

# Test-Driven Development with JAVA





Somkiat Puisungnoen

Somkiat Puisungnoen

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Intro

Software Craftsmanship

Software Practitioner at สยามชัมนาภิกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Somkiat Puisungnoen 15 mins · Bangkok · ⚙️

Java and Bigdata



somkiat.cc

Page Messages Notifications 3 Insights Publishing Tools Settings Help ▾

somkiat.cc  
@somkiat.cc

Home Posts Videos Photos

Like Following Share ...

Help people take action on this Page. + Add a Button



# Slide

<https://github.com/up1/course-tdd-with-java-2020>





# Testing in general



# Technical excellence



SPECIFICATION BY EXAMPLE



CONTINUOUS  
INTEGRATION



CONTINUOUS DELIVERY



TEST AUTOMATION



TECHNICAL  
EXCELLENCE



THINKING ABOUT TESTING



ARCHITECTURE  
& DESIGN



ACCEPTANCE  
TESTING



CLEAN CODE



TEST-DRIVEN DEVELOPMENT



UNIT TESTING

<https://less.works/less/technical-excellence/index>





SPECIFICATION BY EXAMPLE



TEST AUTOMATION



THINKING ABOUT TESTING



CONTINUOUS  
INTEGRATION



TECHNICAL  
EXCELLENCE



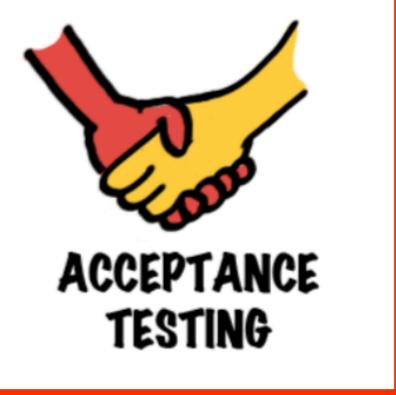
TEST-DRIVEN DEVELOPMENT



CONTINUOUS DELIVERY



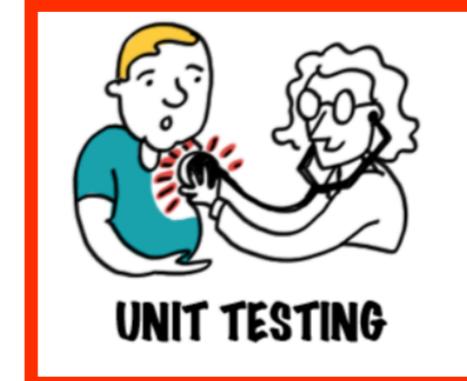
ARCHITECTURE  
& DESIGN



ACCEPTANCE  
TESTING



CLEAN CODE



UNIT TESTING





SPECIFICATION BY EXAMPLE



TEST AUTOMATION



CONTINUOUS  
INTEGRATION



TECHNICAL  
EXCELLENCE



DEVELOPMENT  
TEST-DRIVEN DEVELOPMENT



CONTINUOUS DELIVERY



ACCEPTANCE  
TESTING



ARCHITECTURE  
& DESIGN



CLEAN CODE



UNIT TESTING



**"Program testing can be used  
to show the presence of bugs,  
but never their absence."**

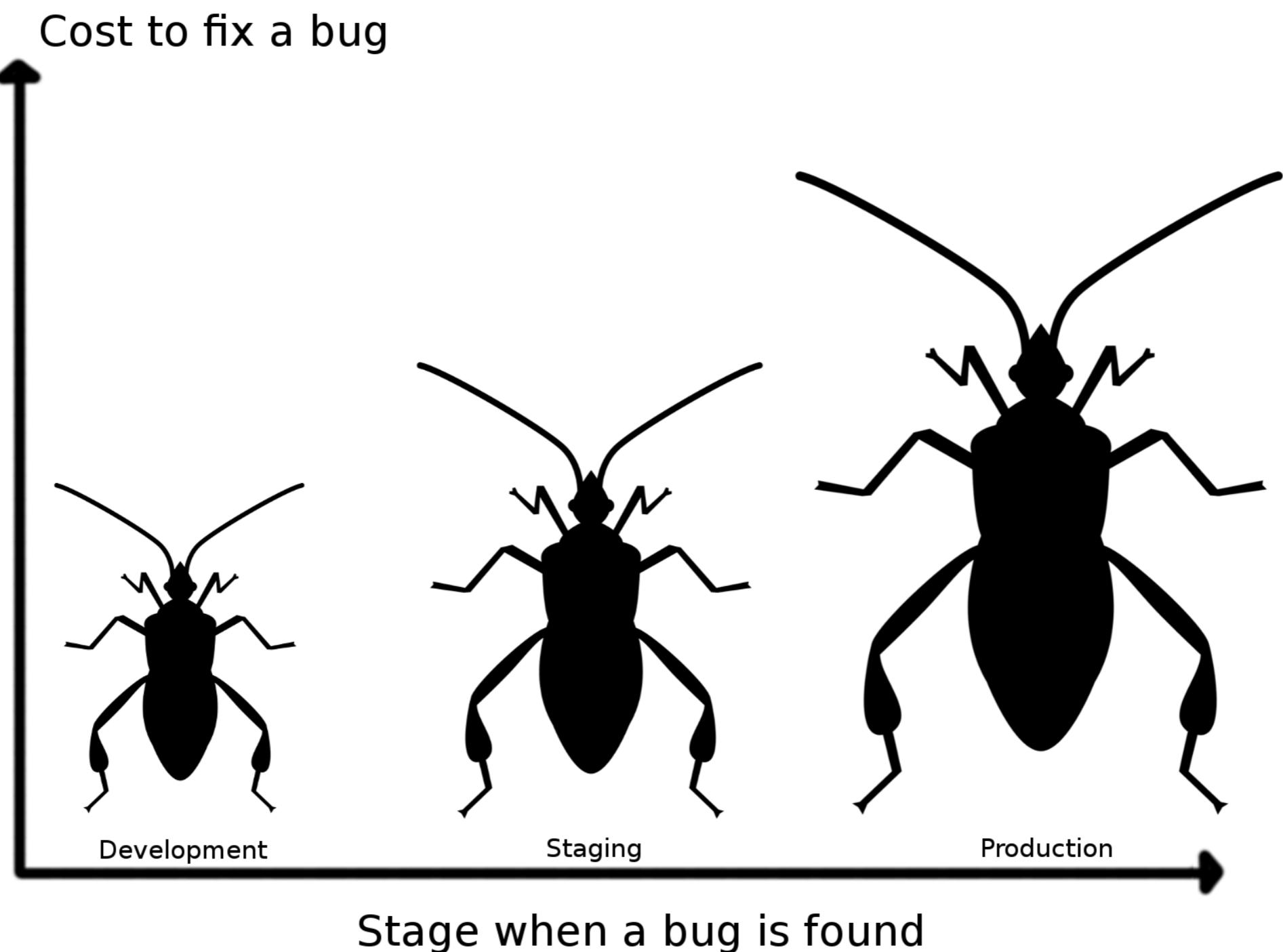
Edsger W. Dijkstra, 1970, Notes on Structured Programming

