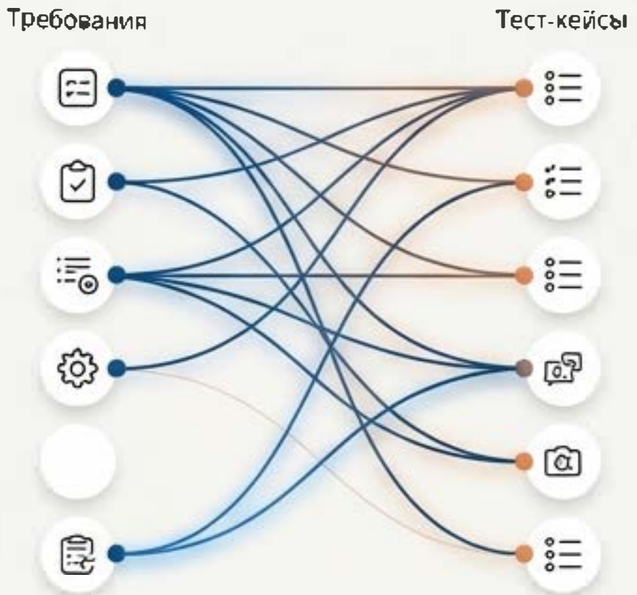
Отчеты и Трассировка

Test Summary Report (Итоговый отчет о тестировании):



- Документ, который подводит итоги этапа тестирования (например, перед релизом).
- Содержит: что было протестировано, сколько тестов пройдено/провалено, сколько найдено багов, и главное — **вывод о качестве текущей версии продукта** и рекомендация к выпуску.

Traceability Matrix (Матрица трассируемости):



- Таблица, которая связывает требования с тест-кейсами, которые их проверяют.
- **Цель:** Убедиться, что **каждое** требование покрыто тестами, и ничего не упущено. Позволяет визуально оценить полноту тестового покрытия.

Финал: Сквозной E2E пример

Проследим путь одной фичи от идеи до пользователя:

QA — это не грость "поиск ошибок".

* Это обеспечение и управление качеством продукта на **всех** этапах его жизненного цикла.
* Ваша роль — быть адвокатом пользователя и хранителем стабильности продукта.
* **Ключ к успеху — системный подход, любознательность и эффективная коммуникация.**
* Эта структура — ваша основа для уверенного старта.

Спасибо за внимание и удачи на собеседованиях!