
# Spring Boot Ripper
# TDD Edition

# Кто делал

Вы кто такие и зачем пришли, представьтесь :)

JUG.RU

@tolkv

@lavcraft

alfalab ЦИАН

# Дополнительные материалы

1. Spring Boot Test для тестировщиков — презентация
2. Доклад с конференции Heisenbug — Curse of Spring Boot Test

# Что будет

- Постараемся разобрать как работает методология и технологии
- В двух частях
  - Часть первая – меньше про Spring, больше про тестирование
  - Часть вторая – больше про Spring *


Что такое TDD?

# Что такое TDD?

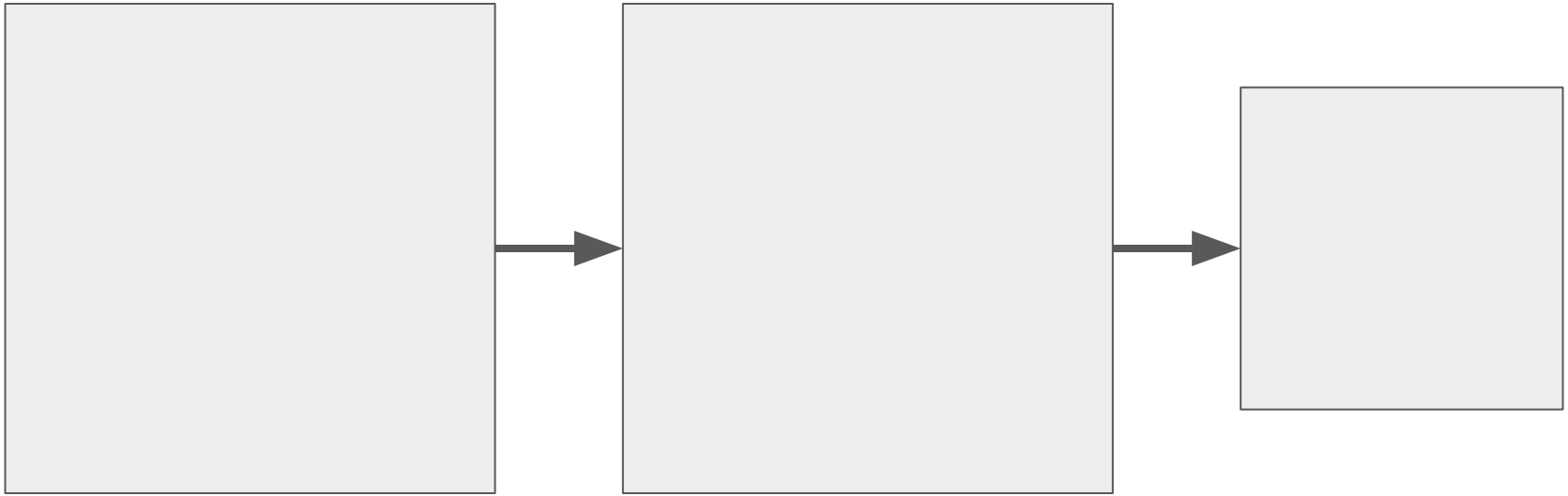
Простым языком*

Тесты **до** Кода

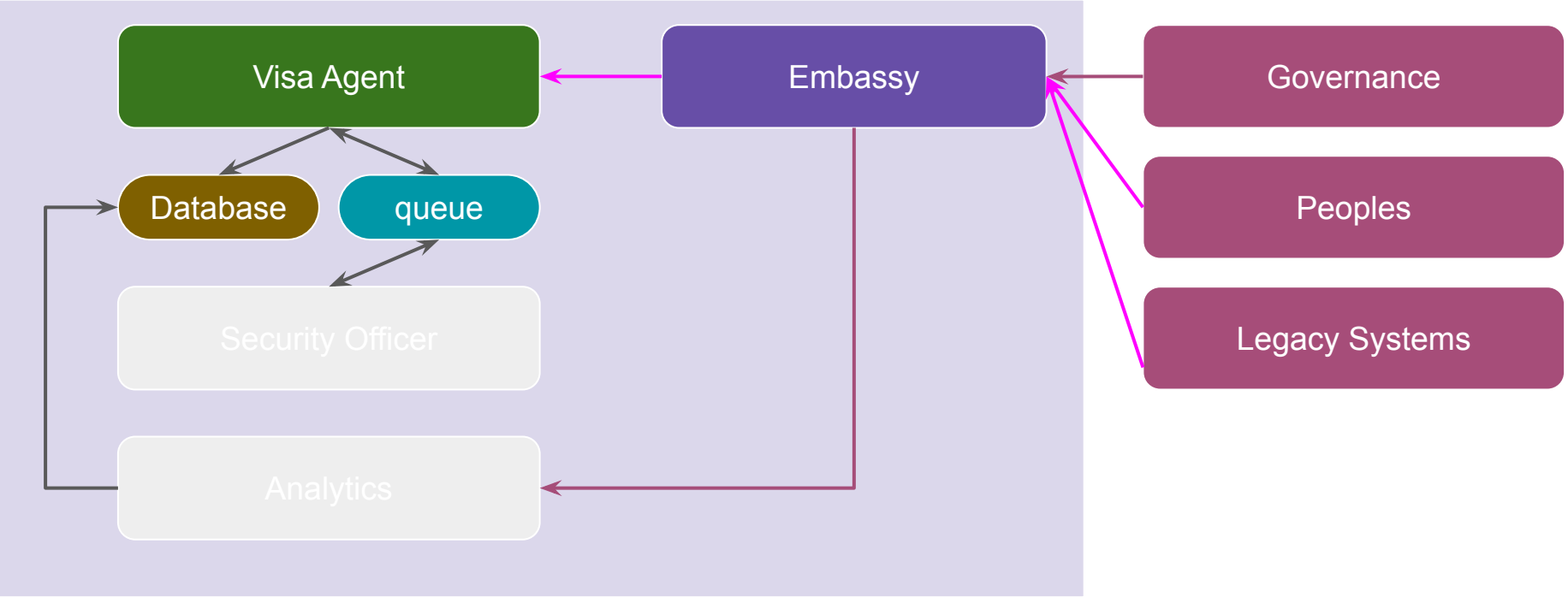
# Что есть

1. Уже договорились об архитектуре

# Архитектура того что ~~мучаем~~ тестируем



# Архитектура того что ~~мучаем~~ тестируем



# Что есть

1. Уже договорились об архитектуре
2. 3 независимые команды которые её будут реализовывать
   a. Команда Посольства
   b. Команда Выдачи Виз
   c. Команда Анализа и Безопасности

Embassy

Visa Agent

Security Officer

Analytics

# Как работаем

- Манифестирует что делаем

# Как работаем

- Манифестирует что делаем
- Справляемся с трудностями

# Как работаем

- Манифестирует что делаем
- Справляемся с трудностями
- Вносим правки

# Как работаем

- Манифестирует что делаем
- Справляемся с трудностями
- Вносим правки
- Справляемся с трудностями
- Справляемся с трудностями
- Справляемся с трудностями

# Часть 1 – Визовый центр

База – Rest Spec, Unit, Component Tests

# Манифестируем задачу команды Посольства

T3 Embassy Service

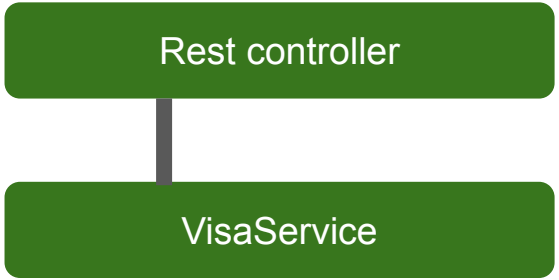
1. Принимает запрос с ID пользователя
2. Отвечает 200 +  номер заявки
3. Пересылает все запросы по визам Visa Agent

# Что тестируем

Rest controller

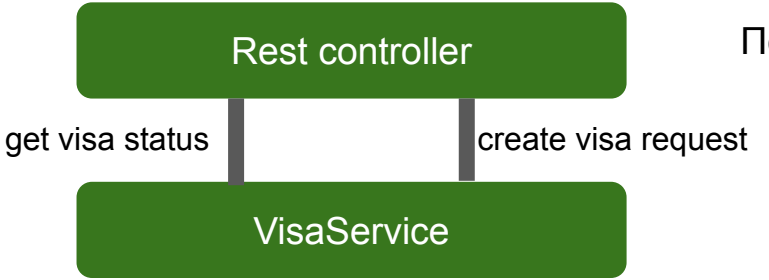
Первый тест на приём

# Что тестируем




Первый тест на приём (пара тестов)

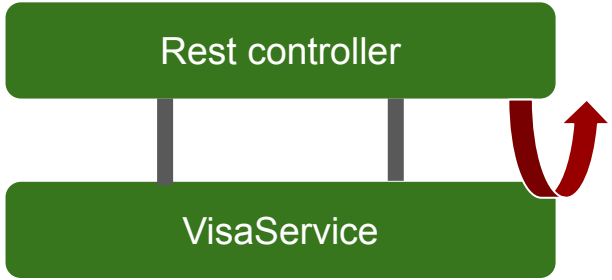
# Что тестируем




Первый тест на приём (пара тестов)

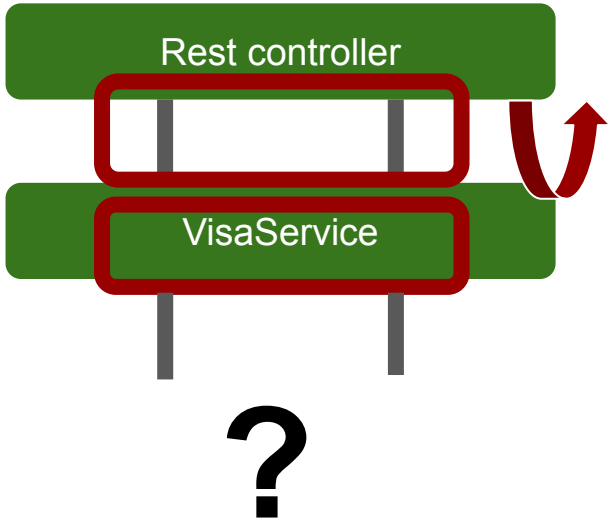
# Business logic time

# Что тестируем



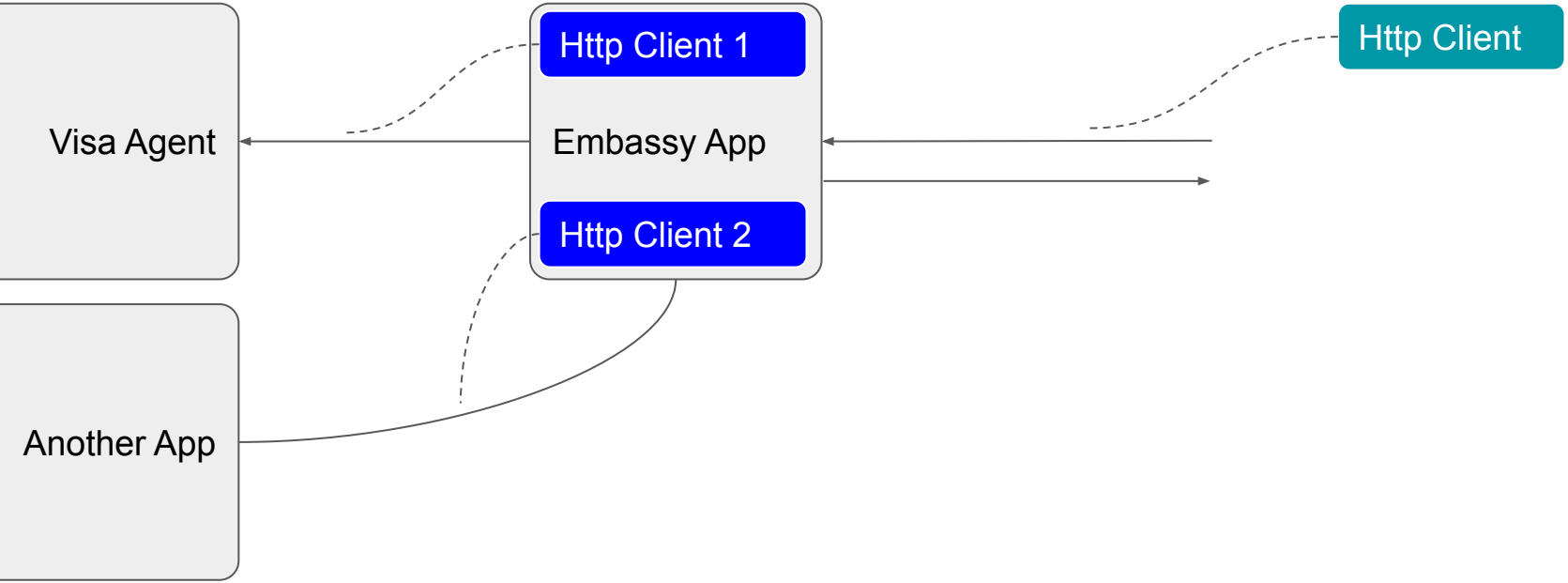

Первый тест на приём (пара тестов)