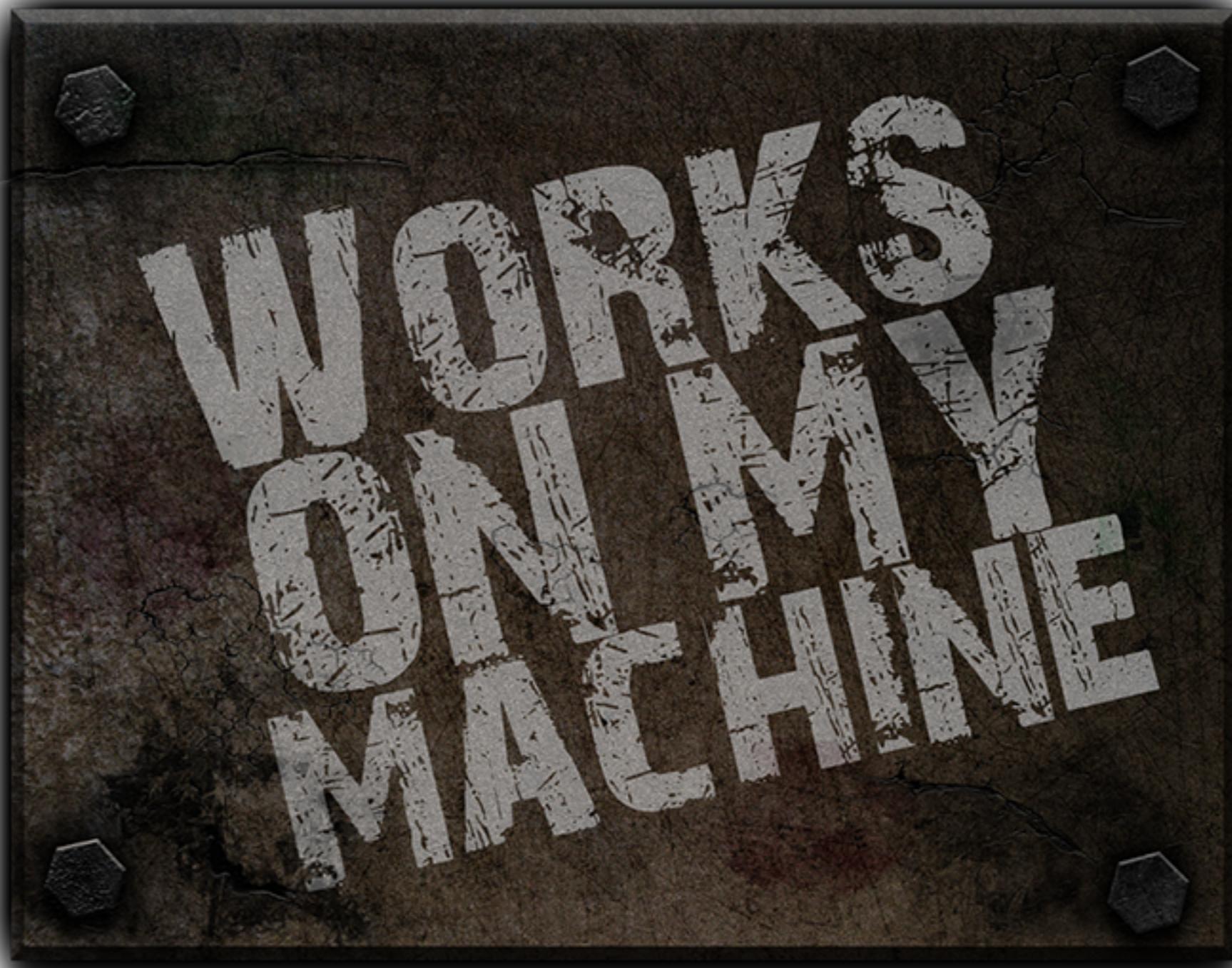


# **Start with Why ...**







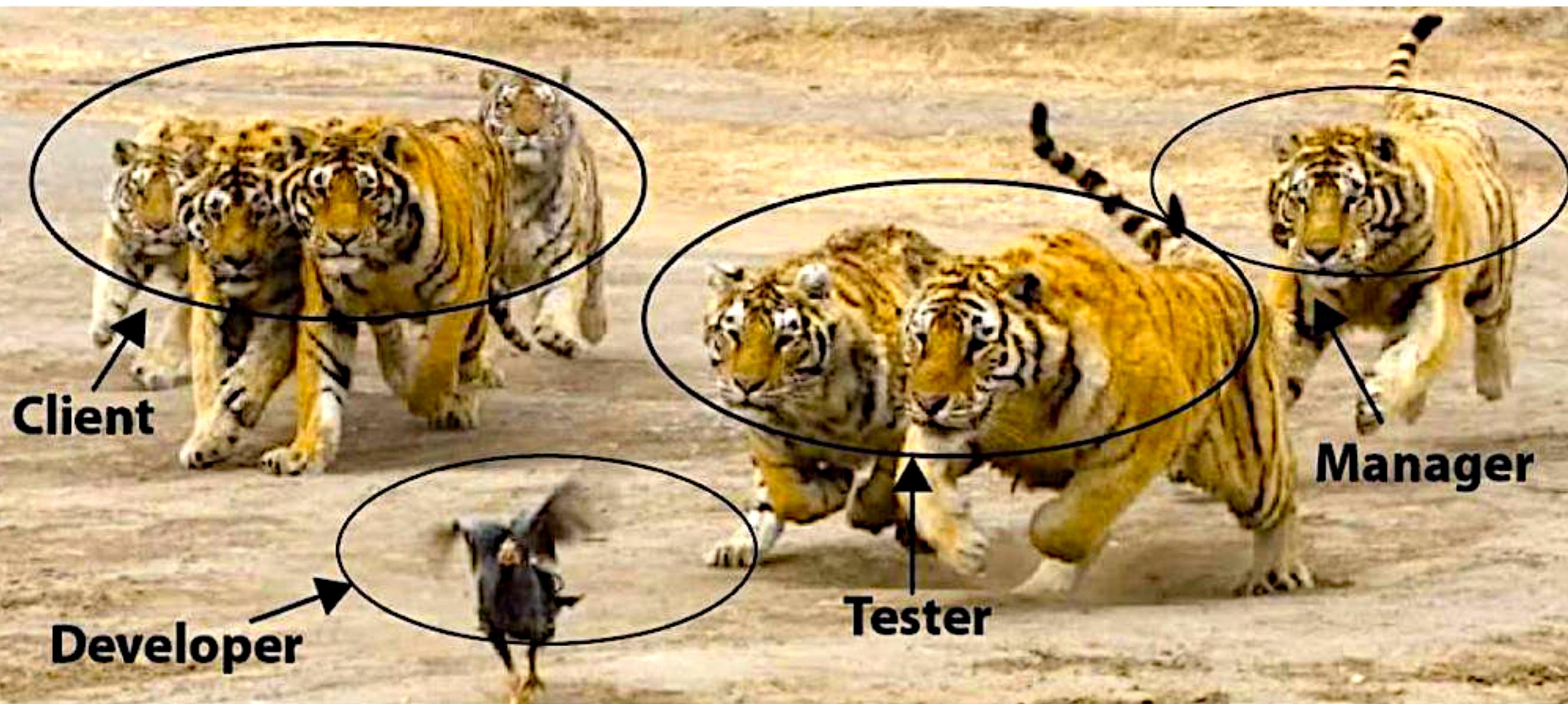


# DDD

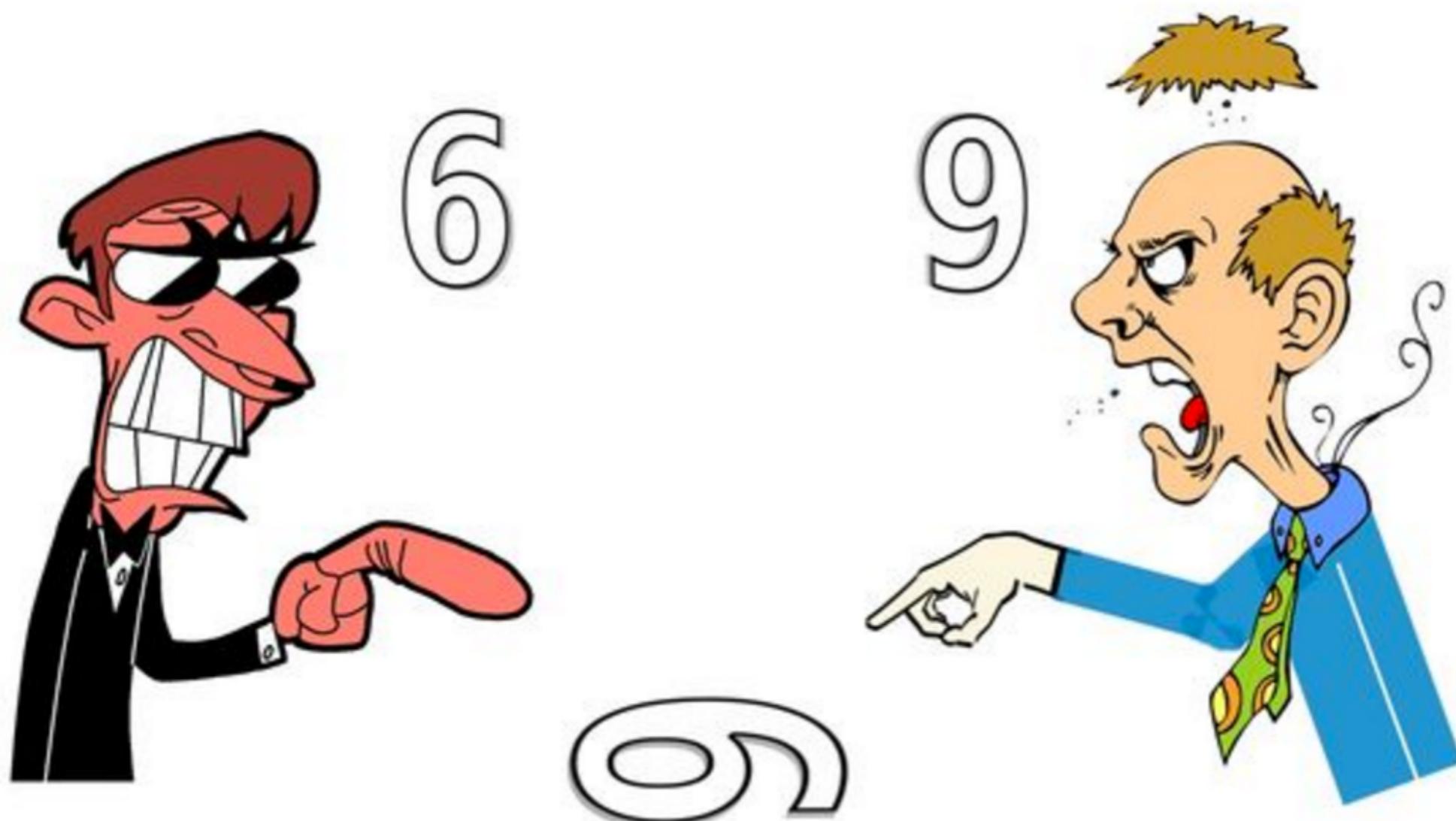


# Deadline Driven Development





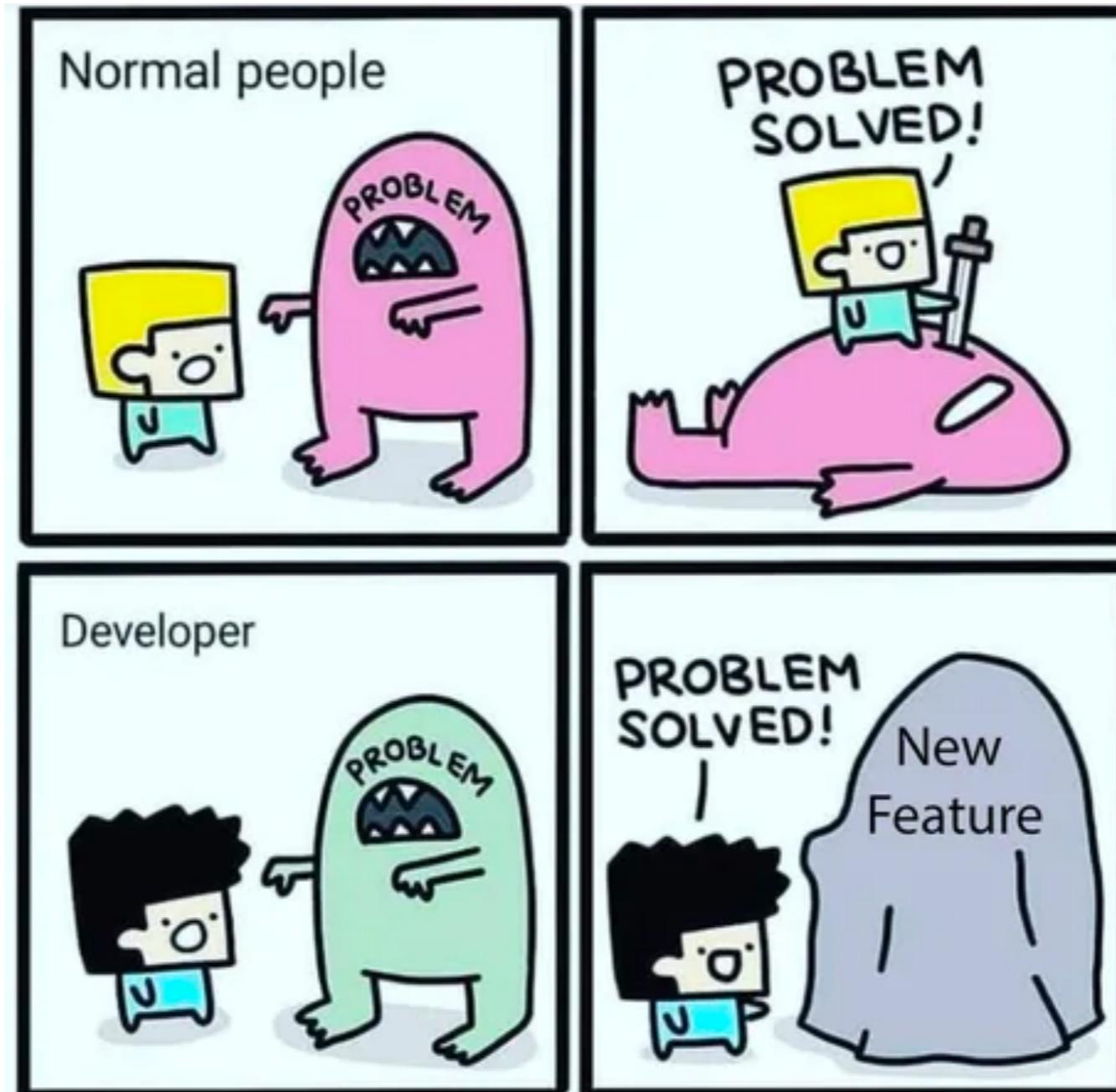
# Developer vs Tester





[www.cartoonistshilpa.com](http://www.cartoonistshilpa.com)





# Every time a developer changing the code !!







**TONIGHT WE TEST**

**IN PRODUCTION!!!**

memegenerator.net



# Why we need to test ?

Help you to **catch bugs**

Boosted confidence

Quality code

Enforce **modularity** of your project

Develop features **faster**

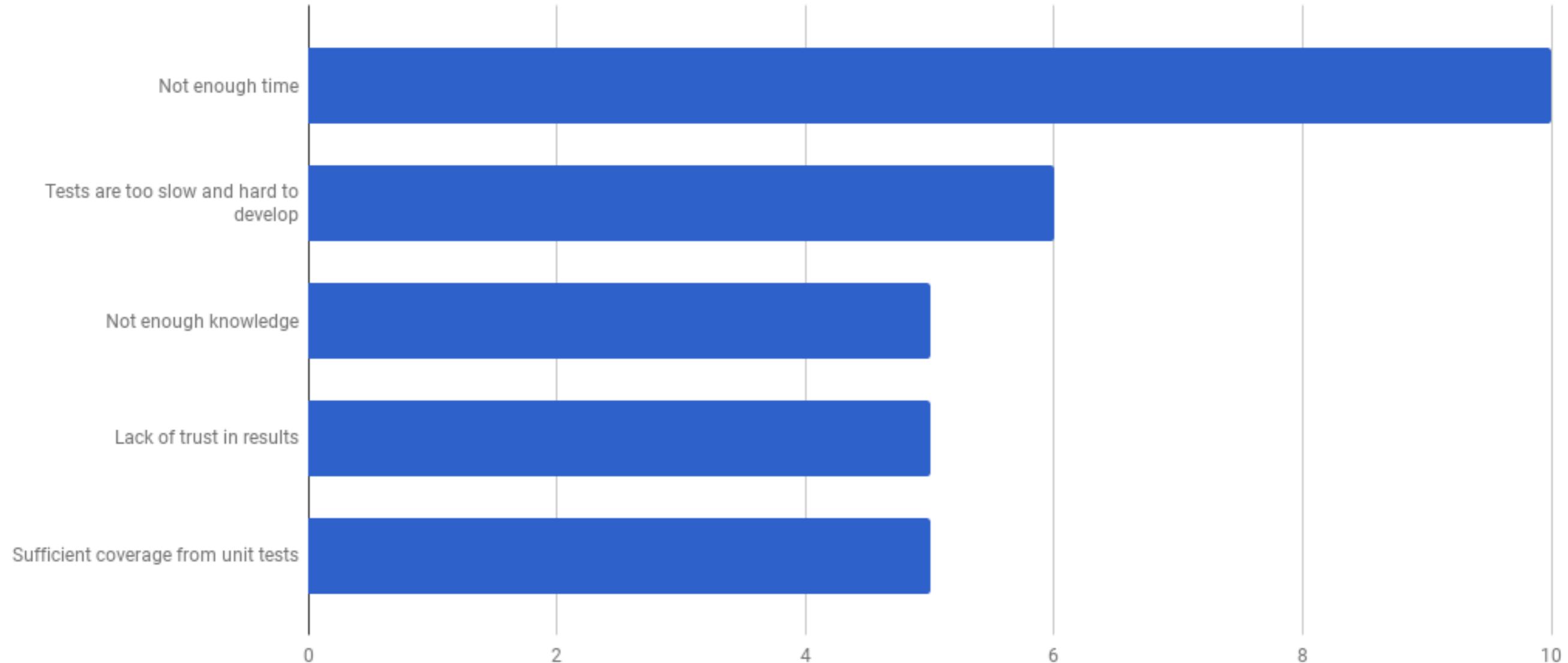
Better documentation



But,  
It's take time to learning and  
practice !!



