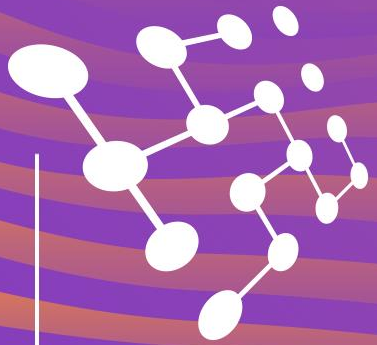


ISTA 2024

Dare. Explore. Change

Effective Unit and Integration Testing

Petyo Dimitrov | Senior Software Architect

Agenda

What is a good test?

Anti-patterns

How to improve your tests?

What is a good test?

Provides Confidence

ISTD 2024

Quick Feedback

ISTD 2024

Robust

Maintainable

Anti-patterns and fixes

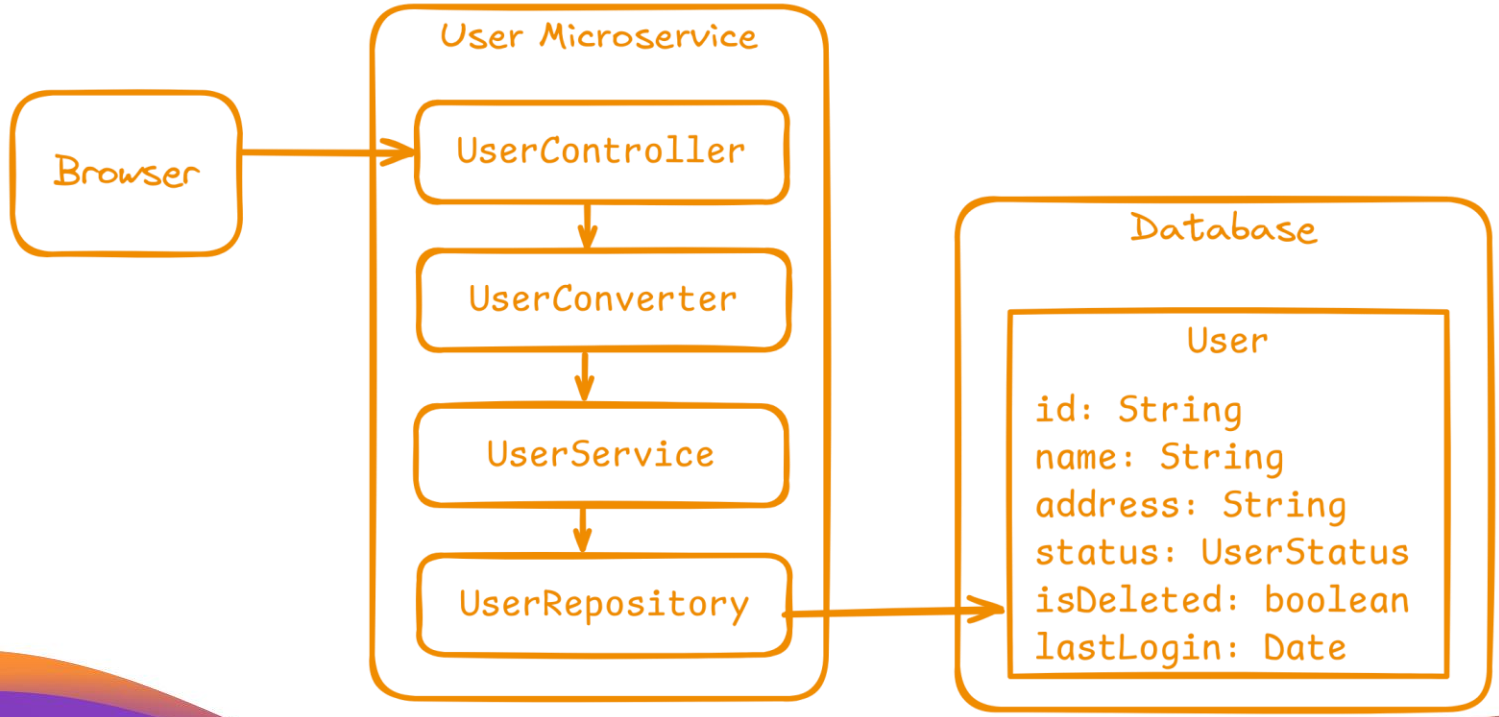
Test structure

```
def "test method name"() {
    given: "prepare"
    ...
    when: "stimulus"
    ...
    then: "assertion"
    ...
}
```

Specific syntax

```
def user = Mock(User)

user.name >> "Johnny Bravo"

user.isDeleted == false

1 * converter.convert(user)

0 * converter._
```

Context


User Microservice
Browser
UserController
UserConverter
UserService
UserRepository
Database
User
id: String
name: String
address: String
status: UserStatus
isDeleted: boolean
lastLogin: Date