# Что тестируем

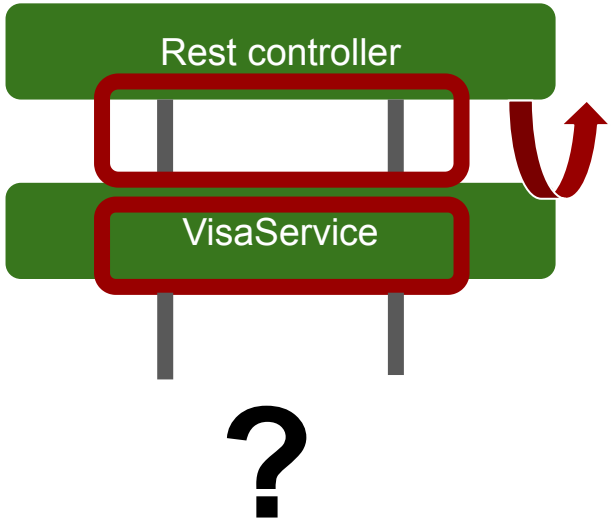



Первый тест на приём (пара тестов)

# А что делает VisaService



# Что тестируем




Первый тест на приём (пара тестов)

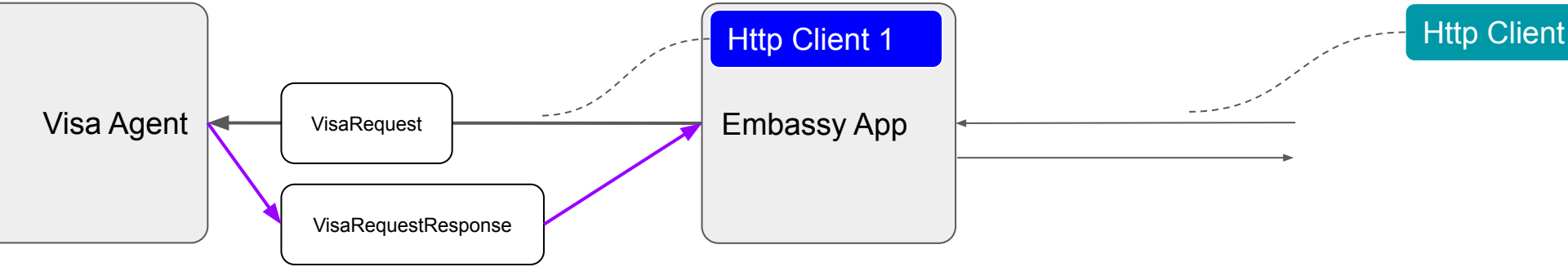
Делаем мок → Пишем второй тест

# Цель тестирования?

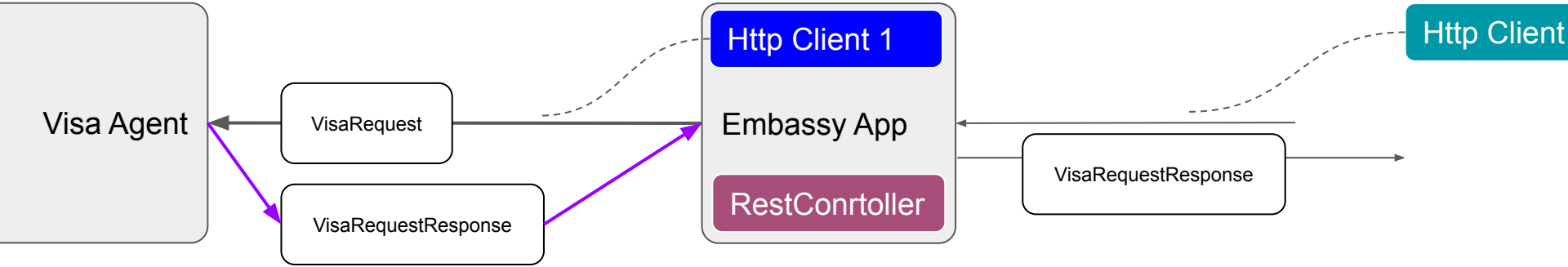

wasted
14:48
$00700911
CV
OG


tested
14:48
$00700911
CV
OG

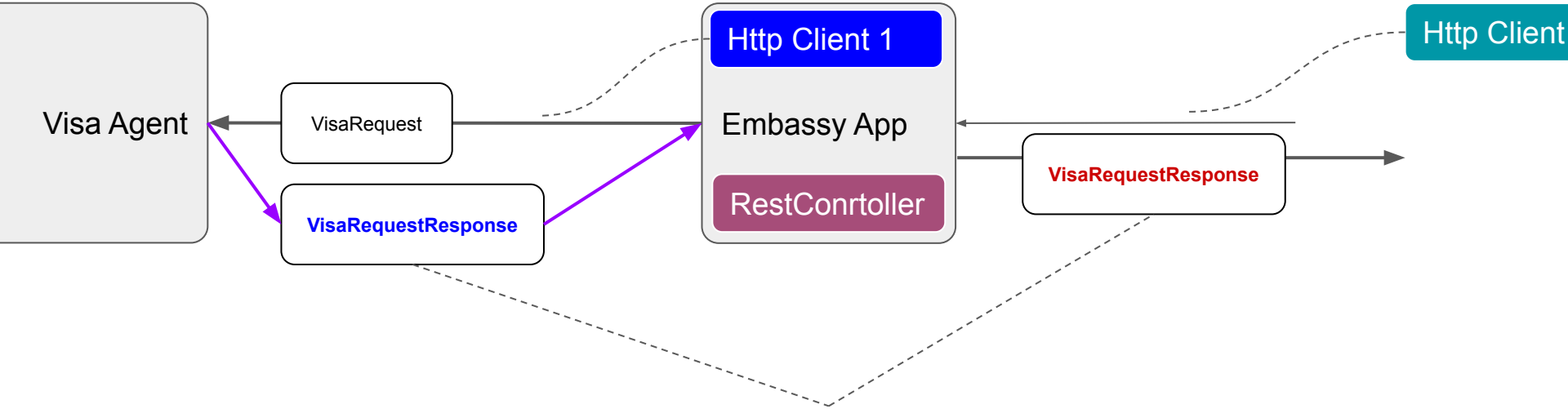
# А что делает VisaService



# А что делает VisaService



# А что делает VisaService




Могут быть разными – DAO

# Что тестируем

# TDD for Business Logic

Тестируем вызов Http Client 1

# TDD for Business Logic Puzzler

Тестируем вызов Http Client 1

```
when(this.httpClientMock.getForEntity(any(), any()))
        .thenReturn(ResponseEntity.ok(VisaRequestResponse.builder().build()));
```

Where is problem Bro?

# TDD for Business Logic Puzzler

Тестируем вызов Http Client 1

```
when(this.httpClientMock.getForEntity(any(), any()))
        .thenReturn(ResponseEntity.ok(VisaRequestResponse.builder().build()));
```

Where is problem Bro?

**String url**, Class<Object> responseType, Object... uriVariables

**String url**, Class<? extends Object> responseType, Map<String, ?> uriVariables

**URI url**, Class<T> responseType

# TDD Costs

```
ResponseEntity<VisaRequestResponse> forEntity = this.template.getForEntity(..., ...);
```

# TDD Costs

```
ResponseEntity<VisaRequestResponse> forEntity = this.template.getForEntity(..., ...);
```


wasted
14:48
$00700911
CV
OG

# TDD Costs

# TDD Costs

**java.lang.NullPointerException: Cannot invoke "org.springframework.http.ResponseEntity.getBody()" because "forEntity" is null**

```
    at com.governance.embassy.service.VisaService.getStatus(VisaService.java:25)
    at
com.governance.embassy.service.VisaServiceTest.should_send_status_request_to_external_service(VisaServiceTest.java:34)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at
java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
    at
java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:568)
    at org.junit.platform.commons.util.ReflectionUtils.invokeMethod(ReflectionUtils.java:727)
    at org.junit.jupiter.engine.execution.MethodInvocation.proceed(MethodInvocation.java:60)
    at
org.junit.jupiter.engine.execution.InvocationInterceptorChain$ValidatingInvocation.proceed(InvocationInter
```

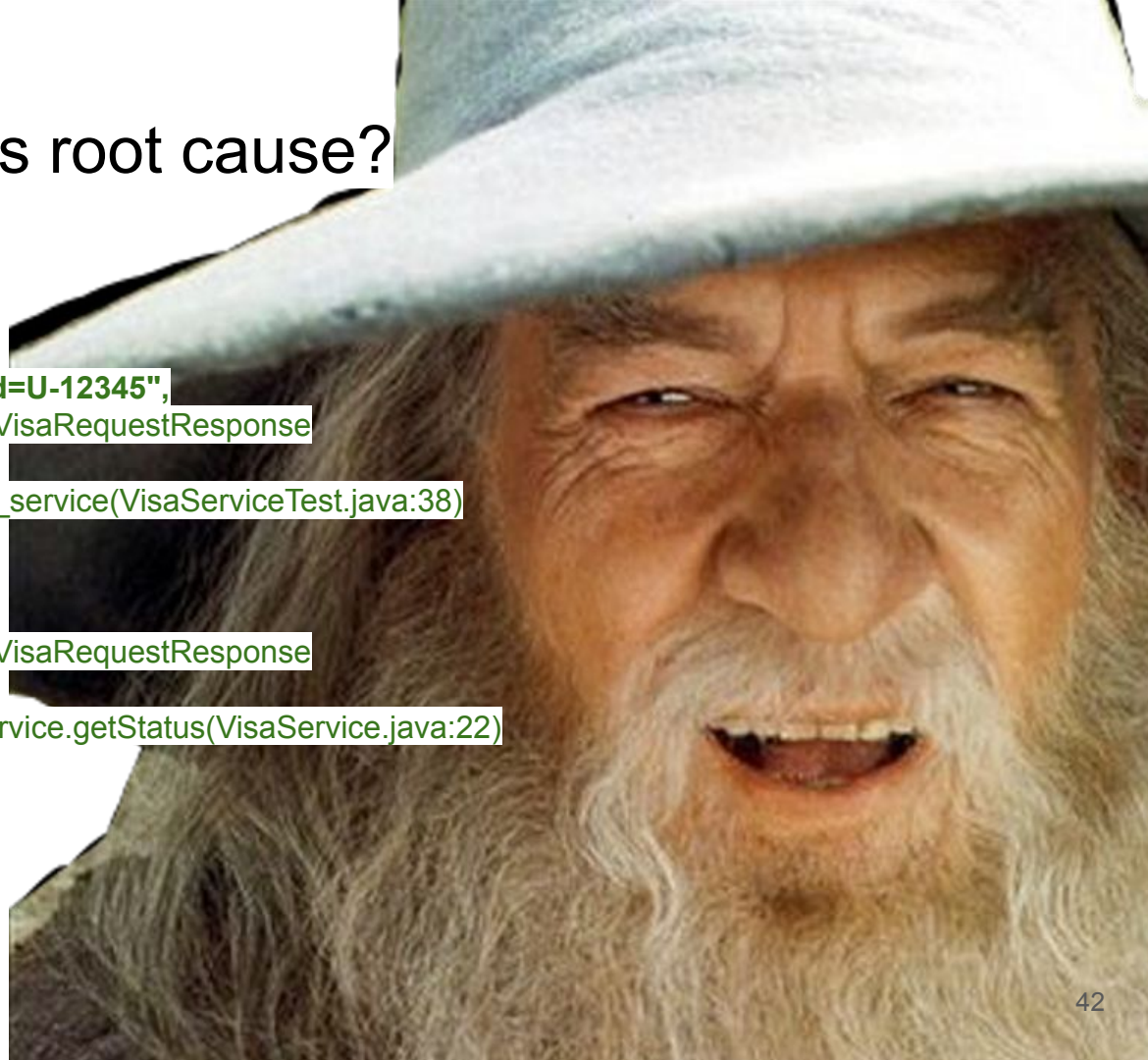
# TDD Costs — Where is root cause?

```
this.template.getForEntity(..., ...).getBody();
```

**java.lang.NullPointerException: Cannot invoke "org.springframework.http.ResponseEntity.getBody()" because "forEntity" is null**

# TDD Costs — Where is root cause?

Argument(s) are different!
**Wanted**:
restTemplate.getForEntity(
    **"http://localhost:8081/visa-request?userId=U-12345",**
    class com.governance.embassy.port.output.VisaRequestResponse
);
-> at should_send_status_request_to_external_service(VisaServiceTest.java:38)
**Actual** invocations have different arguments:
restTemplate.getForEntity(
    **"http://localhost:8081",**
    class com.governance.embassy.port.output.VisaRequestResponse
);
-> at com.governance.embassy.service.VisaService.getStatus(VisaService.java:22)



