ТИНЬКОФФ

Неизвестные пробелы в тестовом покрытии

Понять, расщепить, повторить

Кузнецов Дмитрий

23.04.2024

Дмитрий Кузнецов

Верстальщик-архитектор

НГТУ’09

2ГИС’15

С 2017 в Tinkoff

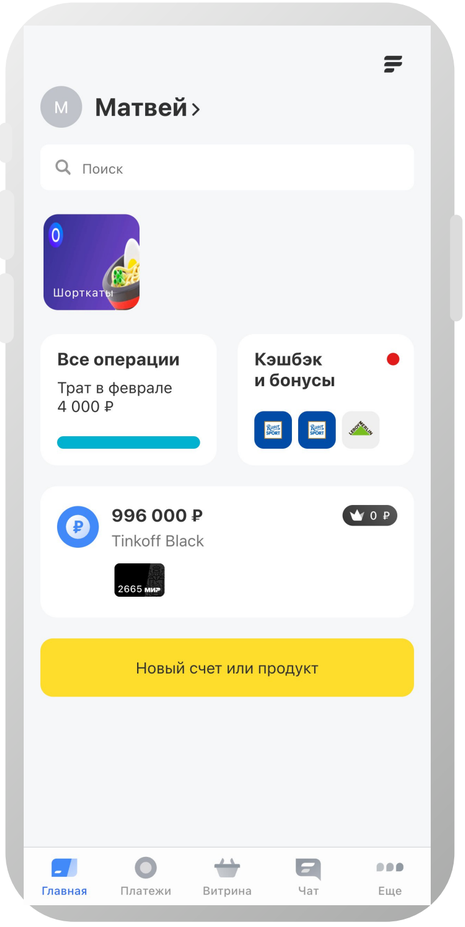
Увлекаюсь тестированием и автоматизацией. Люблю такие архитектурные характеристики testability, maintainability и developer experience.

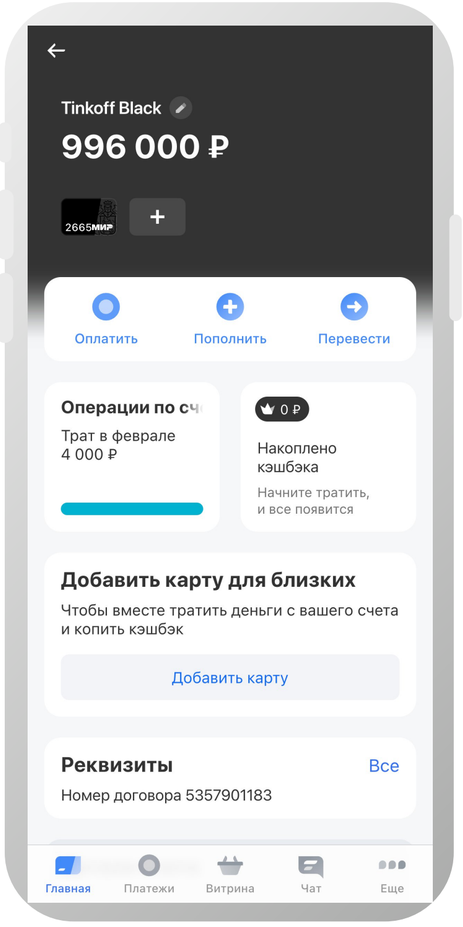
Личный кабинет

Tinkoff PWA

PWA версия — платформа с рекордной скоростью доставки в прод, лишенная санкционных рисков.

- 15+ продуктовых команд
- 100+ фронтовых разработчиков
- Ежедневные релизы





ТИНЬКОФФ

I. Проблемы

Пробелы в покрытии

Пирамида из двух десятков видов тестов

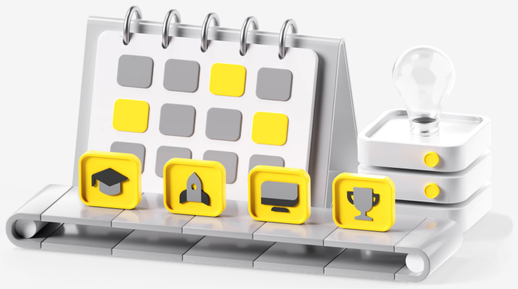
Test case as code

Выгрузка в Allure

Трудно посчитать тестовое покрытие

Тестов много, а баги в проде есть

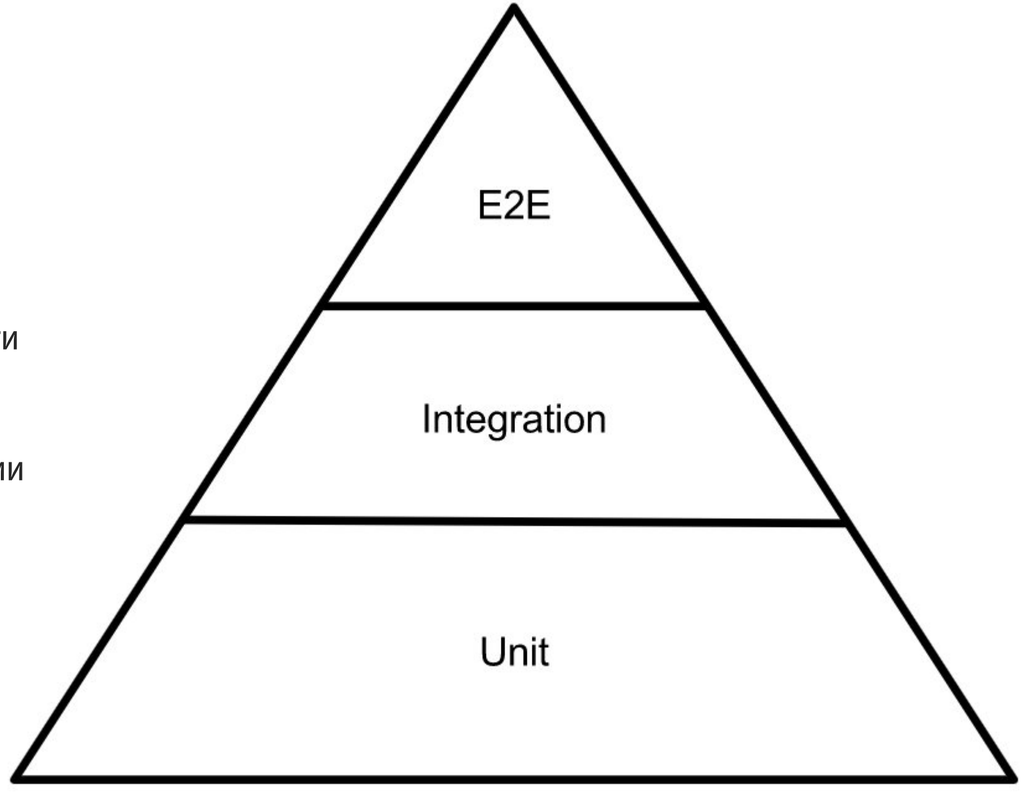
Первые выводы и действия



- Инструкции и обучение
- Изменили процесс написания спецификаций
- Тесткейсы до кода
- Структурирование тесткейсов
- Всё равно баги в проде :(

Странная пирамида

1. Юниты
    1. Позитивные
    2. Негативные
2. Микрофронтенд без браузера
    1. Неинтерактивная логика
    2. Реакция на ролевую модель
    3. Вызов колбеков
    4. Невидимые атрибуты верстки
3. Функциональные
    1. Интерактивная логика
    2. Анимации
    3. Базовые события
    4. Кастомные события
4. Скриншотные
    1. Состояния блока
    2. Вариации данных
    3. Область кликабельности
    4. Скелетон
5. Системные
    1. Отрисовка в приложении
    2. Отображение денег
    3. Движение денег



ТИНЬКОФФ

II. Примеры

- Клик по ссылке
- «Снежинка»
- Логирование ошибок

Клик по ссылке

Система должна переходить на url при клике в ссылку

Система

Добрый день

150 177,45 ₽ — 230 ₽
Tinkoff Black Premium
4436 мир 2289 мир 0488 мир 9076 мир

200 000 ₽
Накопительный счет

300 000 ₽
Потрачено

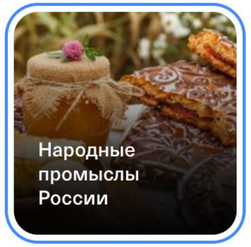
Мобильная связь
Тинькофф Мобайл

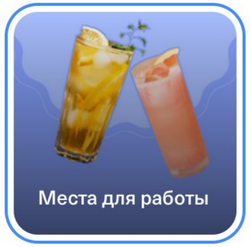
2 полиса
Страхование

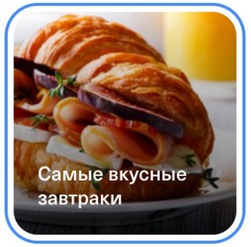
Скрытые продукты

Новый счет или продукт

Поиск







Фильмы недели

Перевести по телефону

Перевести по реквизитам

Оплатить мобильный

Распознать квитанцию

Кэшбэк и бонусы >

Все предложения

Агро-Альянс
Кэшбэк 50%

Delta Sirius
Кэшбэк 10%

Расщепление требования

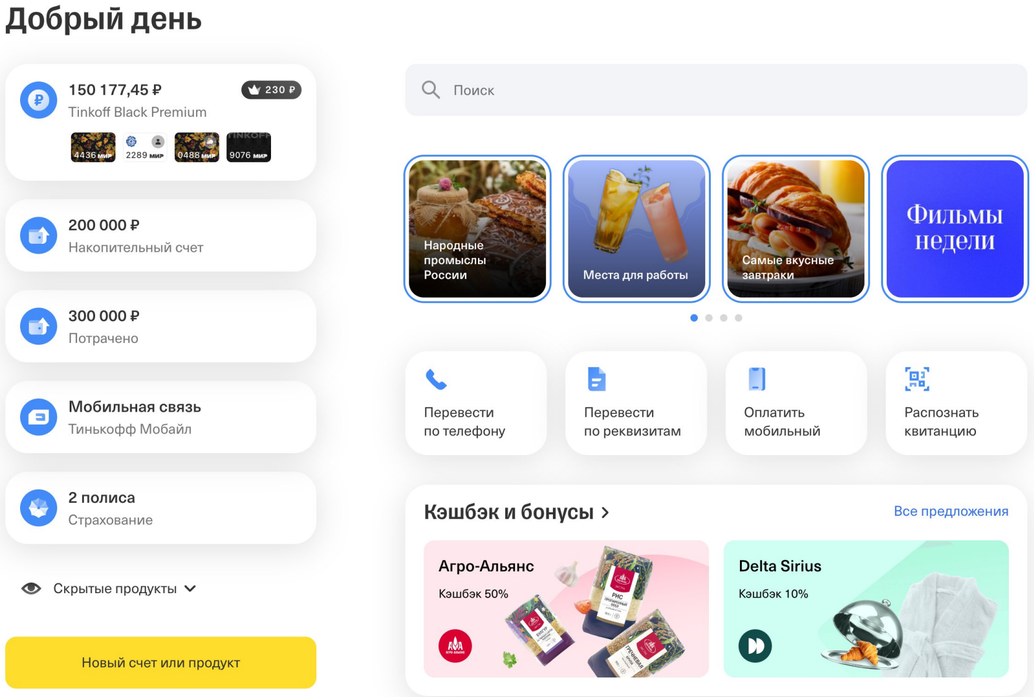
Система должна переходить на url при клике в ссылку

Фронт, при клике, должен приводить к переходу на url, заданному в бэке

Бэк должен отдавать url

- T2 и T3 — производные T1
- T2 и T3 по отдельности проще реализовать и протестировать
- Расщепление стало возможным благодаря архитектуре
- Требования не переносятся, а существуют совместно!