# Why not write automated tests ?

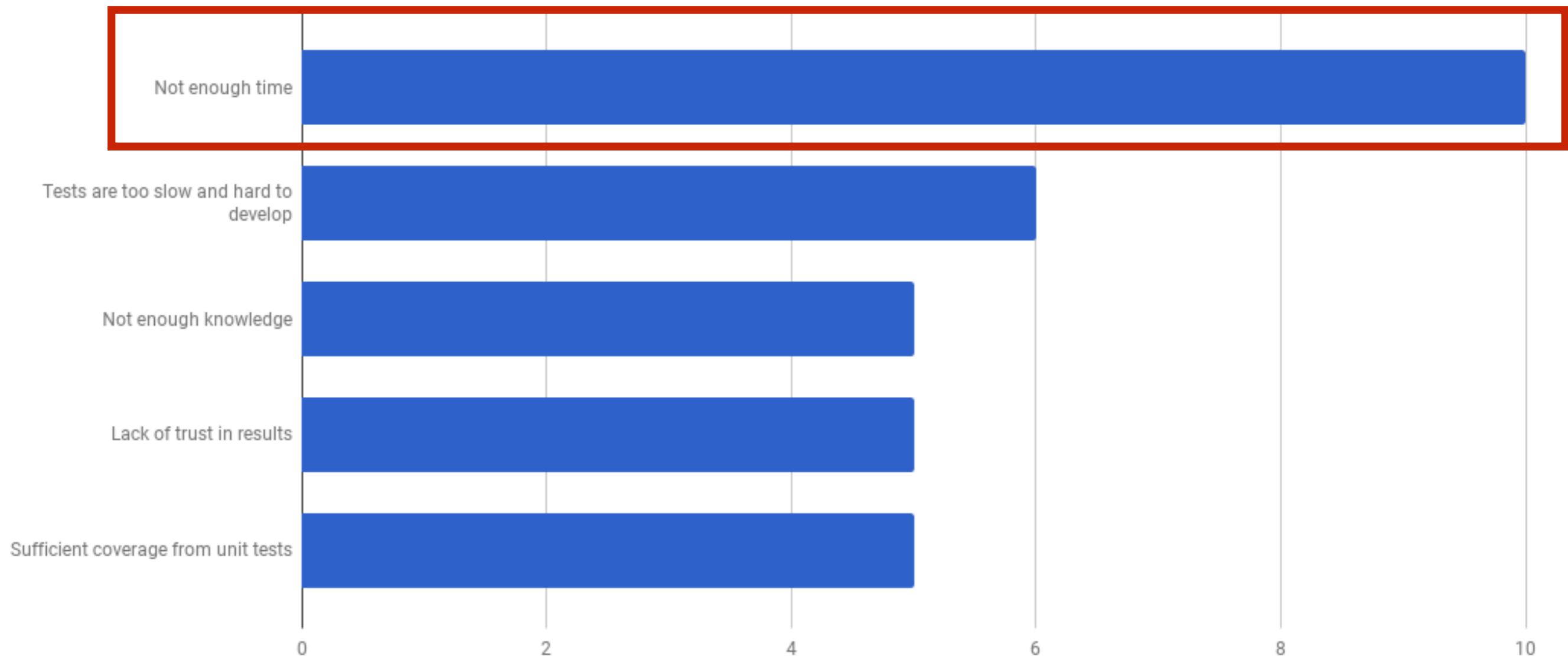


<https://slack.engineering/android-ui-automation-part-1-building-trust-de3deb1c5995>



# Why not write automated tests ?

Not enough time !!!



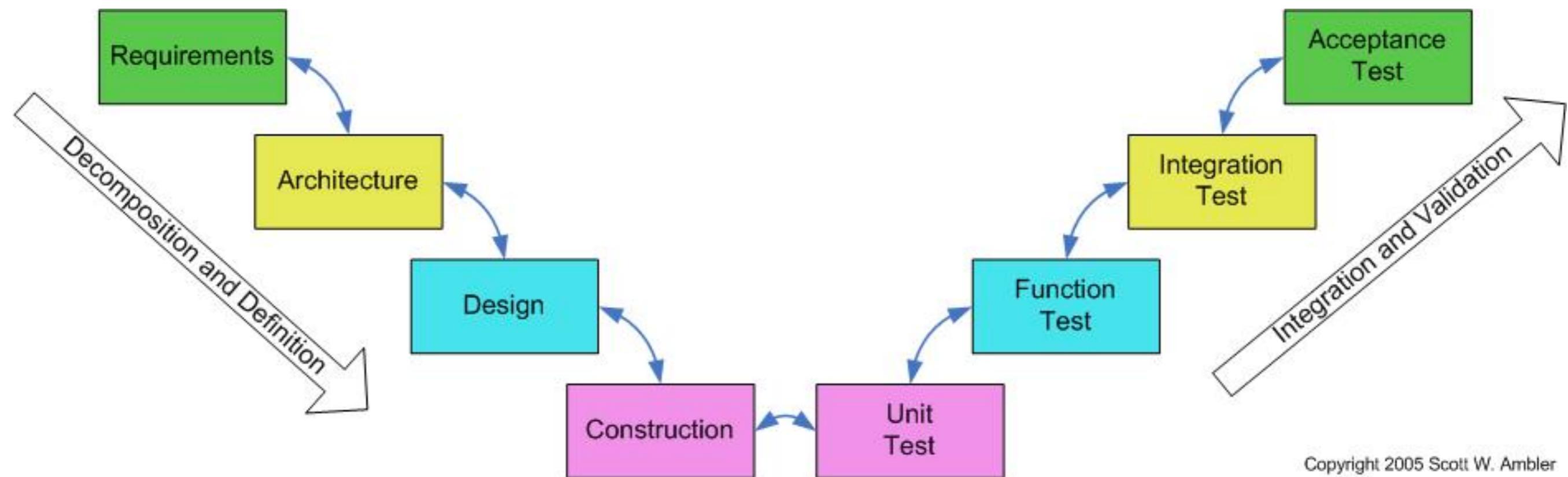
<https://slack.engineering/android-ui-automation-part-1-building-trust-de3deb1c5995>



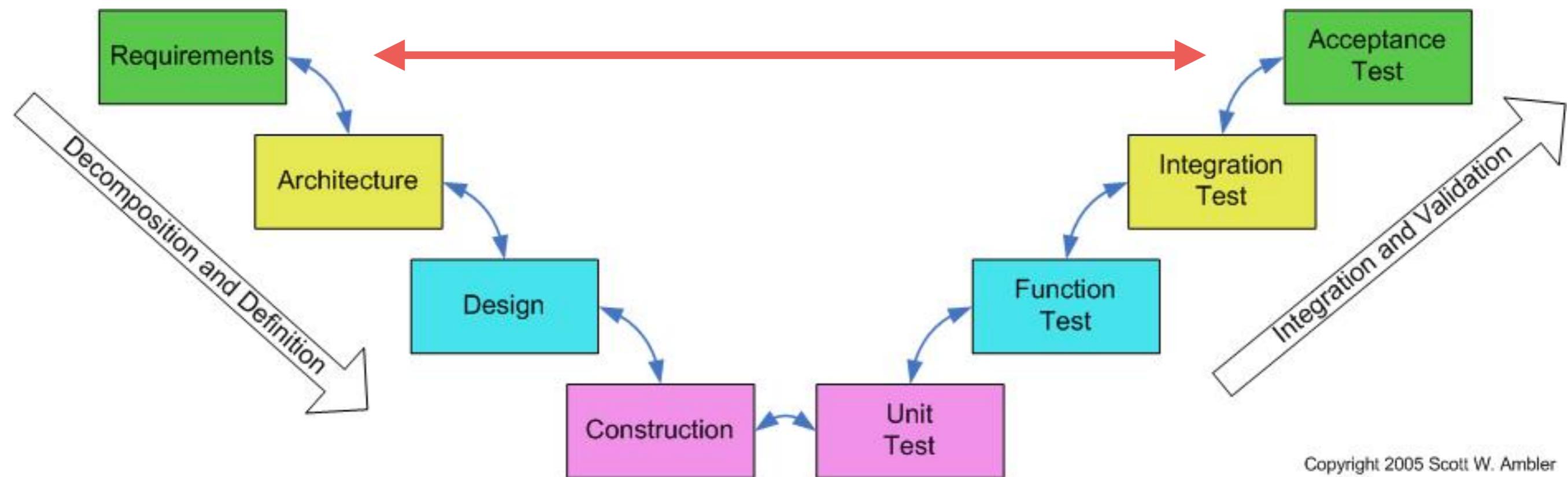
# **What kind of test should we write ?**



