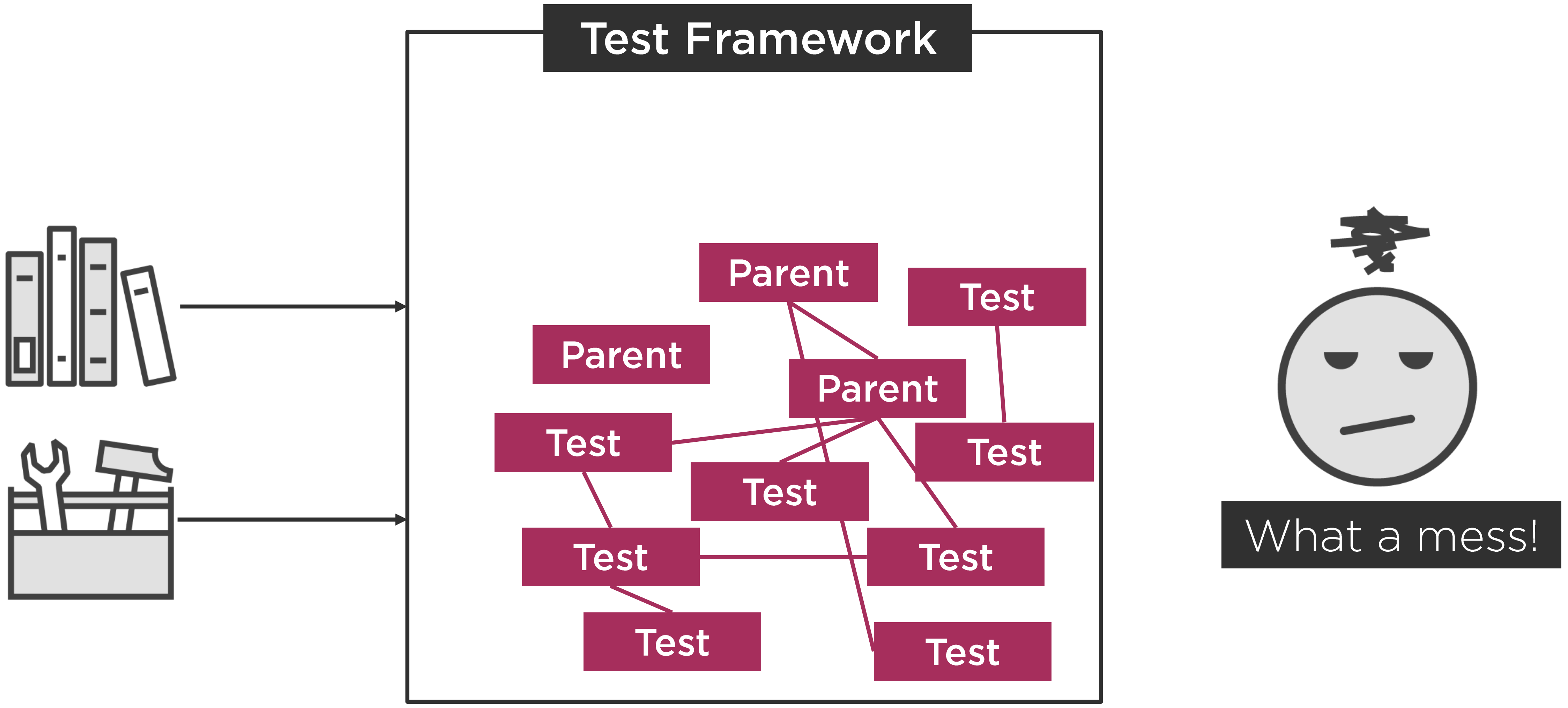
Evolving into a Multi-module Test Framework


Test Framework
Parent
Test
Parent
Parent
Test
Test
Test
Test
Test
Test
Test
What a mess!