Фронт-бэк



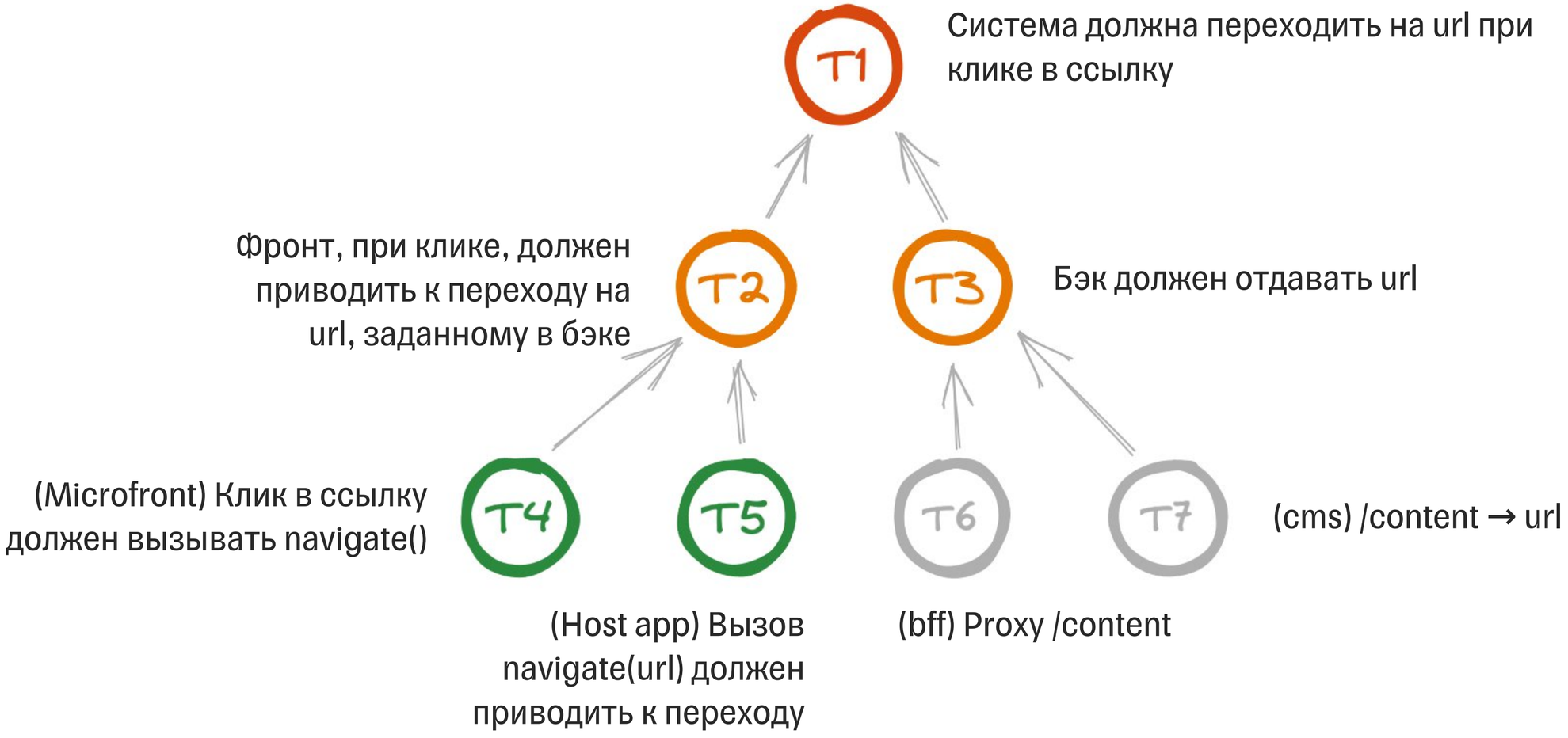
Контент шапки

Заголовок* 15/30
Кэшбэк и бонусы

Подзаголовок 15/15
Все предложения

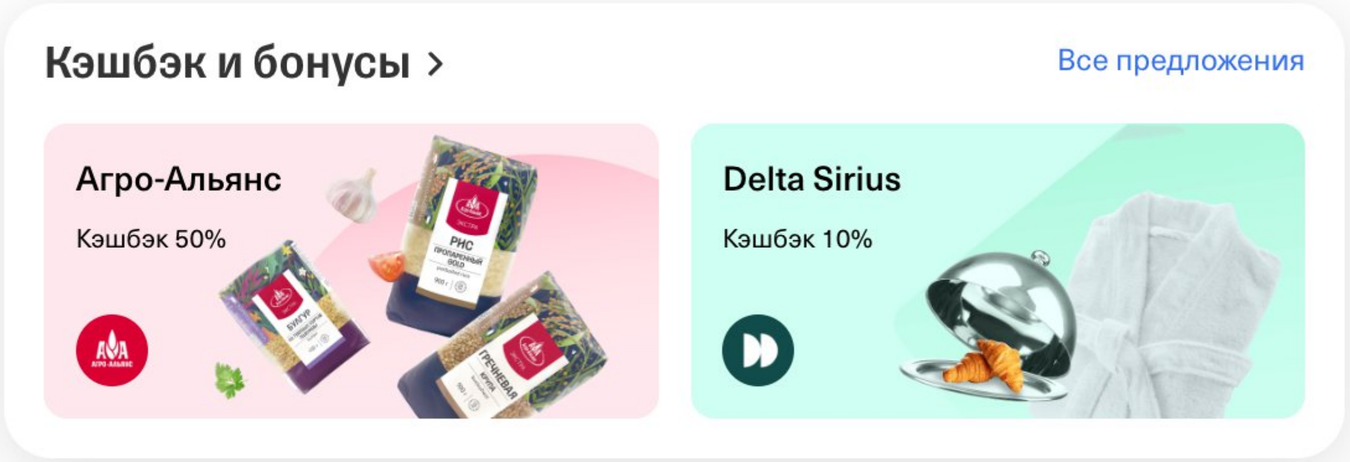
Общая ссылка заголовка и подзаголовка*
/mybank/bonuses/

