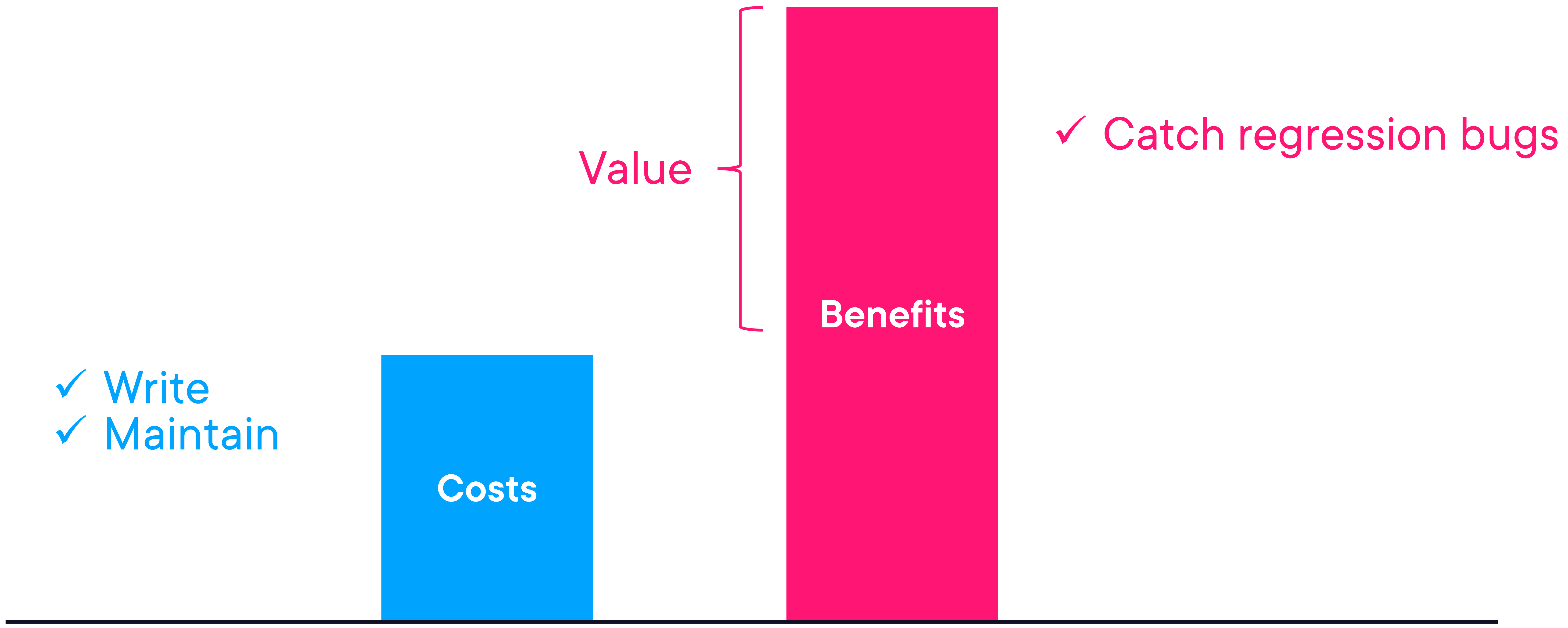
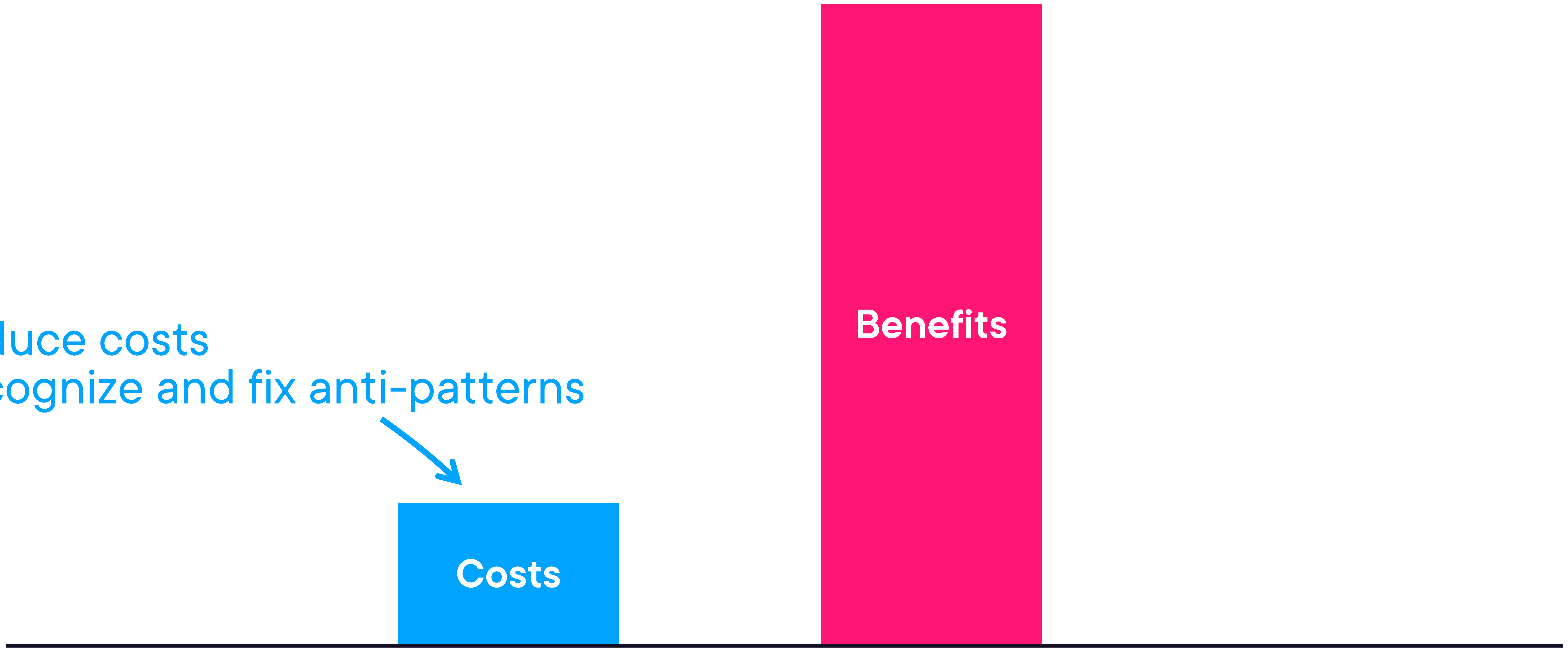
Creating Better Tests