Using a shiny new tool

!=

Test Automation Engineer Pro

Overview

Review clean test code techniques applied so far

Test Refactoring Pyramid

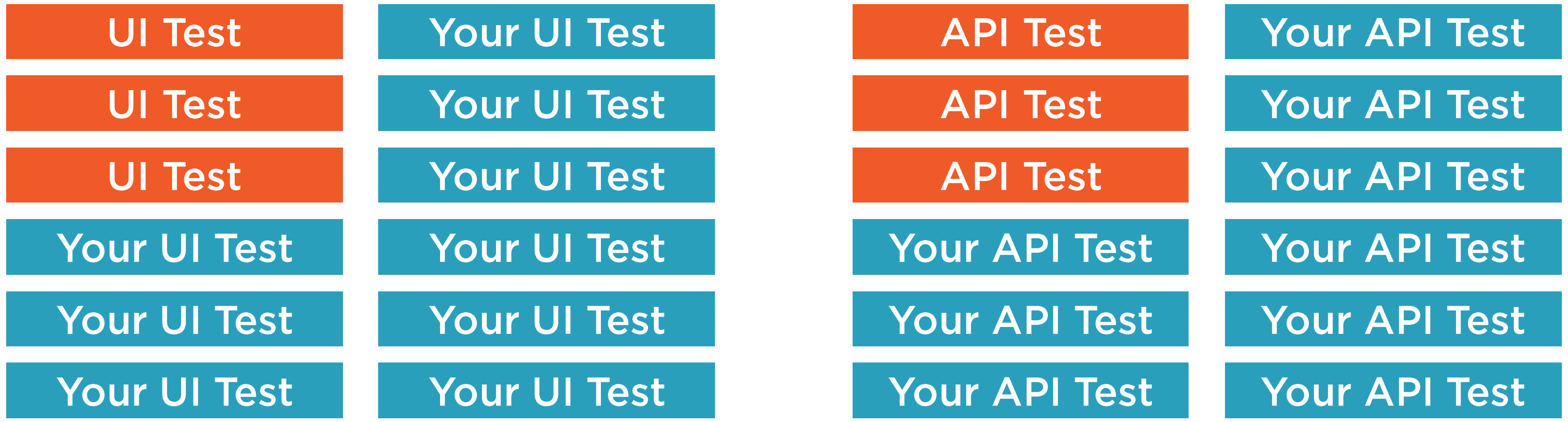
Evaluate abuse of Inheritance

Replace Inheritance with Composition

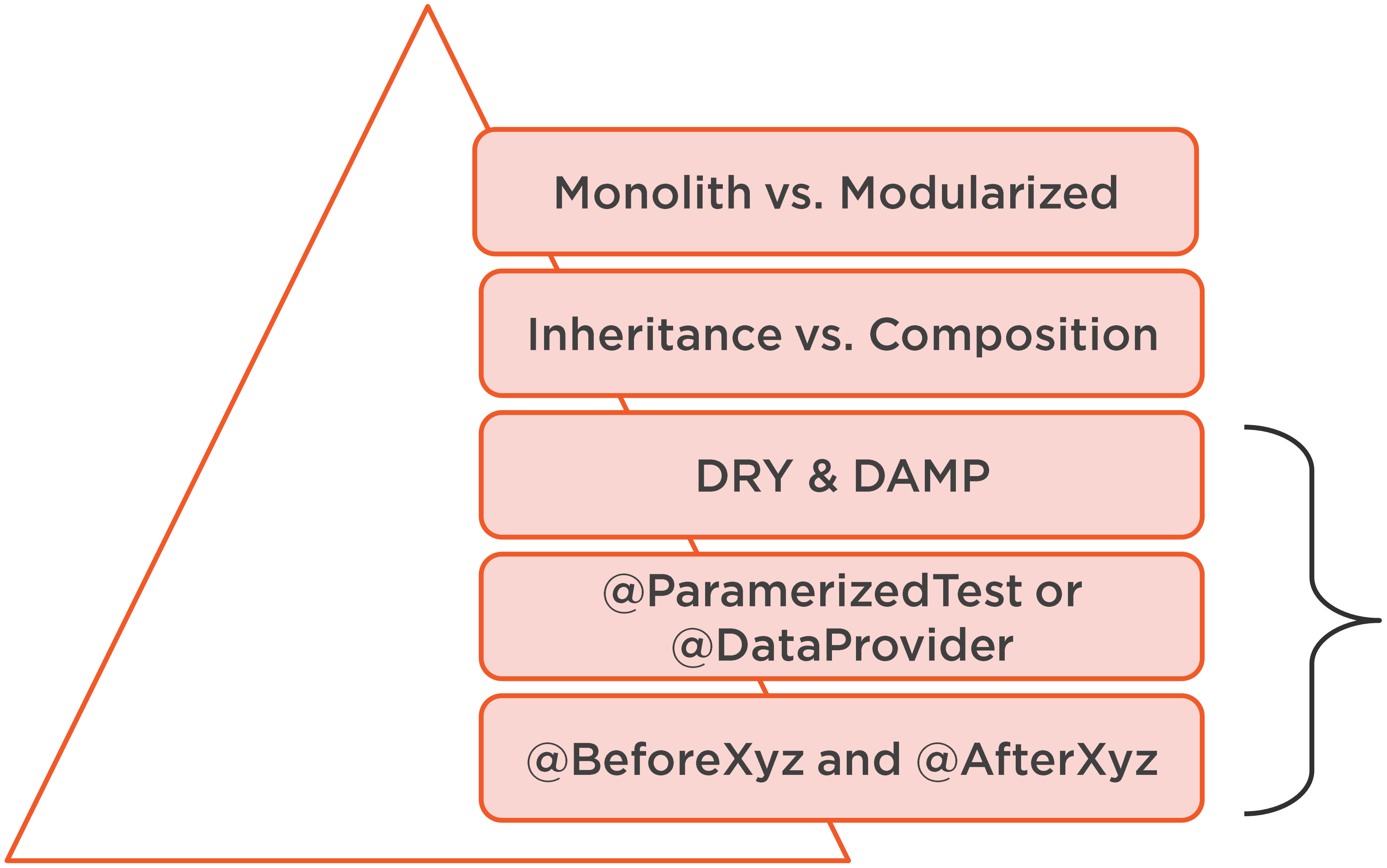
Convert to a multi-module framework

Further course recommendations

As Time Goes By…

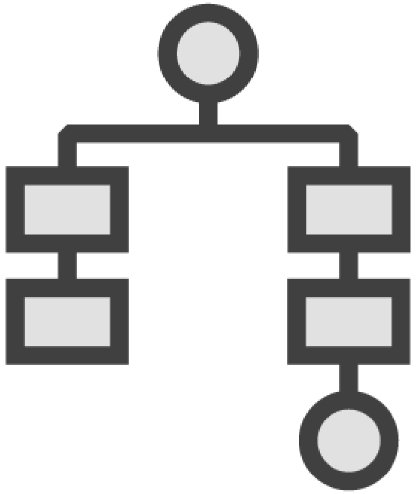


Test Refactoring Pyramid


Monolith vs. Modularized
Inheritance vs. Composition
DRY & DAMP
@ParamerizedTest or @DataProvider
@BeforeXyz and @AfterXyz