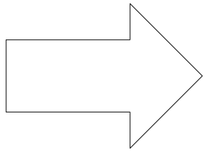
Микрофронт-хост



Микрофронт-хост

```
void router.navigate(url)
```





Host App

Disclaimer! Атомарные требования

В докладе мы говорим об **атомарных требованиях.**

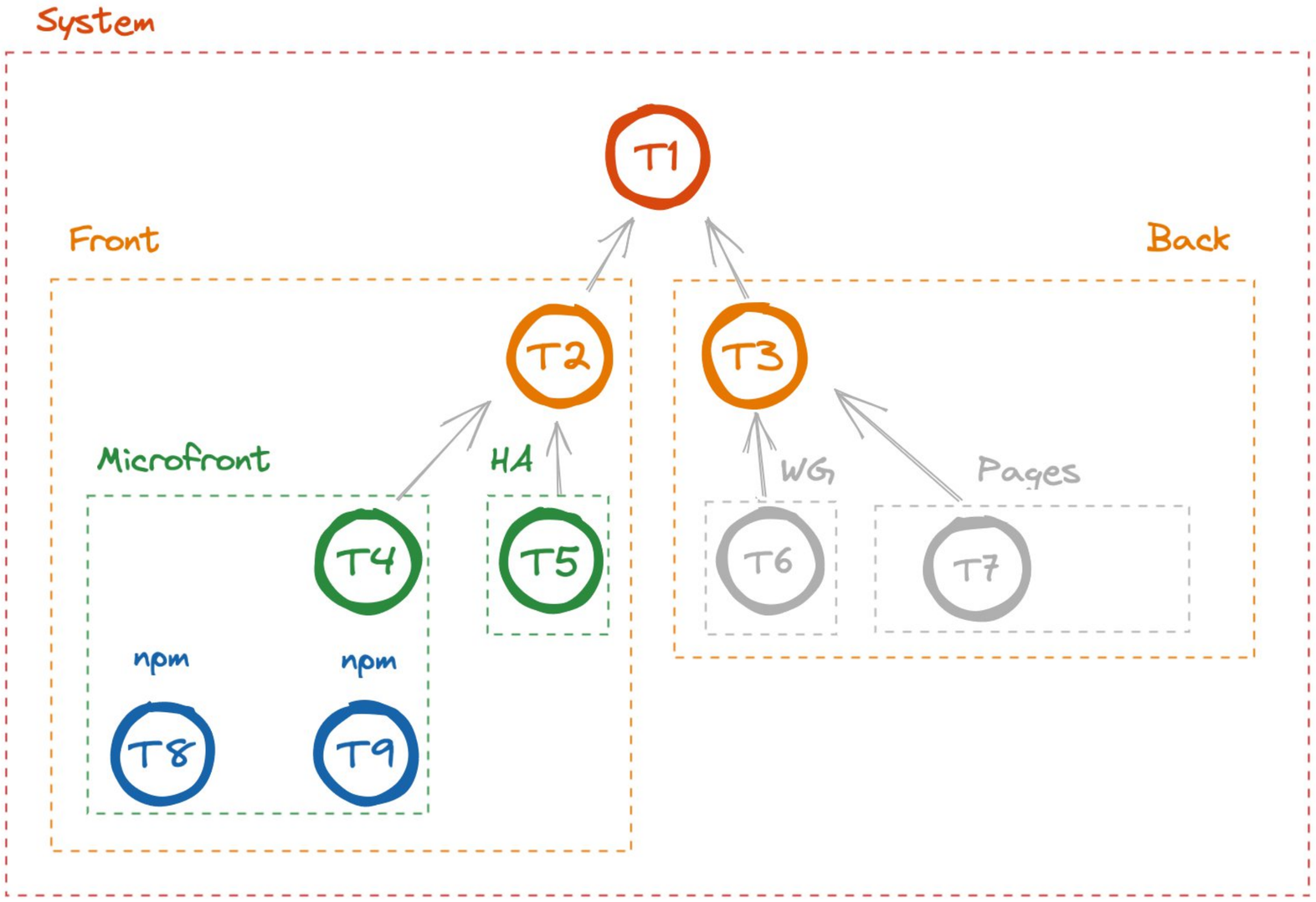
Клик в ссылку … — нет

Клик в ссылку X на странице Y … — да

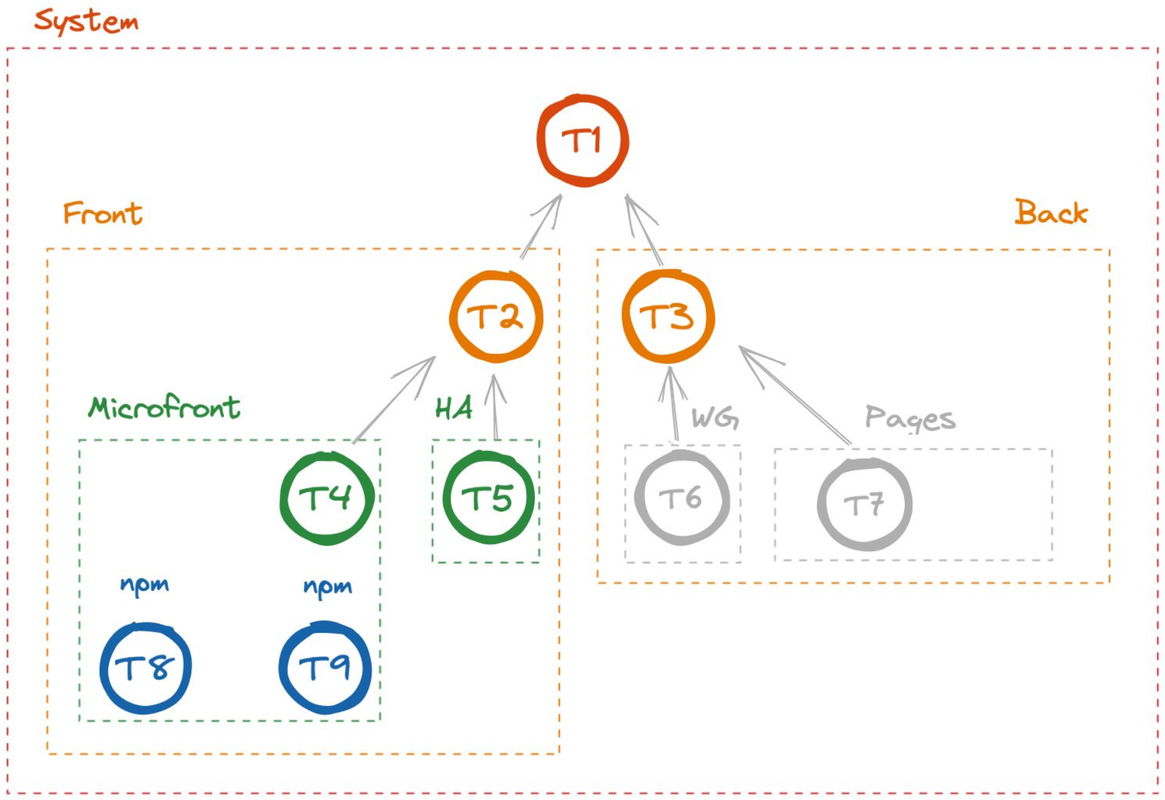
200 микрофронтов по 10 ссылок = 2000 системных требований

До каких пор можем расщеплять?

- Без изоляции нельзя протестировать
- Изоляция возможна благодаря архитектурным границам



Архитектурные границы



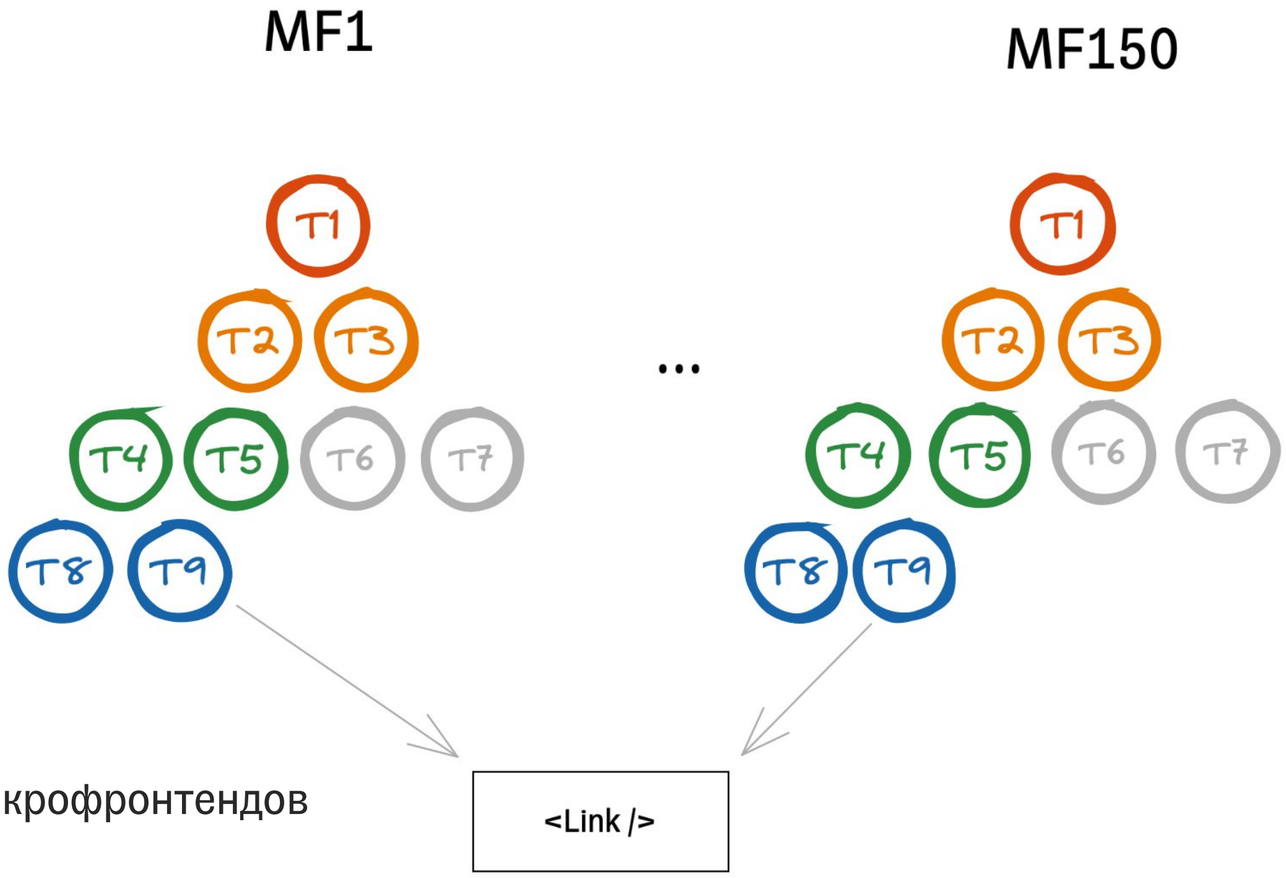
Арх. слой		Граница
S	System	---
CS	Client – server	HTTP + swagger
HAM	Host app – Microfront	Microcontract
N	Npm package	Public interface

Все требования — производные от системных требований к продукту

Вы хотите сказать, что спецификация на CSS Grid — производная от требований к Tinkoff PWA?!

Требования к компоненту Link

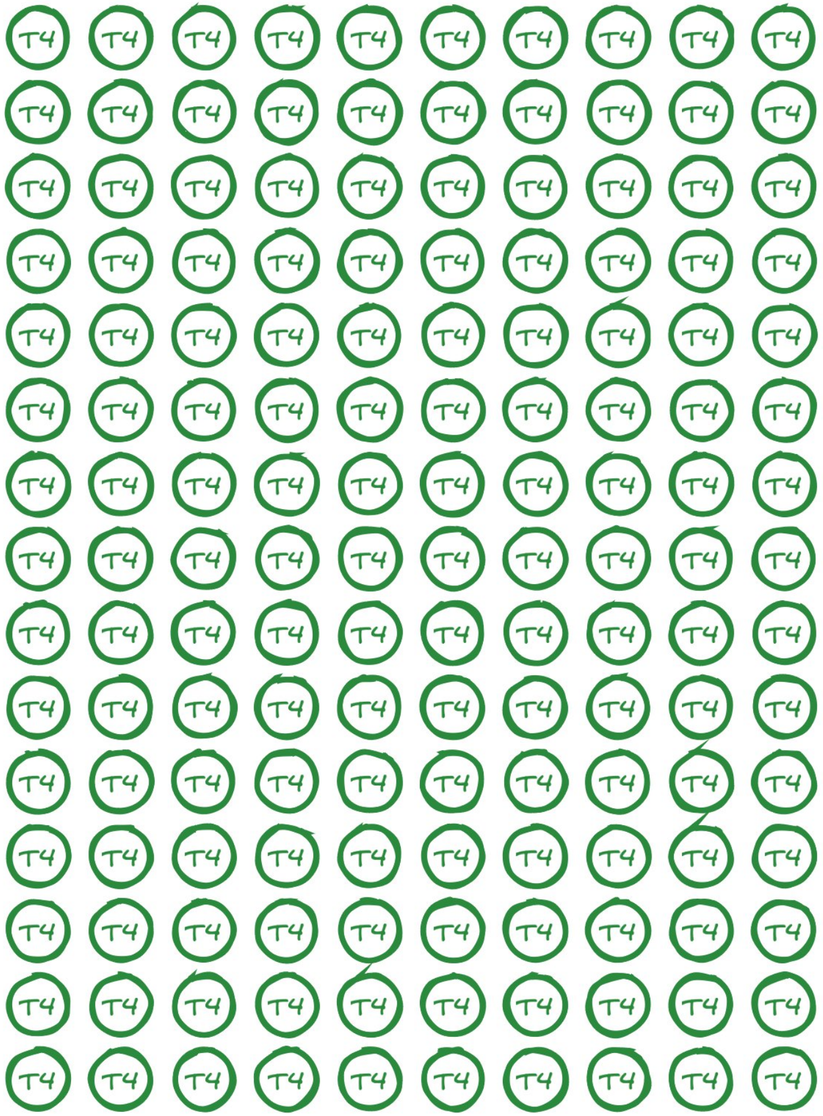
```
const AtomLink = ({ text, callback }) => {
  <span onClick={callback}>
    {text}
  </span>
}
```



- Требование к Link — результат мержа T9 разных микрофронтендов

Спуск позволяет мержить требования и тесты!