# TDD Costs — How to fix

```
ResponseEntity<VisaRequestResponse> response = this.template.getForEntity(..., ...);
```

```
if(response == null) {
    return null;
}
```

Avoid Null Pointer. Null is possible?

```
response.getBody();
```

# TDD Costs

```
ResponseEntity<VisaRequestResponse> response = this.template.getForEntity(..., ...);
```

```
if(response == null) {
    return null;
}
```

Avoid Null Pointer. Null is possible?

```
@Override
public <T> ResponseEntity<T> getForEntity(String url, Class<T> responseType, Object... uriVariables)
        throws RestClientException {

    RequestCallback requestCallback = acceptHeaderRequestCallback(responseType);
    ResponseExtractor<ResponseEntity<T>> responseExtractor = responseEntityExtractor(responseType);
    return nonNull(execute(url, HttpMethod.GET, requestCallback, responseExtractor, uriVariables));
}
```

# TDD for Business Logic Http Client 1 Puzzler

```
httpClientMock = Mockito.mock(RestTemplate.class);
```

# TDD for Business Logic Http Client 1 **Puzzler**

1. `httpClientMock = Mockito.mock(RestTemplate.class);` Pure Mockito
2. `@MockBean RestTemplate httpClientMock;` Spring Boot Test
3. `@Mock RestTemplate httpClientMock;` Mockito inited via MockitoExtension

## Feel the difference

```
if(response == null) {
   return null;
}
```

Avoid this bullshit code

# TDD for Business Logic Http Client 1 **Puzzler**

```
@Mock(strictness = Mock.Strictness.LENIENT)
```

# Что тестируем


tested
14:48
$00700911
CV
OG

# И тут пришёл Operations [опять вносим правки]

Нельзя часто
звать Visa Agent`a

TDD for

Business Logic

Http Client 1

**Cache?**

```
@Cacheable("visa-request")
```

Что лучше?

```
private CacheManager cacheManager;
...
cacheManager.get(...)
```

TDD for

Business Logic

Http Client 1

**Cache?**

Ну и где твой TDD?

# TDD for Business Logic Http Client 1 **Puzzler**

```
@Mock(strictness = Mock.Strictness.LENIENT)
```

Опять вернулись к проблеме

TDD for

Business Logic

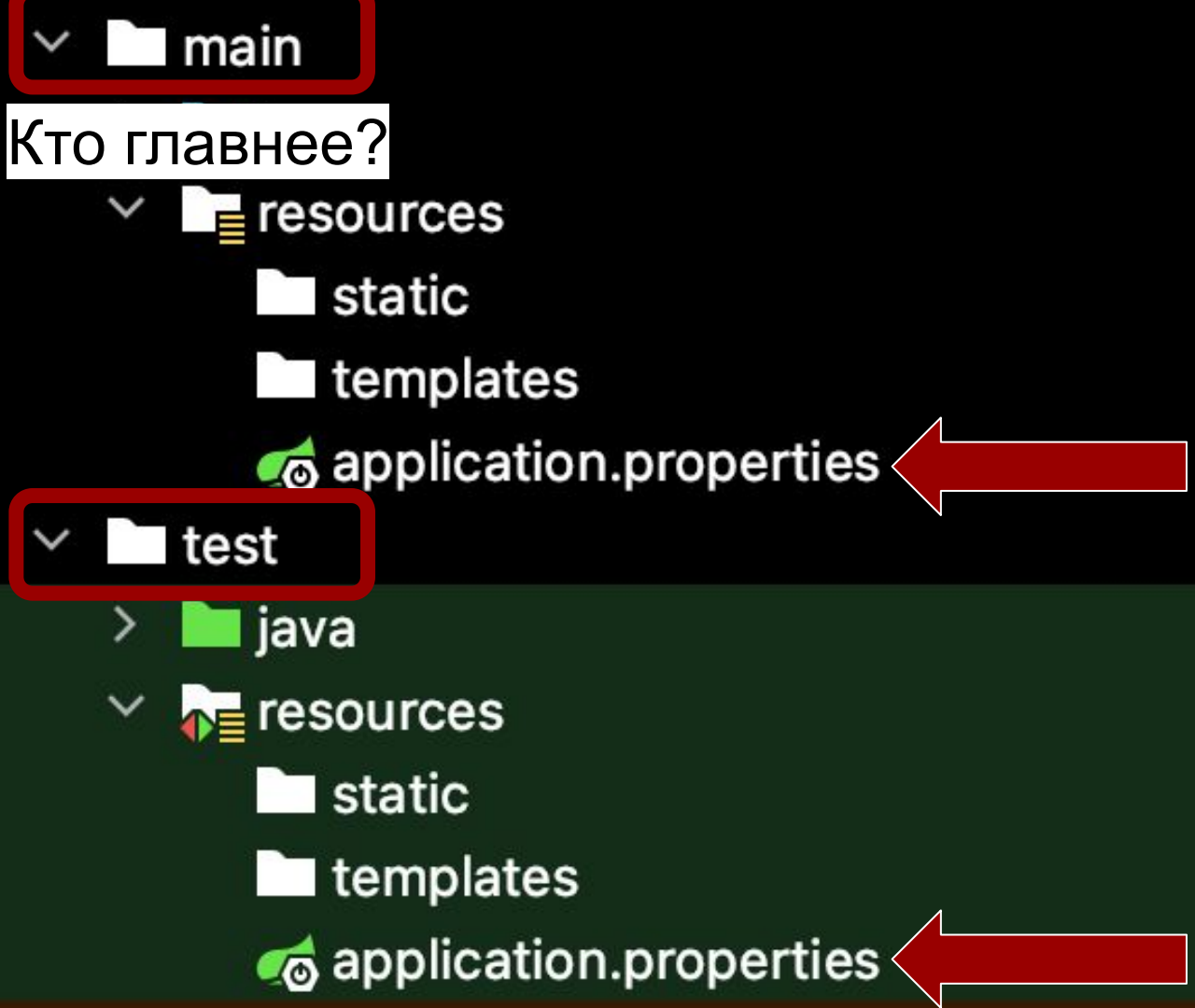
Http Client 1

**Cache?**

# Ну и где твой TDD?

А когда удалять из кеша? …

Кто главнее?


main
resources
static
templates
application.properties
test
java
resources
static
templates
application.properties


# Как запускаются Тесты?

A. Как то JUnit запускает

B. JUnitStarter

C. java -jar *Test.class

D. С божьей помощь

# Java build guide

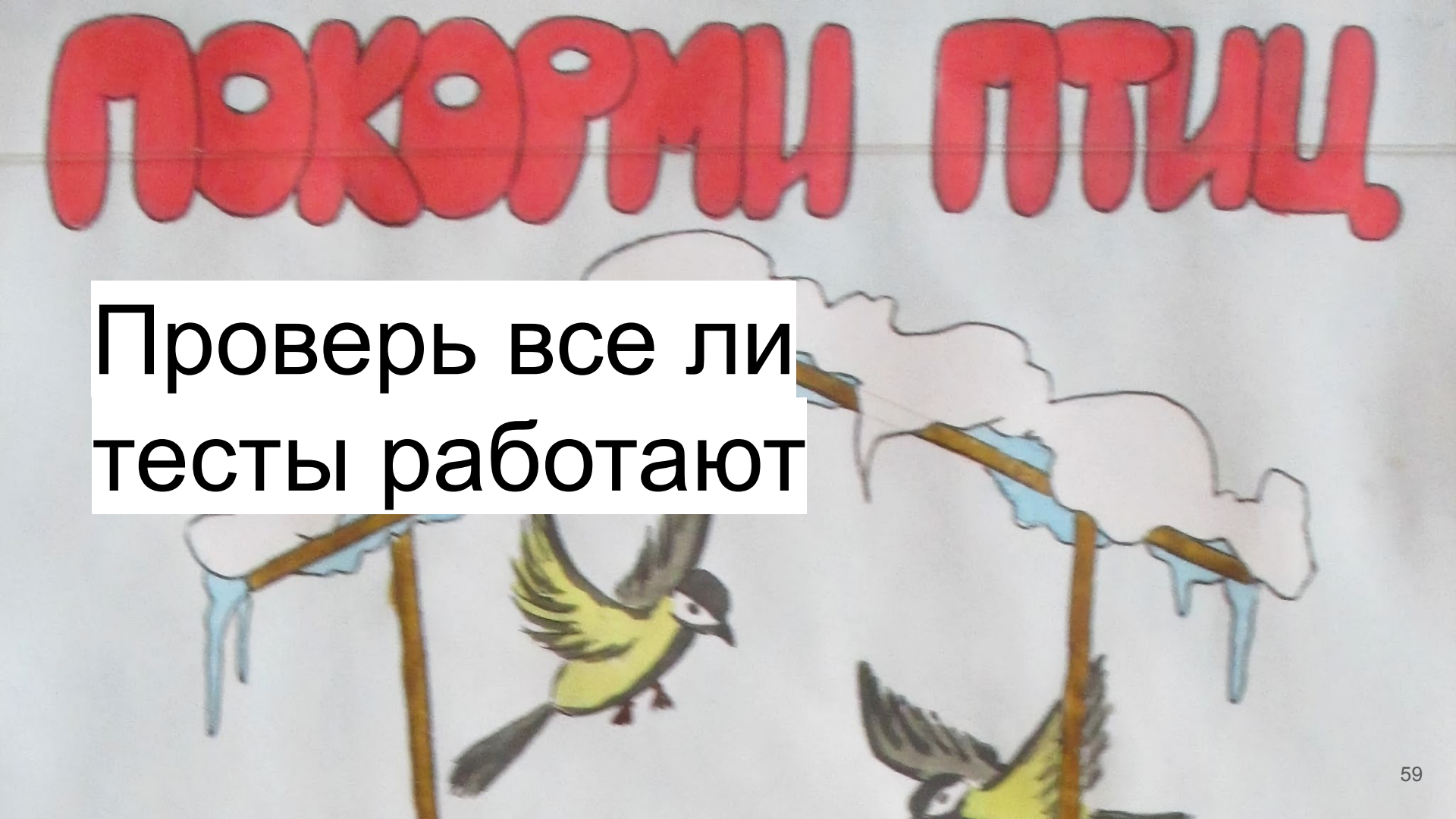
1. Compile
2. Configure ClassPath
3. Run some compiled class with classpath

```
> java -cp $CLASSPATH org.junit.Runner
```

Compiled Main + Test + Libs

# Подведём итоги

1. Посмотрели на разные типы моков
2. Протестировали послойно приложение
3. Нашли особенности работы файлов конфигурации
4. Изучили как работает Cacheable если не знали :)




Проверь все ли тесты работают

Где Business Logic грузить данные?

Load old data


```
@PostConstruct
```

```
@EventListener
(ContextRefreshedEvent.class)
```

# Lazy init — Good, Bad, Is ok

1. application.properties:

```
spring.main.lazy-initialization=true
```

<u>OR</u>

2. In code

```
@SpringBootTest(
        properties = "spring.main.lazy-initialization=true"
)
```

TDD for

Business Logic

Http Client 1/Rest Spec

**Serde?**

А где проверили как сериализуются классы?