What we've done so far

Your Big(ger) Enemies



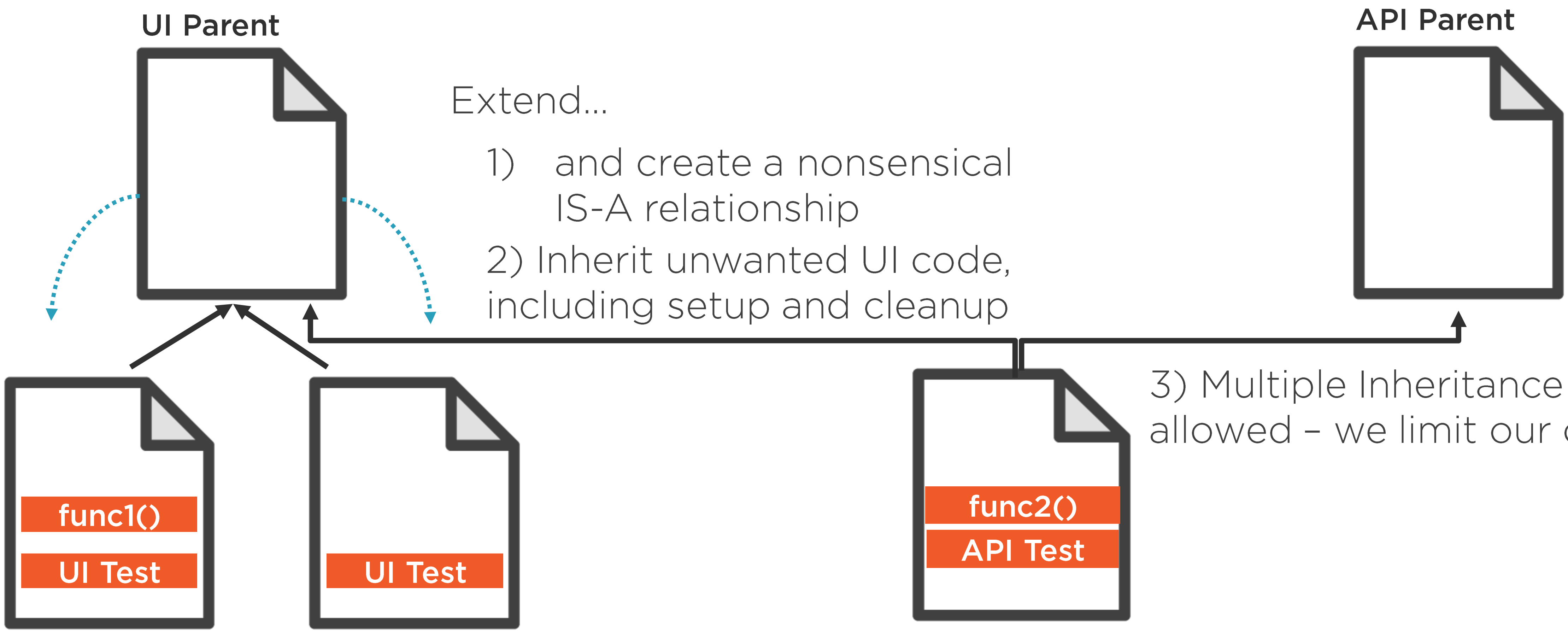
(Abuse of) Inheritance

Code becomes very inflexible

Monolith

A big blob of code

Use inheritance? ❌

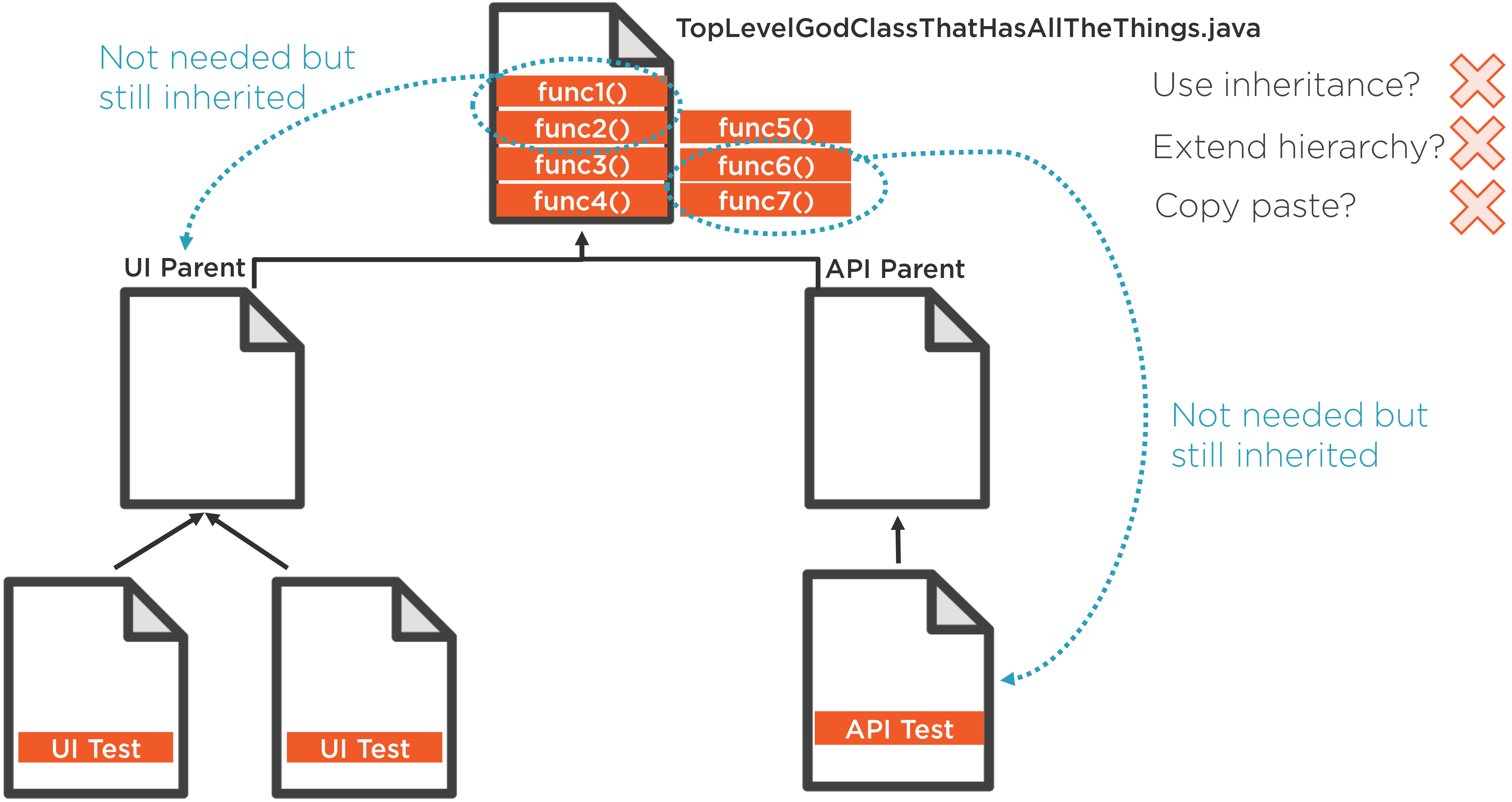


Extend…

1) and create a nonsensical IS-A relationship

2) Inherit unwanted UI code, including setup and cleanup

3) Multiple Inheritance is not allowed – we limit our options

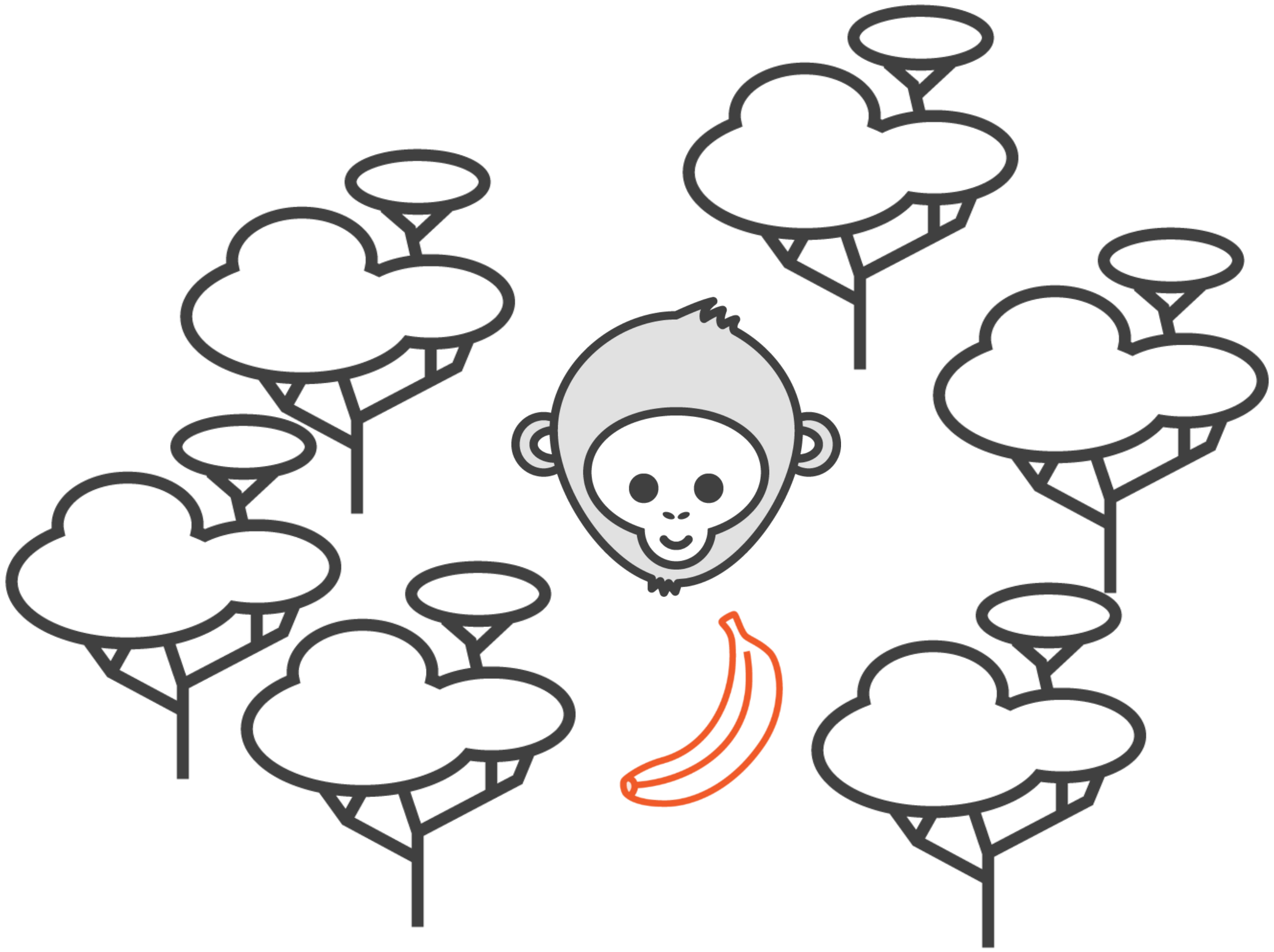

TopLevelGodClassThatHasAllTheThings.java
Not needed but still inherited
func1()
func2()
func3()
func4()
func5()
func6()
func7()
Use inheritance?
Extend hierarchy?
Copy paste?
UI Parent
API Parent
Not needed but still inherited
UI Test
UI Test
API Test

Inheritance creates very tight coupling

God class

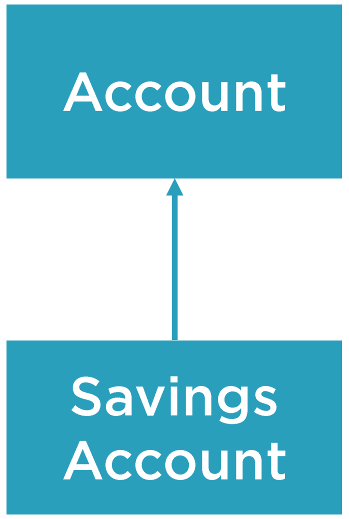
A class that knows too much or does too much. It is connected to way too many other classes and has grown beyond all logic.