Andrejs Doronins
Software Developer in Test


✓ Write
✓ Maintain
Costs
Value
Benefits
✓ Catch regression bugs

Reduce costs
Recognize and fix anti-patterns



Antipatterns

Most common ones

Applicable to many test types

Overview

Single test:

- Naming
- Small and focused
- DAMP principle

Multiple tests:

- Independence

Test class structure and organization

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

You'd be surprised...

Fails? Why?

```
@Test

searchFails(x);

searchFailsInvalidInput(x);

searchRejectsInvalidInput(x);
```

Coneys intent, 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?

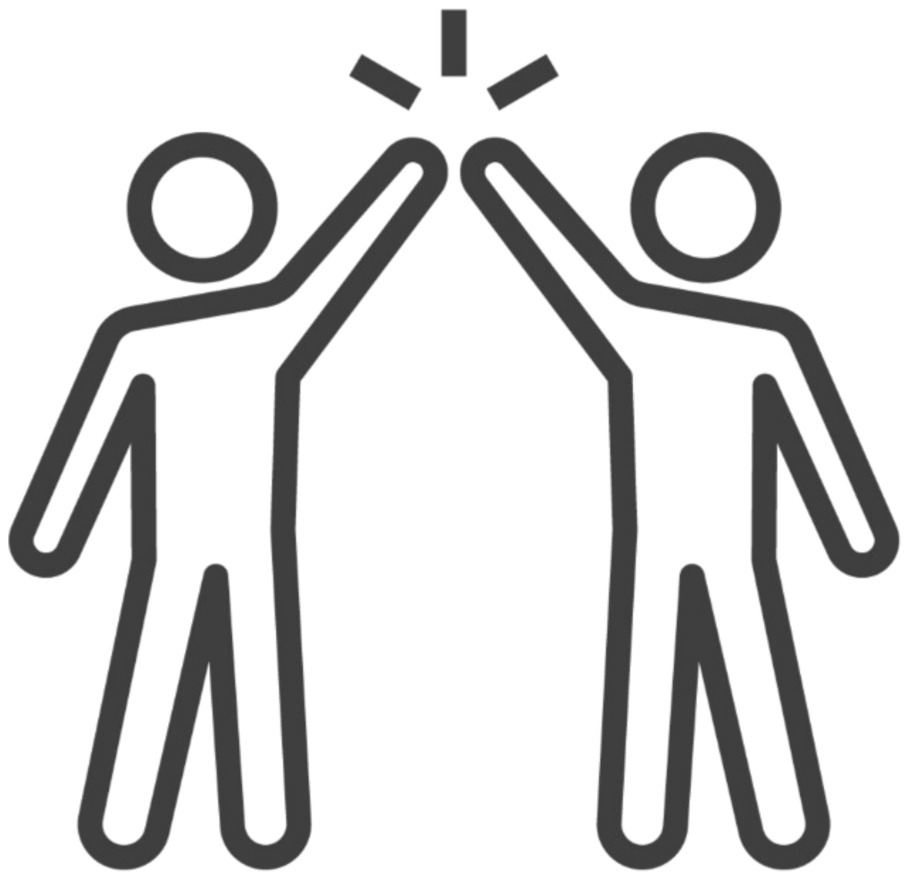
A clear name gives you a head start

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

About
Stuff
?

Poor Name Test

Found some!



@Test

Max string length of 20 characters

Search must only accept alphanumeric characters

Reject input that contains not allowed characters and display a message

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

Failure Reasons

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

Broken functionality (for a very specific reason)

Now excludes "max length" criteria

```
@Test

searchRejectsGivenInputWithInvalidCharacters(x);


…InputWithInvalidCharactersOrLengthTooLong(x); ?
```