Anti-pattern #1:
Mocking overuse

```
def user = Mock(User)
user.name >> "Johnny Bravo"
user.address >> "Sofia"
user.status >> UserStatus.Active
user.isDeleted >> false
user.lastLogin >> new Date()
```

Use real objects

```
def johnny = new User("Johnny Bravo",
                      "Sofia", UserStatus.Active)

johnny.isDeleted = false
johnny.lastLogin = new Date()
```

Invest in test data

```
def johnny = TestData.JOHNNY

def littleSuzy = TestData.littleSuzy()
```

Object Mother pattern

```
UserMother.johnny()
AddressMother.sofia()

UserMother.johnny()
   .withAddress(
       AddressMother.sofia().build()
   ).build()
```

Anti-pattern #2:
Repetition

```
def "user conversion works"() {
  given:
    def input = new User("Johnny Bravo",
                         "Sofia", UserStatus.Active)
  when:
    def result = converter.convert(input)

  then:
    result.name == "Johnny Bravo"
    result.address == "Sofia"
    result.status == Active
}
```

```
def "user conversion works"() {
  given:
    def input = new User("Johnny Bravo",
                         "Sofia", UserStatus.Active)
  when:
    def result = underTest.convert(input)

  then:
    result.name == input.name
    result.address == input.address
    result.status == input.status
}
```

```
def "user conversion works"() {
  given:
    def input = new User("Johnny Bravo",
                         "Sofia", UserStatus.Active)
  when:
    def result = underTest.convert(input)

  then:
    Utils.matchProperties(result, input,
                  "name", "address", "status",...)
}
```

Anti-pattern #3:
Testing the implementation

```
def "user service can create user"() {
  given:
    def user = UserMother.johnny().build()
    def userForm = new UserForm(user.name, ...)

  when:
    def result = underTest.createUser(userForm)

  then:
    1 * otherService.isNameOK(_ as UserForm) >> true
    1 * userRepository.persist(_ as User) >> user
    0 * userRepository._
    result == user
}
```

```
def "user service can create user"() {
  given:
    def user = UserMother.johnny().build()
    def userForm = new UserForm(user.name, ...)
    // name matches rules
    otherService.isNameOK(_) >> true
    // user is persisted
    userRepository.persist(_) >> user

  when:
    def result = underTest.createUser(userForm)

  then:
    0 * userRepository._
    result == user
}
```

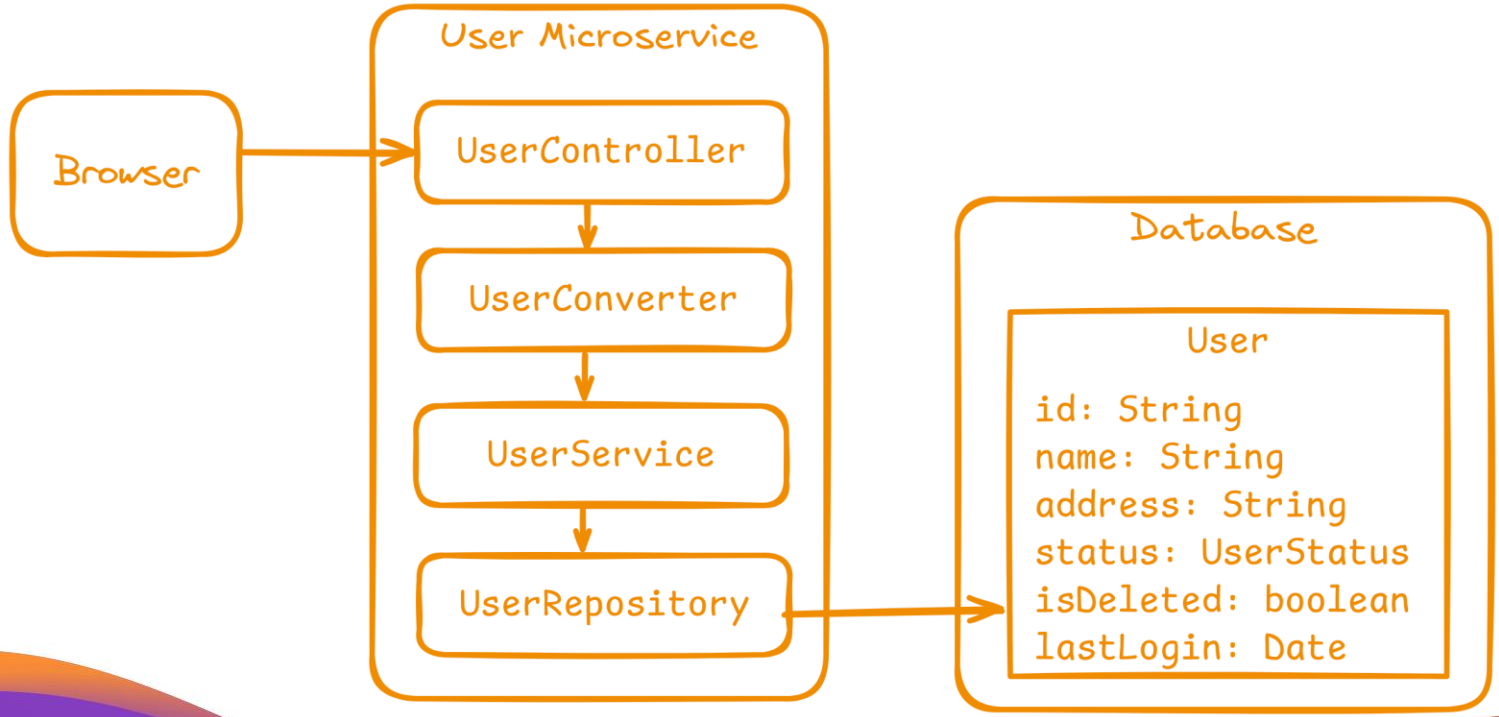
Anti-pattern #4:
Ineffective tests

```
@PostMapping
@ResponseBody
public UserDto create(@RequestBody @Valid UserForm form){

  User user = userService.create(form);

  return userConverter.convert(user);

}
```

```
def "controller calls the service and converter"() {
  given:
    def user = UserMother.johnny().build()
    def userForm = new UserForm(user.name, ...)
    userService.create(_) >> user
    userConverter.convert(_) >> ...

  when:
    def result = underTest.create(form)

  then:
    result.name == user.name
    result.address == user.address
}
```

