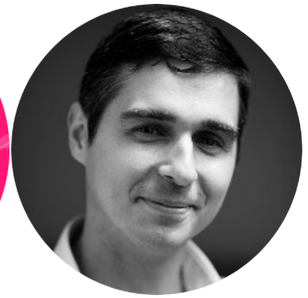
TDD Nuances and Best Practices



Andrejs Doronins
Software Developer in Test

Overview

TDD is

- not always easy to apply
- not a silver bullet

TDD and mocking

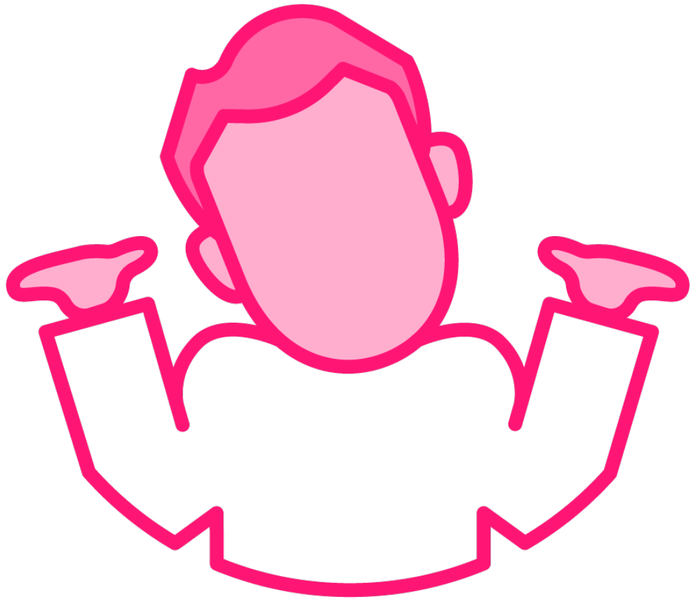
Test code anti-patterns

Is TDD impossible to apply somewhere?

Is TDD ~~impossible~~ difficult to apply somewhere?

Yes - legacy systems

Legacy System Definitions



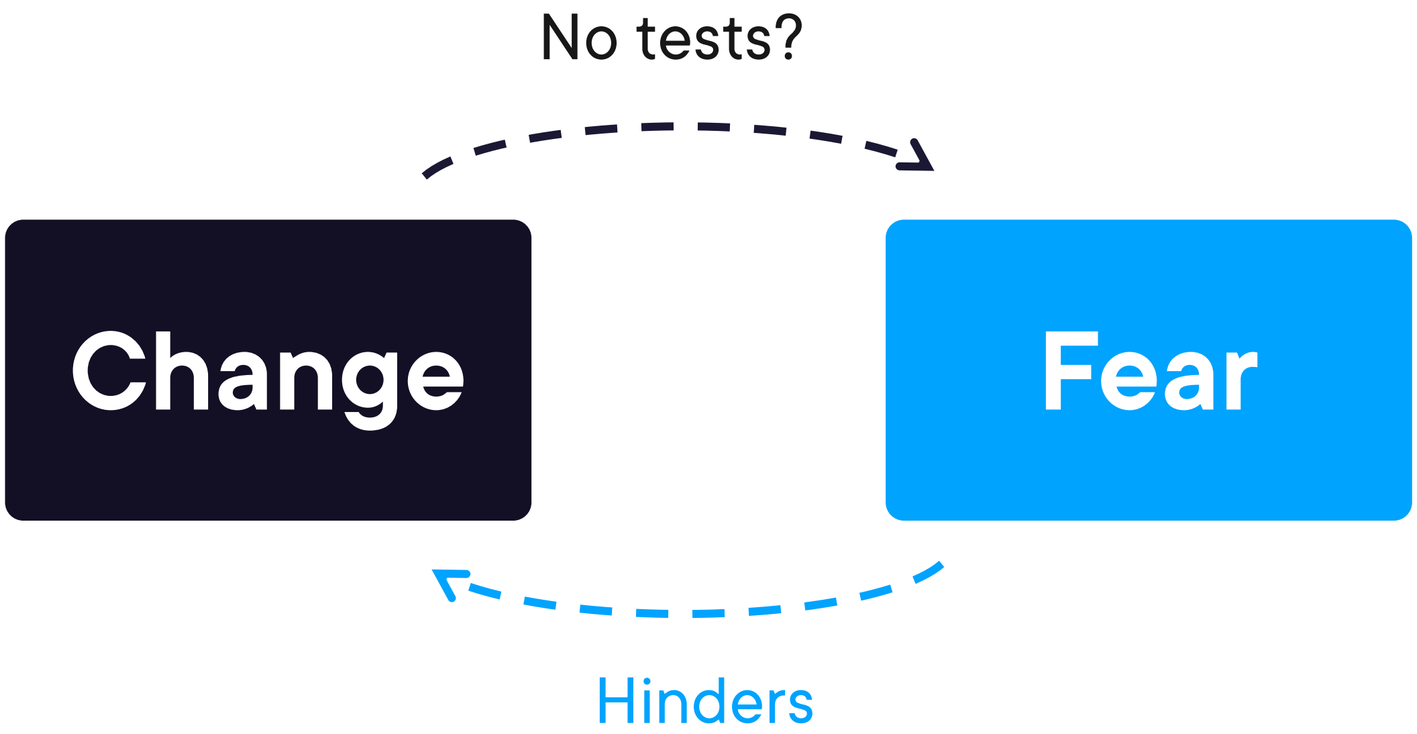
Written decades ago & written in an obsolete language

