Avoiding Common Test Anti-patterns

Andrejs Doronins
TEST AUTOMATION ENGINEER

Anti-pattern

Represents common bad practices and pitfalls

Patterns vs. Anti-patterns

Patterns	Anti-patterns
Good examples	Bad examples
Do things the right way	Learn to recognize them
	Improve them

Recognizing Anti-examples

Avoid doing them yourself

Fix them when you encounter them

Anti-pattern Types

Common

Specific to a layer:

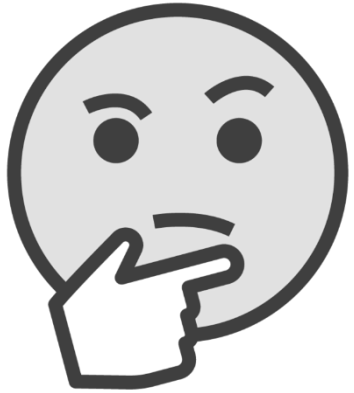
- Unit
- Integration
- UI

Specific to certain technologies, e.g. Web Services or Databases

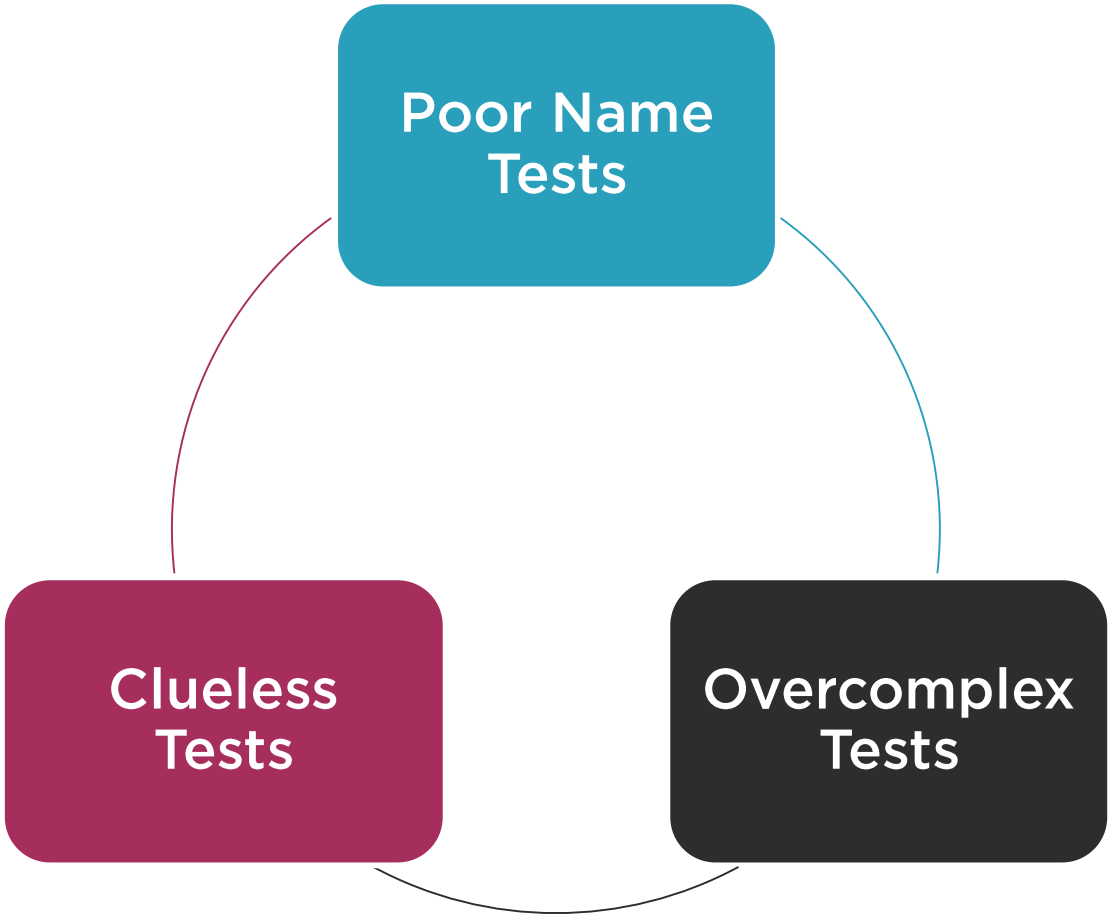
Criteria:

- ✓ Frequency
- ✓ Impact

Most common **universal** anti-patterns?