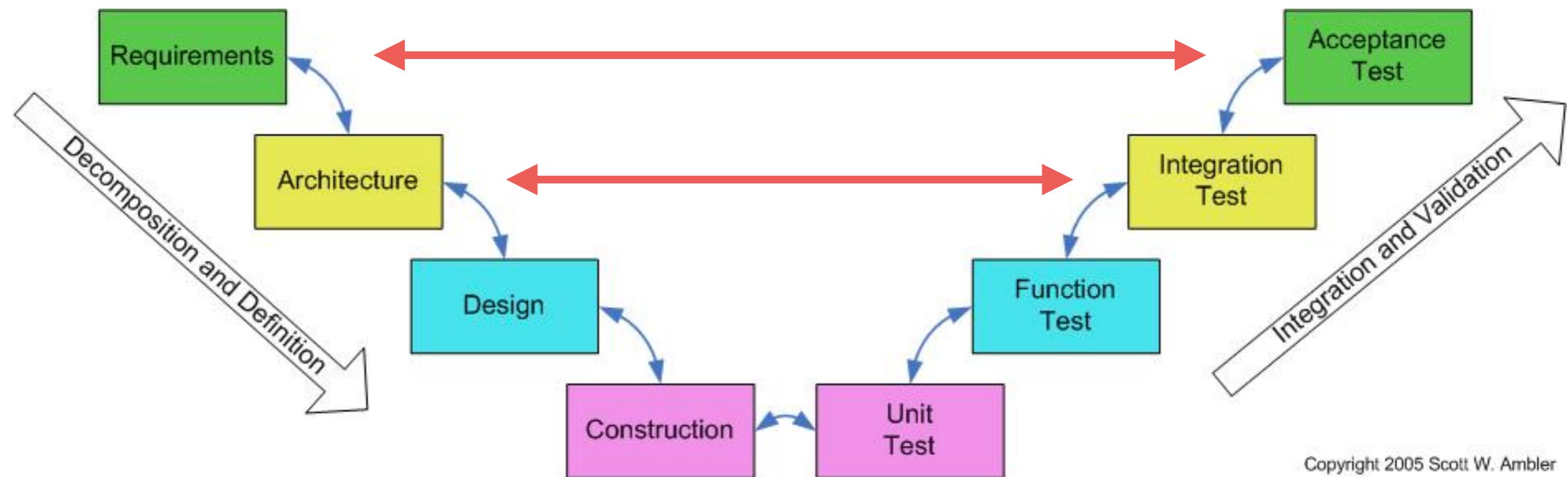
# V Model



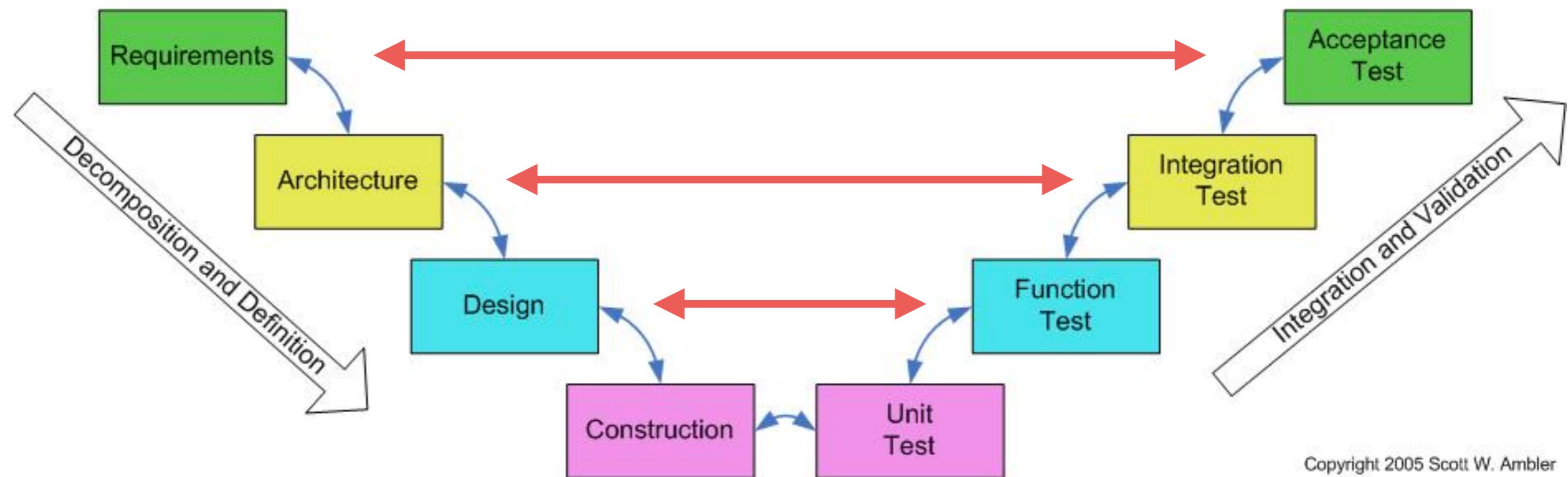
# V Model



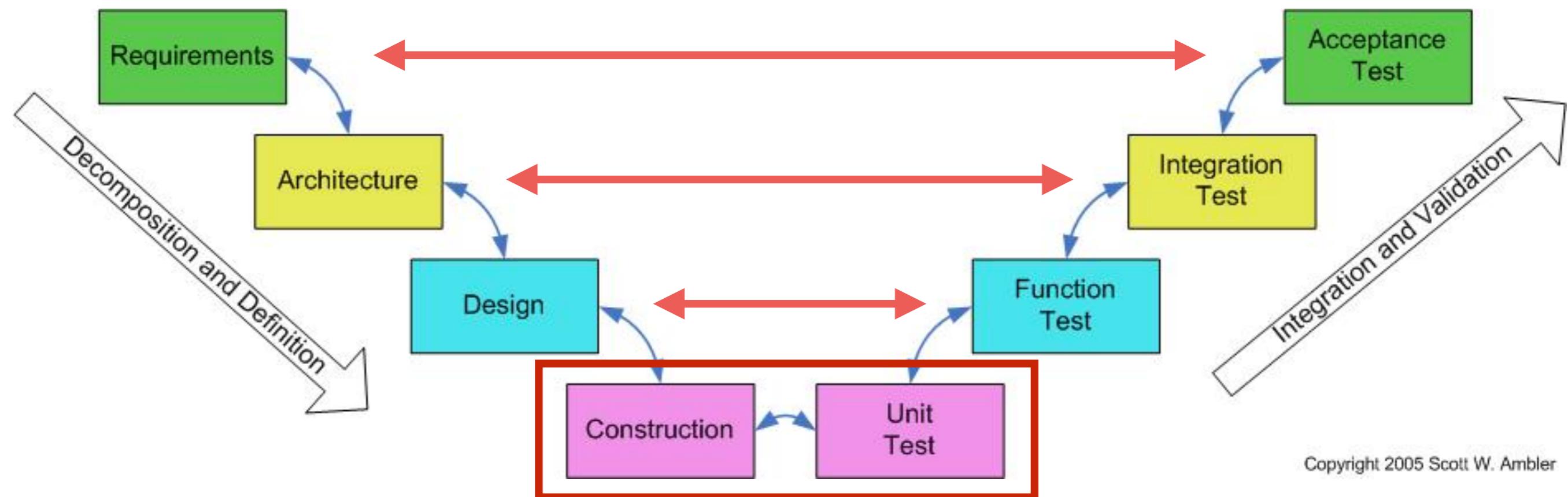
# V Model



# V Model



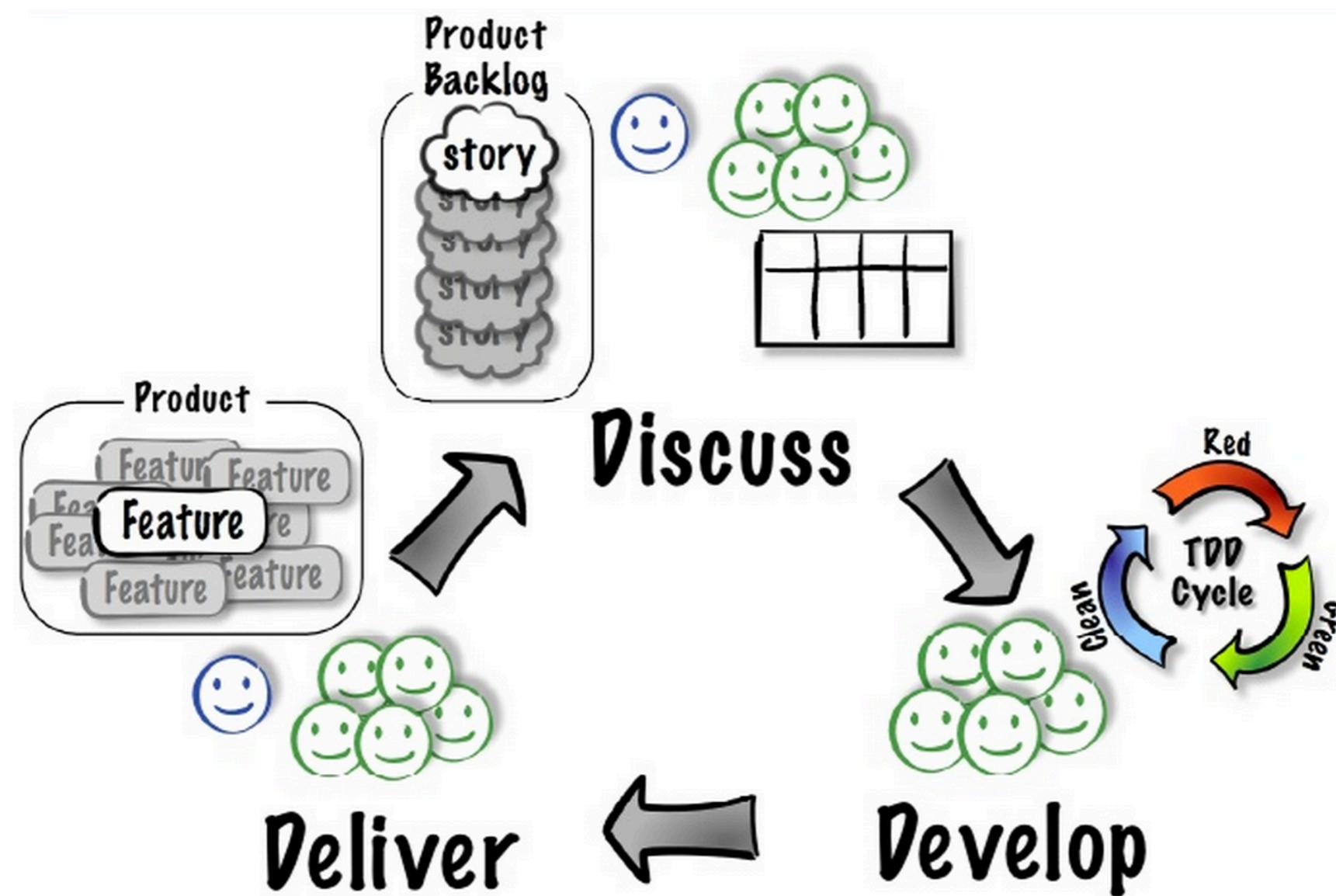
# V Model



# **THINK before coding**



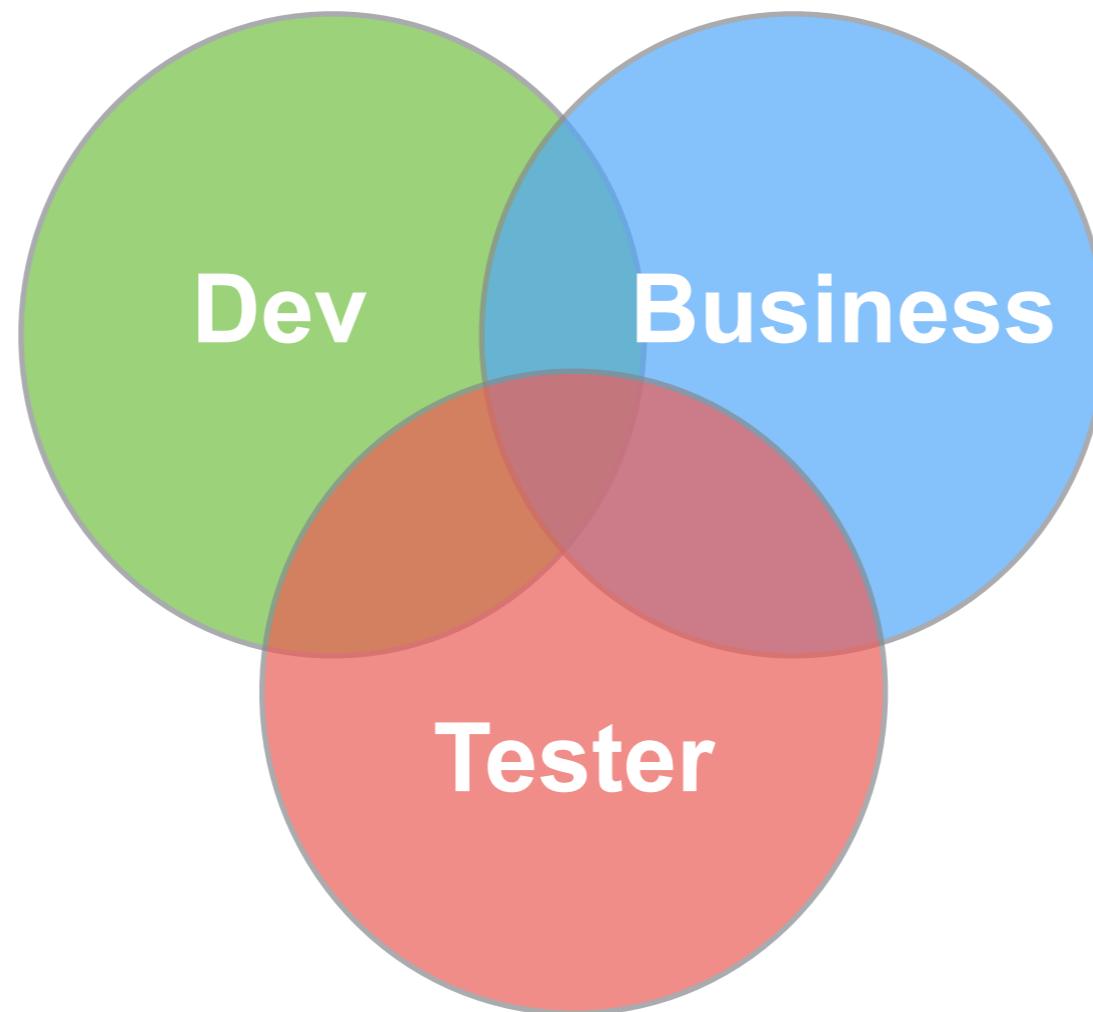
# Acceptance Test-Driven Development



*(Model developed with Pekka Klärck, Bas Vodde, and Craig Larman.)*



# Acceptance Test-Driven Development



# **Acceptance Tests**

=

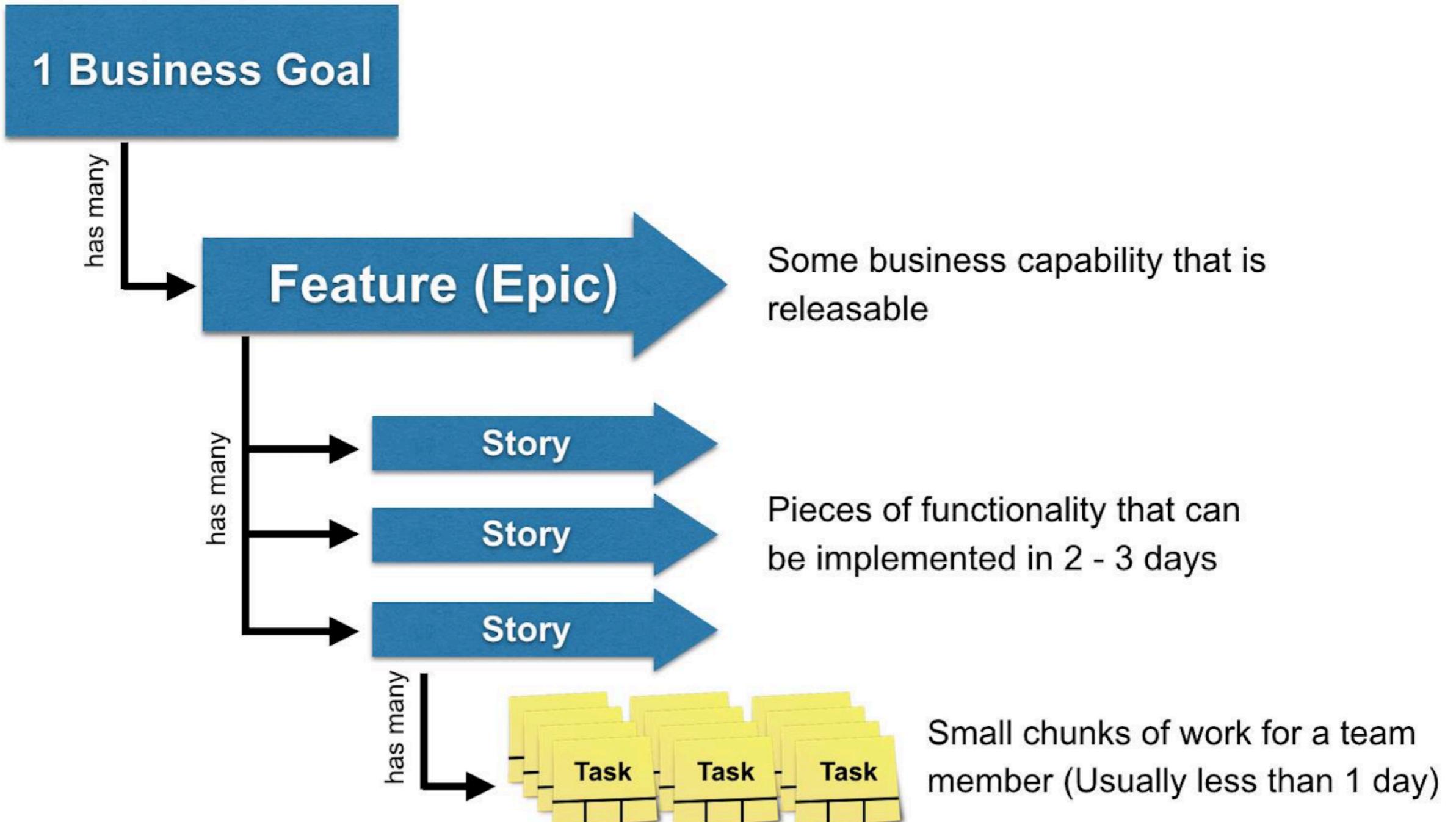
## **Business Criteria**

+