Inheritance and the Extra Baggage

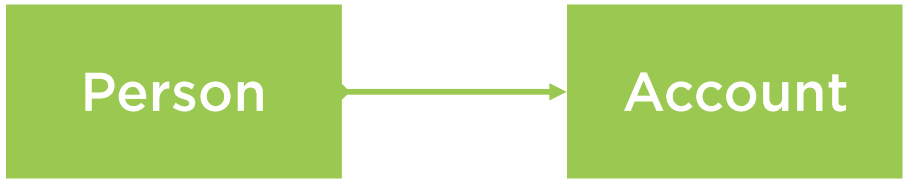


Replace Inheritance with Composition

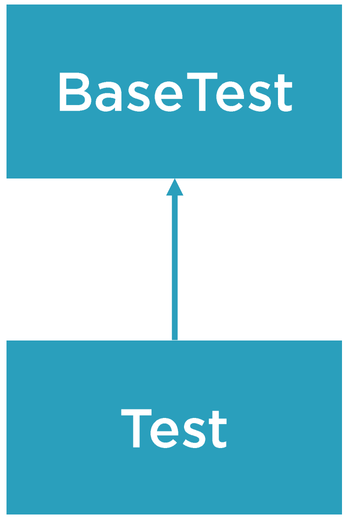
HAS-A
vs.
IS-A

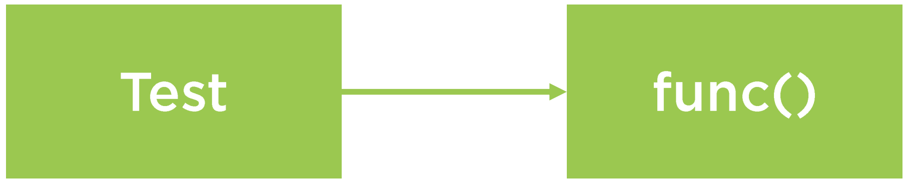


Inheritance: IS-A



Composition: HAS-A



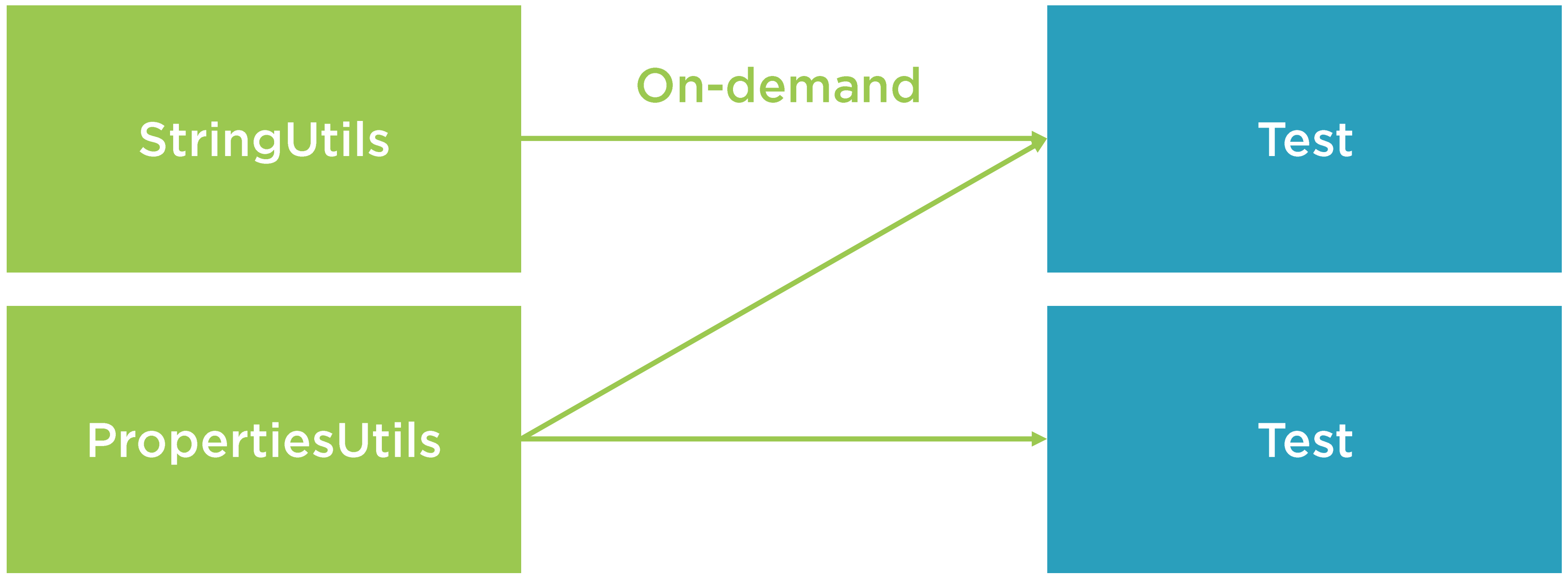


Inheritance: IS-A

Tests needs to run common **setup** and **cleanup**

Composition: HAS-A

Tests needs some specific functionality