# TDD for Business Logic Http Client 1 **Serde?**

```
//when
ResponseEntity<VisaResponse> forEntity =
template.getForEntity("/visa?userId=..", VisaRequestResponse.class);

VisaRequestResponse body = forEntity.getBody();
```

# TDD for Business Logic Http Client 1 Serde?

```
//when
ResponseEntity<VisaResponse> forEntity =
template.getForEntity("/visa?userId=..", VisaRequestResponse.class);

VisaRequestResponse body = forEntity.getBody();
```

# TDD for Business Logic Http Client 1 Serde?

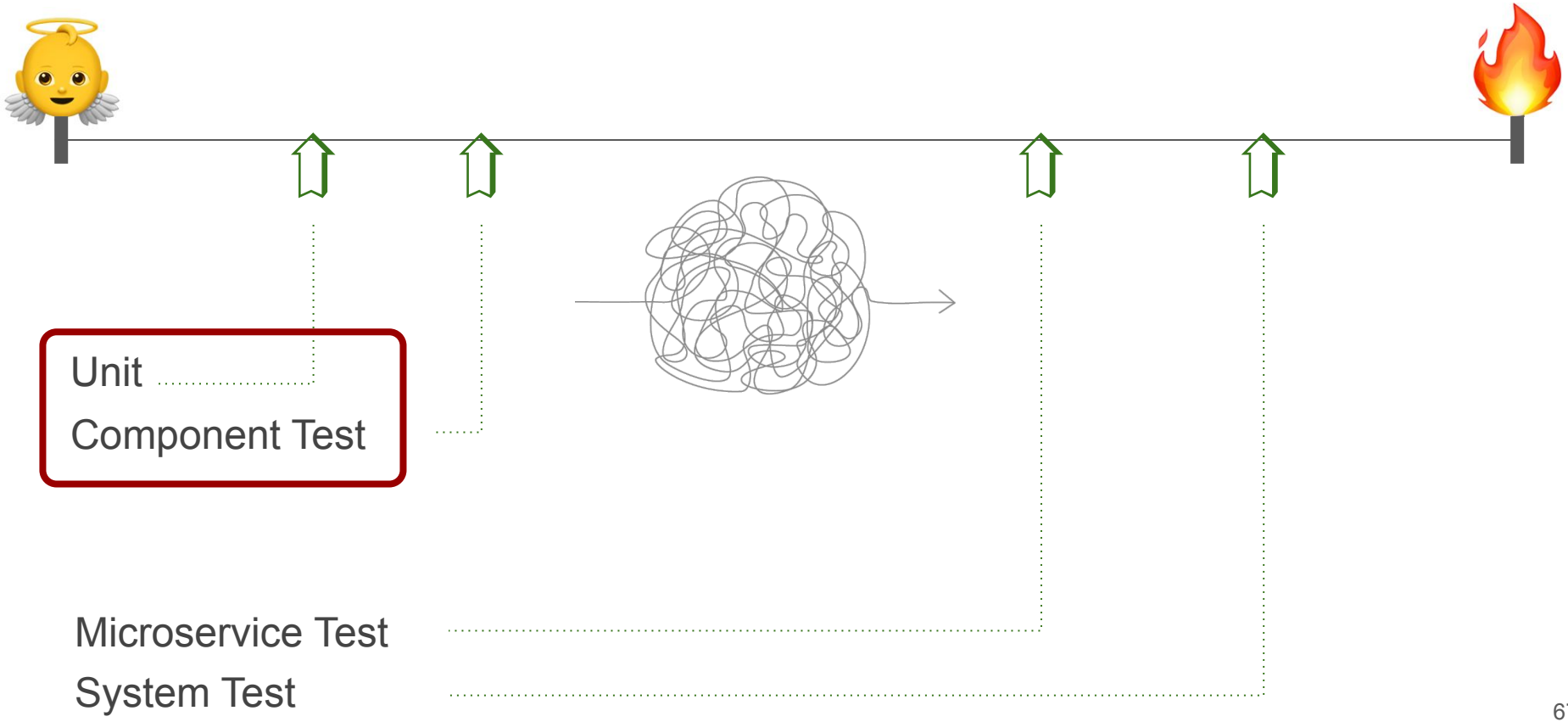
```
//when
ResponseEntity<VisaResponse> forEntity =
template.getForEntity("/visa?userId=..", VisaRequestResponse.class);

VisaRequestResponse body = forEntity.getBody();
```

TDD for Business Logic HTTP Spec **Serde?**

```
@WebMvcTest
public class VisaRequestControllerWebMVCTest {
```