## **Examples (data)**



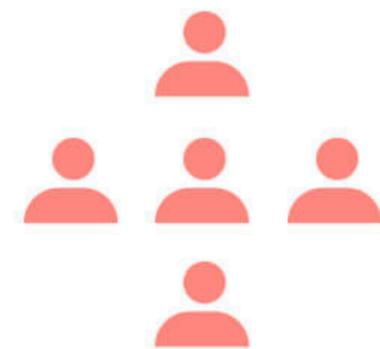
# Work break down



# Whole team approach

## Functional

Common functional expertise



System analysts



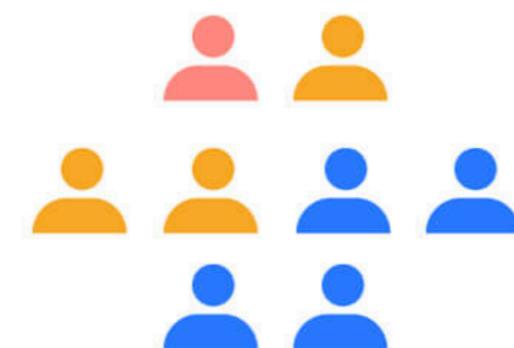
Developers



Testers

## Cross - Functional

Representatives from the various functions



Development Team



# Key success factors

Whole team solve problems

Whole team thinks about testing

Whole team **committed to quality**

Everyone collaborates



# Iterative and incremental process

Feature 1

Time



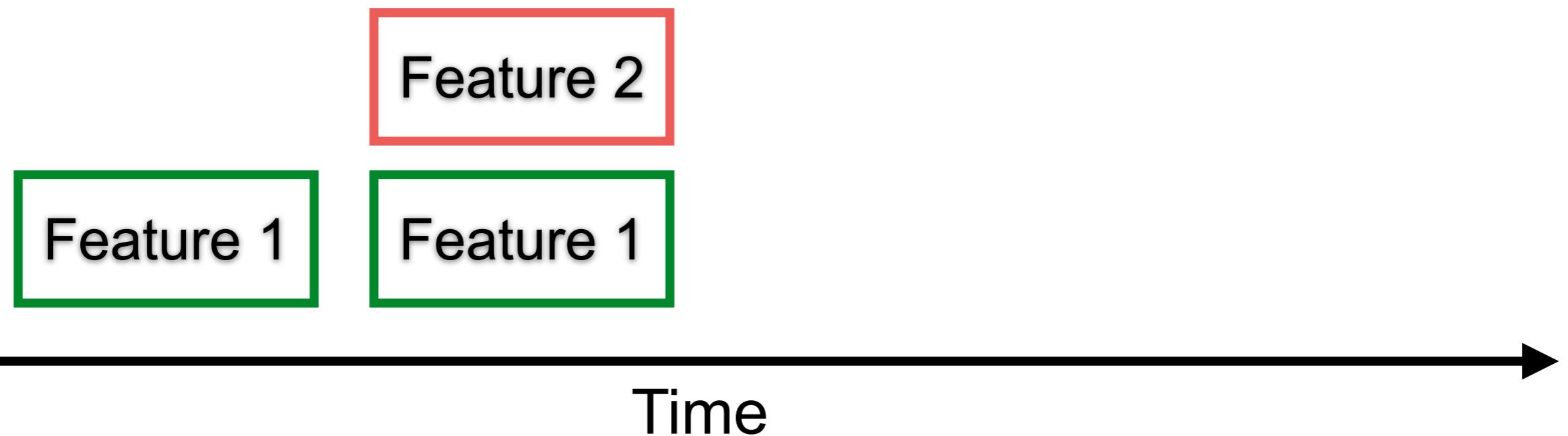