StringUtils
On-demand
Test
PropertiesUtils
Test

Benefits of Composition

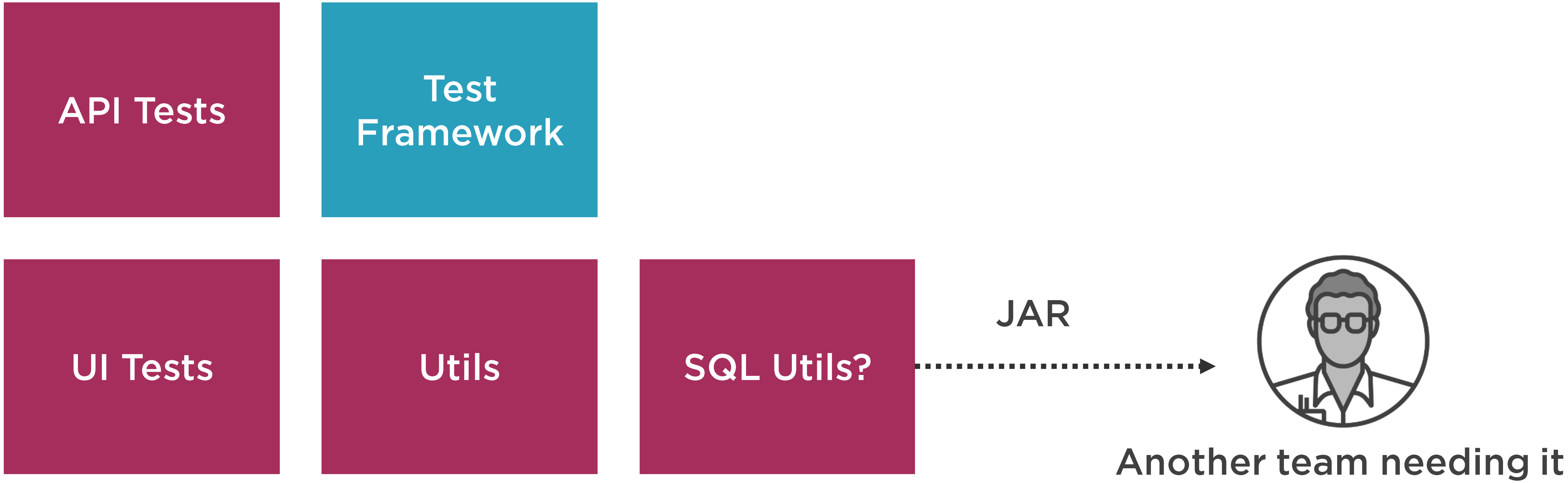
Avoid anti-patterns, such as God Class

Greater flexibility

On-demand functionality

Allows to easily modularize our framework

Test Framework


API Tests
Test Framework
UI Tests
Utils
SQL Utils?
JAR
Another team needing it

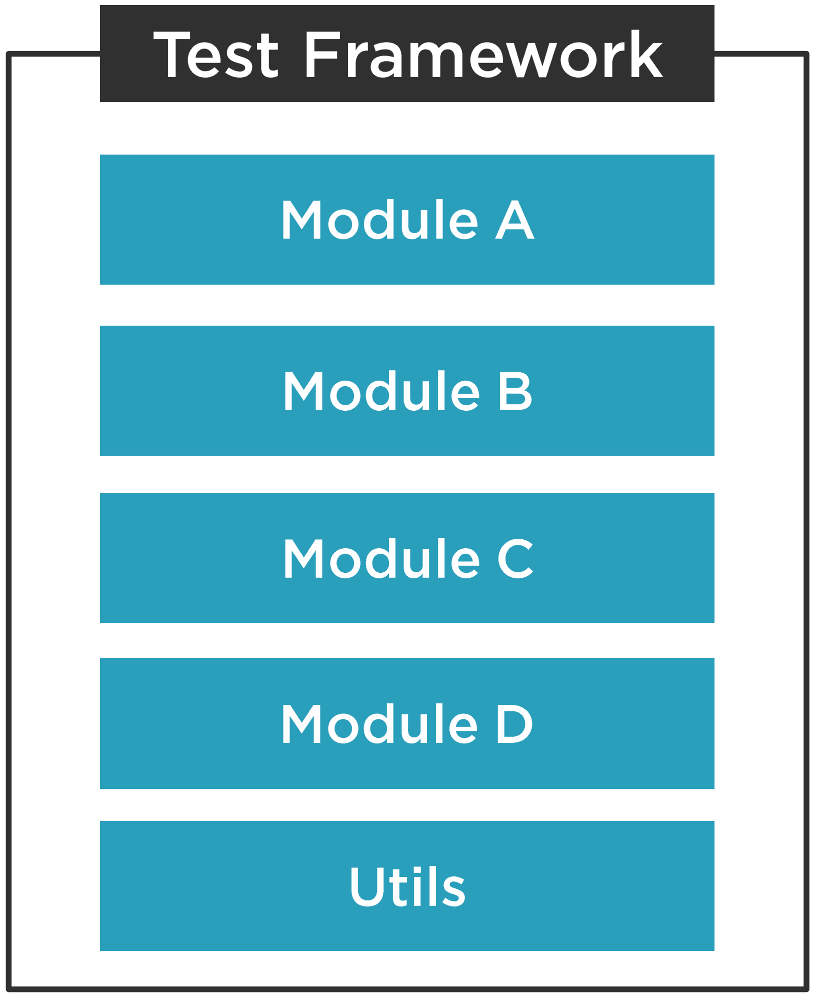
Benefits of Modularization