Спуск снижает количество тестов!



Определим термины

Требование

Информация о том, как должна работать программа

Требования формулируются на языке человека.

Клик по ссылке должен приводить к переходу на url

Мы рассматриваем атомарные требования.

Код

Информация о том, как должна работать программа

На языке компьютера.

text

Лишь код существует объективно :(

Тесткейс

Инструкция проверки соответствия работы программы требованию

На языке человека.

1. Кликаем по ссылке
2. Ожидаем переход

Автотест

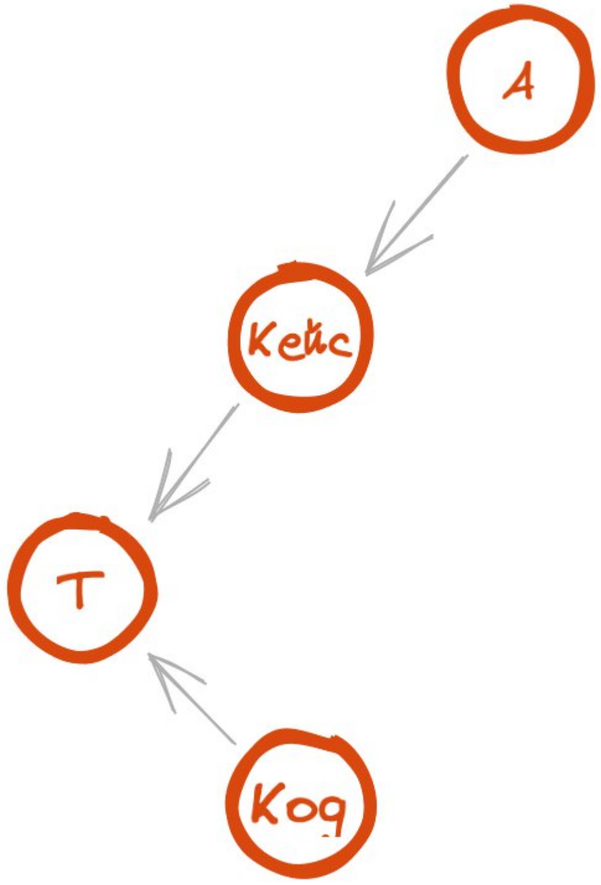
Инструкция проверки соответствия работы программы требованию

На языке компьютера.

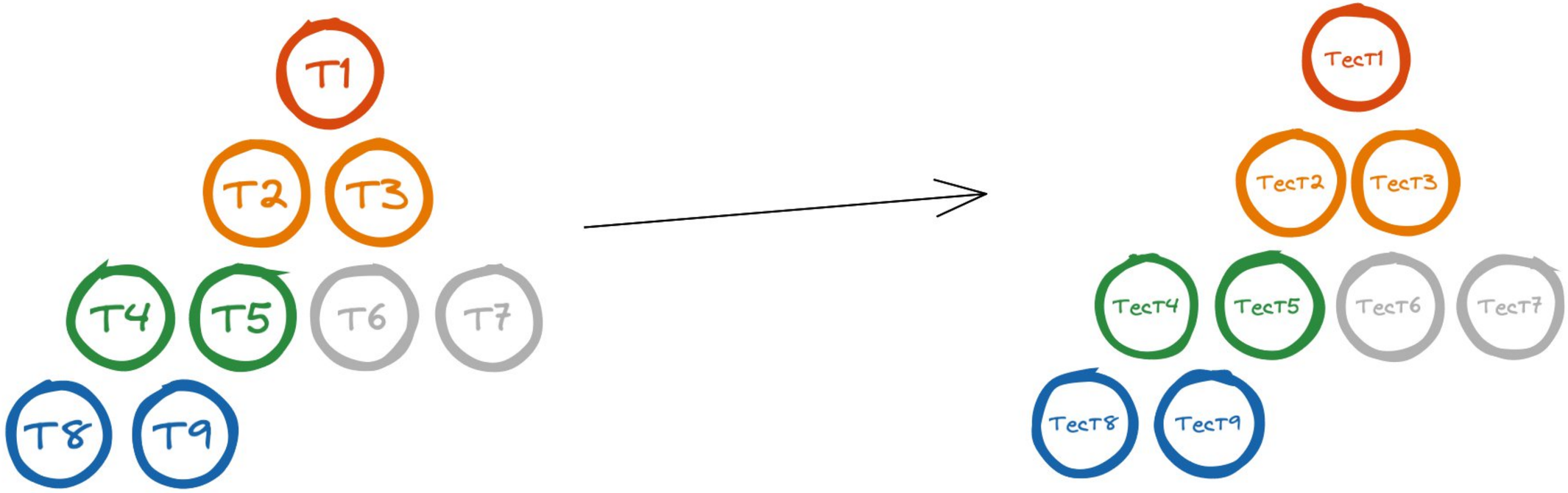
```
await page.click('.link')
expect(page).toHaveUrl(url)
```

Требования — первичны

- Код — производная от требования
- Тесткейс — производная от требования
- Автотест — производная от тесткейса



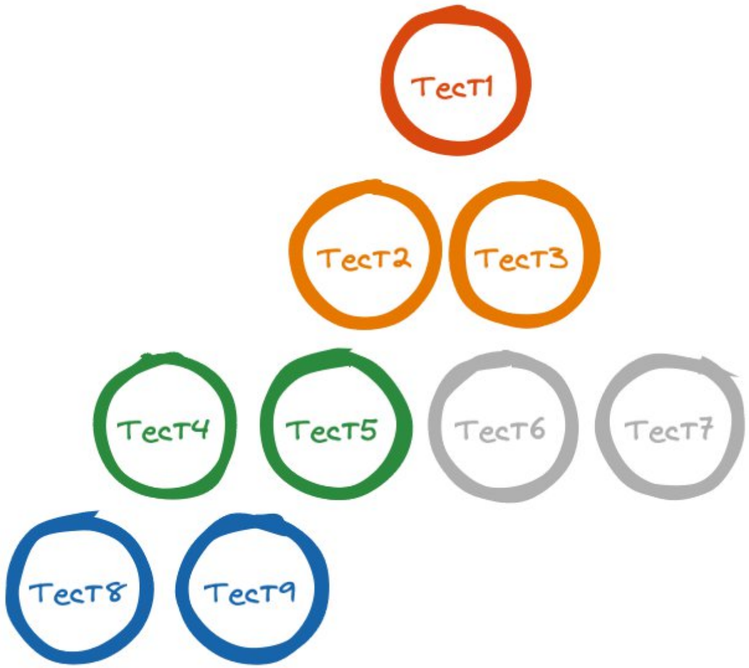
Защитим требования



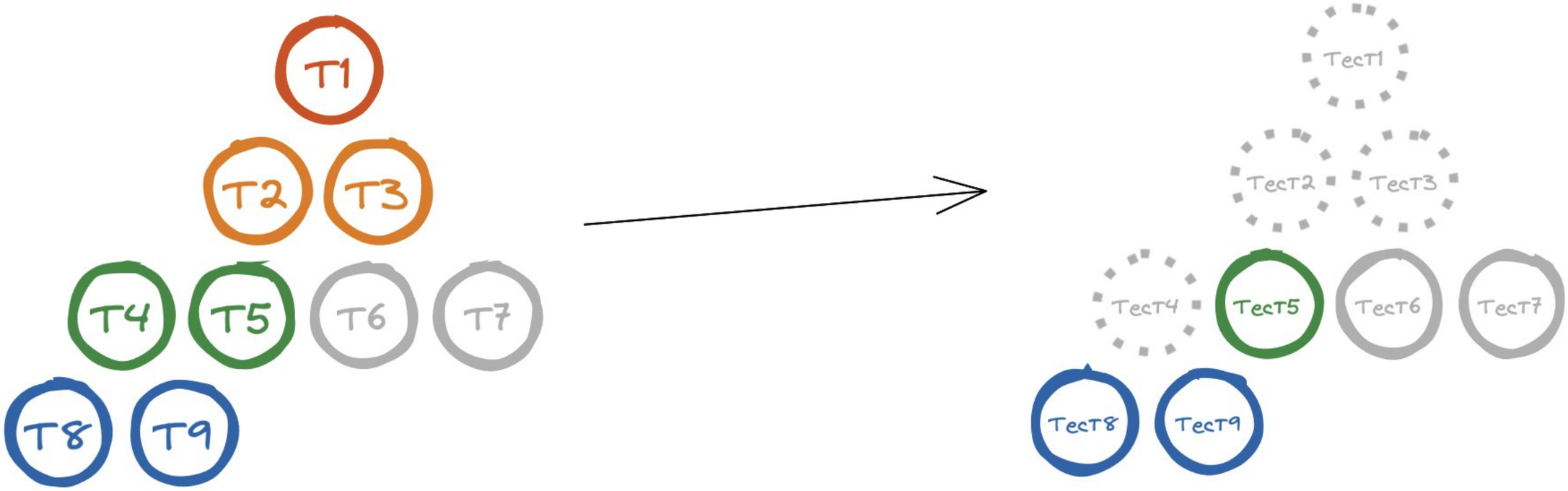
9 тестов на ссылку :)

Какие тесты выбрать?