Inherited software

Working Effectively with Legacy Code

Book definition:

- any code or software that is not covered by tests


No tests?
Change
Fear
Hinders

Code without tests [...] not designed for testability.

Introduce tests [...] need to change the code.

Change the code with confidence [...] need to introduce tests.

[...] chicken-and-egg problem.

"Working effectively with legacy code", Michael Feathers

Working Effectively with Legacy Code

Principles and guidelines:

- I Don’t Understand the Code Well Enough to Change It
- My Application Has No Structure
- I Need to Change a Monster Method and I Can’t Write Tests for It
- How Do I Know That I’m Not Breaking Anything?

Characterization tests

CTDD

1) Assume current behavior is correct

2) Write a test for the current behavior. Let it fail.

3) Change the test to make it pass.

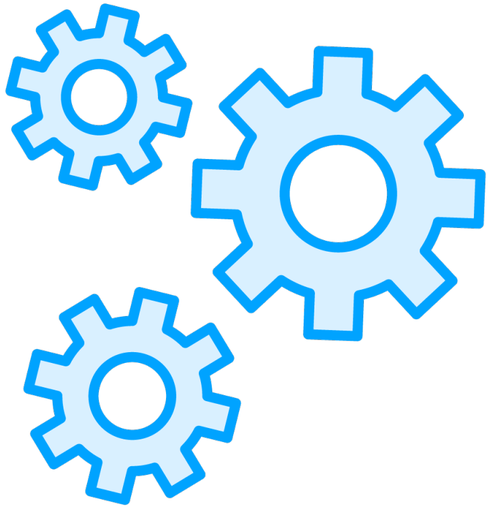
4) Repeat

5) Change code with confidence

More Information

Unit Testing Legacy Code in Java SE 8

Jim Weaver



System Challenge

People Challenge

Convincing the Team

TDD has a learning curve

Needs justification

To convince:

- lead by example
- organize workshops
- knowledge transfers
- active support


SILVER

TDD

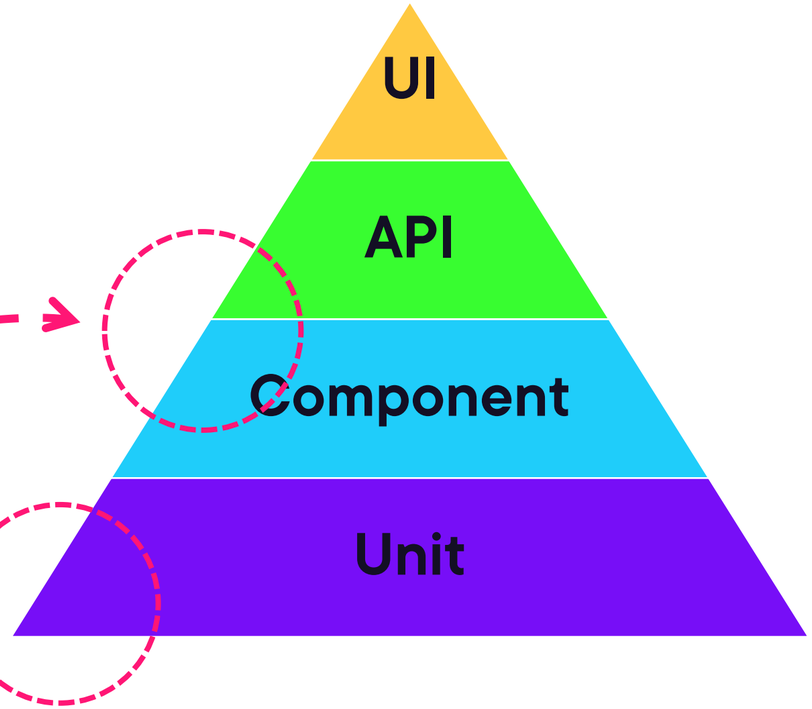
TDD helps produce well-designed, well-tested, and well-factored code in small, verifiable steps