# Шкала Тестов

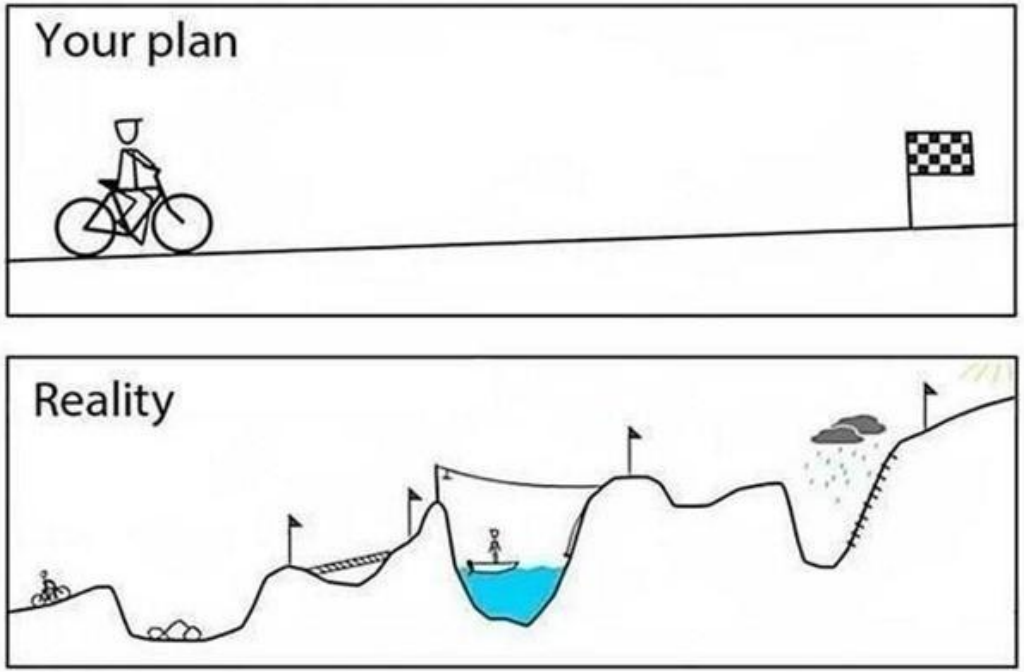


# Так, а что мы написали?

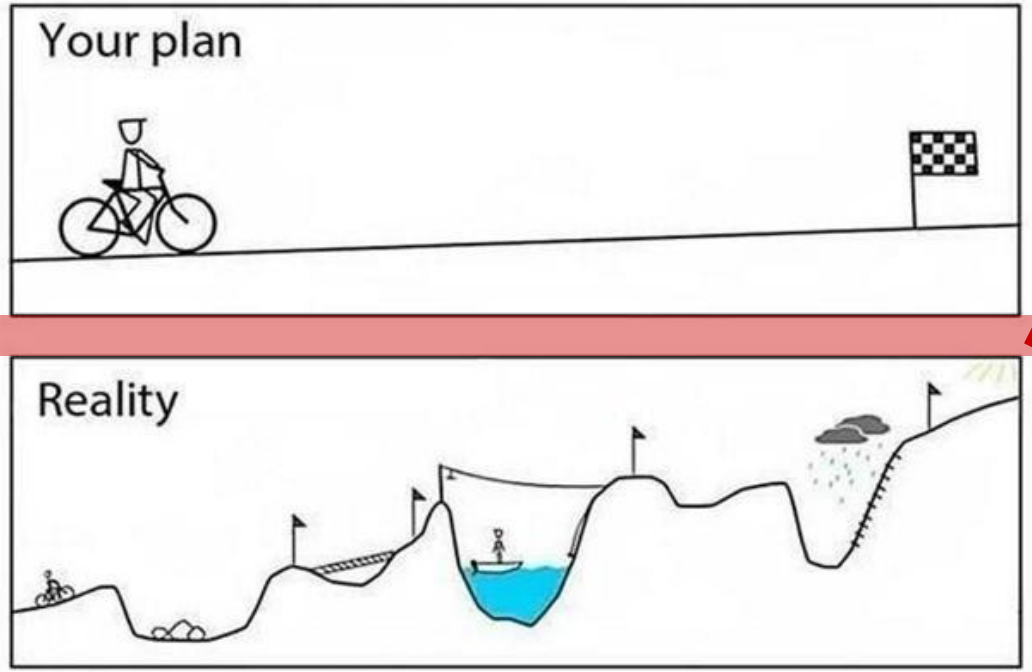
И для чего

# Unit/Component тесты. Для чего?

# Unit/Component тесты. Для чего?

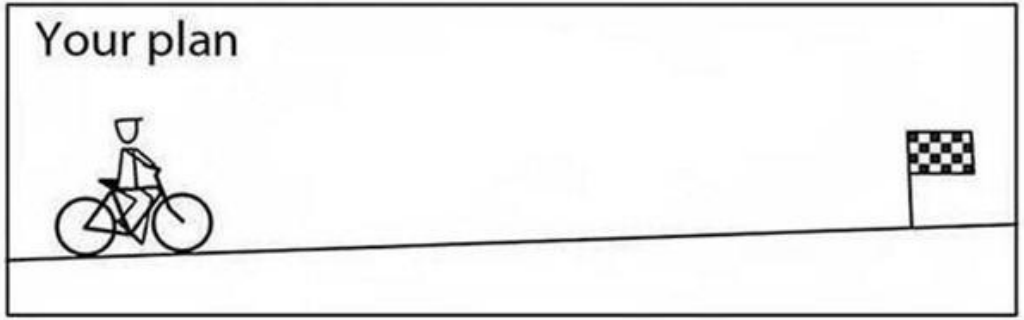


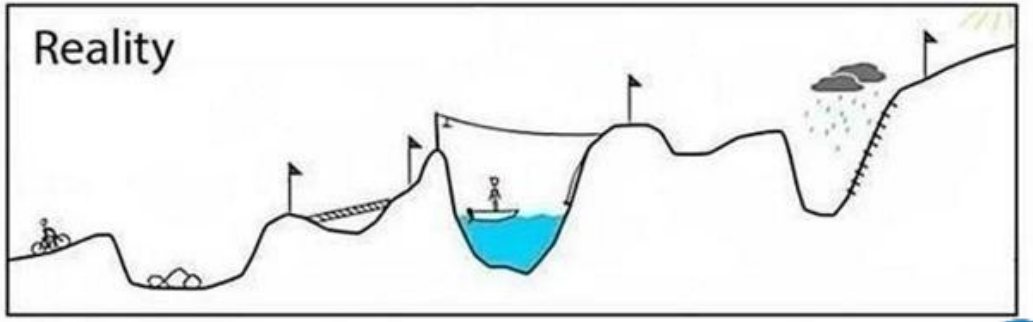

# Unit/Component тесты. Для чего?




**Ваши тесты Тут**

# Unit/Component тесты. Для чего?

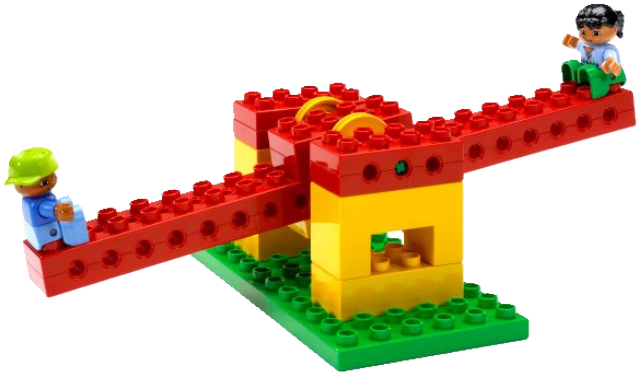


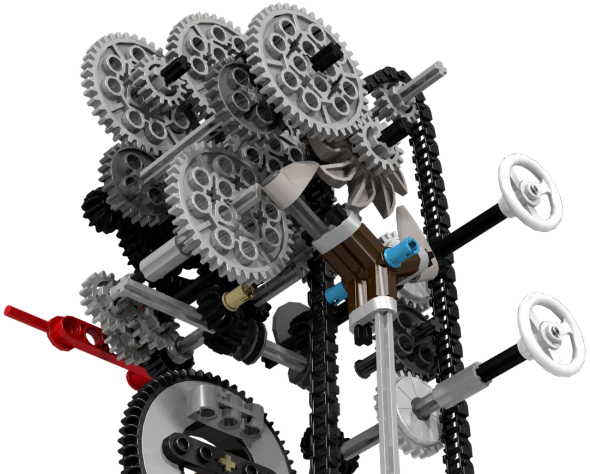

**Тесты уменьшают неопределённость**

# Есть два типа тестов

Простой

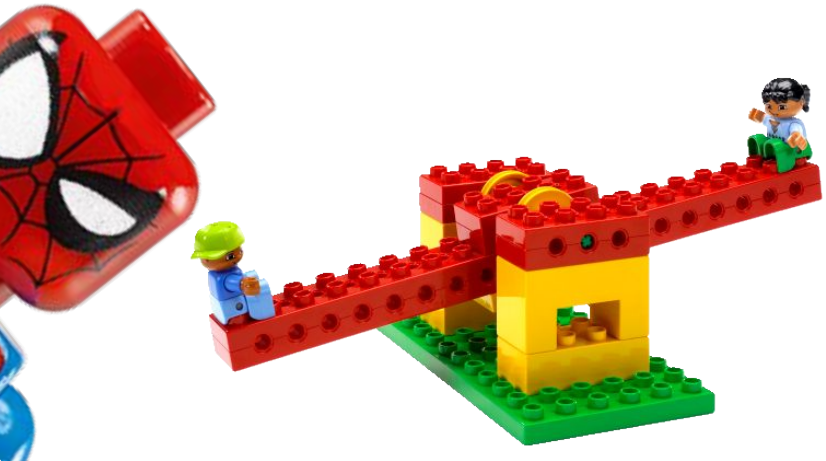


Сложный

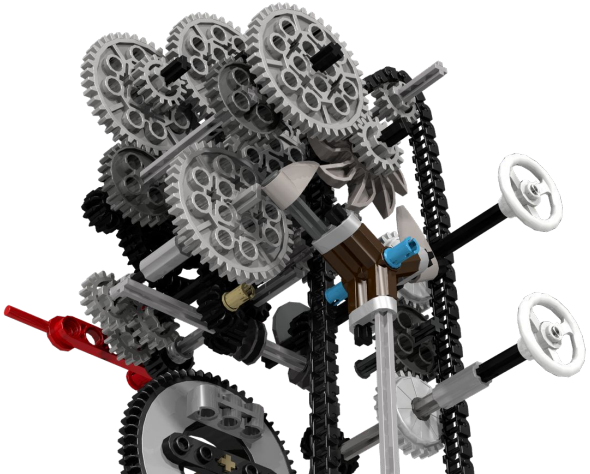


# Есть два типа тестов
# Какой сам выберешь

Простой

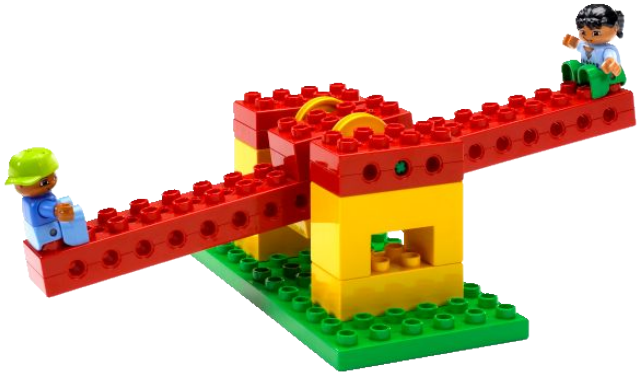


Сложный

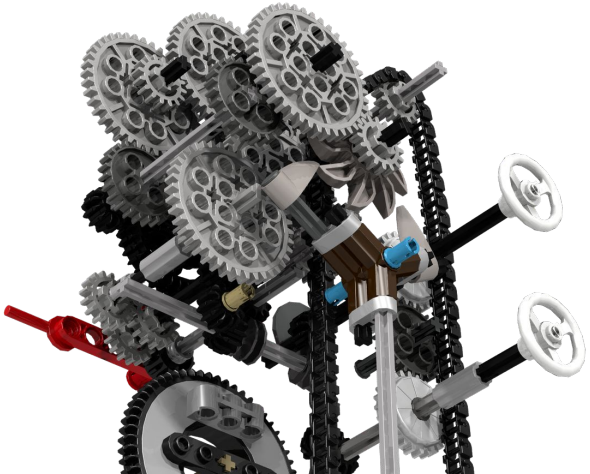


# Есть два типа тестов
# Какой сам выберешь,
# а какой разработчику оставишь?

**Простой**



**Сложный**



# Когда пишут тесты?

# Когда пишут тесты?

1. Требование заказчика

# Когда пишут тесты?

1. Требование заказчика
2. **Культура**

# Когда пишут тесты?

1. Требование заказчика
2. **Культура** → **Перед кодом**

# Когда пишут тесты?

1. Требование заказчика
2. **Культура**

**Перед кодом**

**Вместе кодом**

# Когда пишут тесты?

1. Требование заказчика
2. **Культура**

**Перед кодом**

**Вместе кодом**

**После кода**

# Про какие тесты будем говорить?

Unit

Component Test

**Перед кодом**

**Вместе кодом**

**После кода**

# Про какие тесты будем говорим?

Unit

**Component Test**

**Перед кодом**

**Вместе кодом**

После кода

# Что тестируем

# TDD for Business Logic

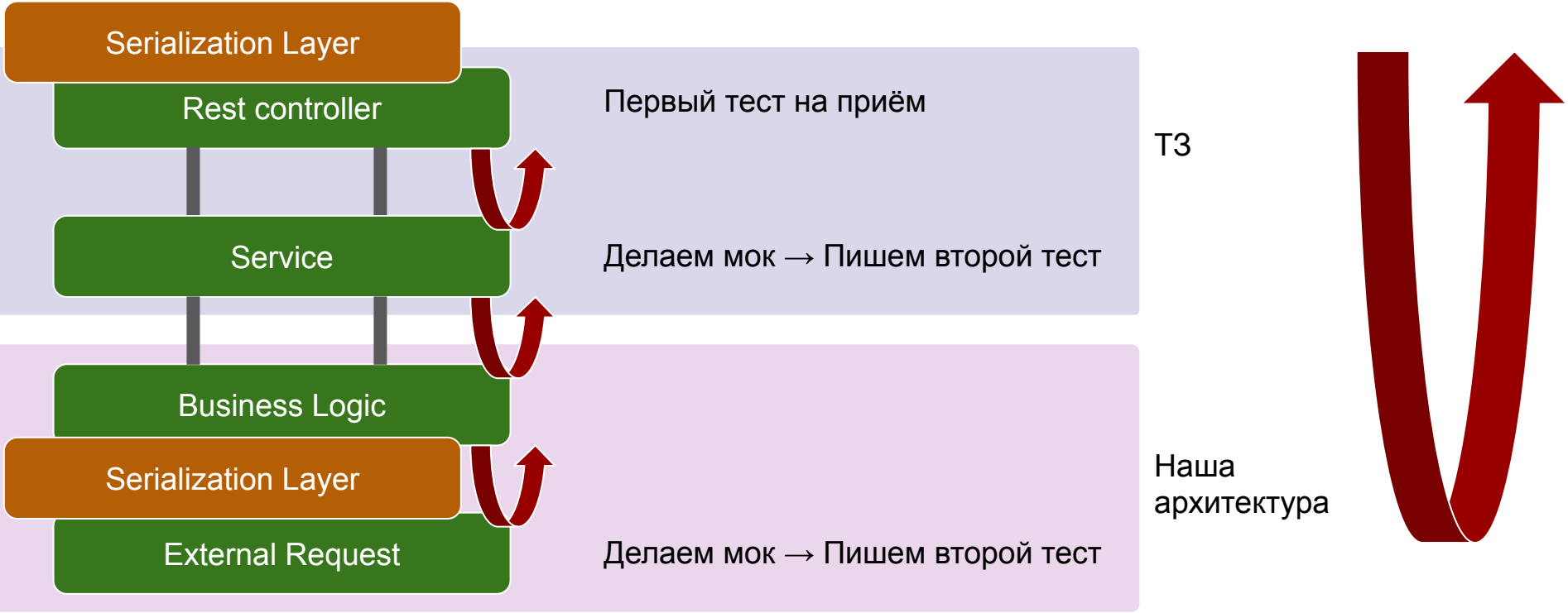
Serde Test

```
@JsonComponent
```

```
@JsonTest
```

```
JacksonTester<T> tester;
```

# За кадром (см @WebMvc, @JsonTest)

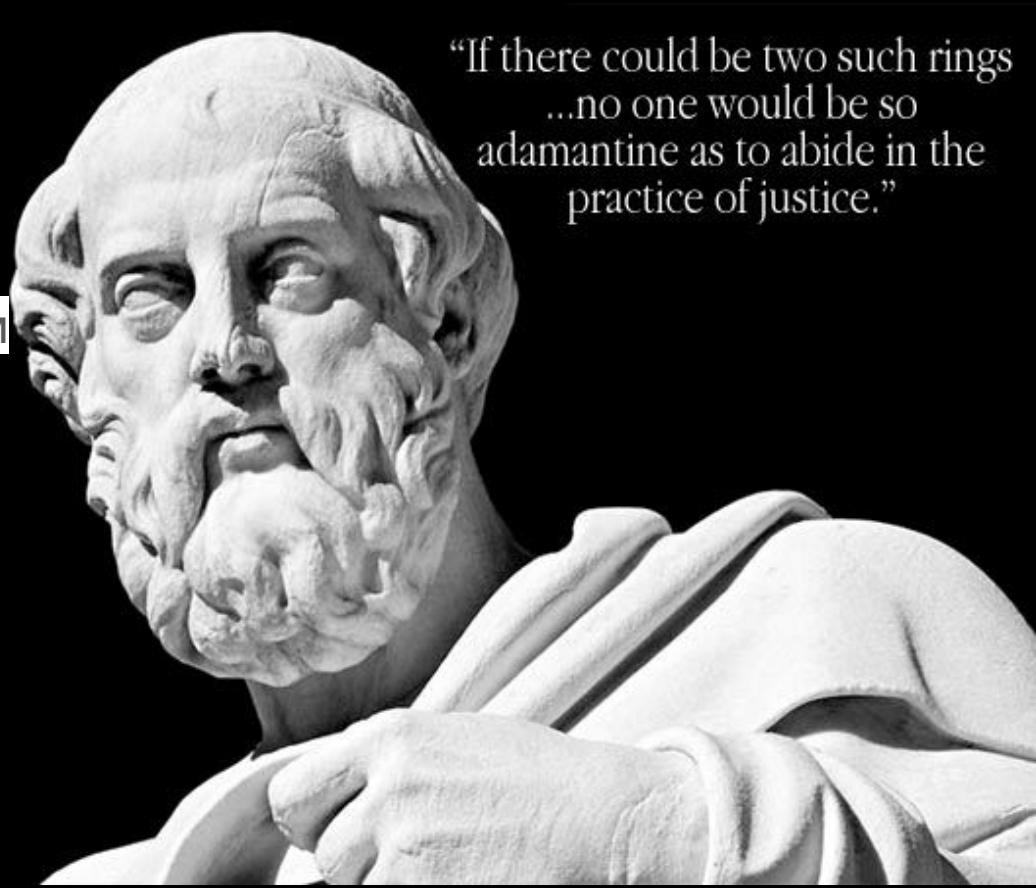


# Выводы

Типы тесты, разработка через TDD

# Выводы

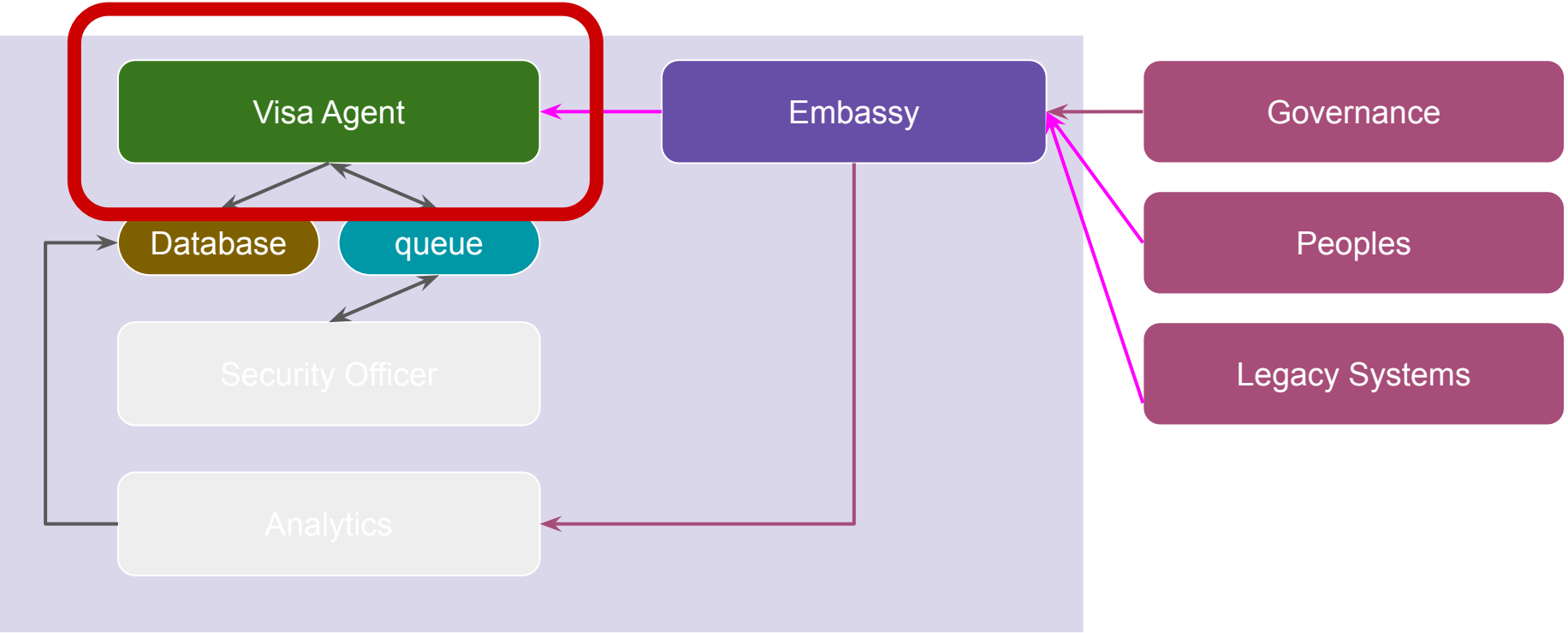
Писать тесты – Это как ходить по лезвию, ваша задача не переборщить со скоупом и при этом не переусложнить тест одновременно

# Часть 2 – Visa Agent

Комплексные приложения – Слои,
Структура интерференция тестов

# Архитектура того что ~~мучаем~~ тестируем



# Манифестируем работу Visa сервиса

ТЗ Visa Agent APP

1. Ответственен за заявки на визы
2. Создаёт заявки
3. Предсотавляет доступ к ним
4. Уведомляет о создании других

# Манифестируем работу Visa сервиса

ТЗ Visa Agent APP

1. Ответственен за заявки на визы
2. Создаёт заявки
3. Предсотавляет доступ к ним
4. Уведомляет о создании других

И наше требование тренинга – **Well Tested** :)

# Опять тестировать?

# Пропустим

1. Rest Controller тесты с помощью WebMVC
2. Создание Business слоя

# А что нового?

1. Data tests (postgresql)
2. Messaging tests (kafka)

# А что тестировать?

Data

1. Сохранились ли данные в базу?
2. Не сохраняются данные если объект невалиден? (статус?)
3. Статус обновляется?
4. Проиницилазировалась ли база данных с помощью FlyWay

Messaging

1. Ушло ли сообщение в очередь
2. Пришло ли в нужный топик
3. Правильно ли сериализовалось/десериализовалось
4. Откуда начинает считываться очередь при старте

# Что нужно чтобы протестировать работу с БД?

1. Проверить подключение?
2. Инициализацию?
3. CRUD…

# А кто в этом поможет?

1. DataJpaTest?
2. H2?
3. TestContainers?

# Testcontainers vs H2

Кого выбирать?