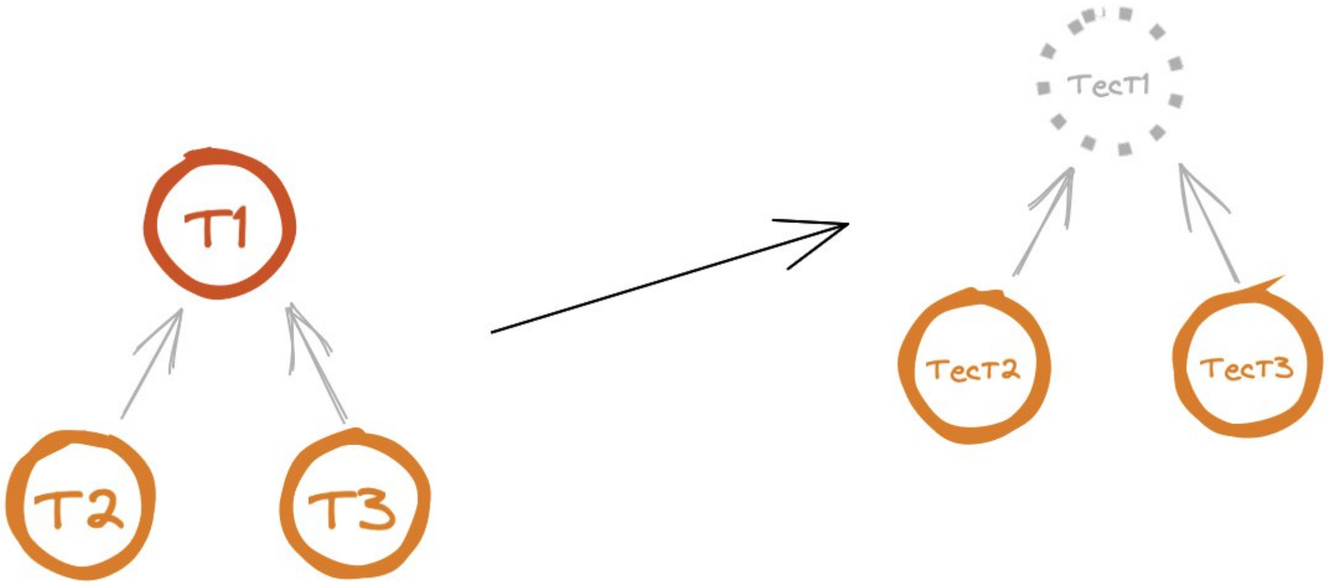
- Время регресса — как часто вы сможете релизиться
- Тестовое покрытие — риски для прода
- Скорость и стабильность — как быстро вы получите обратную связь
- Локализация багов — качество обратной связи



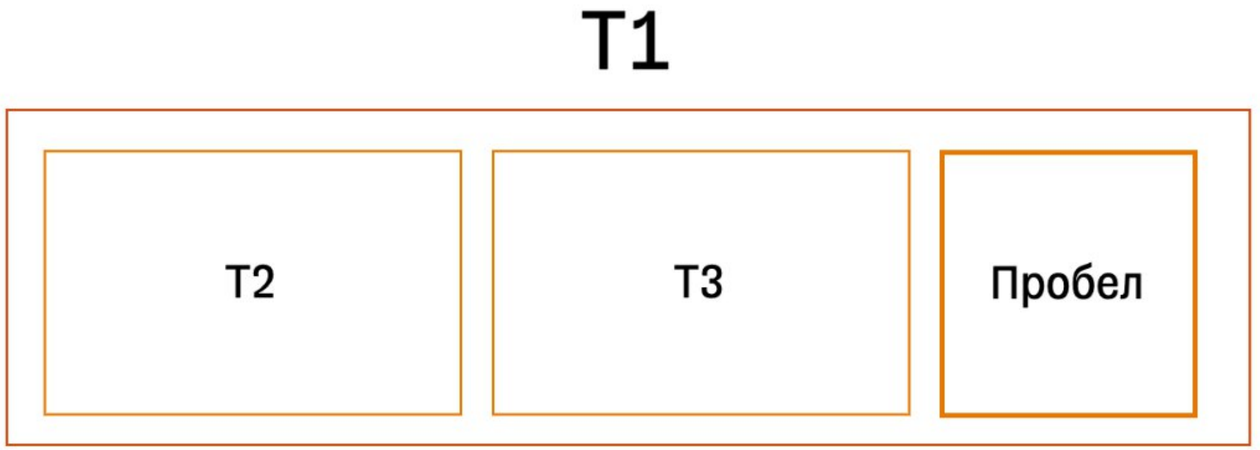
Будут ли пробелы?



Будут ли пробелы?

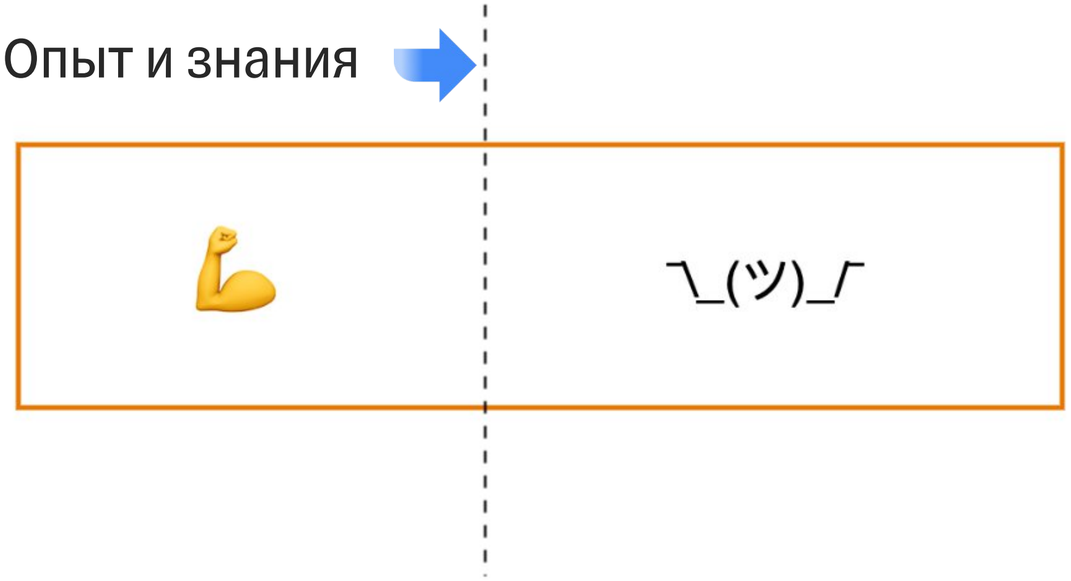


Мы не знаем *



* но, скорее всего, да.

Пробел



1. Интеграция
   - Контракт между фронтом и бэком
   - Сетевые доступы
2. Интерференция требований
   - Поповеры
   - CSP
   - Асинхронность
   - preventDefault() или stopPropagation() стороннего скрипта
   - ...

?

Спуск по «пирамиде» — Источник пробелов

Поиск баланса

Скорость

Локальность

Время регресса

Тестовое покрытие

Домашнее задание

Найдите пробелы в своем покрытии

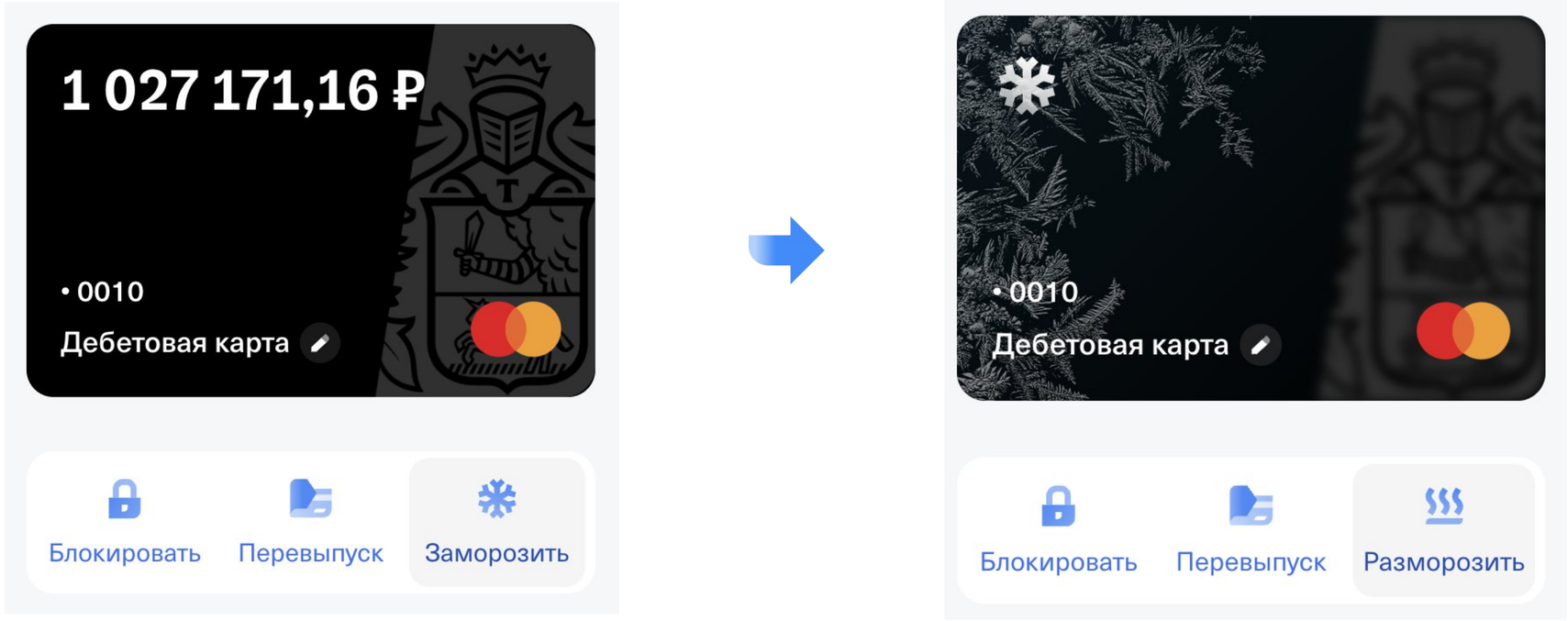
1. Возьмите любой тест, например юнит-тест
2. Выведите требование, которое он покрывает (может быть несколько)
3. Найдите всех предков этого требования, до системного уровня
4. Попробуйте найти тесты на всех предков

Клик по ссылке

«Снежинка»

Логирование ошибок

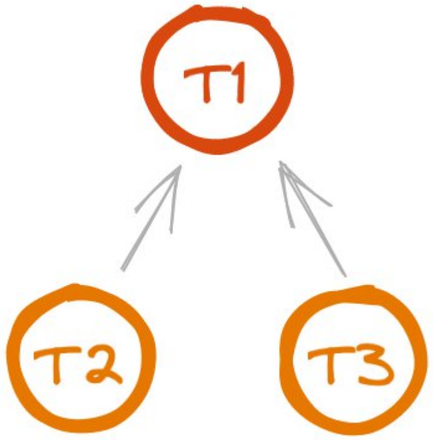
«Снежинка»