# Iterative and incremental process

Done = coded and tested



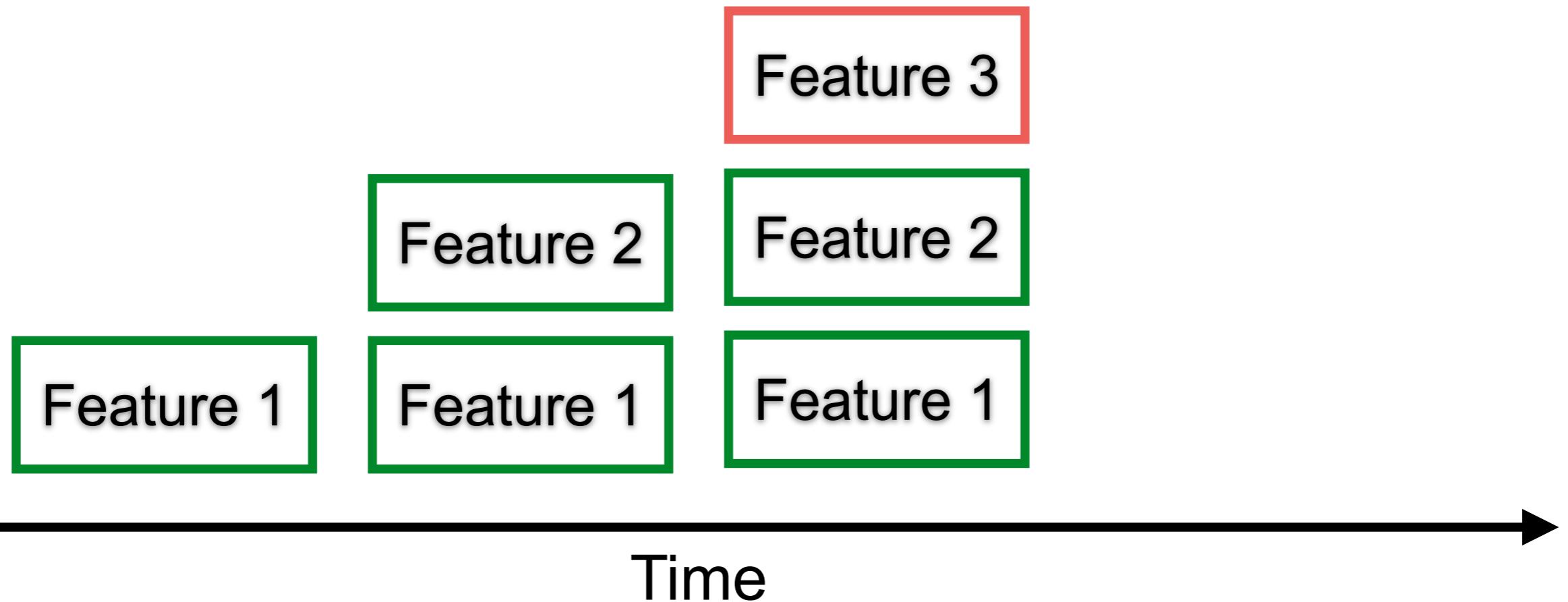
# Iterative and incremental process

Done = coded and tested



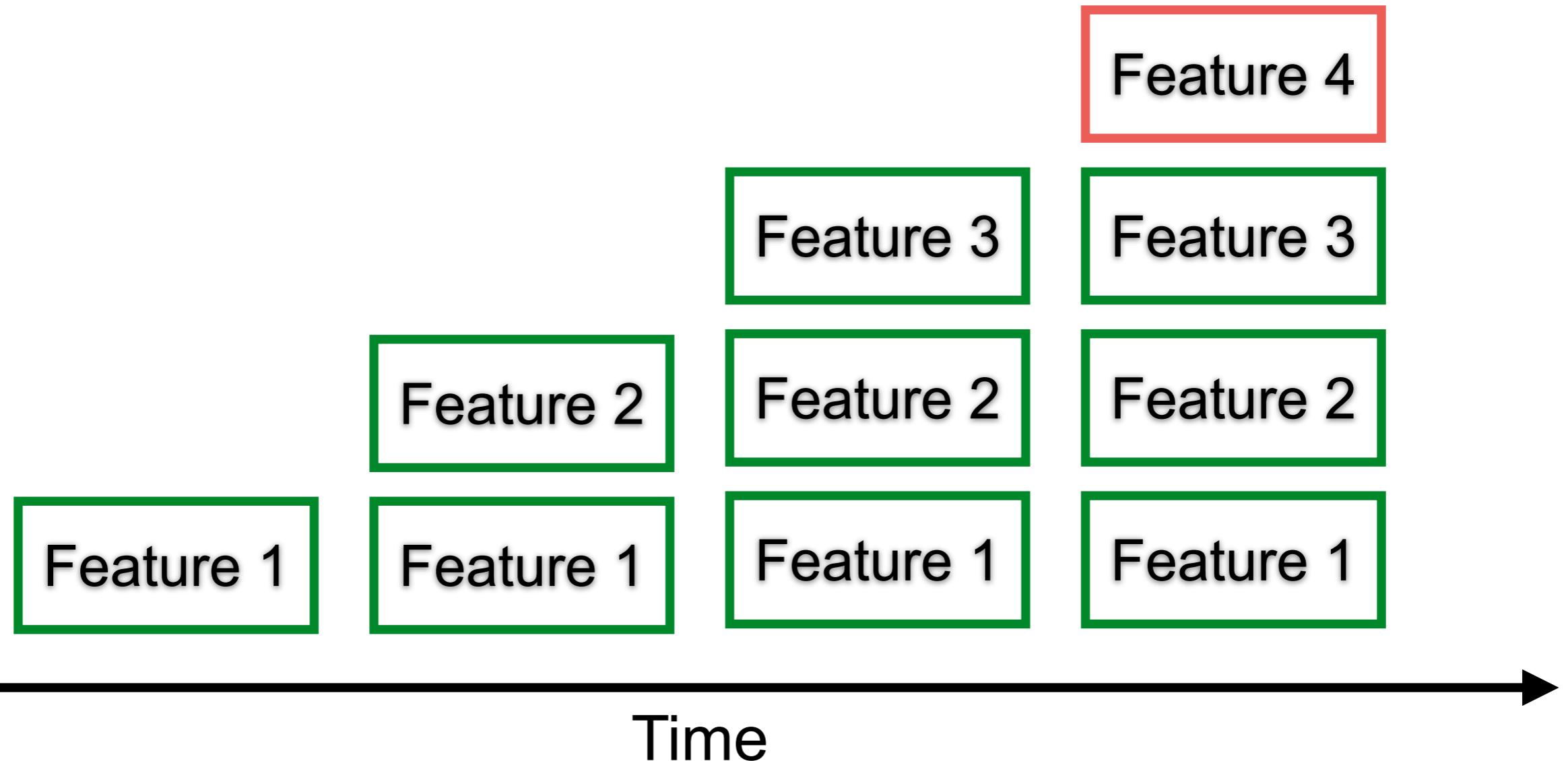
# Iterative and incremental process

Done = coded and tested



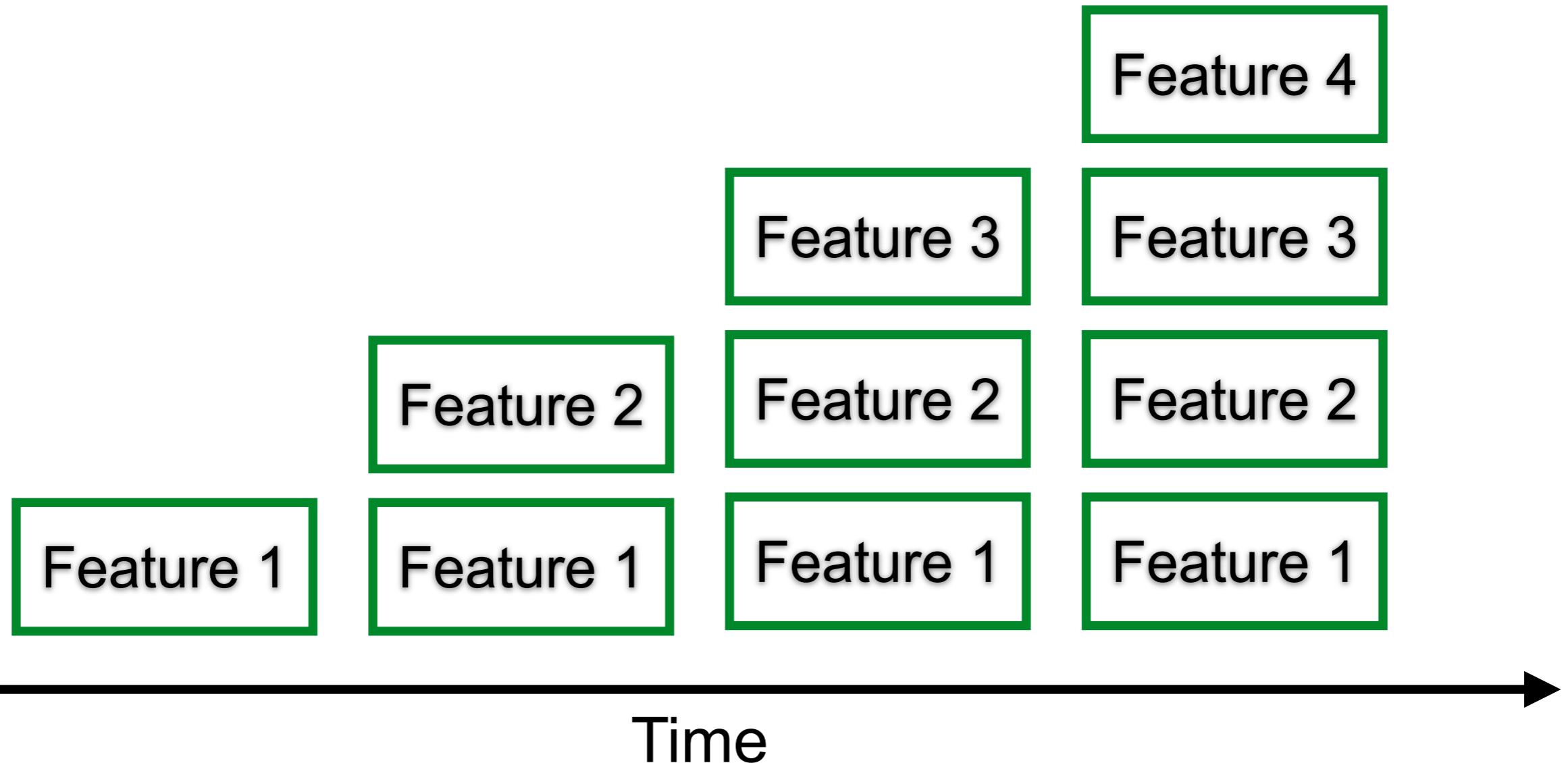
# Iterative and incremental process

Done = coded and tested



# Iterative and incremental process

Done = coded and tested



# Testing is activity

~~Test phase~~

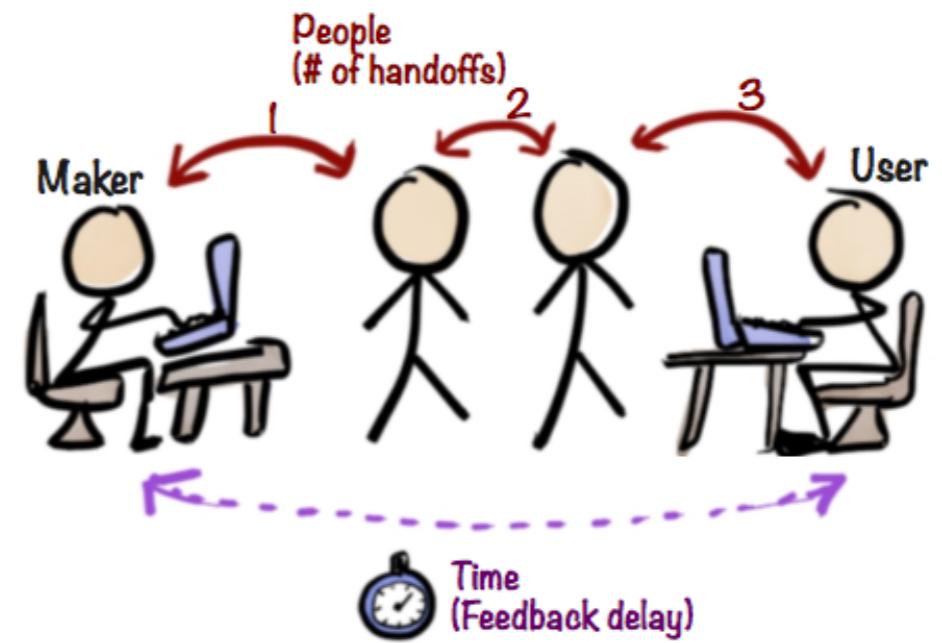
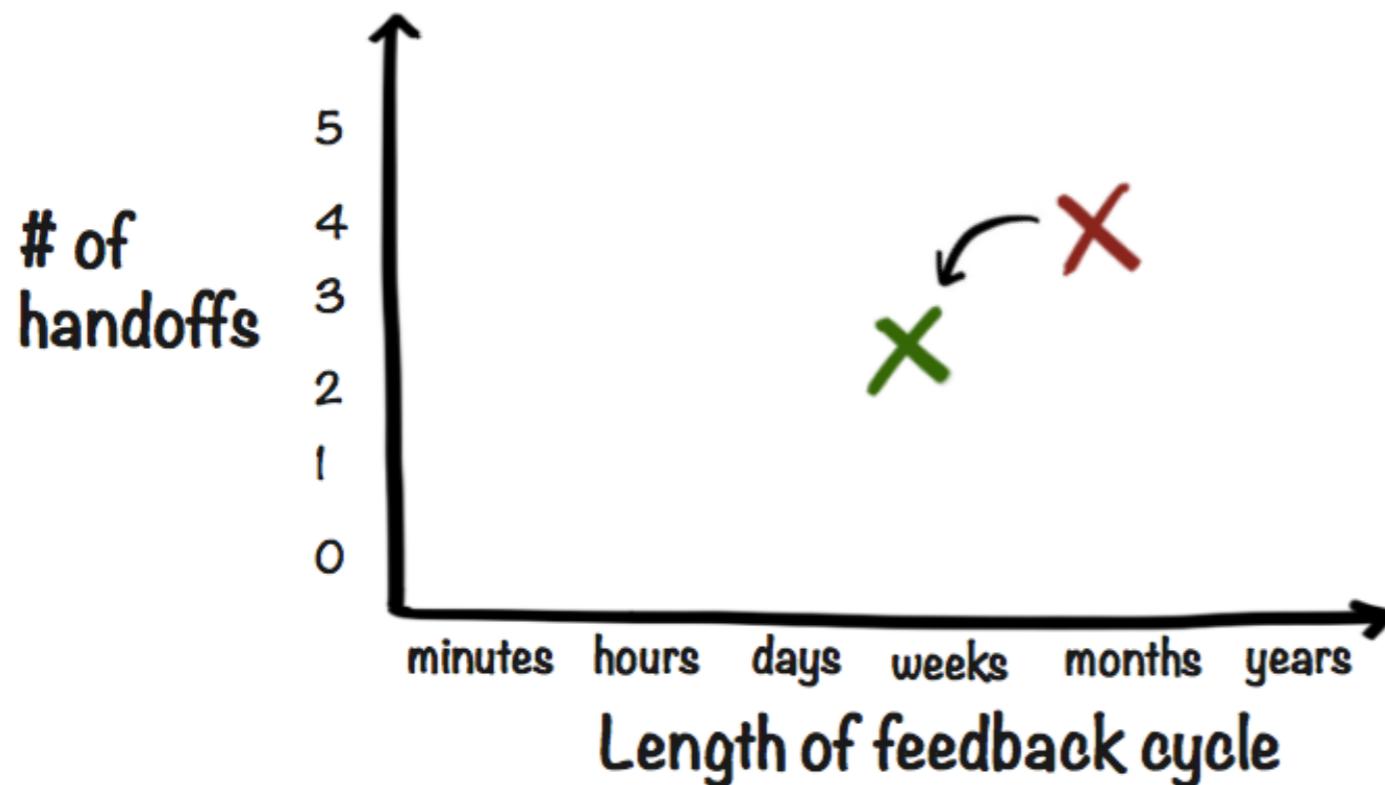
~~Test team~~