Book: The Art of Agile Development

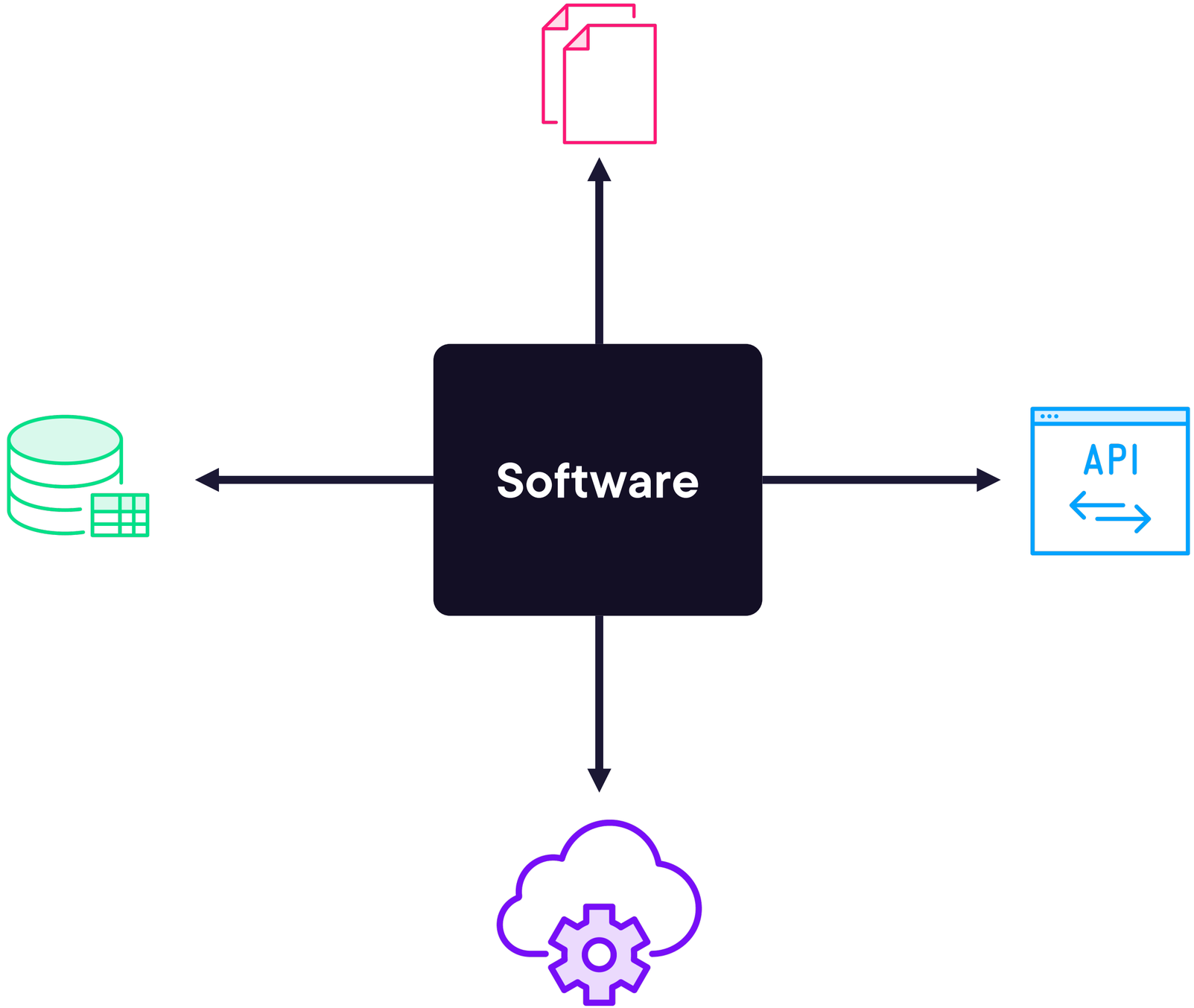
```
class HugeThingService {

    void doMisleadingThing(a,b,c,d,e) {

    }



}
```