Система-фронт-бэк

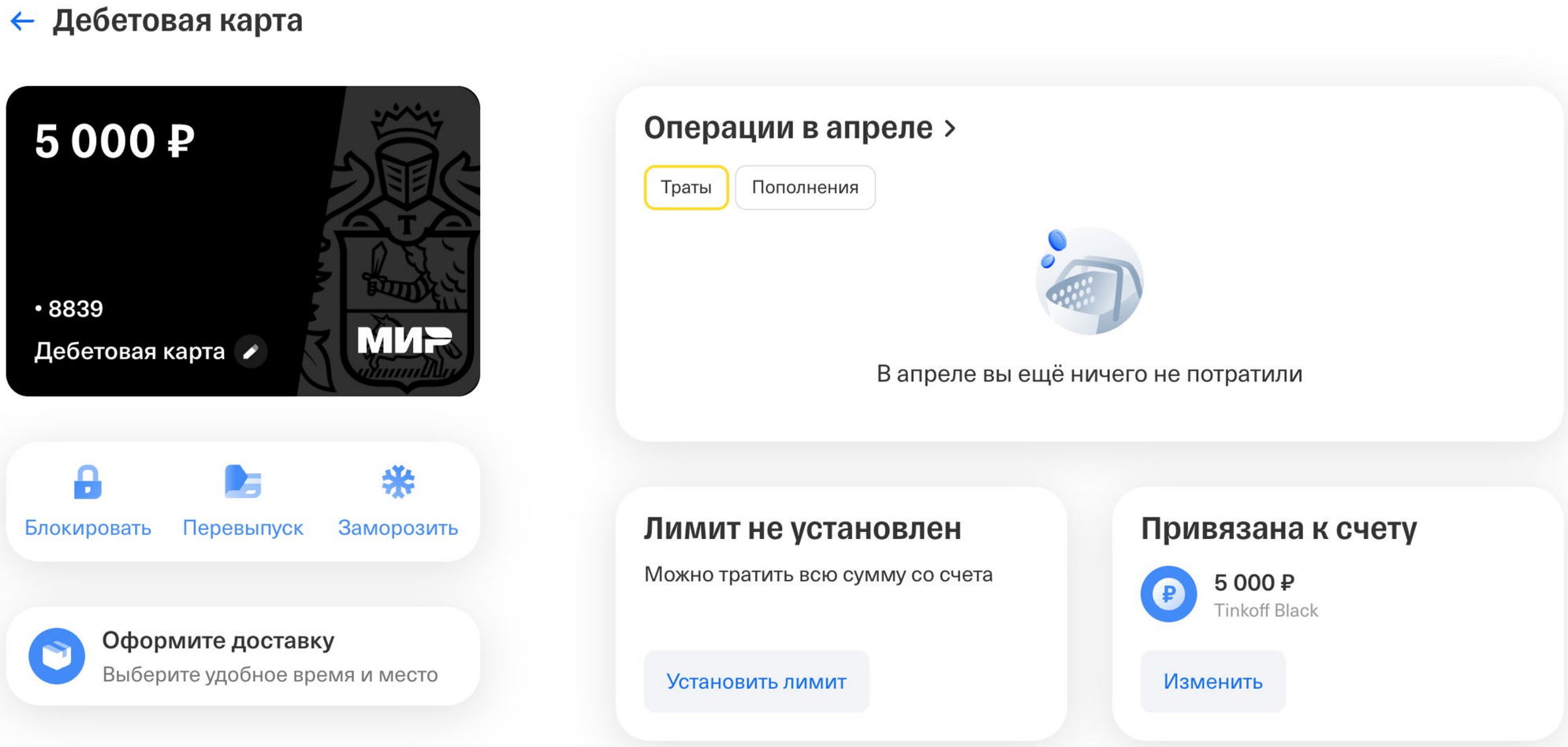
Клик в «снежинку» блокирует карту

Клик в «снежинку» вызывает /freeze и инвалидирует /defails, который обновляет «снежинку»



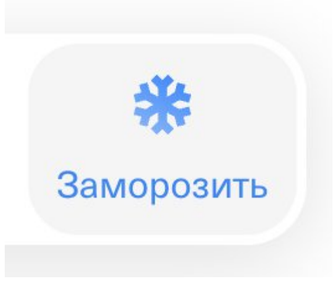
Вызов /freeze блокирует карту

Система-фронт-бэк

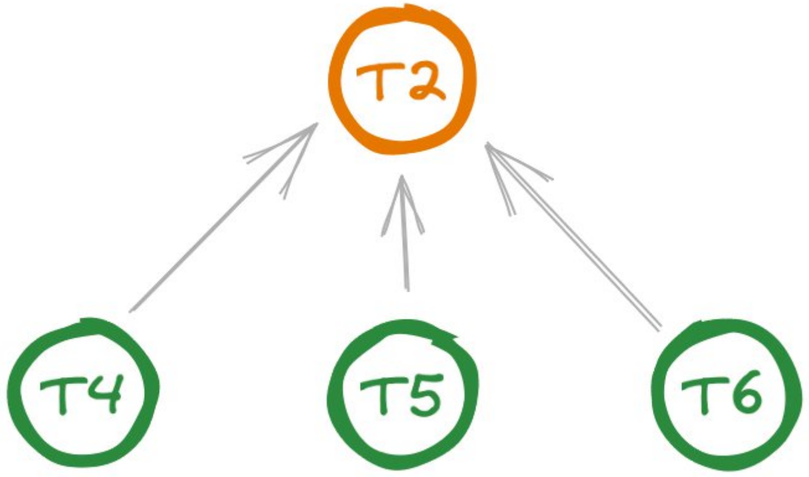


Фронт-микрофронт-хост

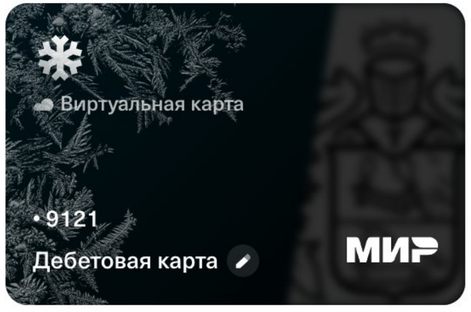
Клик в «снежинку» вызывает /freeze
и инвалидирует /defails, который
обновляет «снежинку»



Микрофронт 1: клик в «Заморозить» должен вызывать freeze api и инвалидировать card details api

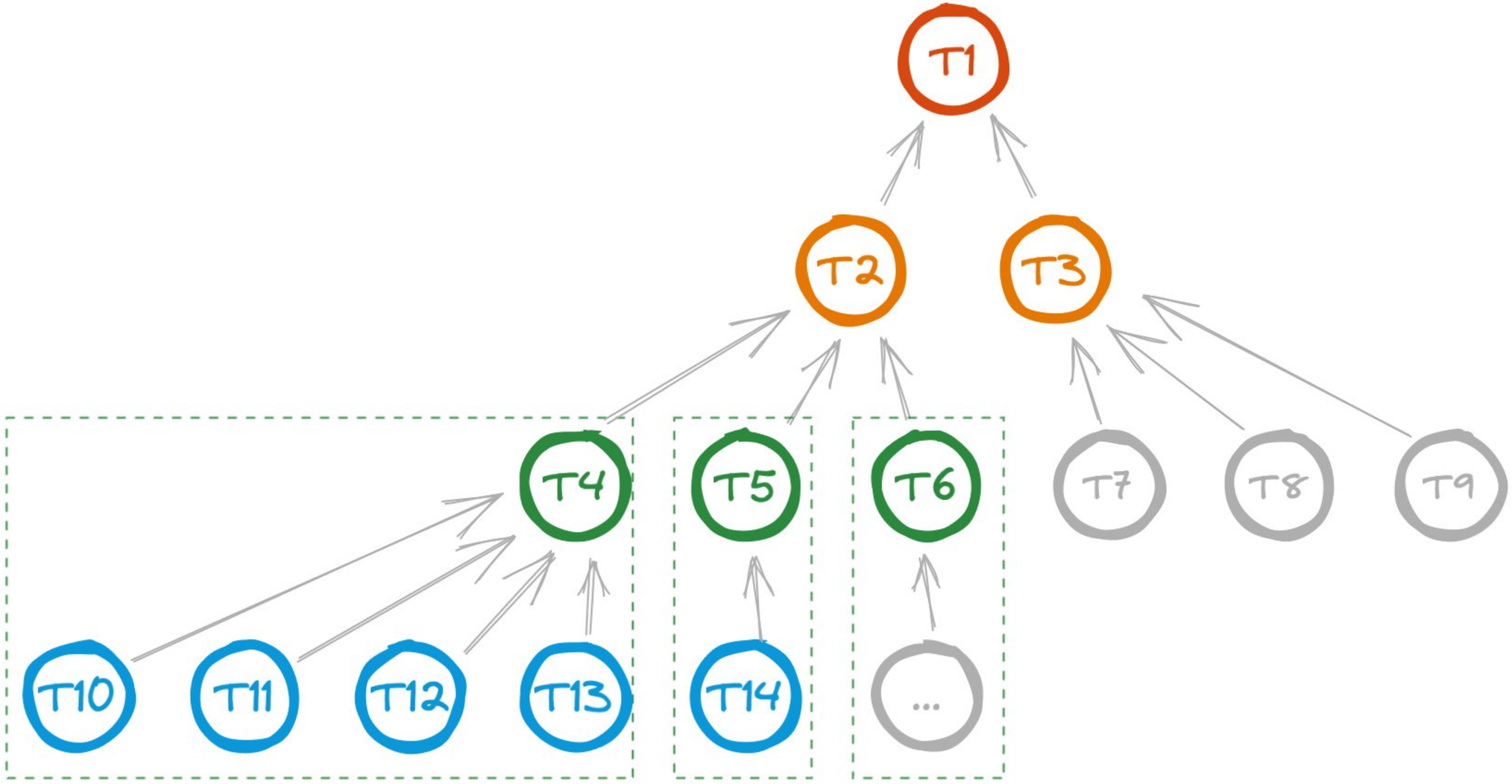


Хост: должен обеспечить связь микрофронтов



Микрофронт 2: должен рисовать снежинку при флаге isFrozen

На что похоже? :)