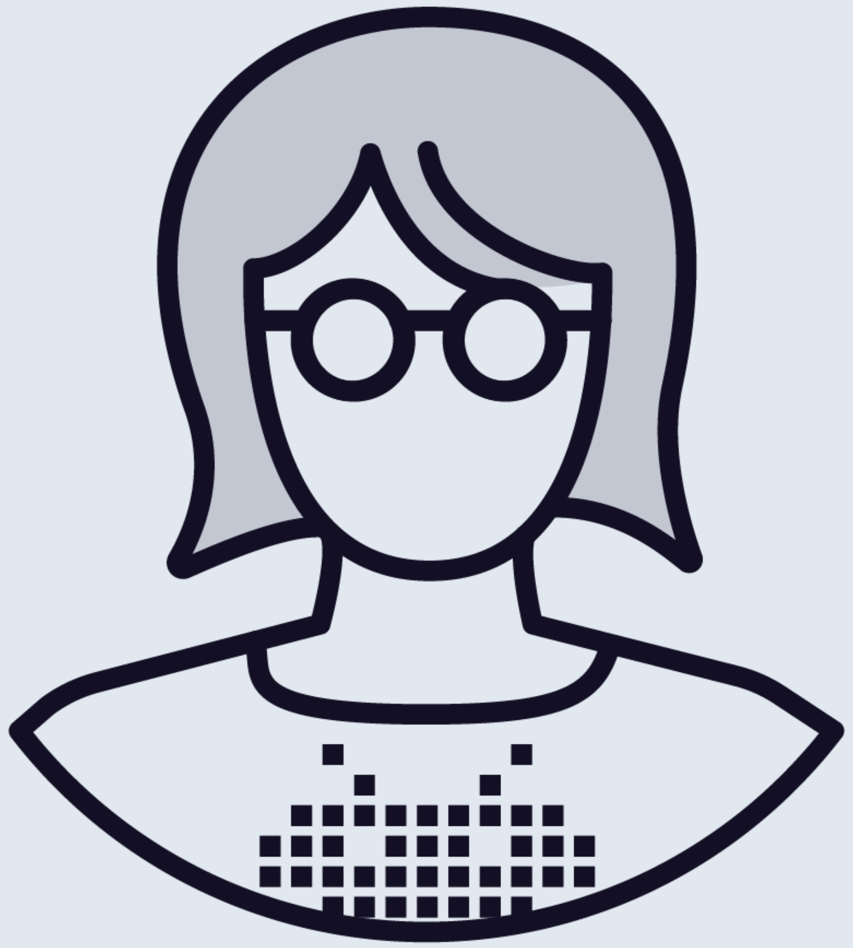
Something's wrong…

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

Does the name tell me the main reason it would fail?

Does the name tell me the main reason it would fail?

```
@Test
void testInvalidInput() {


}
```



Excludes
“max length” criteria

```
@Test

searchRejectsInputWithInvalidCharacters(x);


…InputWithInvalidCharactersOrLengthTooLong(x); ?
```

Something’s
wrong…

Tests should be small and focused

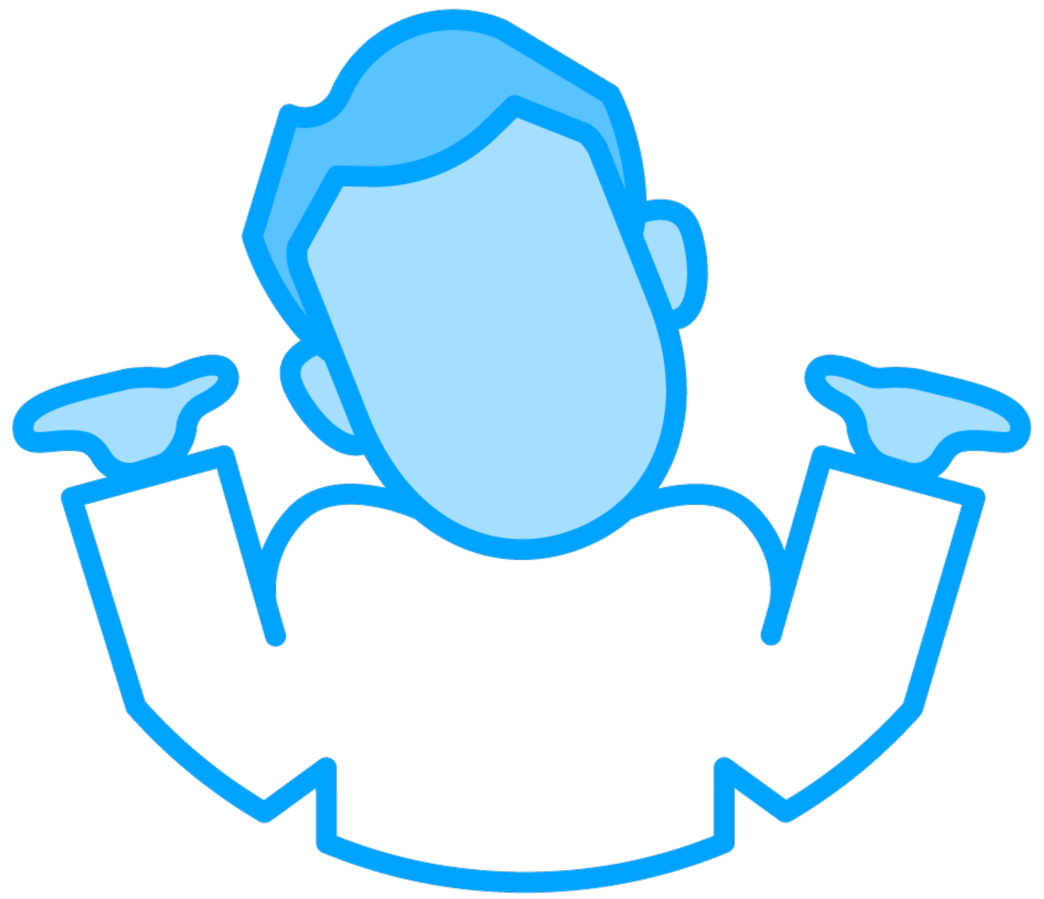
Clueless Test

What is the point of this test?