The Wicked Testing Trio



How hard can it be to give a test a good name?

You'd be surprised...

```
@Test
searchFails(x);
searchFailsInvalidInput(x);
searchRejectsInvalidInput(x);
```

Fails? Why?

Mirrors requirements

Poor Name Test

Needs (much) more time to just understand what it is about

Is the test failing because of a real bug or there is a problem with the test itself?

A clear name gives you a head start

If it’s not clear what the test is verifying, then its value is not clear either


?
About
Stuff

Requirements

@Test

Max string length of 20 characters

Search must only accept alphanumeric characters

The exception is that it should allow quotation marks

Reject input that contains not allowed characters and display a message

```
@Test
searchFails(x);
searchFailsInvalidInput(x);
searchRejectsInvalidInput(x);
searchRejectsGivenInputWithInvalidCharacters(x);
```

Failure Reasons

We pass in a valid string (verify the test data)

Broken functionality (for a very specific reason)

A test name should ideally reveal the reason why it would fail

Now excludes
“max length” criteria

```
@Test
searchRejectsGivenInputWithInvalidCharacters(x);

…InputWithInvalidCharactersOrLengthTooLong(x); ?
```

Something’s
wrong…

Clueless Test

What is the point of this test?

- The answer should come without much effort

Why did this test just fail?

- Unit: one clear answer
- Higher-level: ideally also only one*

***Excluding environment issues**

Clueless Test

Answer should match a simple pattern:

- If we send input <A>, then the system should do <B>
- BDD: Given X, When we Y, Then Z

This test verifies this... and that...
and also the other thing...

Jack of all trades,
master of none test

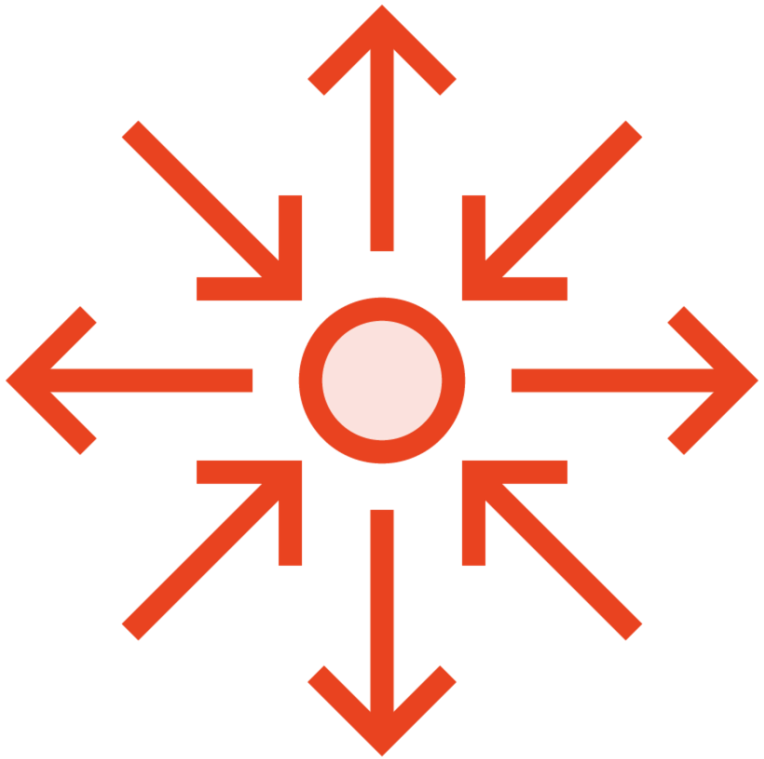
Clueless Test Signs

The test is clueless if it:

- Has a descriptive name, but contains “and”, “or”
  - Does multiple checks*

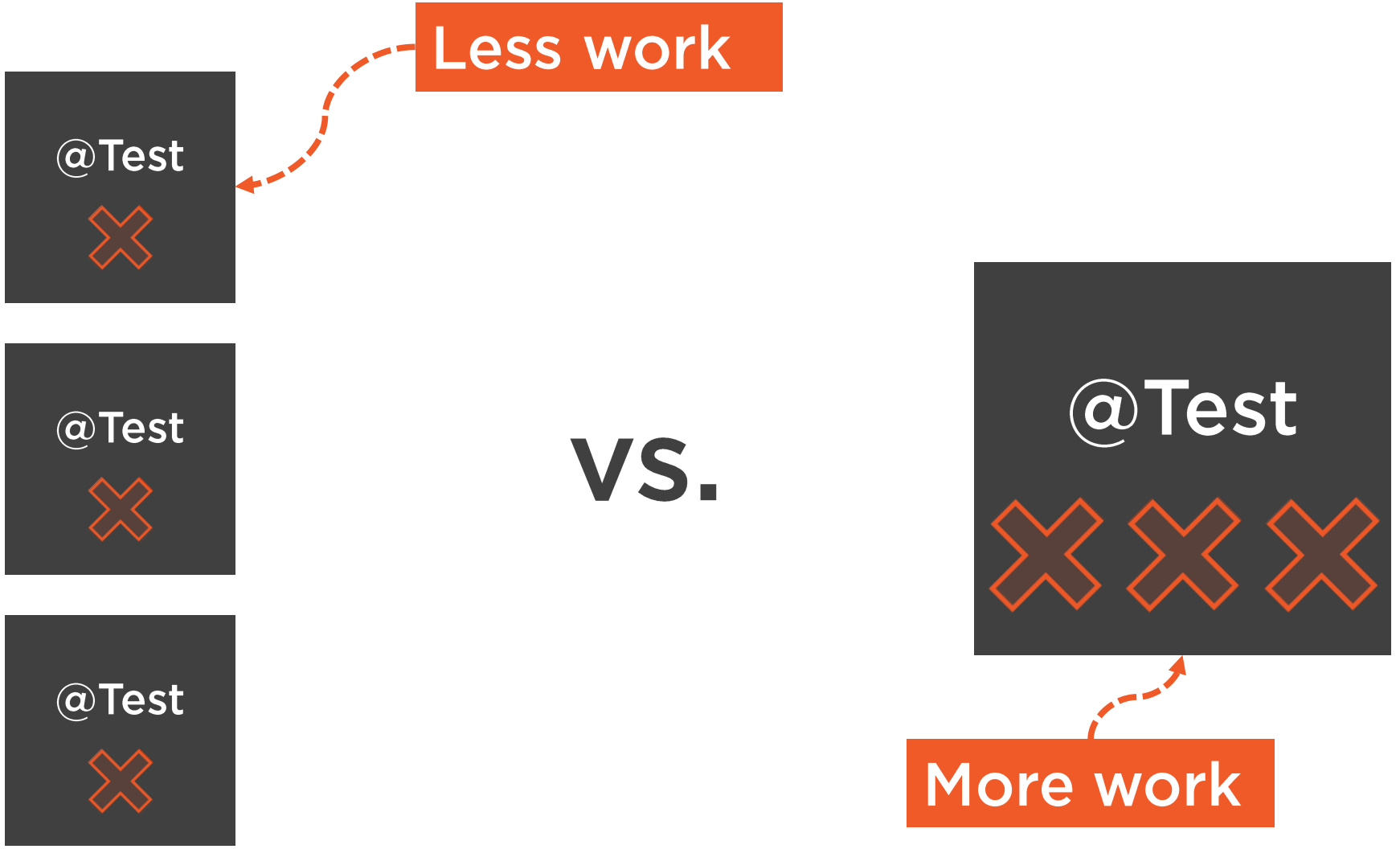
***Excluding complex integration and end-to-end tests**

