Хватит писать тесты

Пора писать спецификации

Зачем вообще тесты?

Быстро добавлять фичи

Чистый и стабильный код

Рефакторинг

Уверенность

Тесты

Зачем писать тест до кода?

- Тестирует ли тест что должен
- Падает ли он вообще?
- Нужно “ломать” код, чтобы проверить тест
- Инструмент для декомпозиции
- Архитектурный инструмент

TDD

Test Driven ~~Development~~ Design

TDD подход поможет

- Декомпозировать задачу
- Определить структуру кода
- Понять границы и ответственности компонентов

Red-Green-Refactor

RGR цикл это суть и смысл TDD

- Пишем красный падающий тест
- Делаем его зеленым
- Рефакторим

Walking skeleton

Минимум кода, чтобы запустить TDD цикл.

Все тестовые скрипты и обвязки!

Настроенный pipeline.

```
P(tests) = 1 / manual-steps
```

Тесты и спецификации

```java
@Test
void worksForExpectedUser() throws Exception {
    this.mockDatabase();
    this.mockSession("test-user");
    this.stubExternalDependency();
    this.copyTestFiles("/var/tmp/test-path");
    when(userDetails.getDetails("test-user"))
        .thenReturn(new User("test-user", "", Collections.emptyList()));

    this.mockMvc.perform(post("/api/endpoint")
        .with(user("test-user"))
        .content(getTestRequestFor("test-user")));

    verify(userDetails).getDetails("test-user");
    this.verifyDatabaseHasRecord("test-user");
    this.verifyExternalApiHasBeenCalled();

    this.mockMvc.perform(post("/api/endpoint")
        .with(user("test-user"))
        .content(getTestRequestFor("another-user")))
        .andExpect(status().isBadRequest());

    verifyNoInteractions(userDetails);
    this.verifyDatabaseHasNoRecord("test-user");
    this.verifyExternalApiHasBeenCalled();
}
```

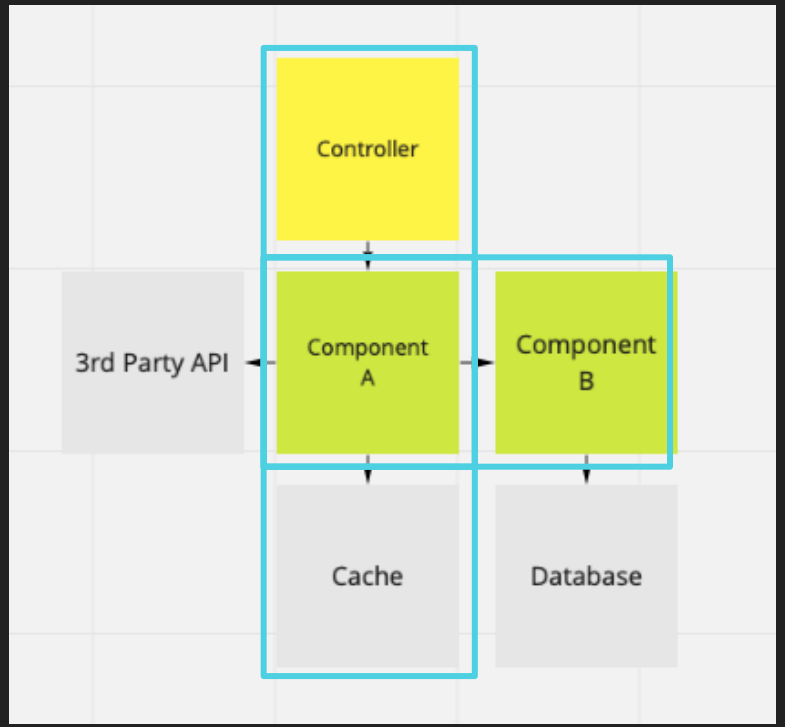
- Непонятно что тестирует
- Слишком много assert-ов
- Падает по целому ряду причин
- Слишком сложный сетап
- Непонятно поведение external api
- Тестирует несколько code path

Хороший тест != хорошая спецификация

Тесты → Спецификации

- Писать спецификации до кода это отличная идея
- Понятно что и где падает
- Мотивирует писать минимум кода
- Заставляет декомпозировать задачи и код
- Добавляет границы между компонентами

Юниты и интеграции



- “Юнит тест” может значит разное . Обычно логический изолированный блок.
- “Интеграция” может значит инфраструктуру
- ...но оставаться внутри “юнита”
- “Интеграция” может значит “другие компоненты”
- ...но при этом не требовать инфраструктуры

В рамках воркшопа

- Интеграционный - требует Spring
- Юнит - не требует Spring

Acceptance или E2E

- Пишутся от лица пользователя
- Не используют знания о внутренностях системы
- В основном happy paths
- Чаще медленные и нестабильные (увы!)

Быстрая проверка “правильный ли у меня E2E тест?” - можно запустить E2E suite как смок тест при деплойменте?

BDD живут здесь и только здесь!

Моки и двойники

Есть много способов изолировать компоненты

- Моки
- Стабы
- Двойники

Первое правило мокистов: не мокайте то, что вам не принадлежит!

Как структурировать тесты?

TDD это не про юнит тесты!

C is for Confidence

...а не Coverage.

“TDD нужно для лучшего покрытия” 😱

Если CI зеленый можно сразу поставить приложение в продакшен?

Покрытие 100%

Какая разница?

Let’s code