- The answer should be short and simple
- Pattern: If we send input {A}, then the system should do {B}

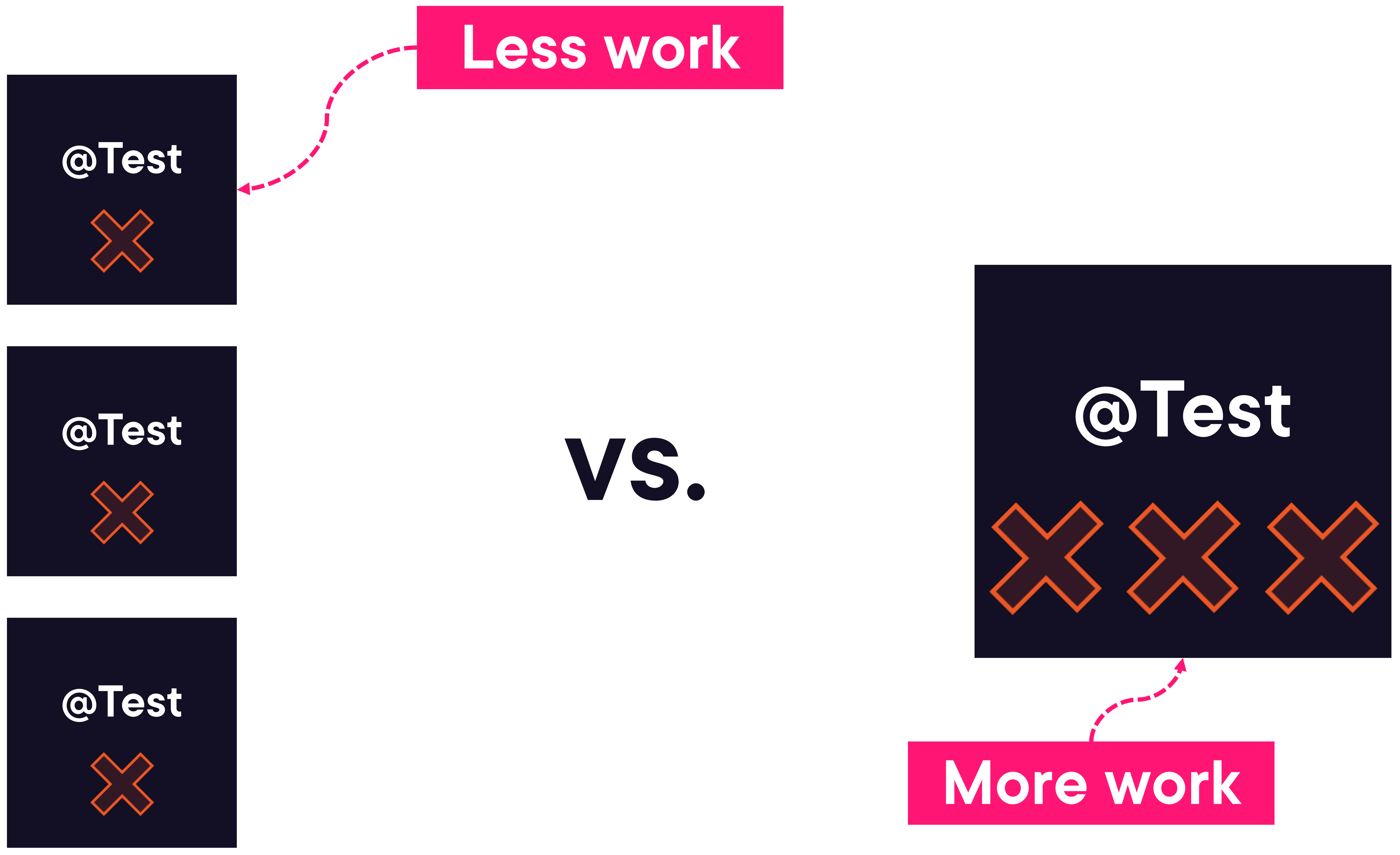
This test verifies this… and that…
and also the other thing…