Clueless Test Downsides



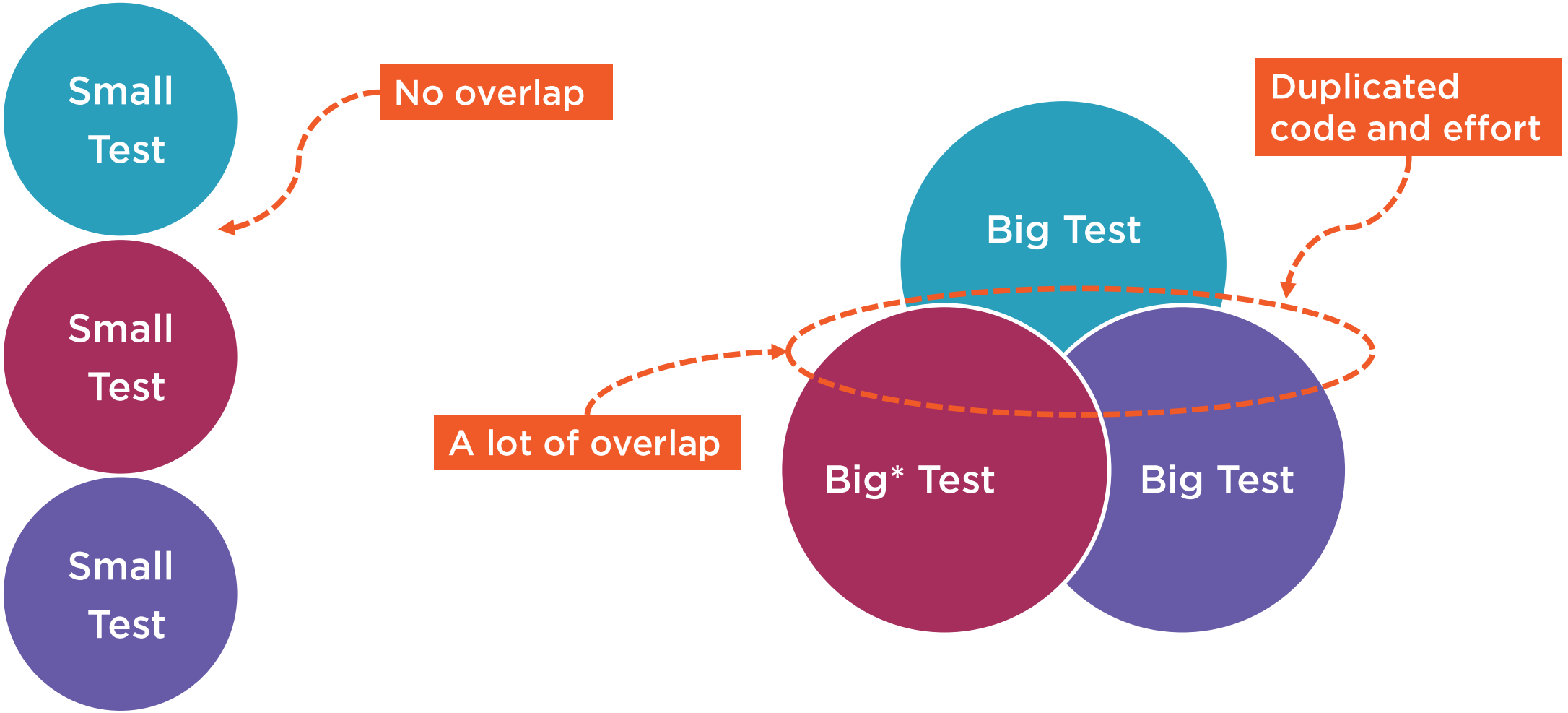
Introduces additional “Points of Failure” (PoF), i.e. reasons to fail

Tests eventually overlap in verification responsibility


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

Why squeeze multiple checks into one test?

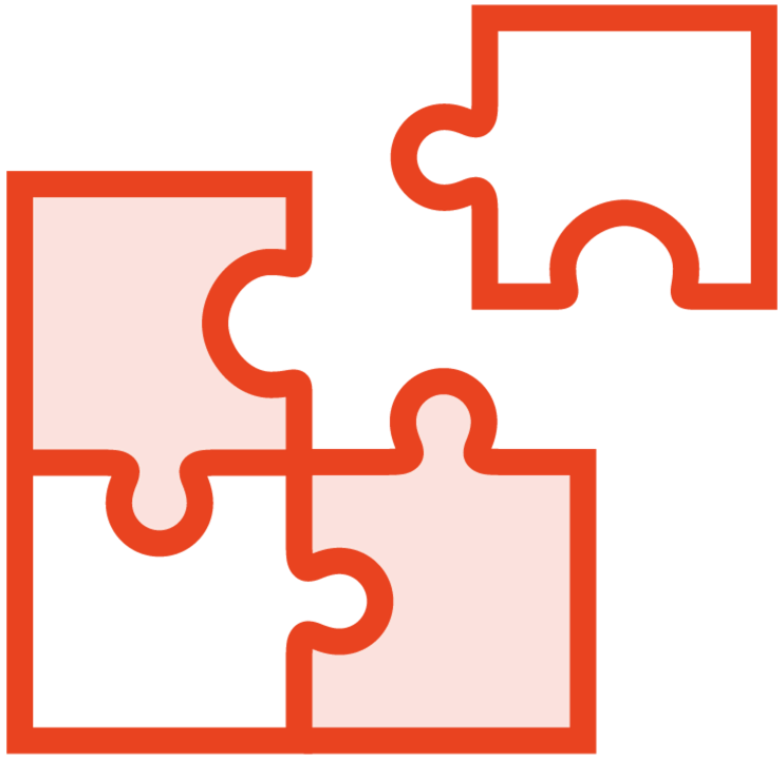
Well, we have all this setup and act code, so we might as well add one extra innocent check