# Testcontainers

Но почему?

# Быть ближе к реальности

Тестовое окружение как прод

→ Вероятность успеха в проде больше

→ Через тесты изучаете как работает реальная система

→ Подбираем решения и технологии которые знаем

# Почему?

```
CREATE FUNCTION AddData() RETURNS INTEGER
AS
$$
BEGIN
   INSERT INTO visa_request (id, user_id) VALUES (1, 'user0' + random());
   RETURN 1;
END;
$$ LANGUAGE plpgsql;
```

# Как выглядят твои тесты

Кто виноват и почему

# Как выглядят твои тесты

| | |
|---|---|
| Test Results | 275 ms |
| VisaPersistenceTest | 166 ms |
| should_save() | 124 ms |
| should_update_entity() | 42 ms |
| CreateVisaControllerTest | 95 ms |
| should_receive_visa_reque | 88 ms |
| should_inform_about_invalid | 7 ms |
| VisaApplicationTests | 2 ms |
| VisaStatusControllerTest | 12 ms |
| should_return_status_for_s | 10 ms |
| should_return_404_when_n | 2 ms |

# Как выглядят твои тесты

- servicevisaagent (com.governance.visaagent) 20 sec 995 m
  - VisaServiceMessagingTest 624 m
    - should_send_visa_request_notification_to_qu 624 m
  - VisaStatusControllerTest 65 m
    - should_return_status_for_saved_request() 63 m
    - should_return_404_when_no_request_available( 2 m
  - CreateVisaControllerTest 24 m
  - NotifyNewVisaRequestToKafkaTest 10 sec 137 m
    - should_send_visa_request_to_queue_af 10 sec 137 m
  - VisaRepositoryTest 57 m
  - VisaUpdateStatusListenerTest 7 m
  - VisaServiceTest 51 m
    - should_return_status_by_ticket_id() 45 m
    - should_return_old_ticket_id_if_visa_request_for 3 m
    - should_persist_new_request_and_return_id()
    - should_send_new_request_to_queue_after_persist(
    - should_throw_exception_when_many_tickets_in 3 m
  - ServiceVisaAgentApplicationTests 2 m
  - VisaUpdateStatusMessagingTest 10 sec 28 m
    - should_receive_visa_request_update() 10 sec 28 m

# Как выглядят твои тесты

- Test Results 275 ms
  - VisaPersistenceTest 166 ms
    - should_save() 124 ms
    - should_update_entity() 42 ms
  - CreateVisaControllerTest 95 ms
    - should_receive_visa_reque 88 ms
    - should_inform_about_invalid 7 ms
  - VisaApplicationTests 2 ms
  - VisaStatusControllerTest 12 ms
    - should_return_status_for_s 10 ms
    - should_return_404_when_n 2 ms

- servicevisaagent (com.governance.visaagent) 20 sec 995 m
  - VisaServiceMessagingTest 624 m
    - should_send_visa_request_notification_to_qu 624 m
  - VisaStatusControllerTest 65 m
    - should_return_status_for_saved_request() 63 m
    - should_return_404_when_no_request_available( 2 m
  - CreateVisaControllerTest 24 m
  - NotifyNewVisaRequestToKafkaTest 10 sec 137 m
    - should_send_visa_request_to_queue_af 10 sec 137 m
  - VisaRepositoryTest 57 m
  - VisaUpdateStatusListenerTest 7 m
  - VisaServiceTest 51 m
    - should_return_status_by_ticket_id() 45 m
    - should_return_old_ticket_id_if_visa_request_for 3 m
    - should_persist_new_request_and_return_id()
    - should_send_new_request_to_queue_after_persist(
    - should_throw_exception_when_many_tickets_in 3 m
  - ServiceVisaAgentApplicationTests 2 m
  - VisaUpdateStatusMessagingTest 10 sec 28 m
    - should_receive_visa_request_update() 10 sec 28 m

# Тесты должны мигать

# Тесты должны мигать

А вы разбираться почему это происходит

# Что реально нужно проверять и знать?

Проверять / фиксировать в тестах

1. Изоляция транзакций
2. Консистентность данных
3. Инициализацию БД в разных условиях
4. …

Что знать

1. DataJpaTest
2. Transaction propagation
3. Flyway/Liquibase
4. TestContainers/H2
5. SQL + DB Specific Language
6. JDBC driver resolution
7. …

1. Настраиваем datasource url
2. Инициализируем БД
3. Встраиваем в тесты

# Что реально нужно проверять и знать?

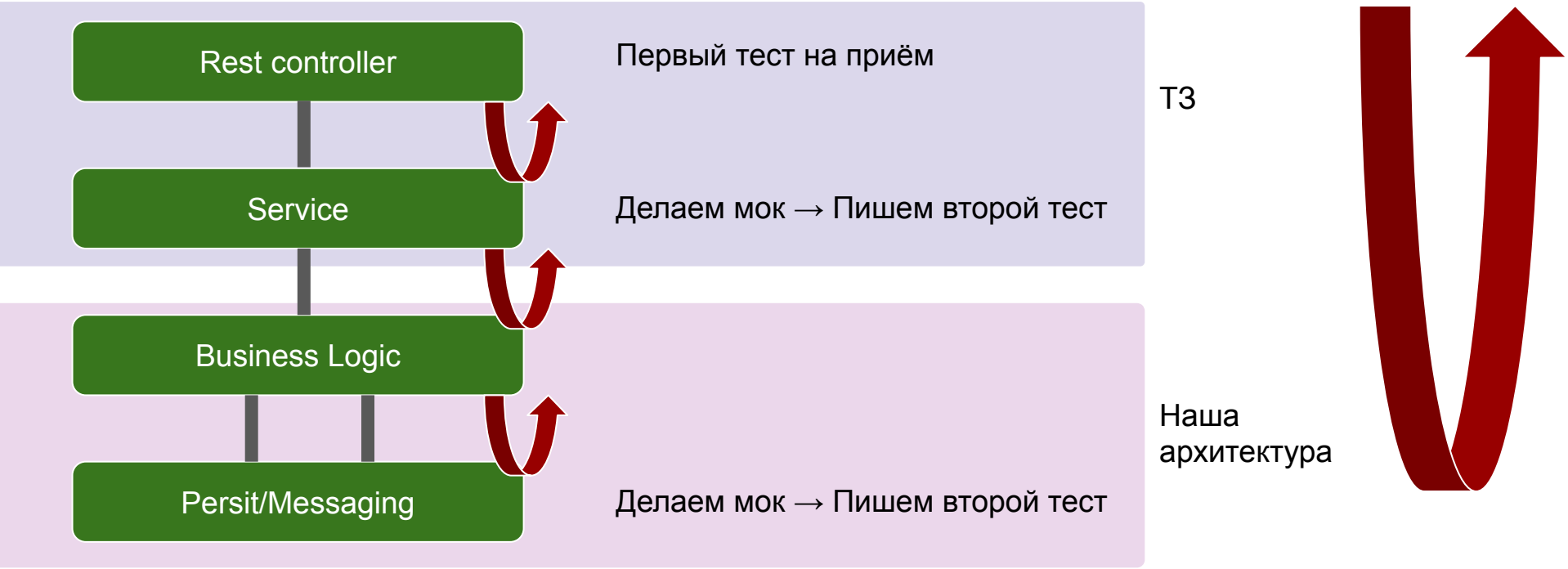
1. Настраиваем datasource url

```
spring.datasource.url =jdbc:tc:postgresql:15.2-alpine:///db?TC_REUSABLE=true
```

2. Инициализируем БД — V1__INIT.sql
3. Встраиваем в тесты

```
@AutoConfigureTestDatabase (replace =
AutoConfigureTestDatabase .Replace.NONE)
```

# Что тестируем




Messaging

# А сообщение отправилось в Kafka?

Как проверить

# Testcontainers vs KafkaEmbedded

Кого выбирать?

# Testcontainers

Но почему?

# Быть ближе к реальности

Тестовое окружение как прод

→ Вероятность успеха в проде больше

→ Через тесты изучаете как работает реальная система

→ Подбираем решения и технологии которые знаем

# Изучаем Kafka через тосты

# Как?

И что для этого нужно знать

# Что и как тестировать?

1. Отправка / получение
2. Сериализация / десериализация
3. Распределение сообщений в брокере
4. Инициализация и подключение

# А как?

1. @SpringBootTest
2. @TestConfiguration
3. @ContextConfiguration
4. @SpringBootConfiguration
5. ApplicationContextInitializer
6. @KafkaListener
7. CountDownLatch
8. TestContainers
9. *AutoConfiguration and exckude

# А как?

1. @SpringBootTest
2. @TestConfiguration
3. @ContextConfiguration
4. @SpringBootConfiguration
5. ApplicationContextInitializer
6. @KafkaListener
7. CountDownLatch
8. TestContainers
9. *AutoConfiguration and exclude specific
10. Kafka specific

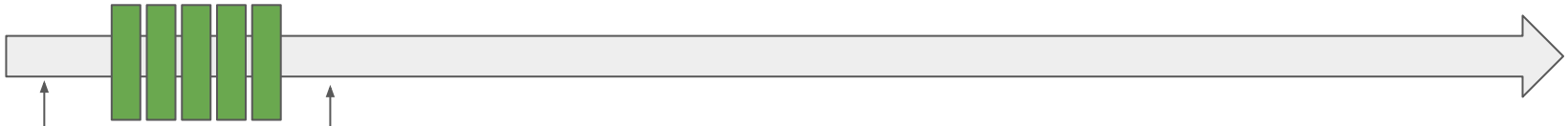
# Что делать

1. Add @Test
2. Add related @TestConfiguration with @KafkaListener
3. Run kafka with @Testcontainers
   a. Add @Container KafkaContainer
   b. Add @DynamicPropertySource for register kafka credentials in Spring context
4. Use @Spy, @MockBean and CountDownLatch for write tests
5. Research how to Kafka really work…

# Что делать

1. Add @Test
2. Add related @TestConfiguration with @KafkaListener
3. Run kafka with @Testcontainers
   a. Add @Container KafkaContainer
   b. Add @DynamicPropertySource for register kafka credentials in Spring context
4. Use @Spy, @MockBean and CountDownLatch for write tests
5. Research how to Kafka really work…

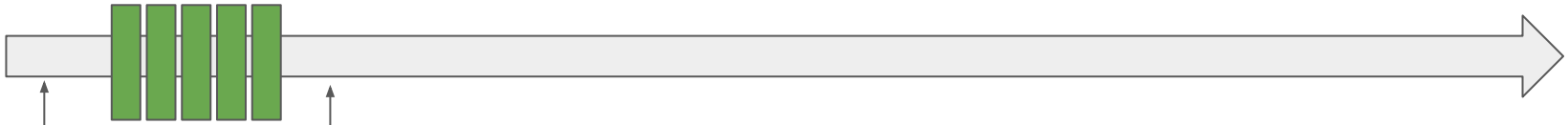
# Kafka specific? Изучаем Kafka через тесты



```
@Tescontainers + @DynamicPropertySource
```

```
static {
    Startables.deepStart(kafka).join();
}
+ @ContextConfiguration(initializers…
```