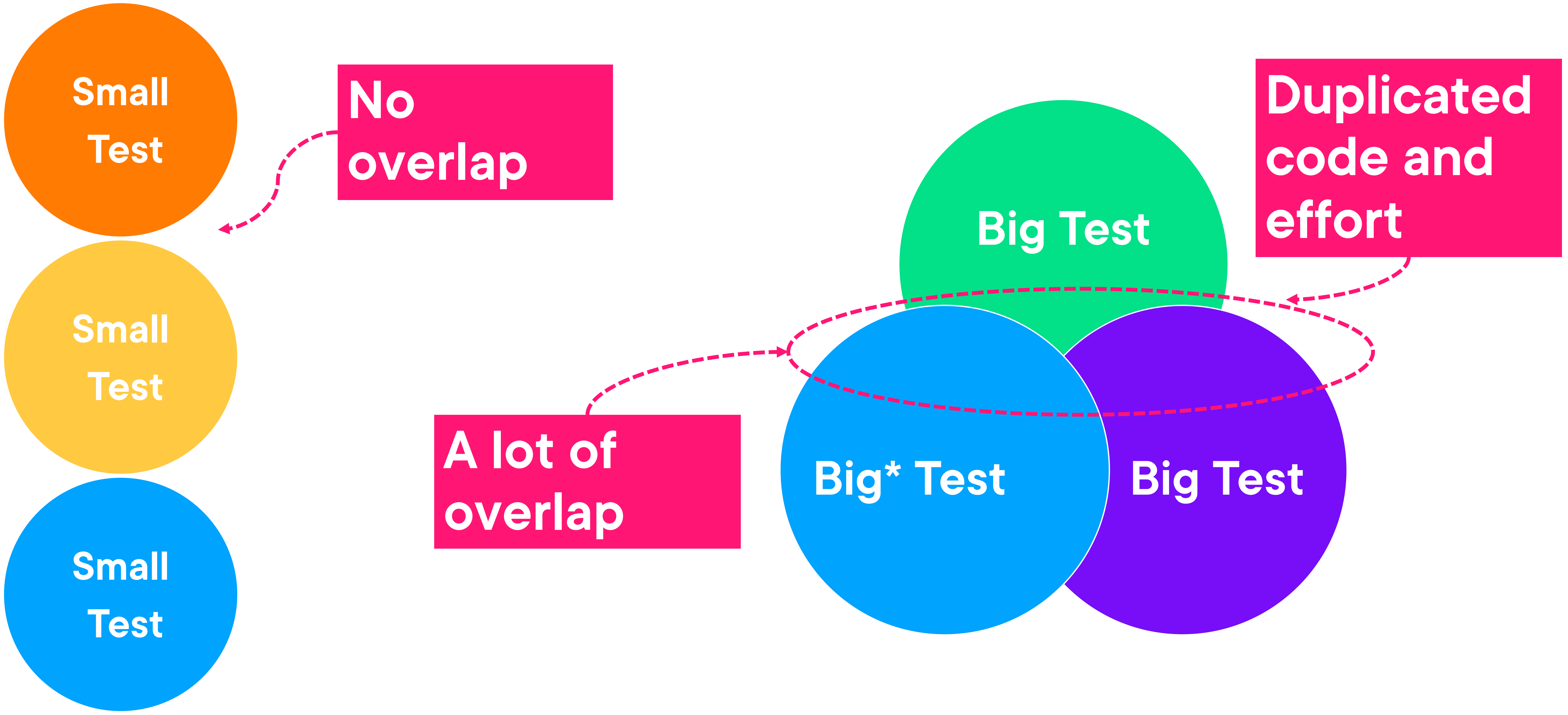
Clueless Test Downsides



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



*Here: a clueless test that verifies multiple things

```java
@Test
searchRejectsInputWithInvalidCharactersOrLengthTooLong(x);

searchRejectsInputWithInvalidCharacters(x);
```

~~searchRejectsInputTooLong(x);~~

~~searchRejectsInputTooLongOrTooShort(x);~~

```java
searchRejectsInputOfInvalidLength(x);
```

DRY

Don’t Repeat Yourself

DAMP

Descriptive and Meaningful Phrases

WET

Write Everything Twice

Can I glance through this test code and understand what it's trying to achieve?

```
@Test
void searchRejectsInvalidChars() {


}
```

DAMP Principle Tips

No loops or branching

- Looping? Can you parameterize?

No low-level code

Consider using a Fluent Interface

```
// generate random int in range of 1 to 10
int randomNum = rand.nextInt((10 - 1) + 1) + min;


// vs.
int randomNum2 = getRandomInt(1, 20);
```

Java Streams

```
someList.stream()
    .filter(...)
    .map(transform)
    .sorted()
    .collect(...);
```

AssertJ

```
assertThat(passengerList)
    .hasSize(50)
    .contains(“Smith”, “Evans”)
    .doesNotContain(“Miller”);
```

RestAssured

```
RestAssured.get("api/url…")
    .then()
    .assertThat()
        .statusCode(200)
    .and()
        .contentType(ContentType.JSON);
```

Custom Fluent Interface

```
search.act()
    .selectTab(Tab.COURSES)
    .selectCourse("Java Fundamentals: The Java Language");

course.verify()
       .freeTrialIsDisplayed()
       .coursePreviewIsDisplayed();
```

NullPointerException

```java
String s1 = null;
s1.toUpperCase();
```

```
Exception in thread "main" java.lang.NullPointerException
```

NullPointerException

```
String s1 = null;
s1.toUpperCase();
```

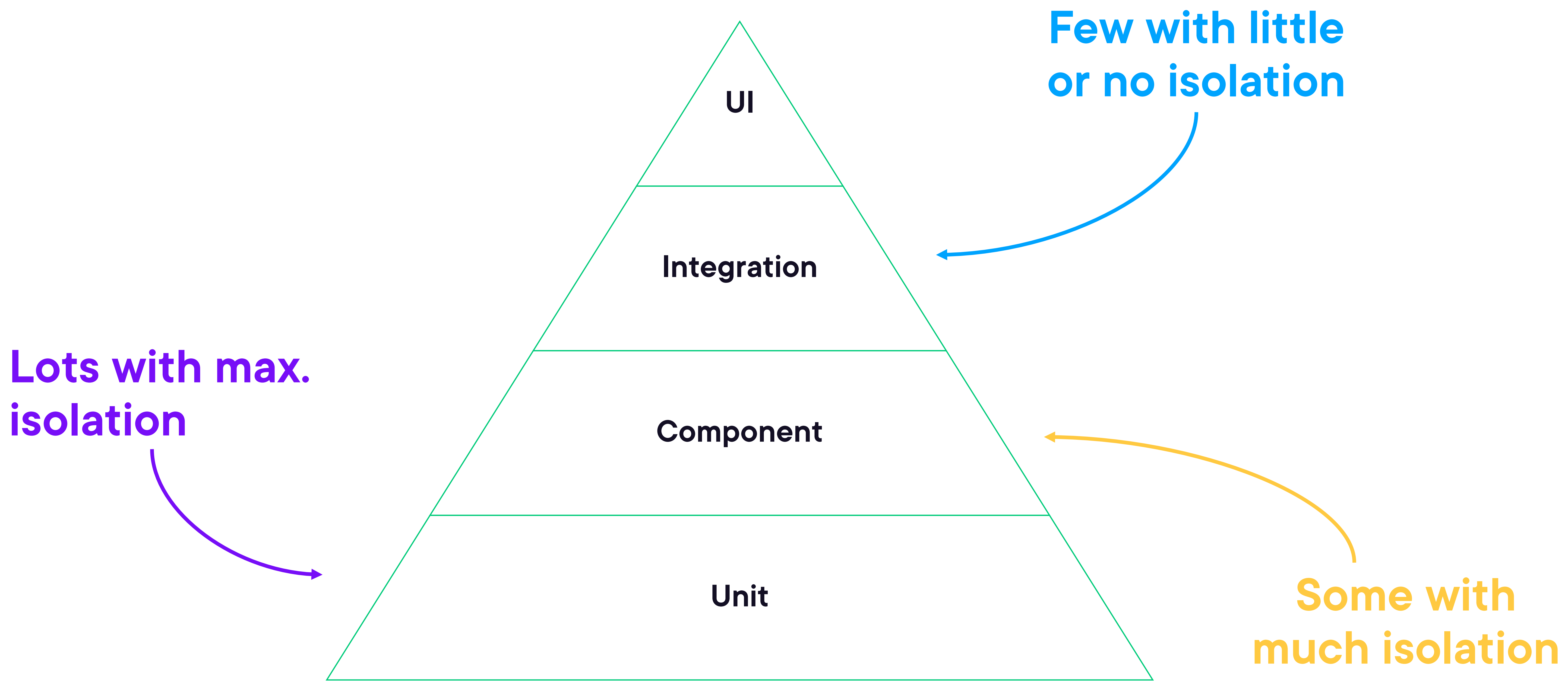
Can we strive for this?

```
Exception in thread "main" java.lang.NullPointerException: on line
5 in Class X the field didn't initialize properly, you need to go
and fix it by changing that other thing over there
```