*Here: a clueless test that verifies multiple things

Many small, focused tests, each with a clearly defined responsibility is good

Few large, clueless tests that have a responsibility of verifying a bit of “this and that” is bad

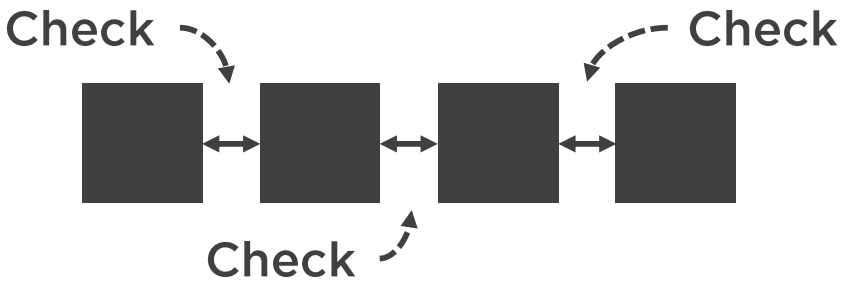


Complex integration and E2E tests:

- Acceptable to have multiple checks

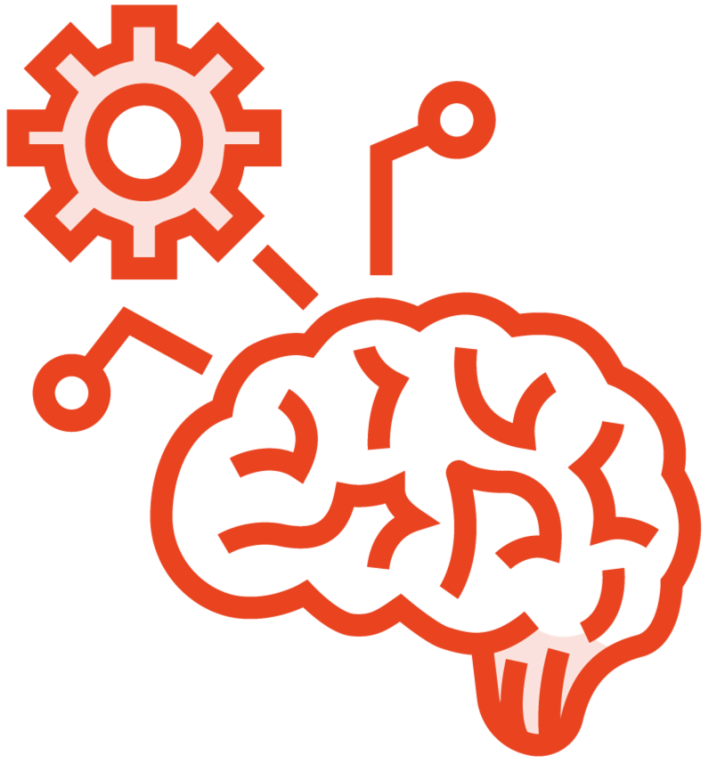
Why?

- To narrow down where the problem is



```
@Test
searchRejectsGivenInputWithInvalidCharactersOrLengthTooLong(x);

searchRejectsGivenInputWithInvalidCharacters(x);

~~searchRejectsGivenInputTooLong(x);~~
~~searchRejectsGivenInputTooLongOrTooShort(x);~~
searchRejectsGivenInputOfInvalidLength(x);
```