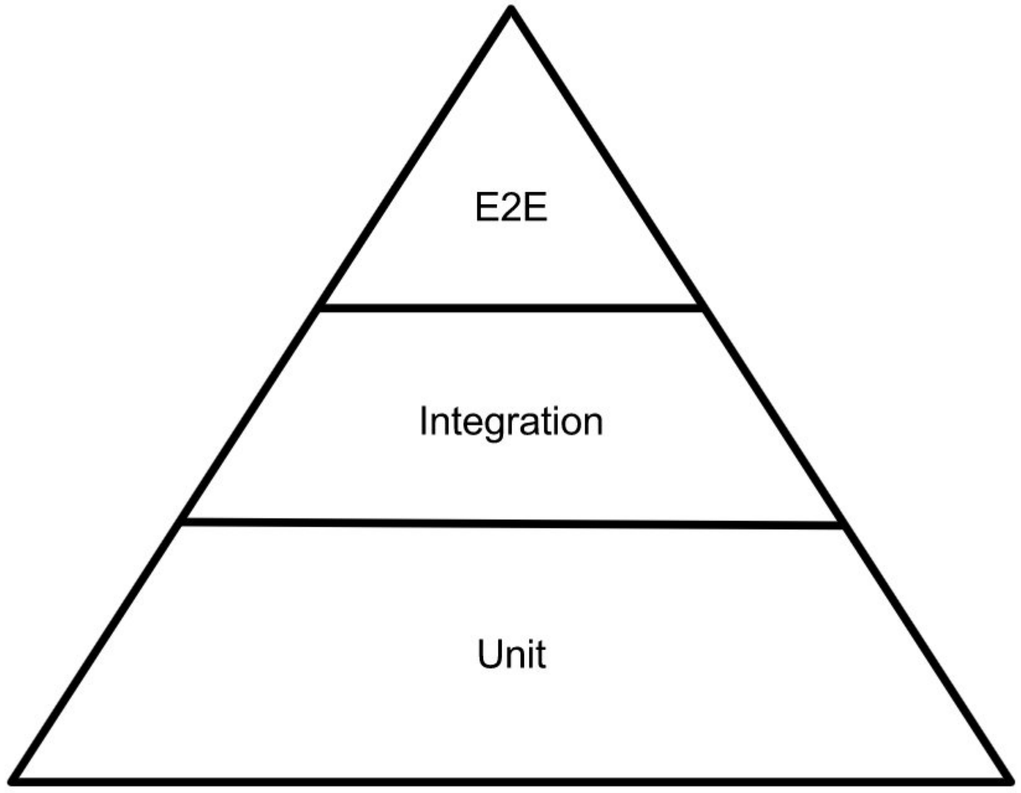


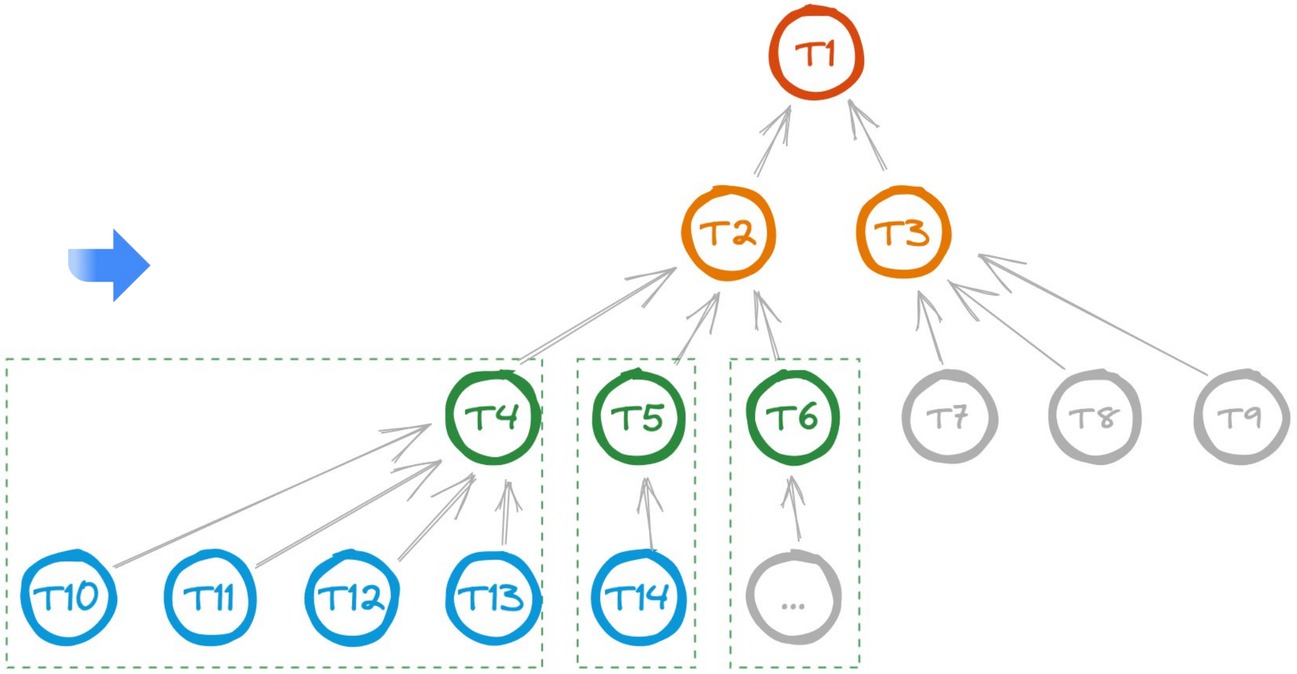
«Пирамида» — производная архитектуры

(Возможна лишь благодаря архитектурным границам)

Как совместить слои с деревом?

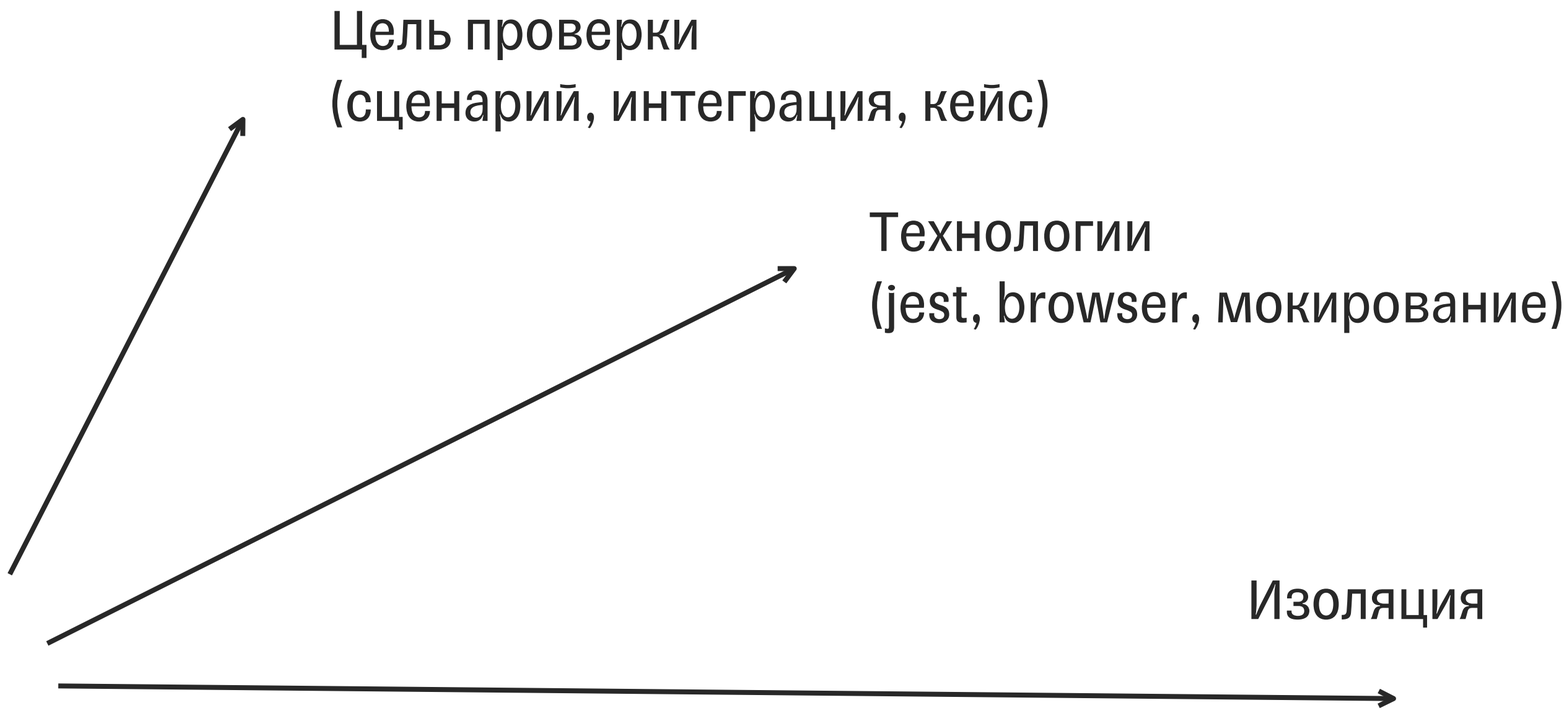
Никак :)