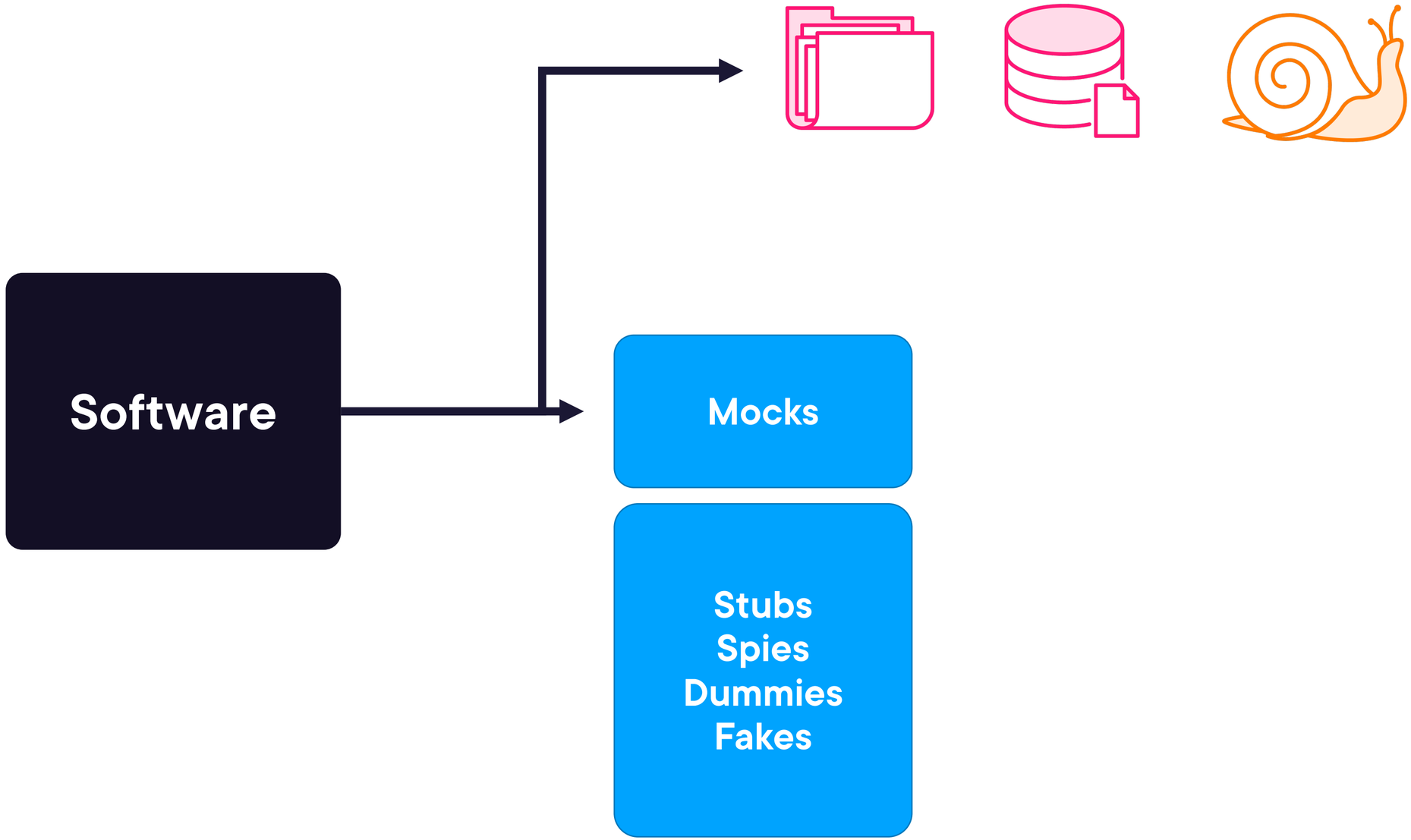
~~UTDD~~

TDD



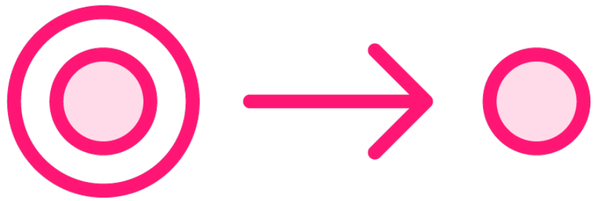
+ system
+ performance
+ security
+ compliance
+ accessibility
+ usability


Software
API


Software
Mocks
Stubs
Spies
Dummies
Fakes

```
Mockito.when(action).thenReturn(thing);