Don't write unit tests!
Do write integration tests!

A refresher


User Microservice
Browser
UserController
UserConverter
UserService
UserRepository
Database
User
id: String
name: String
address: String
status: UserStatus
isDeleted: boolean
lastLogin: Date

```
def "user is created, updated and deactivated"() {
  given:
    def user = UserMother.johnny().build()

  when:
    def createdUserDto = /* trigger create REST API */

  then:
    createdUserDto.name == user.name
    userRepository.findByName(user.name) != null
  ...
```

```
when:
  user.lastLogin = now().minusDays(30)
  def updatedUserDto = /* trigger update REST API */

then:
  updatedUserDto.name == user.name
  updatedUserDto.lastLogin == user.lastLogin
```

```
when:
  jobsUtil.triggerUserPeriodicJob()
  def userDto = /* trigger read REST API */

then:
  userDto.name == user.name
  userDto.status == UserStatus.Inactivate
```


Don't write
integration tests!
Do write unit tests!

```
@Unroll
def "update changes status from #currentStatus to #newStatus"() {
  given:
  def user = new User(status: currentStatus)
  when:
  user.updateStatus(newStatus)
  then:
  user.status == newStatus
  where:
  currentStatus          | newStatus
  UserStatus.Inactive    | UserStatus.Active
  UserStatus.Inactive    | UserStatus.Deleted
  ...
```

Summary

Avoid mock overuse

Don't Repeat Yourself

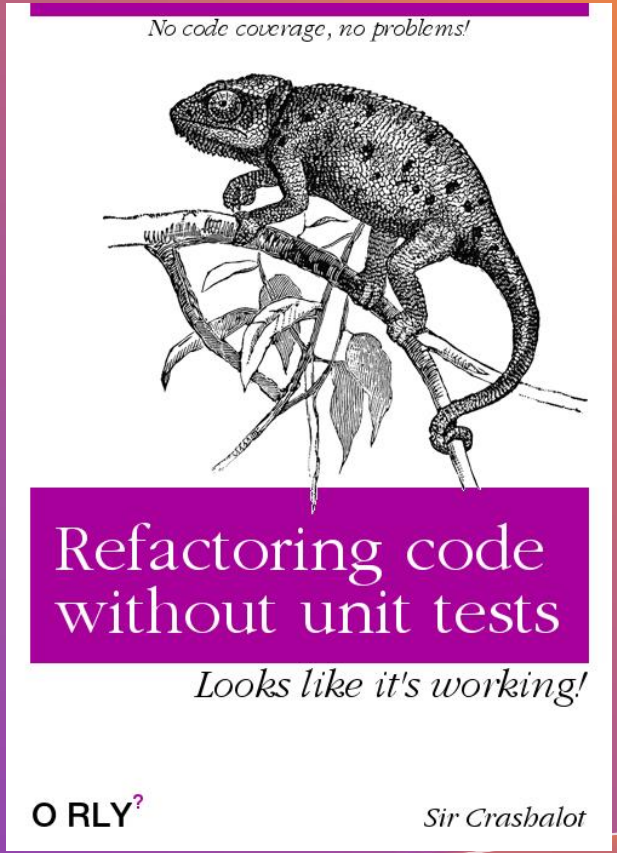
Test contracts, not implementations

Don't test for the sake of testing

Distribute tests across the test pyramid

To me, legacy code is simply code... without tests

Michael Feathers
Working Effectively With Legacy Code



Thanks!

petyo.dimitrov@gmail.com
petyo.dimitrov@musala.com

www.linkedin.com/in/petyo-dimitrov