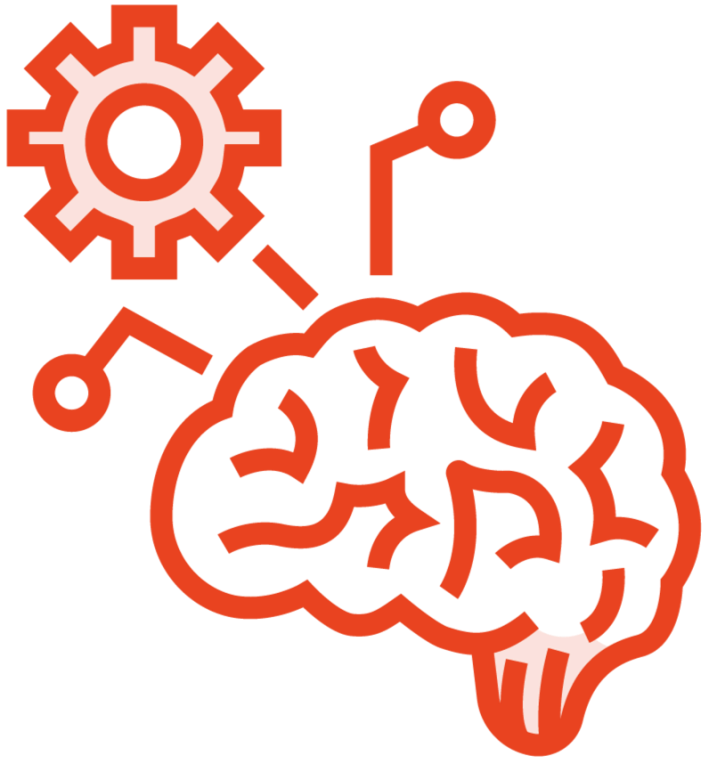
What Is Not Meant by “Complex Test”



An integration or an end-to-end (E2E) test. They are naturally more complex than unit tests

A test verifying complex logic of a sophisticated SUT

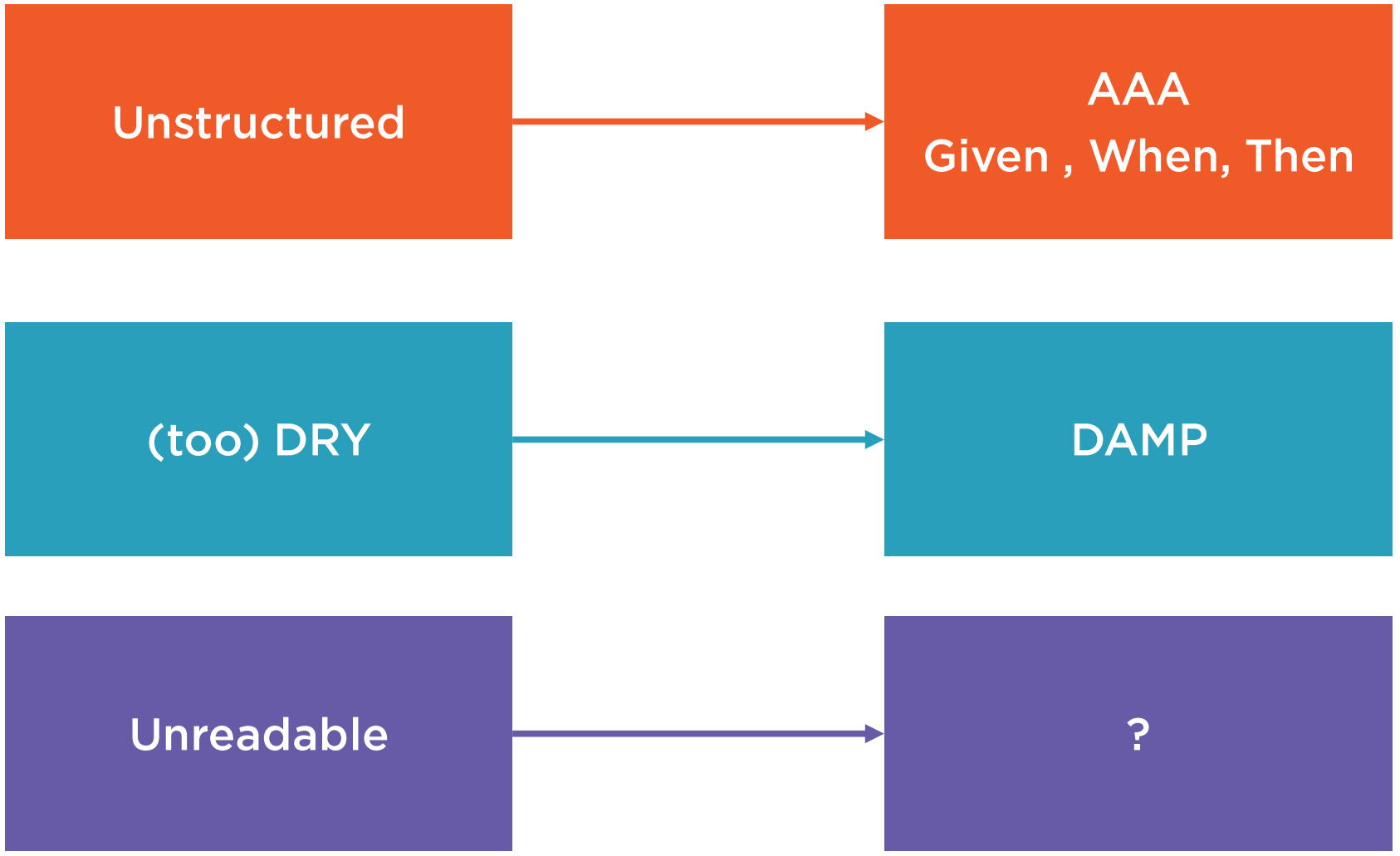
What Is Meant by “Complex Test”