~~Tester role~~

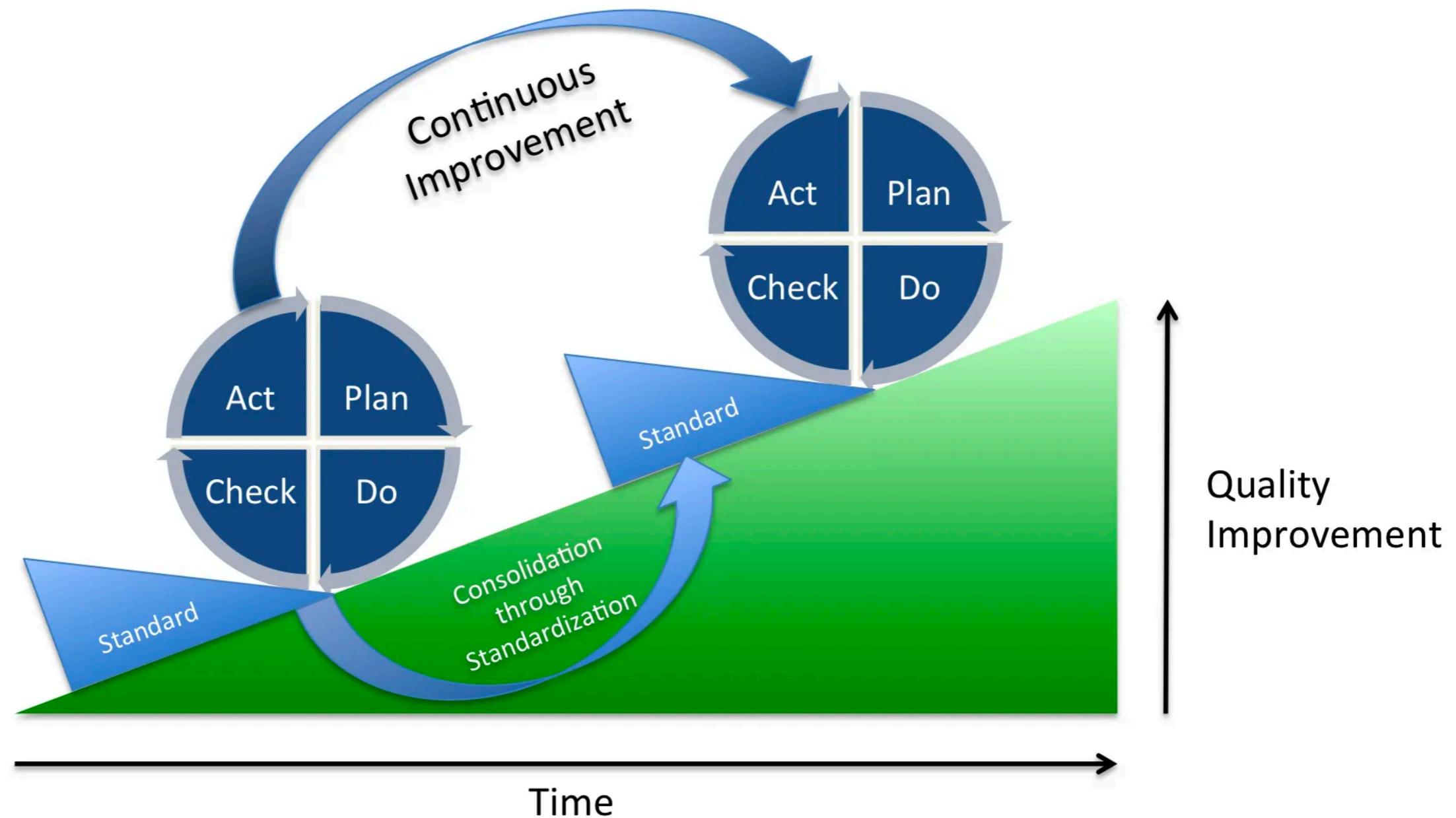


# Fast feedback loop

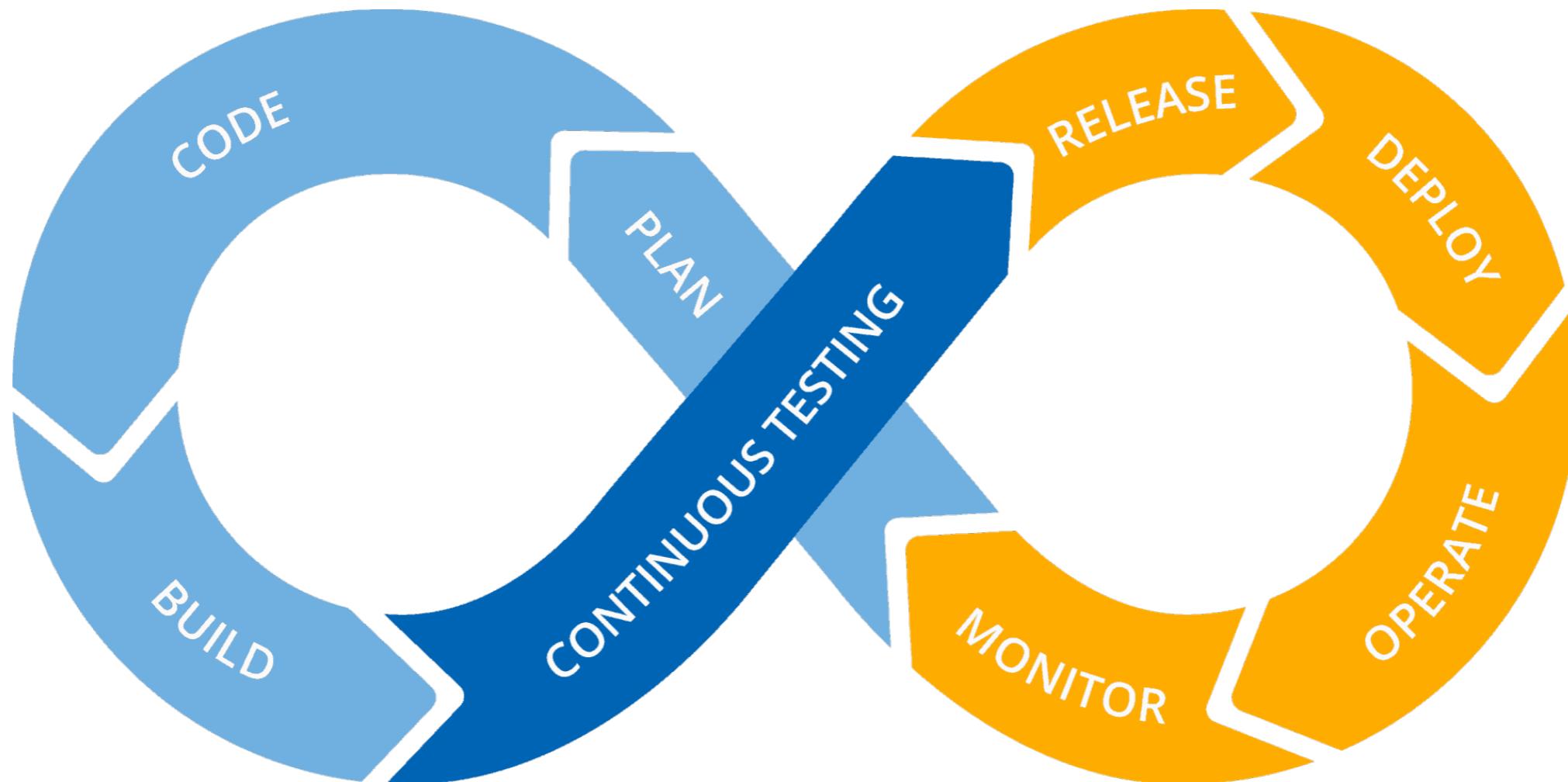
Shorten the feedback loop



# Continuous improvement



# Continuous testing



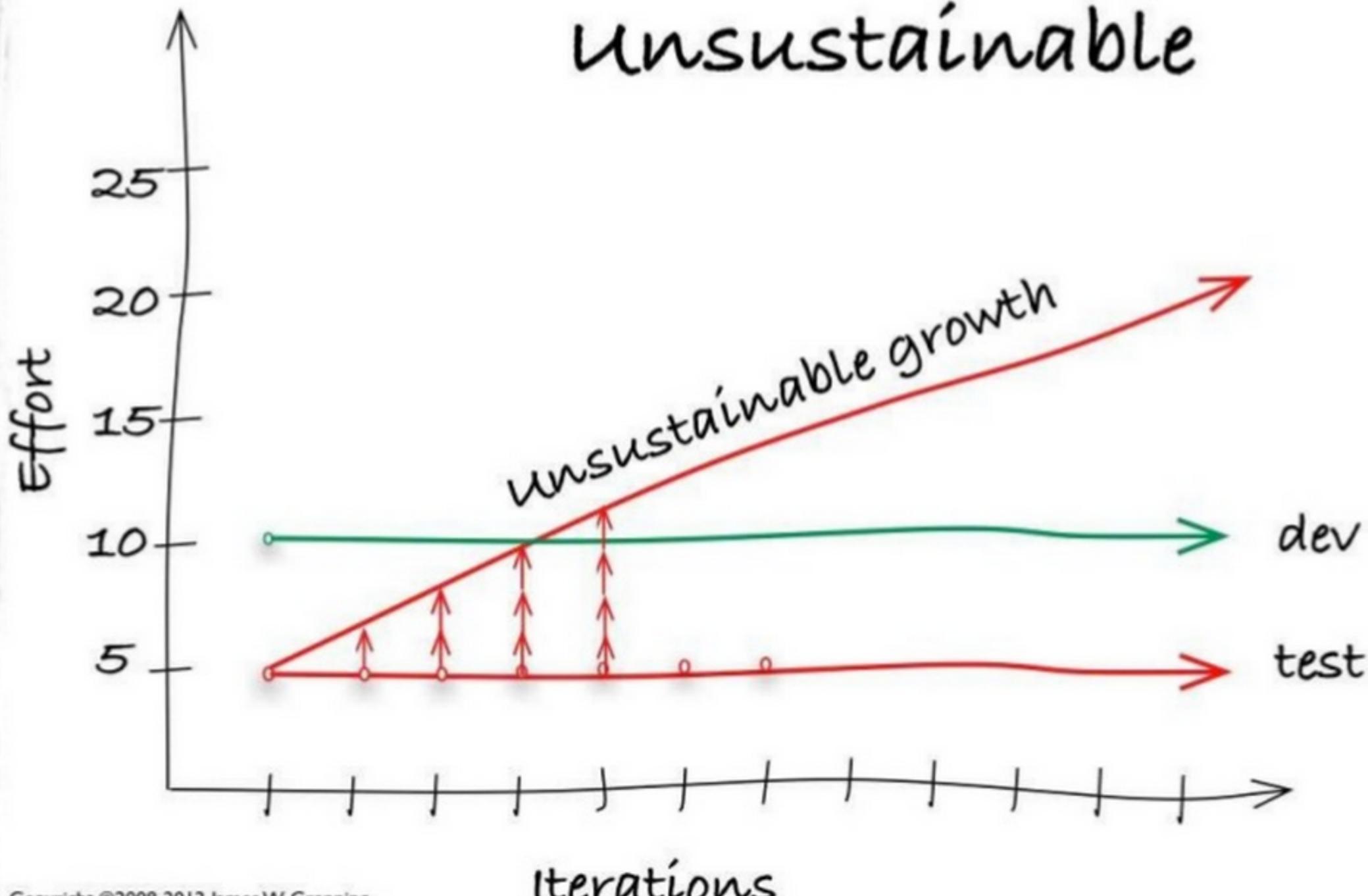
# **But ...**



# Manual testing ?



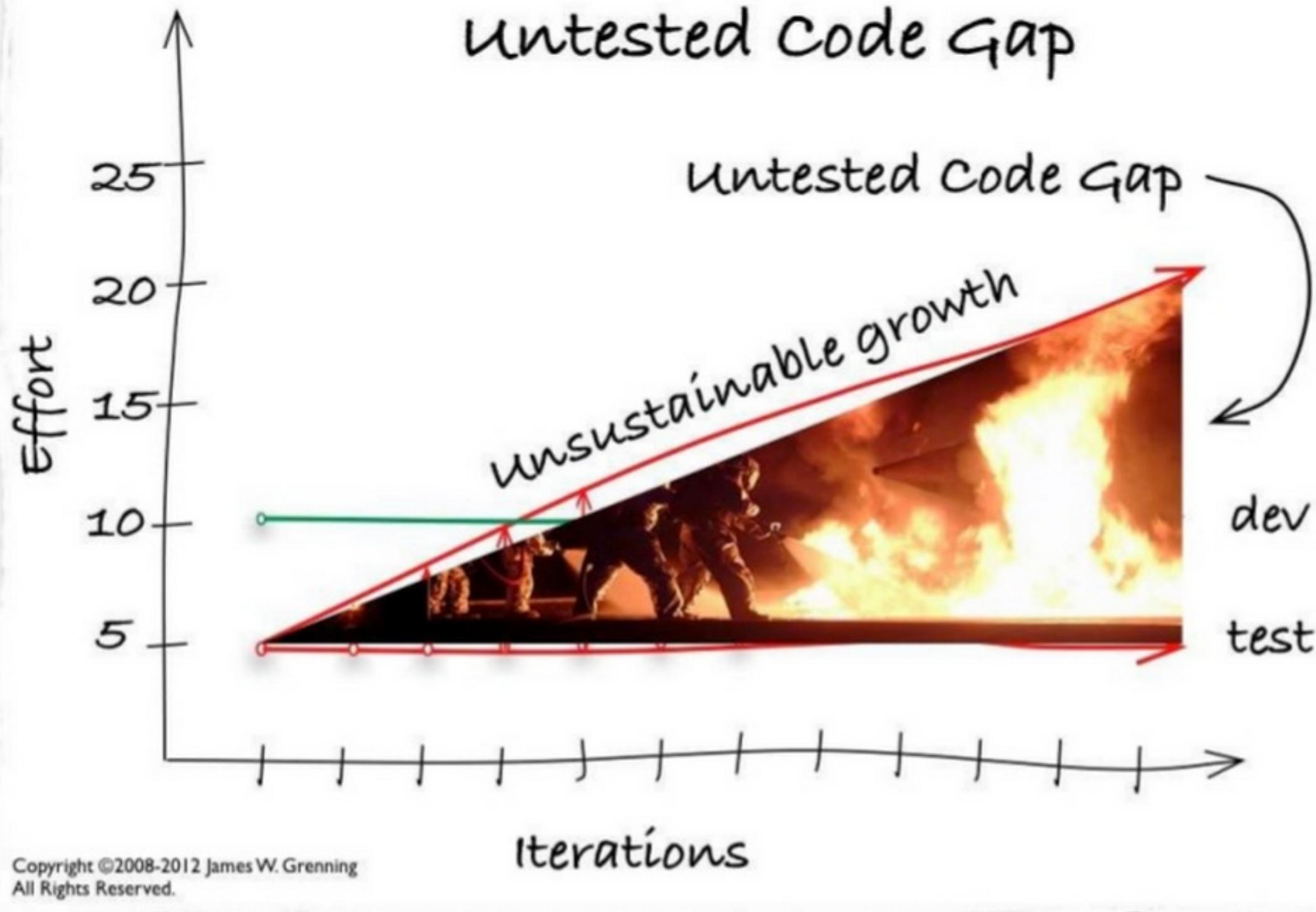
# Manual Test is unsustainable



Copyright ©2008-2012 James W. Grenning  
All Rights Reserved.