```

Mock Framework Courses



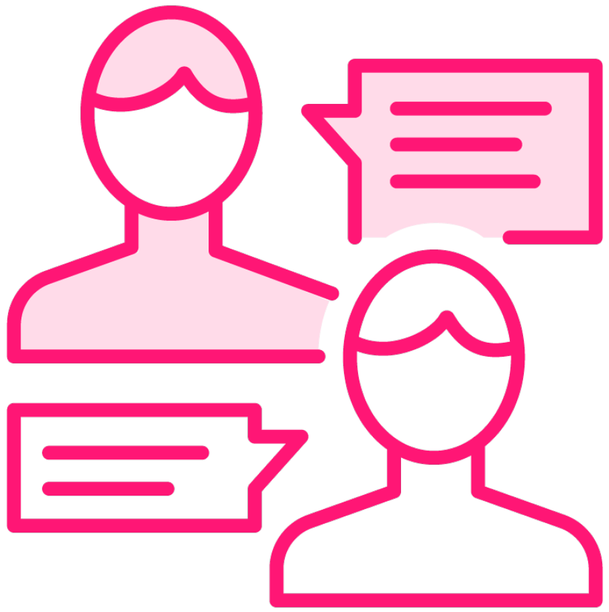
Getting Started with Mockito 2

Getting Started with EasyMock 4

Tests are a big cost!
(And they break)

I believe you, but...

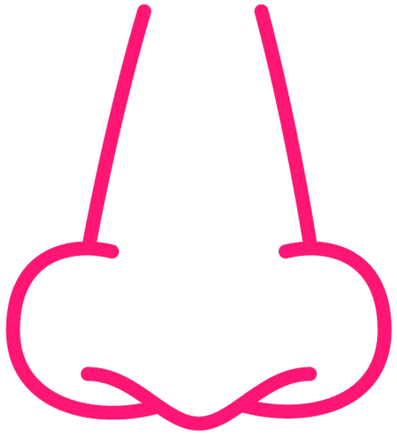
Criticizing TDD or Automated Tests?