More complex than necessary

- Unstructured
- Too DRY
- Unreadable, or more difficult to read than necessary

Such issues cumulatively consume hours and days unnecessarily


Unstructured
AAA
Given , When, Then
(too) DRY
DAMP
Unreadable
?

Making Tests Readable

Consider applying Fluent Interface (FI)

- Import libraries built using FI
- Create your own (requires a lot of work)

Avoid branching and looping (if, switch, for)

Making Tests Readable

A test should read like a small easy-to-digest story

Tests should be “flat” (no branch nesting)

When necessary – create wrapper helper functions

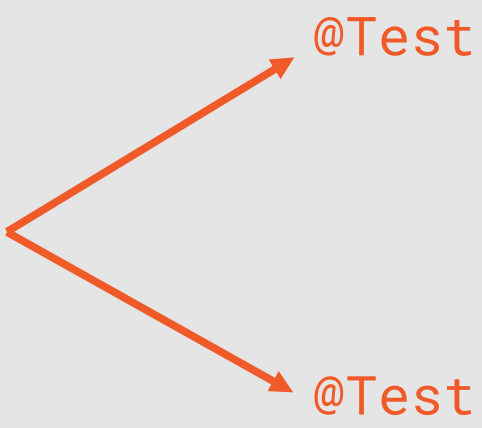
RestAssured (Web APIs)

```
given().
    param("k1", "v1").

when().
    post("/somewhere").


then().
   body(containsString("OK"))
```

AssertJ

```
assertThat(fruitList)
        .hasSize(9)
        .contains(“apple”, “pear”)
        .doesNotContain(“banana”);
```

```
@Test(dataProvider = “someInput”)
void testSomething(String s) {
    // we test quotation marks in the other branch
    if (!s.contains(“’”)) {
        // test one thing
    } else {
        // test another thing
        }
    }
```