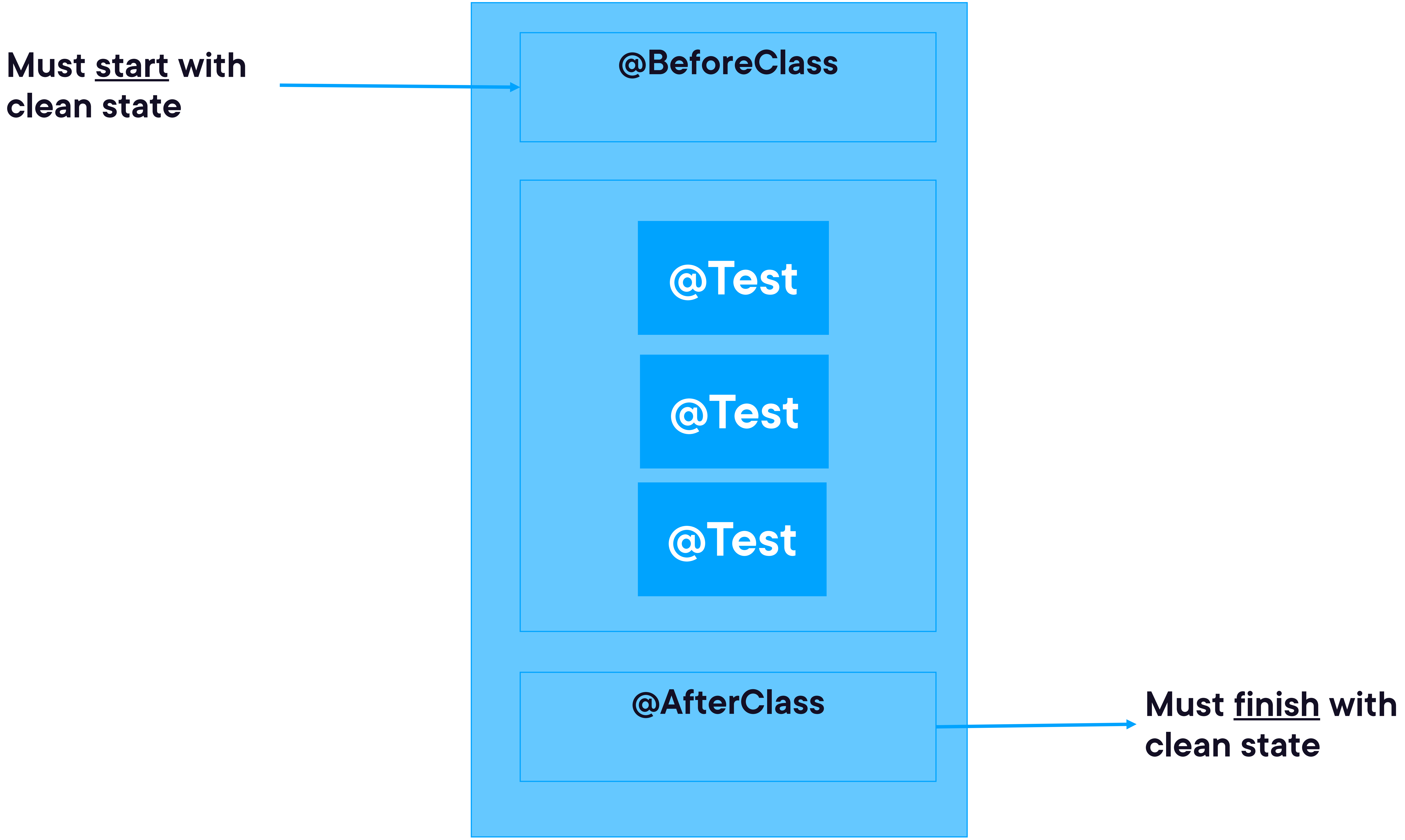
Test Methods

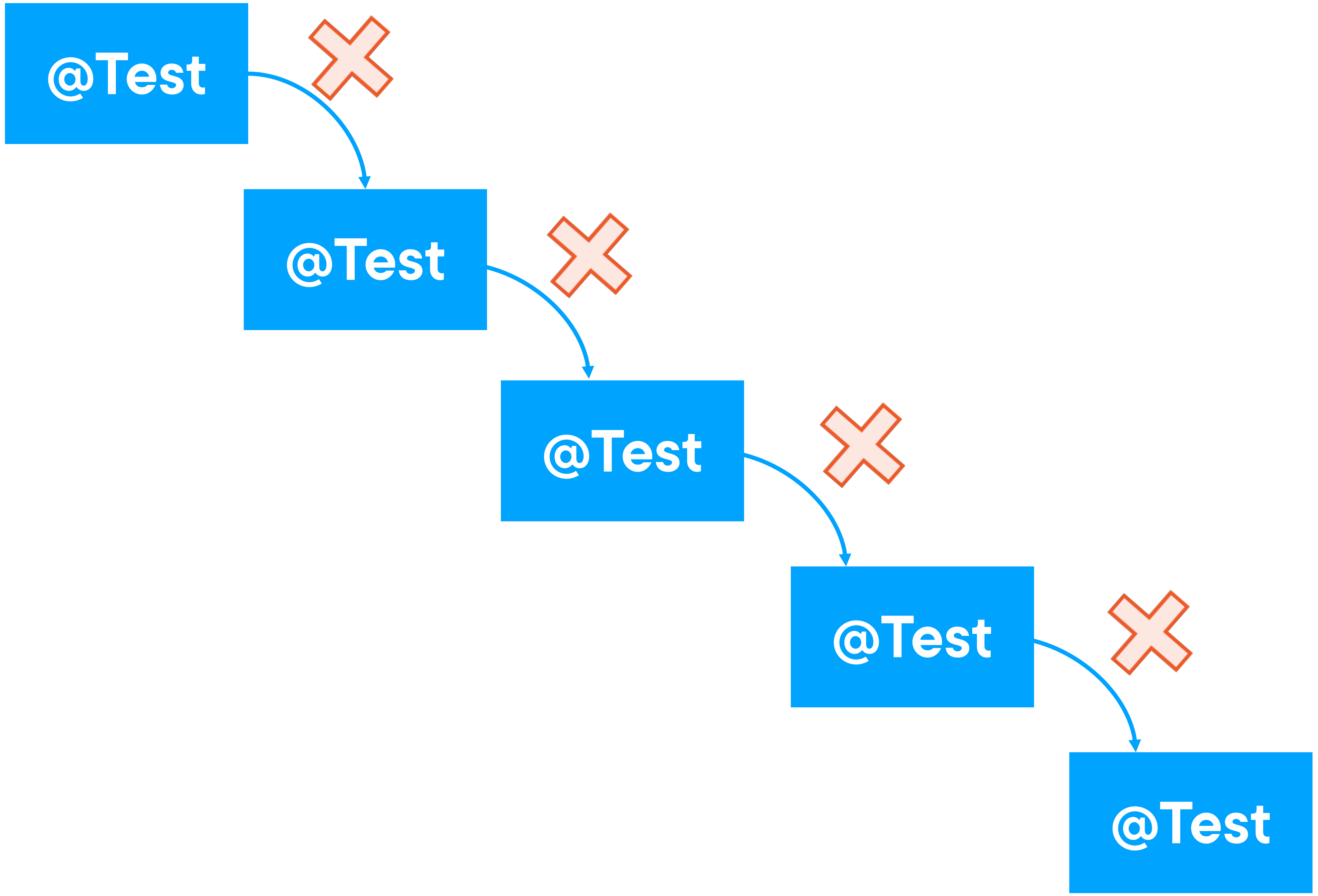
```
assertEquals(expected, actual);

assertEquals(expected, actual, message);
```

Note to your future self


Few with little or no isolation
UI
Integration
Component
Unit
Lots with max. isolation
Some with much isolation

Isolation != Independence


Must start with clean state