Почти каждый узел может быть в каждом слое

Архитектура первична



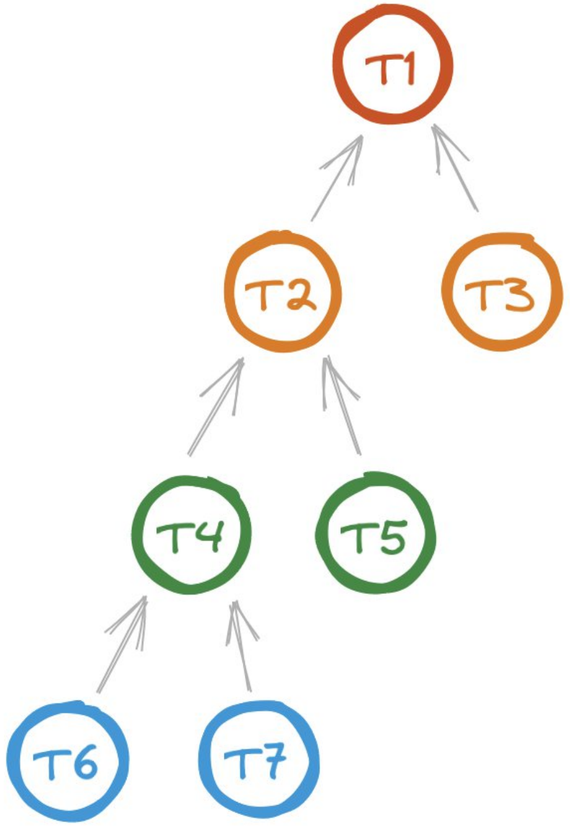
Домашнее задание #2

Установите связь пирамиды с архитектурой

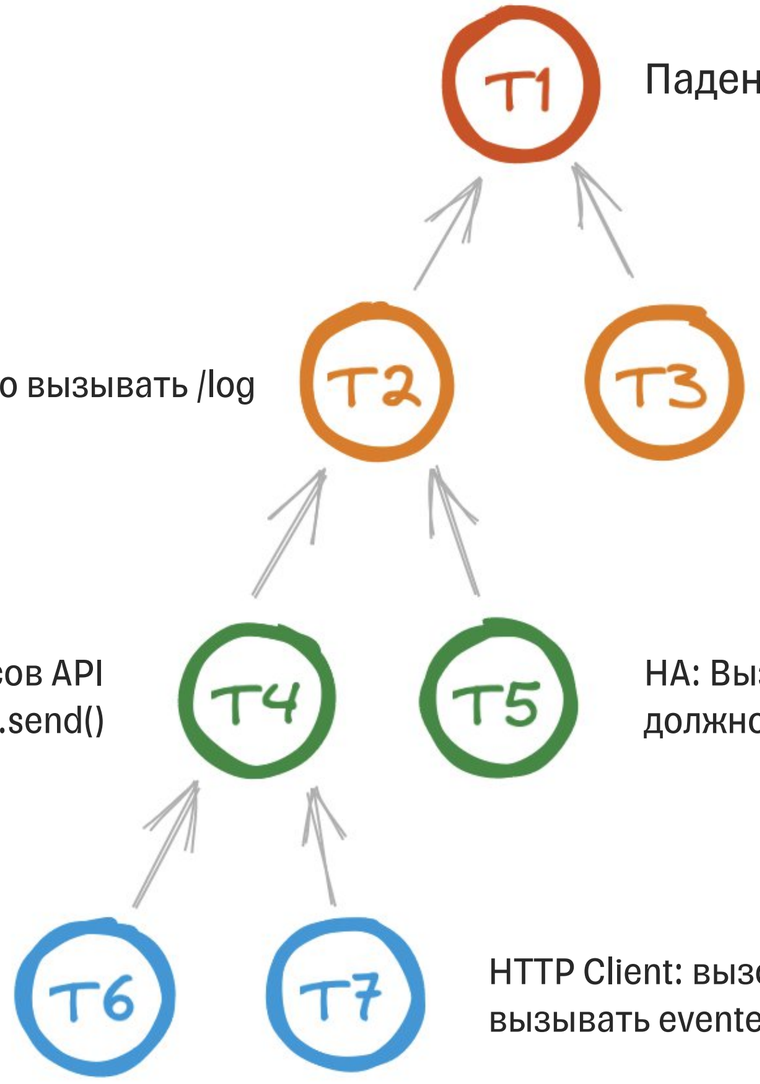
1. Возьмите любое системное требование
2. Разложите его по архитектуре приложения
3. Найдите тесты, связанные с производными требованиями
4. Установите связь видов тестов с узлами архитектуры

- Клик по ссылке
- «Снежинка»
- Логирование ошибок

Логирование ошибок



Дерево



Падение запросов API должно логироваться

Front: Падение запросов API должно вызывать /log

Back: Вызов /log сохраняет лог в систему логирования

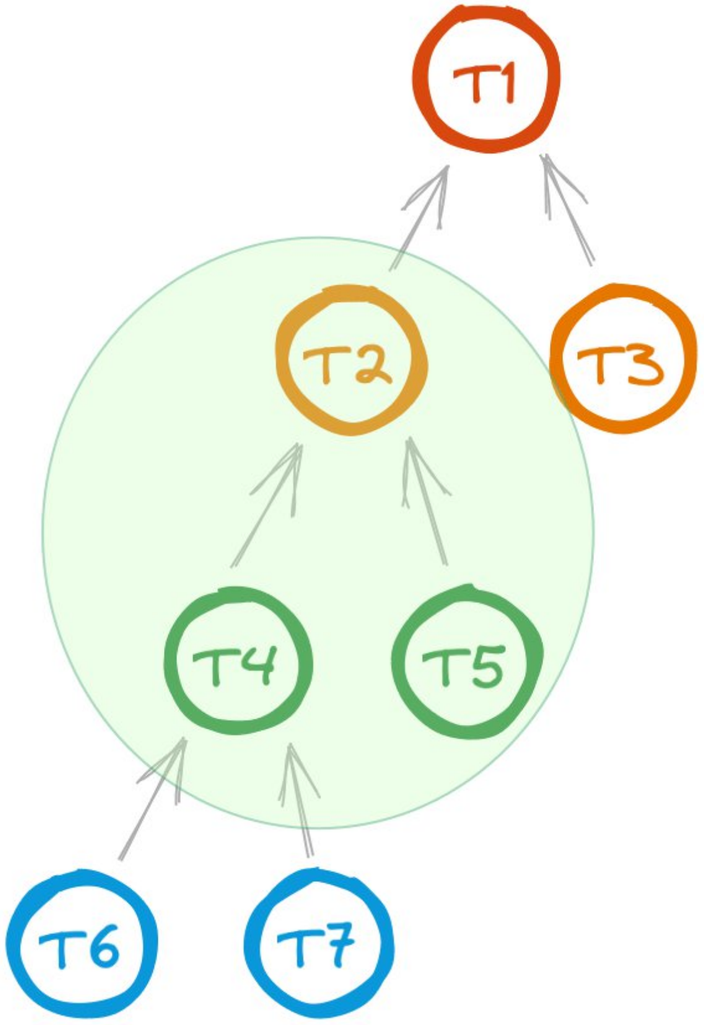
MF: Падение запросов API должно вызывать eventea.send()

HA: Вызов eventea.send() должно вызывать /log

API Client: падение запроса должно вызывать errorCallback в HTTP Client

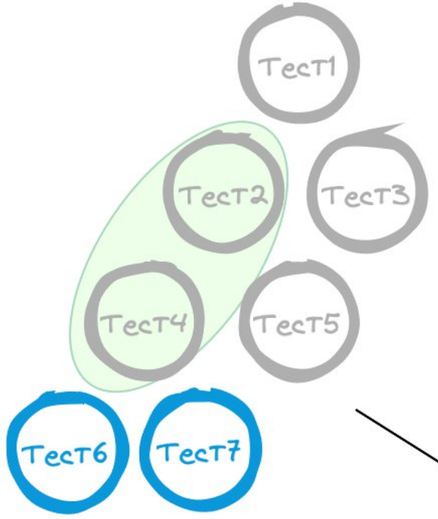
HTTP Client: вызов errorCallback должен вызывать eventea.send()

Области понятности

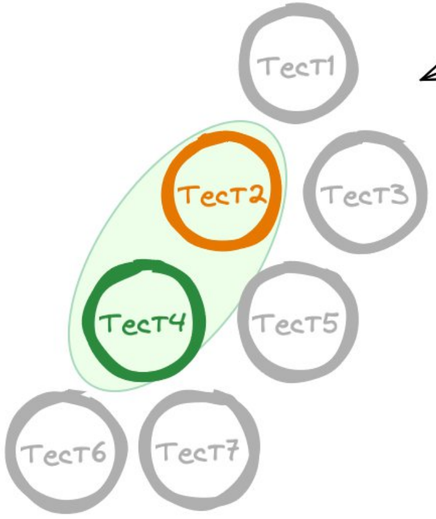


Два выхода из ситуации

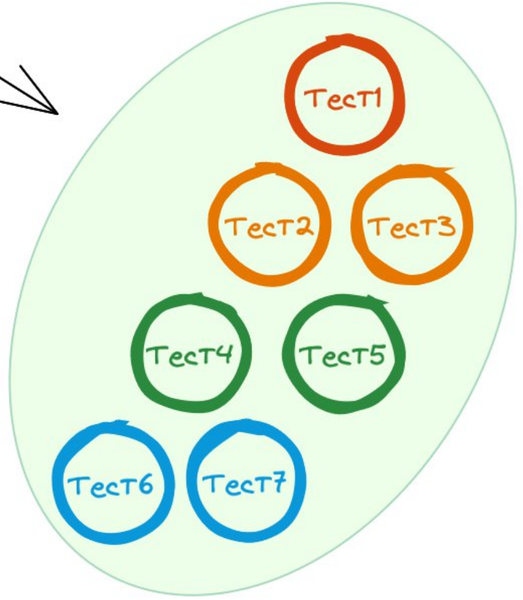
Бесполезные тесты



Скукожиться в область понятности



Разобраться в архитектуре и контрактах



Непонятно = пробел в покрытии

ТИНЬКОФФ

III. Эффективность дерева

Какие узлы дерева нужно покрывать?

- Время жизни проекта и его критичность
- Требования к скорости доставки и частоте релизов
- Архитектура проекта и гранулярность поставки
- Уровень и размер команды

Зачем покрывать новый узел?

- Для снижения затрат на обеспечение качества (при сохранении качества)

Трейдофы на добавление нового узла

1. Ускоряет фидбек от тестов
2. Улучшает проработку требований
3. Ускоряет поиск проблем

1. Больше тестов писать и поддерживать
2. Сложнее анализировать тестовое покрытие
3. Выше порог входа (больше технологий)
4. Больше тестовых обвязок

Эффективность узла

$$\frac{\text{Профит} - \text{Потери}}{\text{Время на качество}}$$

Оценка эффективности узла, пример #1

- Фронтовая команда с опытом автоматизации
- Нестабильный бэкенд
- Еженедельные релизы в течение двух лет ≈ 100 релизов
- Узел: **система**

$$\frac{2 * 100 - 8}{4 * 100} = 50\%$$

Оценка эффективности узла, пример #1

- Фронтовая команда с опытом автоматизации
- Нестабильный бэкенд
- Еженедельные релизы в течение двух лет ≈ 100 релизов
- Узел: **фронт**

$$\frac{3 * 100 - 50}{4 * 100} \approx 63\%$$

Эффективность узлов — разная

Оценка эффективности узла, пример #2

- Фронтовая команда без опыта автоматизации
- Стабильный бэкенд
- Ежемесячные релизы в течение двух лет ≈ 24 релиза
- Узел: **система**

$$\frac{3 * 24}{4 * 24} \quad \frac{\quad}{\quad} \quad 60 \qquad 12\%$$

Оценка эффективности узла, пример #2

- Фронтовая команда без опыта автоматизации
- Стабильный бэкенд
- Ежемесячные релизы в течение двух лет ≈ 24 релиза
- Узел: **фронт**

$$\frac{3.1 * 24}{200} - \frac{150}{} = -38\%$$

Эффективность узлов может быть отрицательной

Когда делать больше видов тестов?

1. Мотивированная команда
2. Долгий срок жизни проекта
3. Высокая частота релизов
4. Высокая критичность проекта
5. Большой масштаб проекта

Каждой команде — своя «пирамида»

Домашнее задание #3

Оцените эффективность видов тестов

1. Какие тесты приносят больше пользы?
2. Какие тесты малополезны и почему?
3. Есть ли тесты, которые вас замедляют?
4. Какие компетенции надо прокачать в команде, чтобы все тесты приносили много пользы?

Выводы

1. Чтобы найти пробелы — расщепи требования по архитектуре
2. Пирамида = дерево, определяемое архитектурой
3. Спуск по пирамиде и зона непонятности — источники пробелов
4. Узлы дерева (слои «пирамиды», виды тестов) имеют разную эффективность

ТИНЬКОФФ

Спасибо!