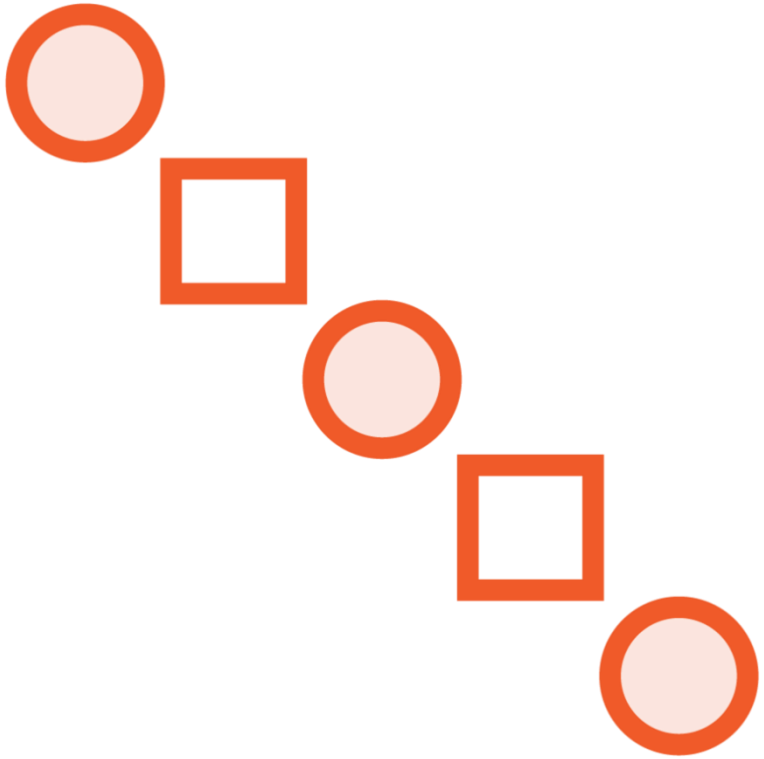
Demo

Fixing tests with anti-patterns

Lean focused tests lead to lean focused test data

Fast comprehension and investigation has priority

Other Anti-patterns

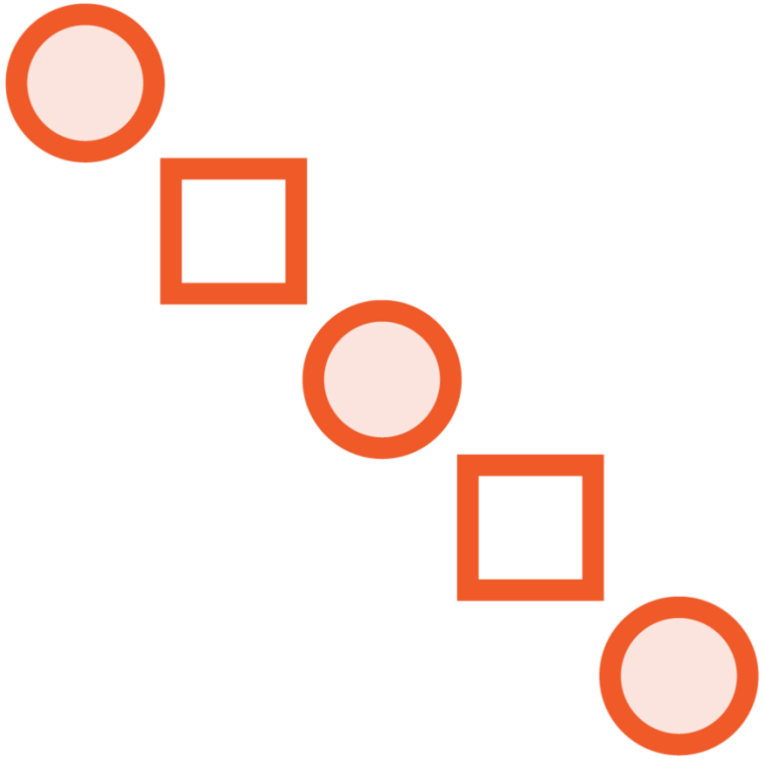


Unit test anti-patterns:

- Unit testing private methods
- Exposing private state to enable unit testing
- Leaking domain knowledge to tests
- Mocking concrete classes

Book: "Unit Testing Principles, Practices, and Patterns" by Vladimir Khorikov

Other Anti-patterns



UI test anti-patterns:

- Using Record & Playback
- Using pauses or explicit waiting

Other test anti-pattern sources:

- http://agileinaflash.blogspot.com/2009/06/tdd-antipatterns.html

It’s beneficial to learn the same thing twice but from two different perspectives

Summary

Anti-patterns exist in test automation

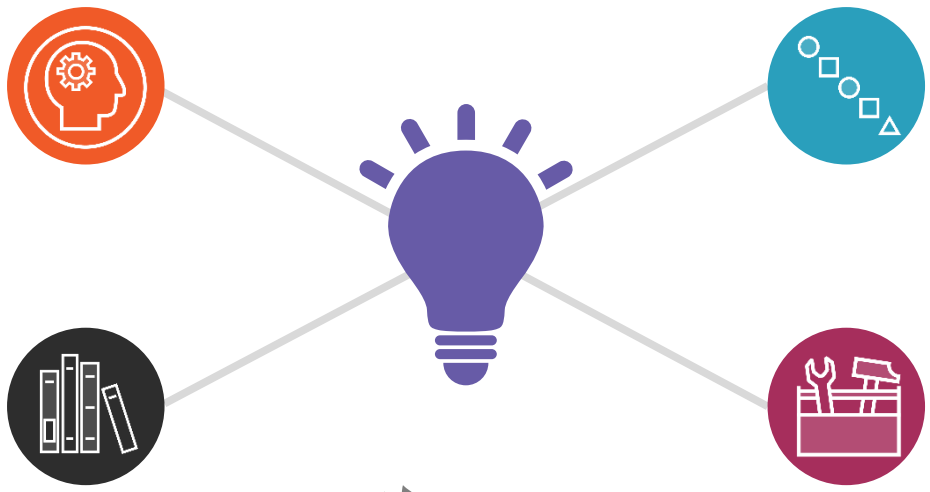
It's important to recognize flawed code in order to improve it

- Poor name tests
- Clueless tests
- Overcomplex tests

The Four Pillars

QA & Testing Skills

Clean Code, SOLID, patterns



Test Automation Principles
(this course)

Expertise in tools & frameworks

Rating

Thank you!

(Happy coding)