@BeforeClass
@Test
@Test
@Test
@AfterClass
Must finish with clean state


@Test
@Test
@Test
@Test
@Test

How can I be sure that I do my clean up right and I don't leave any unwanted artifacts that might interfere with the tests that follow?

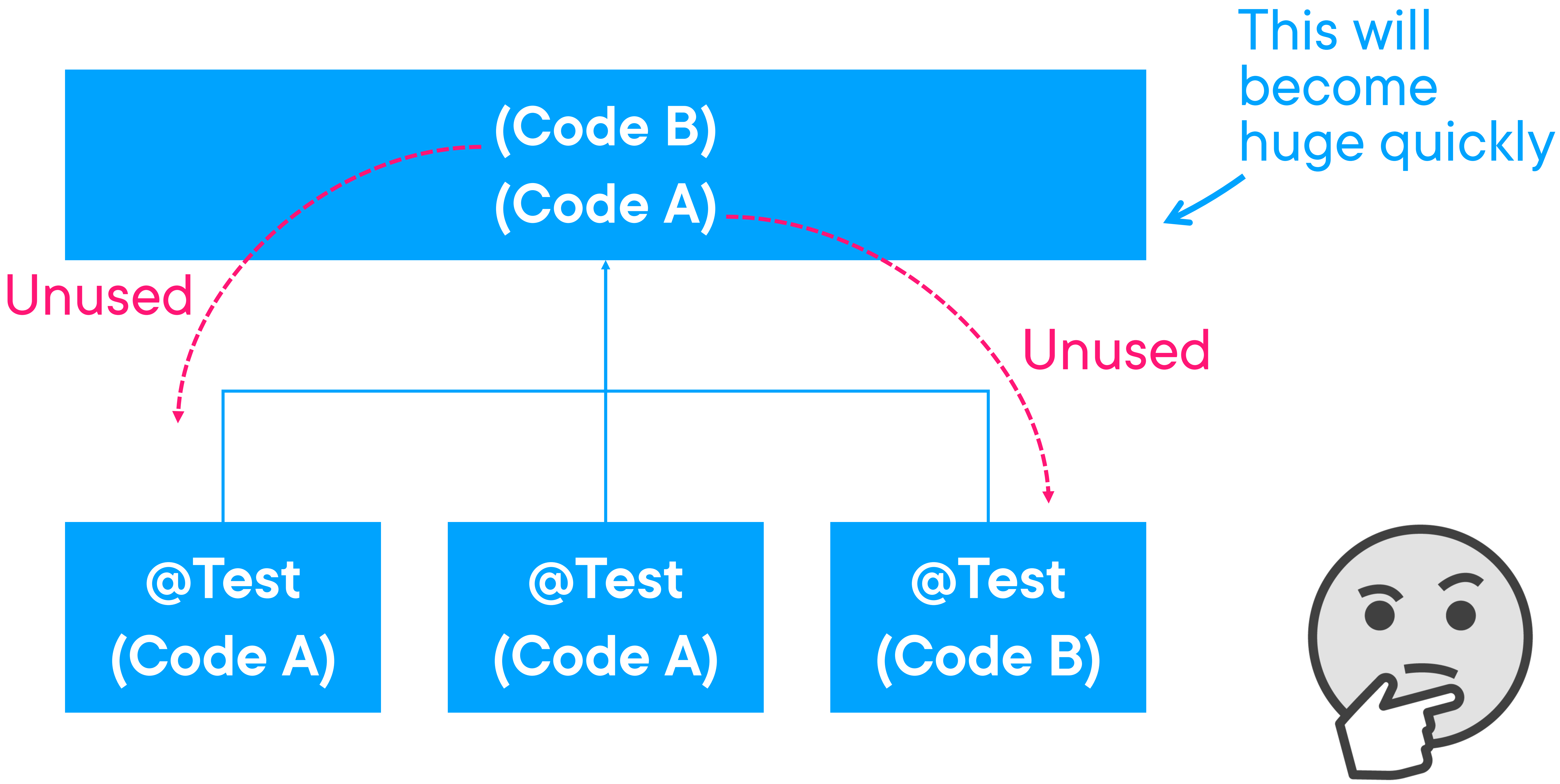
You should be able to run your tests any number of times, in any order