# Изучаем Kafka через тесты



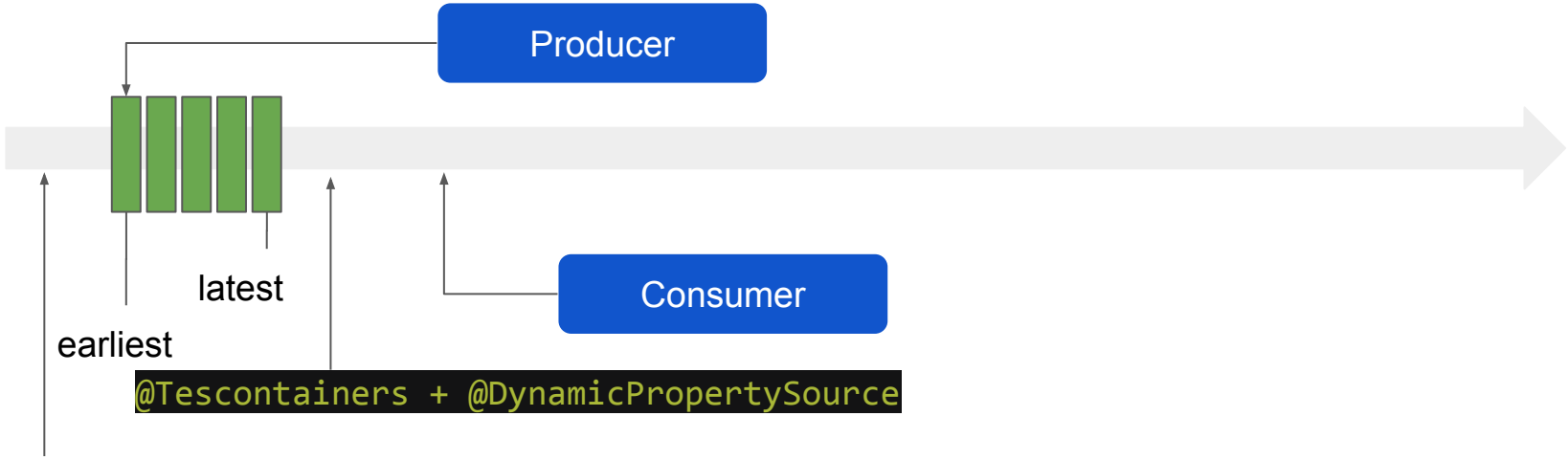
```
@Tescontainers + @DynamicPropertySource
```

```
static {
   Startables.deepStart(kafka).join();
}
+ @ContextConfiguration(initializers…
```

```
spring.kafka.consumer.auto-offset-reset=earliest
```

Используем для всех тестов

# Изучаем Kafka через тесты




```
@Tescontainers + @DynamicPropertySource
```

```
static {
   Startables.deepStart(kafka).join();
}
+ @ContextConfiguration(initializers…
```

```
spring.kafka.consumer.auto-offset-reset=earliest
```

Используем для всех тестов

# Долго

Тесты должны быть быстрыми!

# На что тратится время

1. Старт окружения для контейнеров
2. Старт нужных контейнеров (можно сделать асинхронным)
3. Подключение Spring Boot к контейнером (пулы, соединения, итд)

https://www.testcontainers.org/features/reuse/

# На что тратится время

1. **Старт окружения для контейнеров**
2. **Старт нужных контейнеров (можно сделать асихронным)**
3. Подключение Spring Boot к контейнером (пулы, соединения, итд)

Reuse containers!

```
$ echo "testcontainers.reuse.enable=true" >> ~/.testcontainers.properties
```

https://www.testcontainers.org/features/reuse/

# А сколько раз

Запускается Kafka?

# А сколько раз

Запускается PostgreSQL?

# А что с кэшированием?

Контекст кэшируется для разных тестов?

# DataJpaTest and Transactional problem

```
@good.DataJpaTest →
  @DataJpaTest + @Transactional(propagation =
Propagation.NOT_SUPPORTED)
```

# ORM vs Custom Repo Method

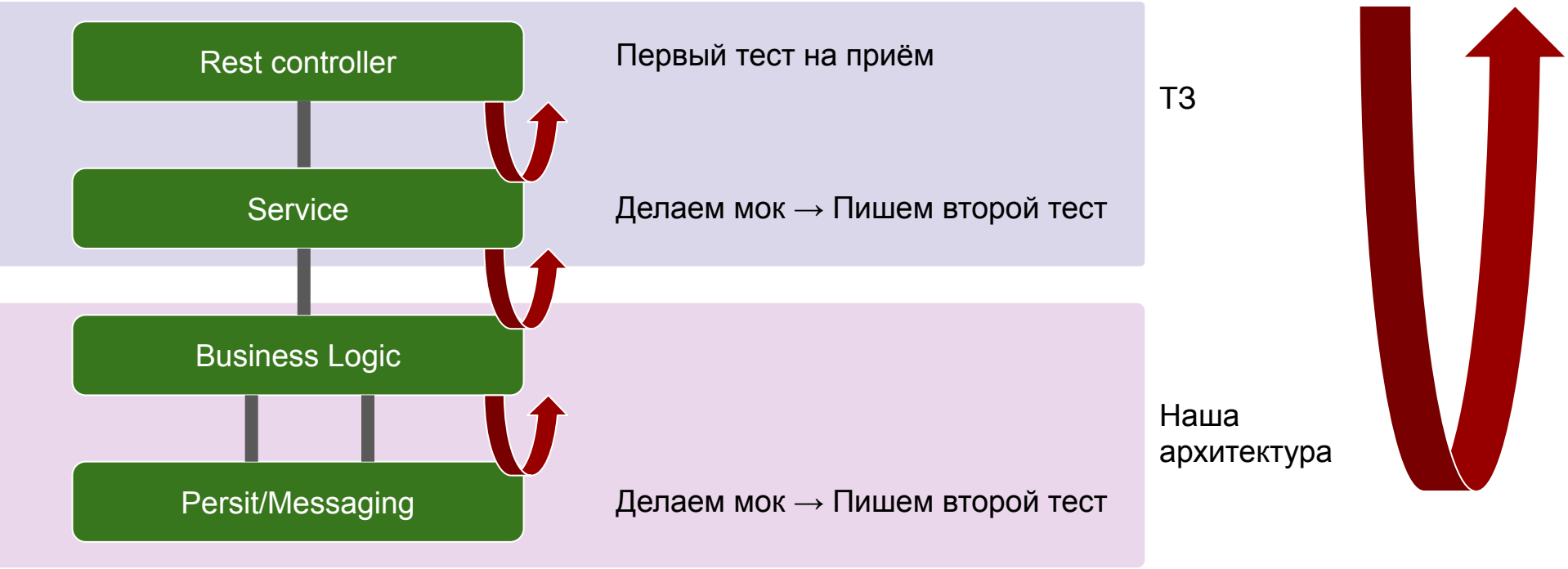
## Update method

```
entity.setStatus("accepted");
→ 2 sql requests
More memory
```

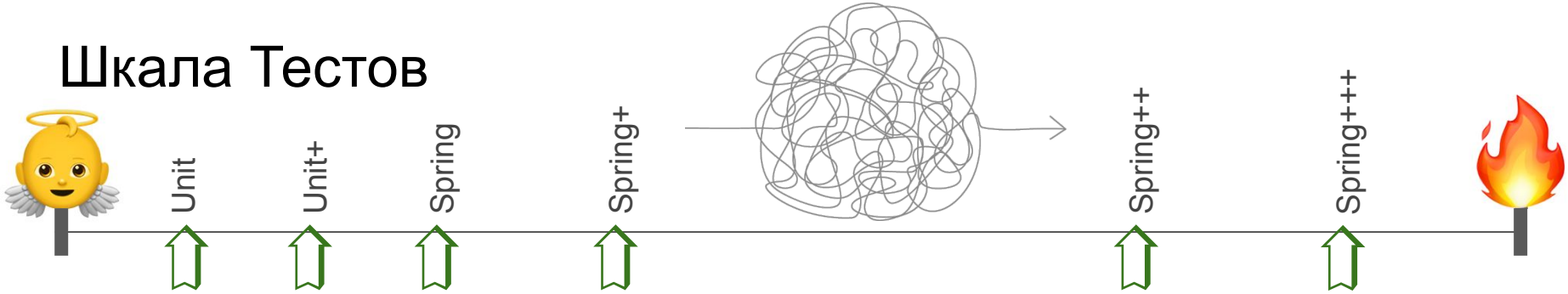
## Update method in repository

```
@Query("update VisaRequest v set v.status = ?1 where v.id = ?2")
int updateStatusById(String status, Long id);
→ 1 sql requests
Less memory
```