Tests cost regardless of TDD

Most of the cost comes from poor test code practices

Code Smells

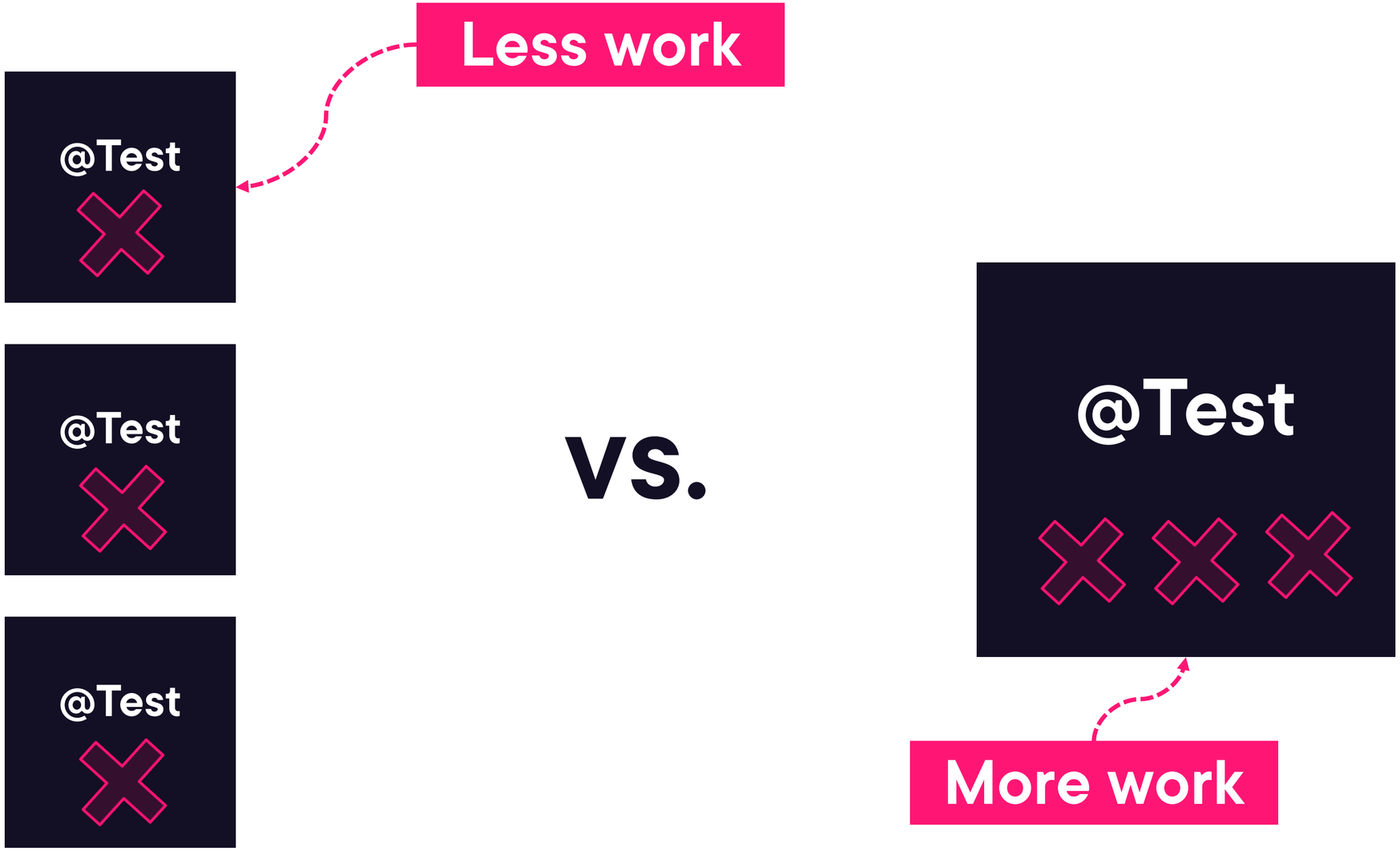


Poor names

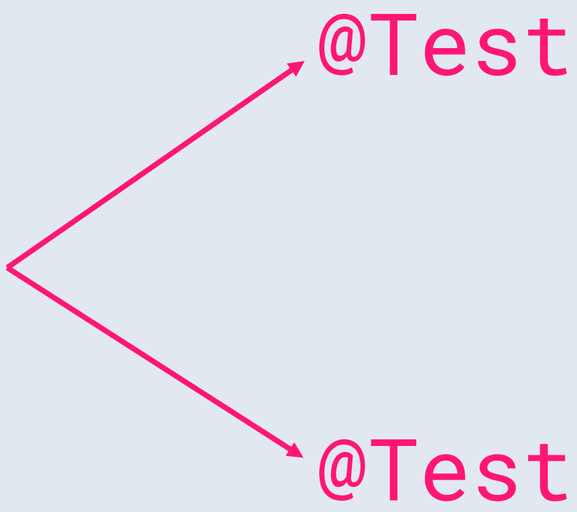
Methods with too many parameters

Methods too big

Test code has its own anti-patterns!


Less work
@Test
@Test
@Test
vs.
@Test
More work

```
@Test
void testSomething() {
  if (...) {
     for(...) {
     // test one thing
     }
  } else {
     // test another thing
     }
   }
```



Summary

Doing TDD well if you:

- Code in tiny steps
- Find programming mistakes in minutes, not hours
- Refactor both production and test code
- Spend little time debugging

Further Learning

Fundamentals of Test Automation in Java

Java Refactoring: Best Practices

Unit Testing Legacy Code in Java

Writing Highly Maintainable Unit Tests

Getting Started with Mockito 2

Getting Started with EasyMock 4

Rating

Thank you!

(Happy TDDing)