Are My Tests Repeatable?

Run a single test multiple times

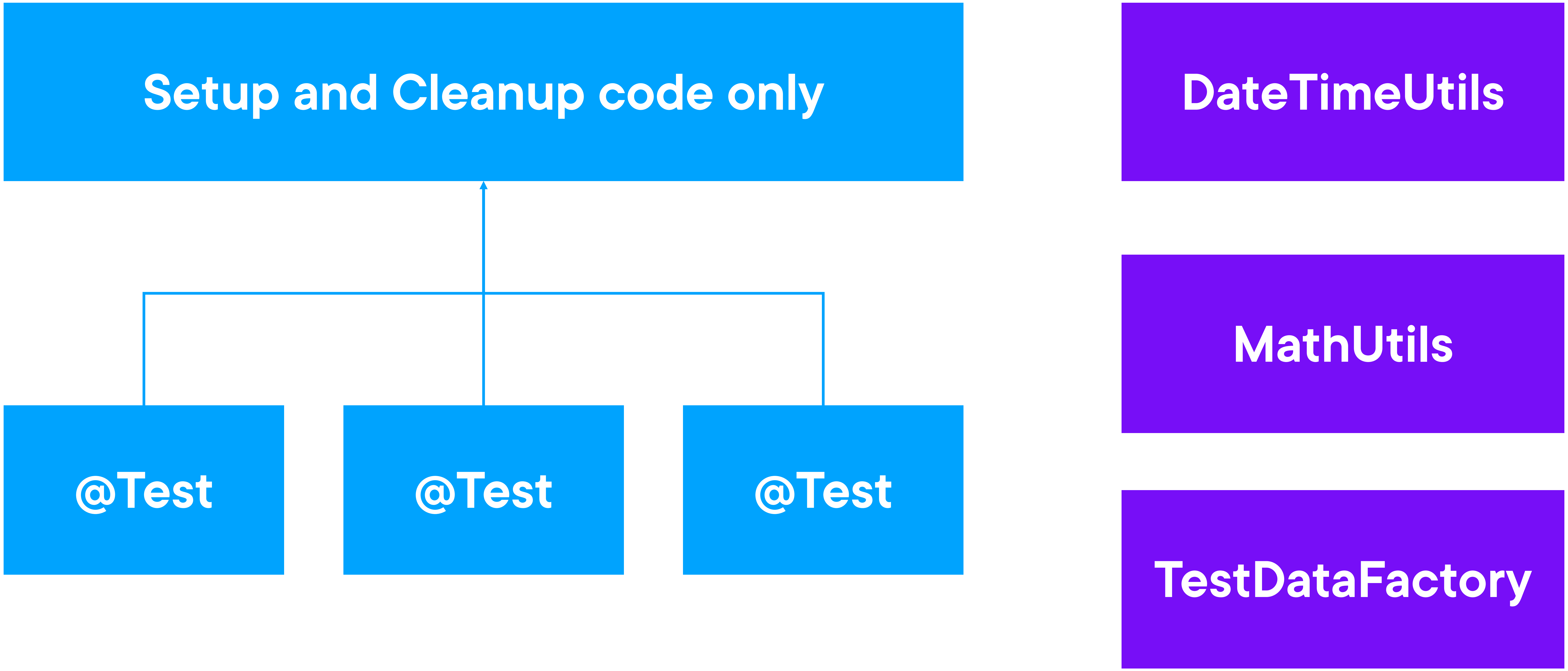
Run a newly added test together with other tests

- Every time the result should be the same



Poor OOP: making child classes inherit things they don’t need

Prefer composition over inheritance


Setup and Cleanup code only
@Test
@Test
@Test
DateTimeUtils
MathUtils
TestDataFactory

Further Study

Fundamentals of Test Automation in Java

- FIRST, BICEP, CORRECT

Java SE 17 Unit Testing with Junit

Writing Highly Maintainable Unit Tests

- (In C#)

Summary

Test Automation anti-patterns

Naming matters

DAMP over DRY

Helpful messages... help!

Tests must be independent

Prefer composition over inheritance