# Что тестируем



# Шкала Тестов




One target class
+ Mocktio

Target class +
dependencies

@ContextConfiguration
+ part of context/profiles

DataJpaTest/JsonTest/etc
Context with
@BootstrapWith + Filters

@SpringBootTest
+ lazy-initializatoin

@SpringBootTest

# Context Bootstrapper & Include/Exclude Filters

Или как сделать свой контекст

# Context Bootstrap

DataJpaTest

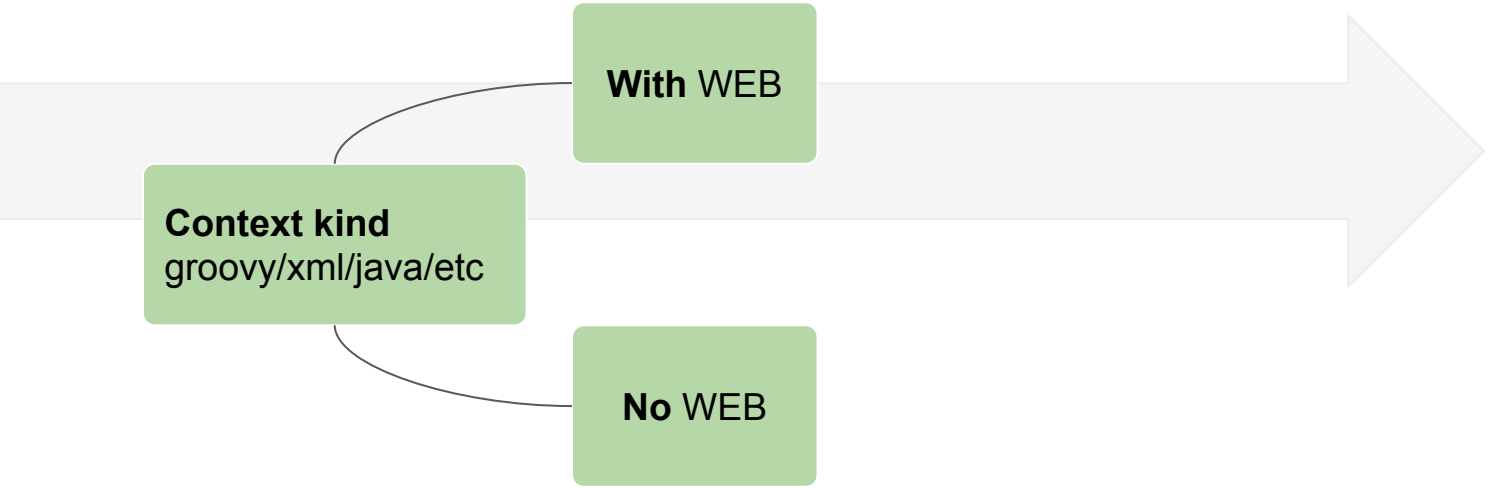
SpringBootTest

…

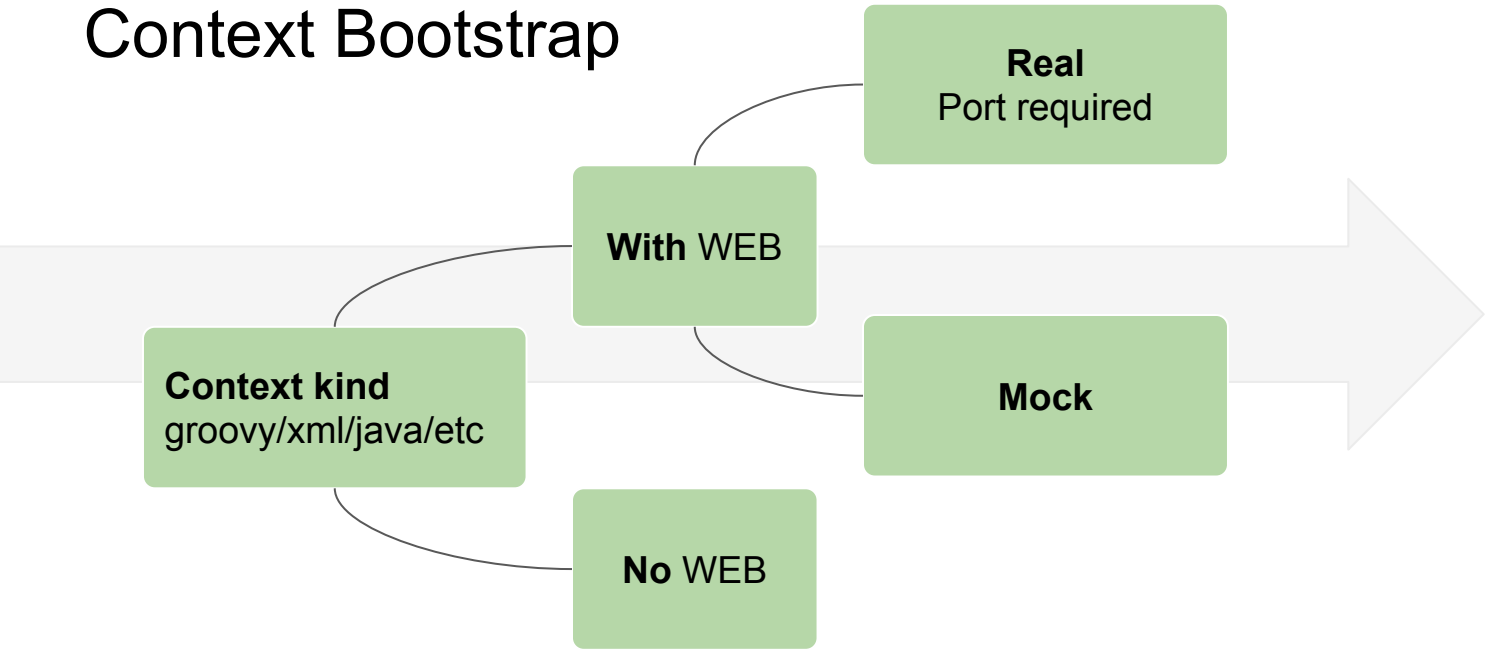
# Context Bootstrap

**Context kind**
groovy/xml/java/etc

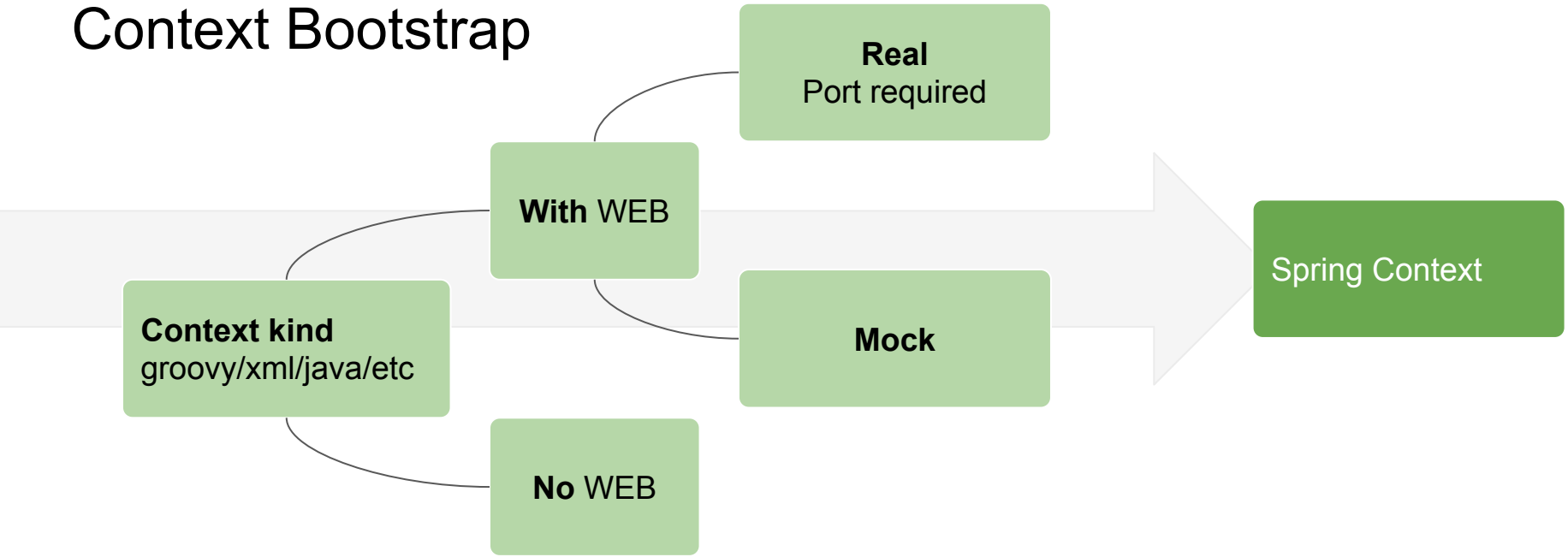
# Context Bootstrap



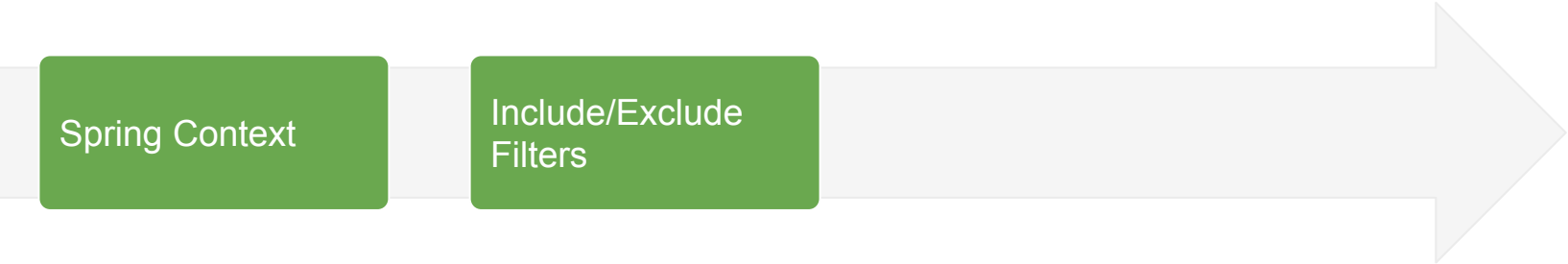
# Context Bootstrap



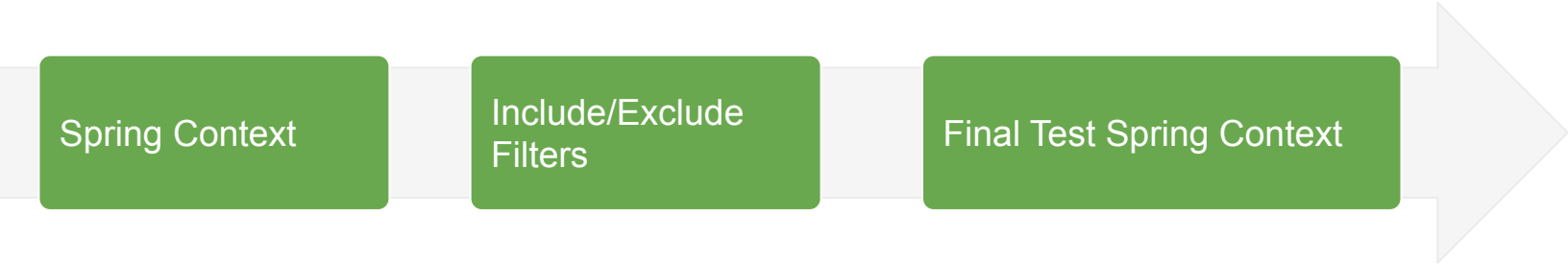
# Context Bootstrap



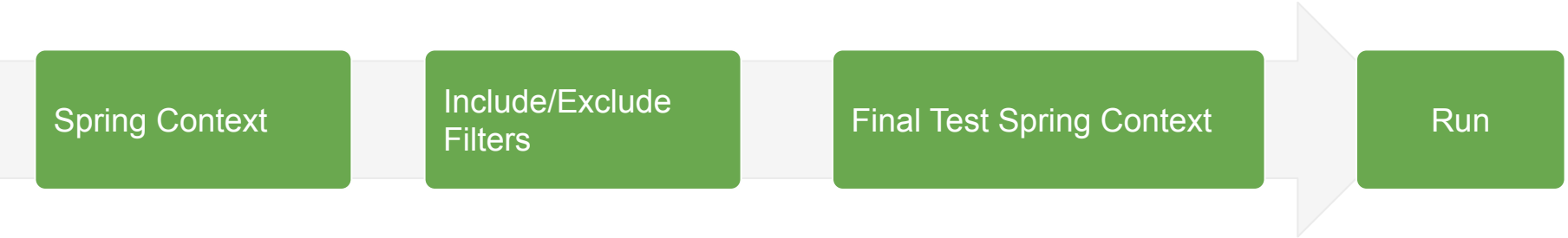
# Context Bootstrap



# Context Bootstrap

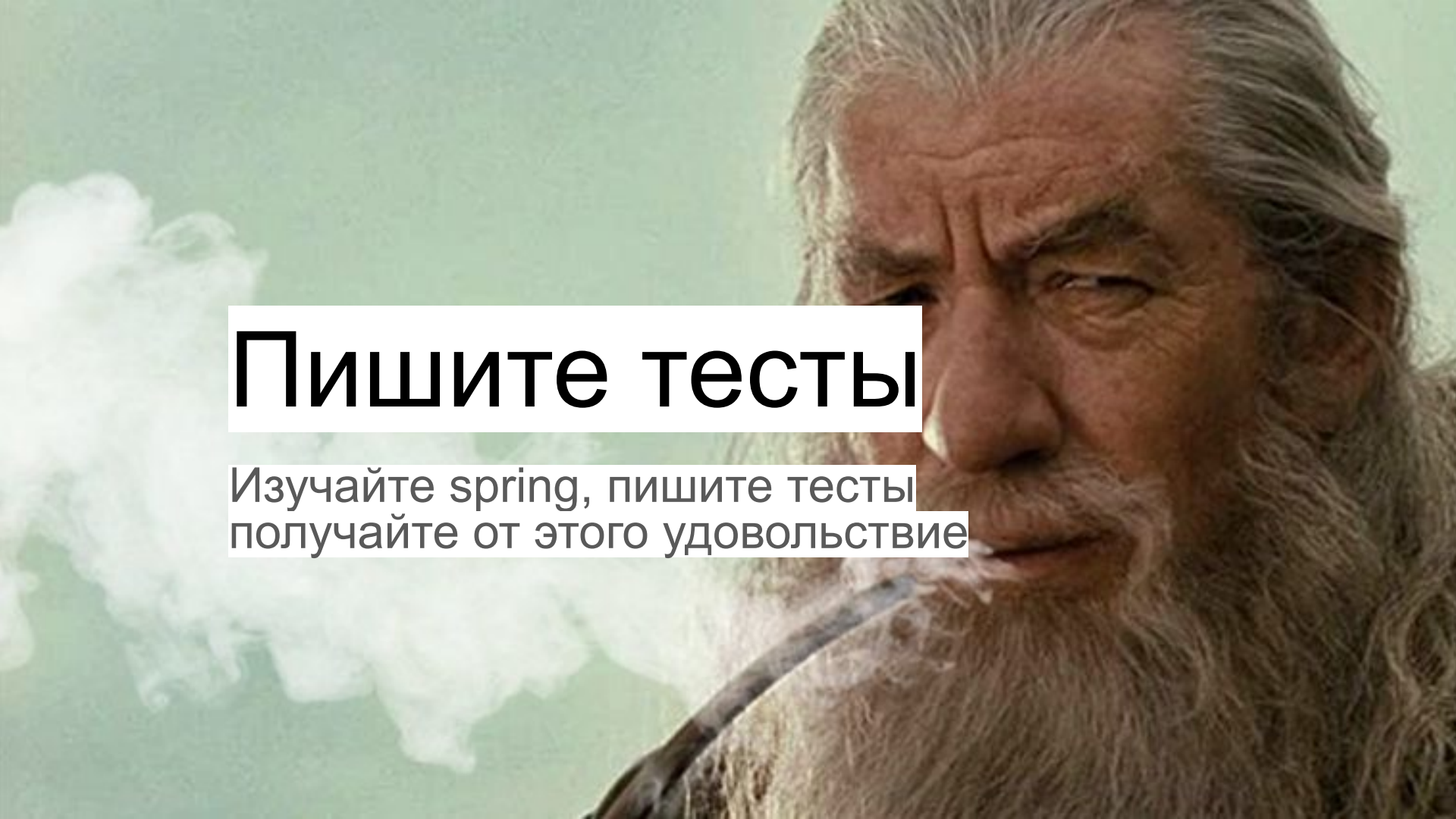


# Context Bootstrap



# А какой вывод то?

1. Лучше узнать свой фреймворк через TDD — **норм**
2. Изучать новые технологии через ~~тосты~~ тесты — **норм**
3. Писать код без оглядки на требованиями/логику/техническими ограничениями — **не норм**
4. Можно ли всё делать с TDD? — **да**
5. Тесты разбирающегося человека и разобравшегося это совсем разные вещи — **да**
6. А нужно ли? — **не факт :)**



# Пишите тесты

Изучайте spring, пишите тесты
получайте от этого удовольствие

# Закон конвея для тестов

- Архитектура ваших тестов отражается на архитектуре вашего приложения. И наоборот