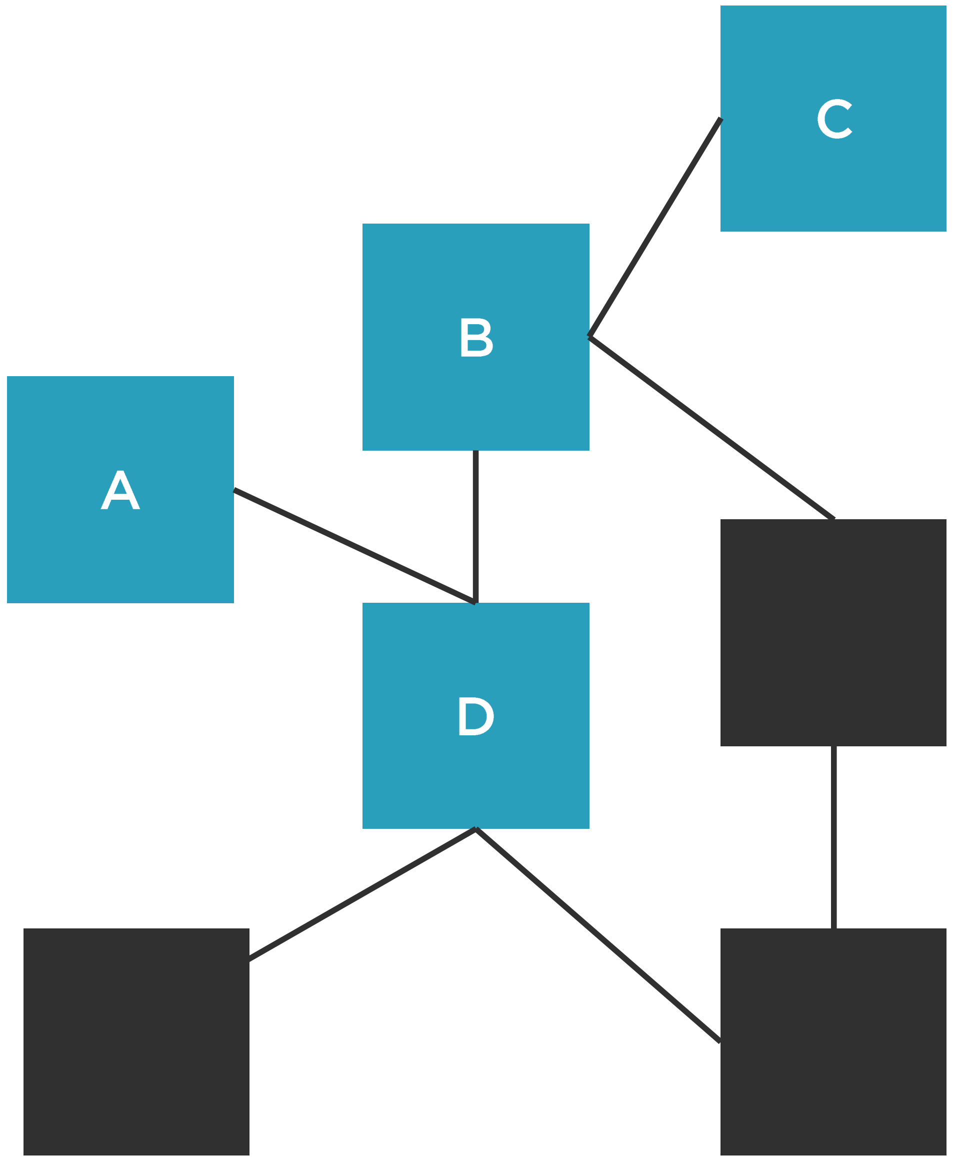
Enhances code organization

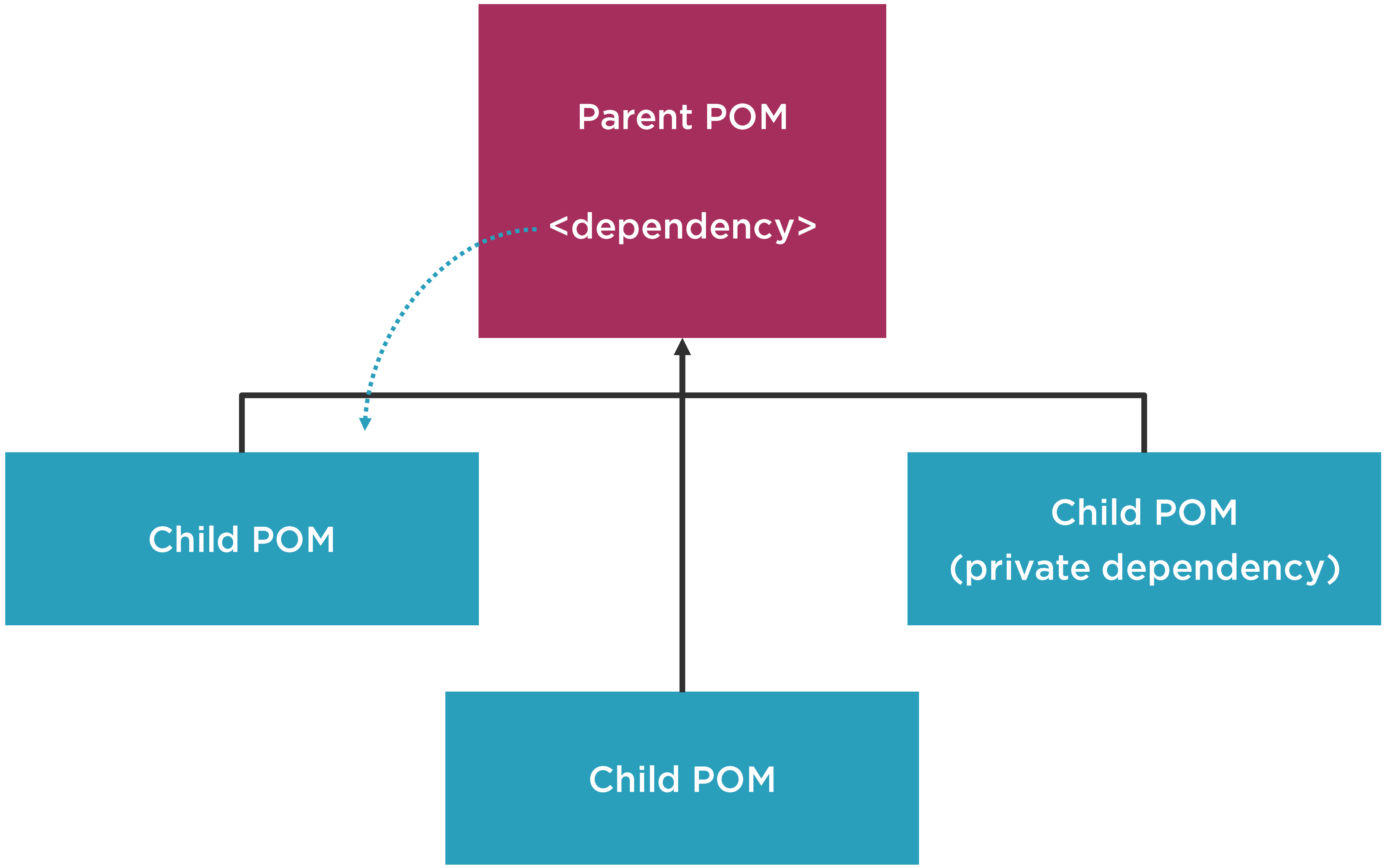
Speeds up orientation in code

Separates concerns

More granular management of dependencies


Test Framework
Module A
Module B
Module C
Module D
Utils


C
B
A
D


Parent POM
<dependency>
Child POM
Child POM
(private dependency)
Child POM

For SDETs:

- Tools are important
- General programming skills are even more so

Further Study

Java: Writing Readable and Maintainable Code

Java Refactoring: Best Practices

SOLID Software Design Principles in Java

Java Design Patterns Path

Summary

Review various techniques to make test code better

- Setup and Cleanup
- Parametrizing
- Applying DRY and DAMP

Prefer composition over inheritance

Modularizing the project allows scaling

Rating

Thank you!

(Happy coding)