# Risk Accumulates in the Untested Code Gap



# We need automation !!



# Why should you automate ?

Manual checking take too long

Manual checks are error prone

Free people to do their best work

Provides living document

Repeatable

Save time



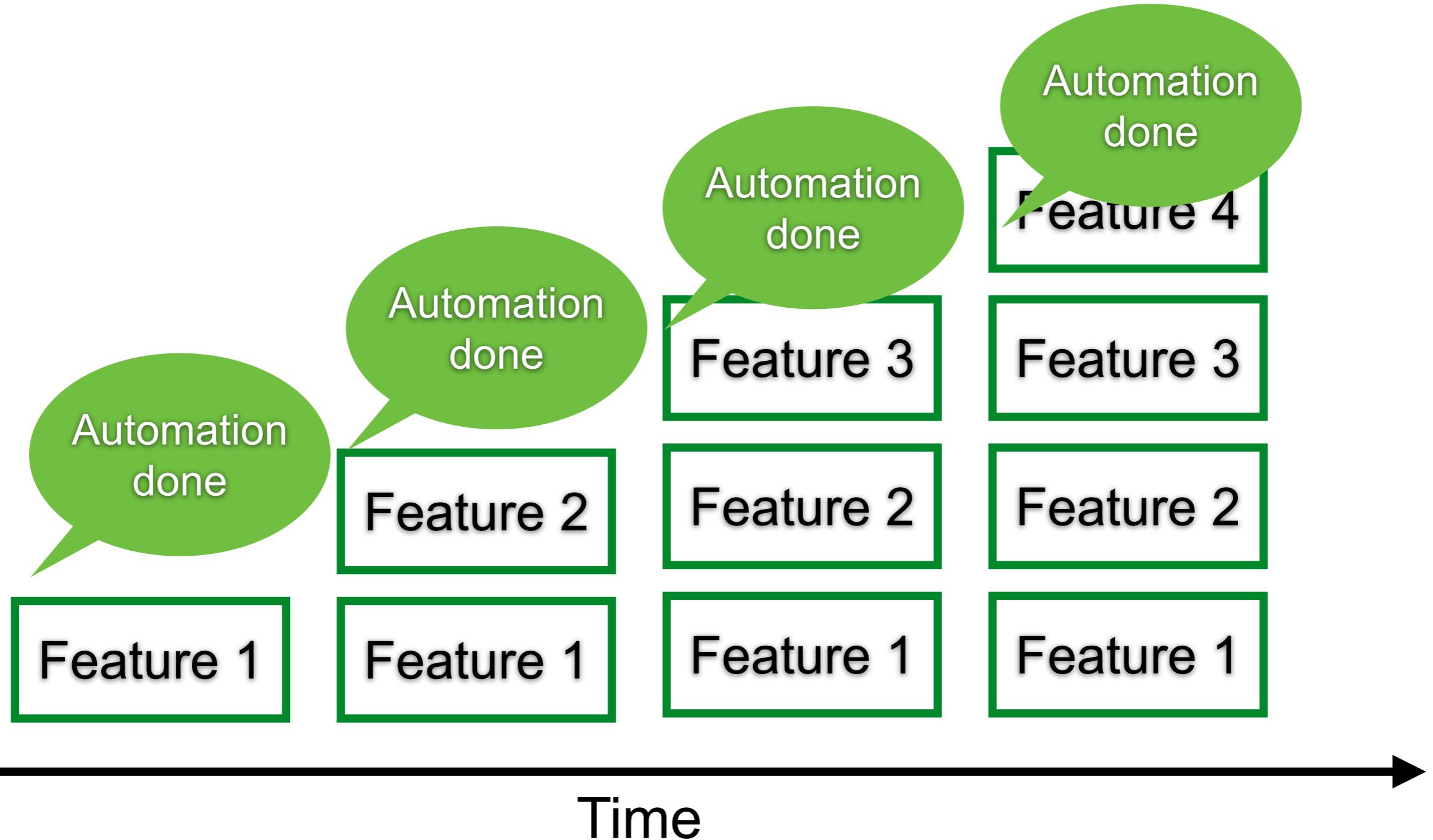
# Workshop

Why aren't you automate ?  
Share obstacles in your group



# Iterative and incremental process

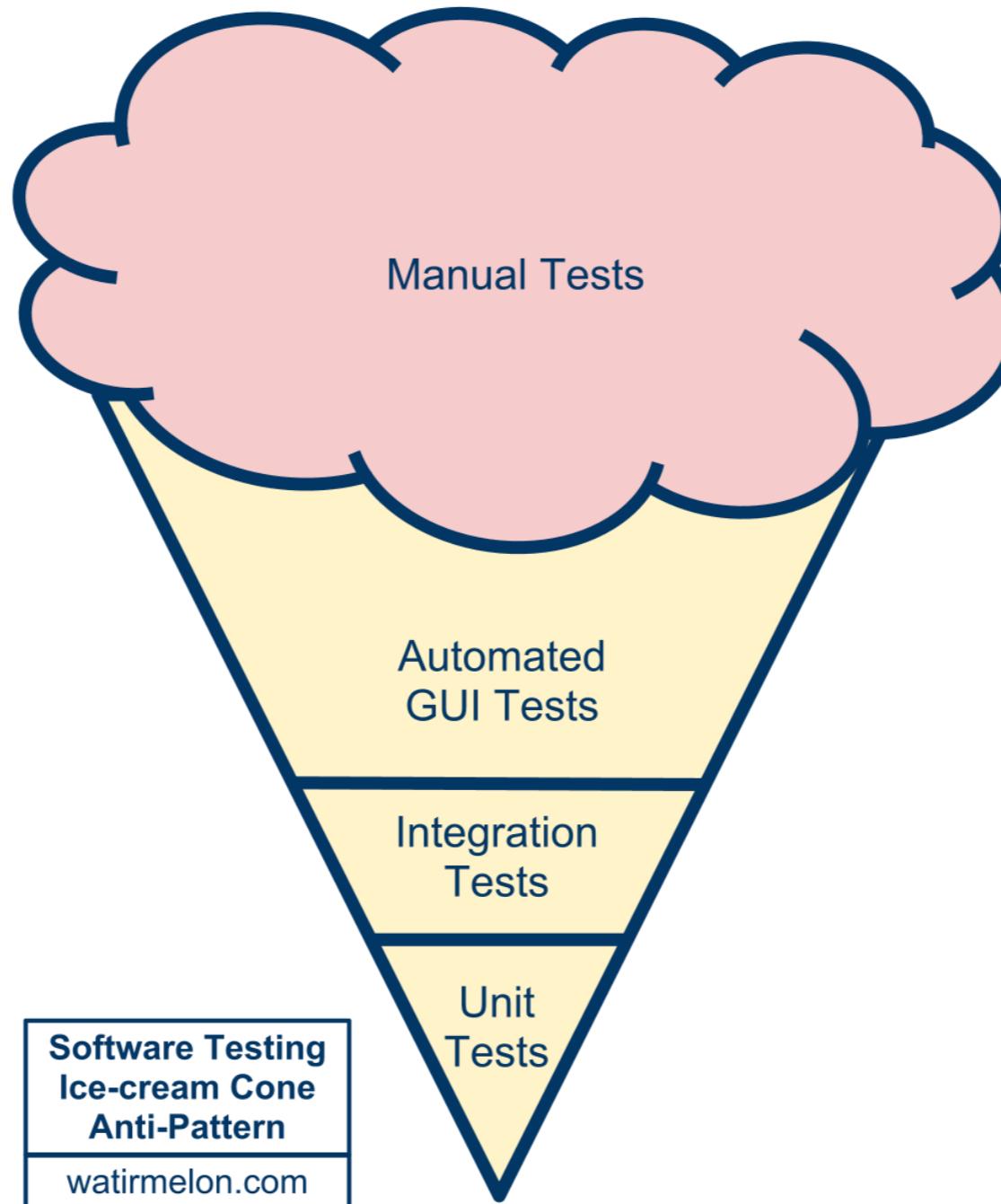
Done = coded and tested



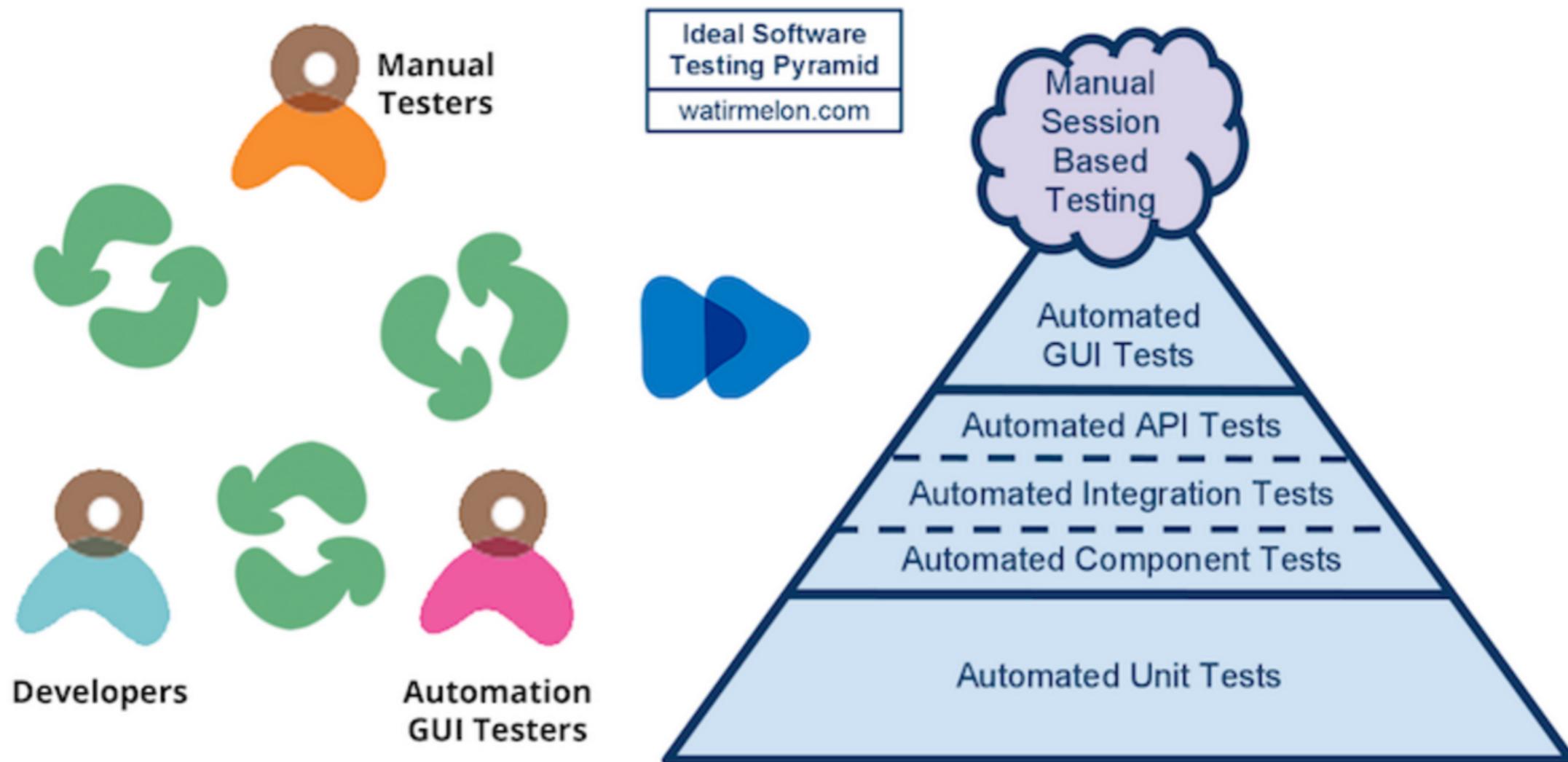
# Testing pyramid



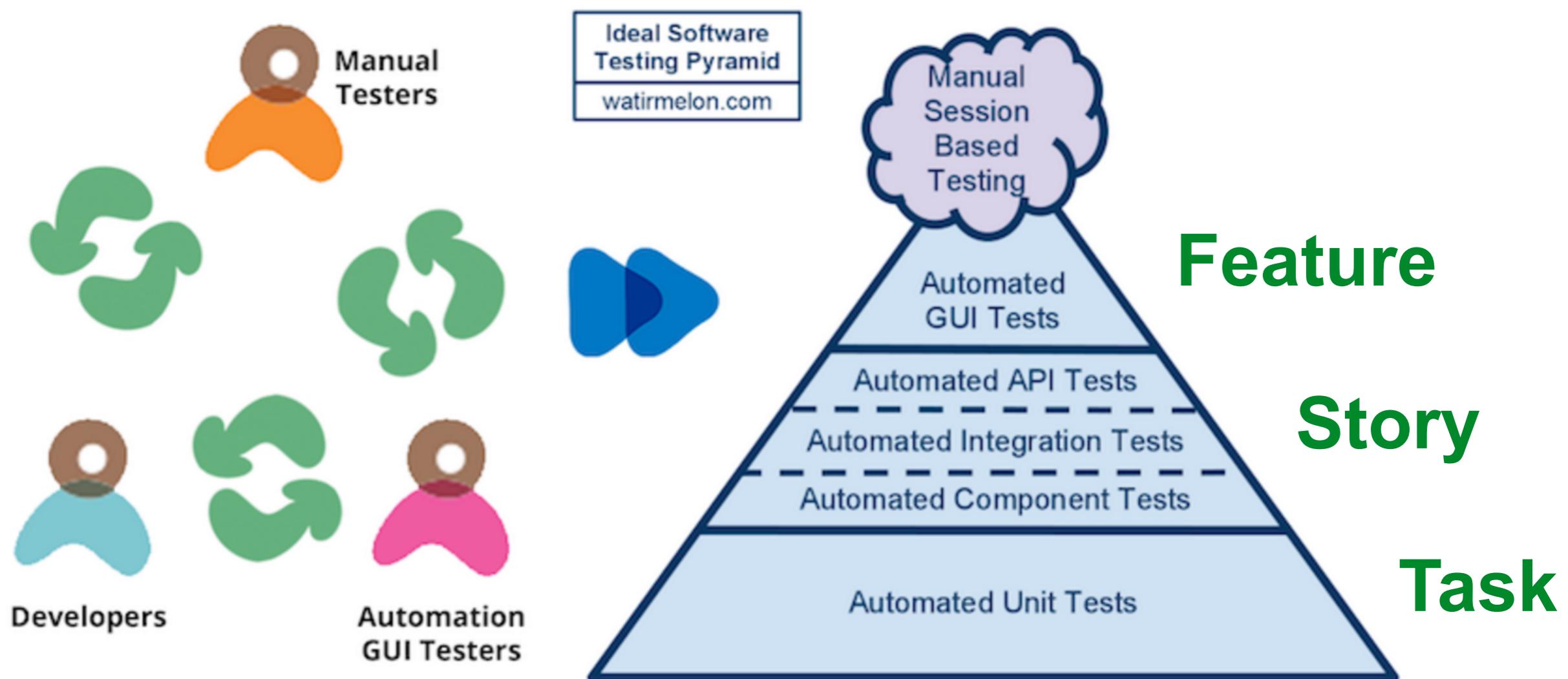
# Ice-cream testing



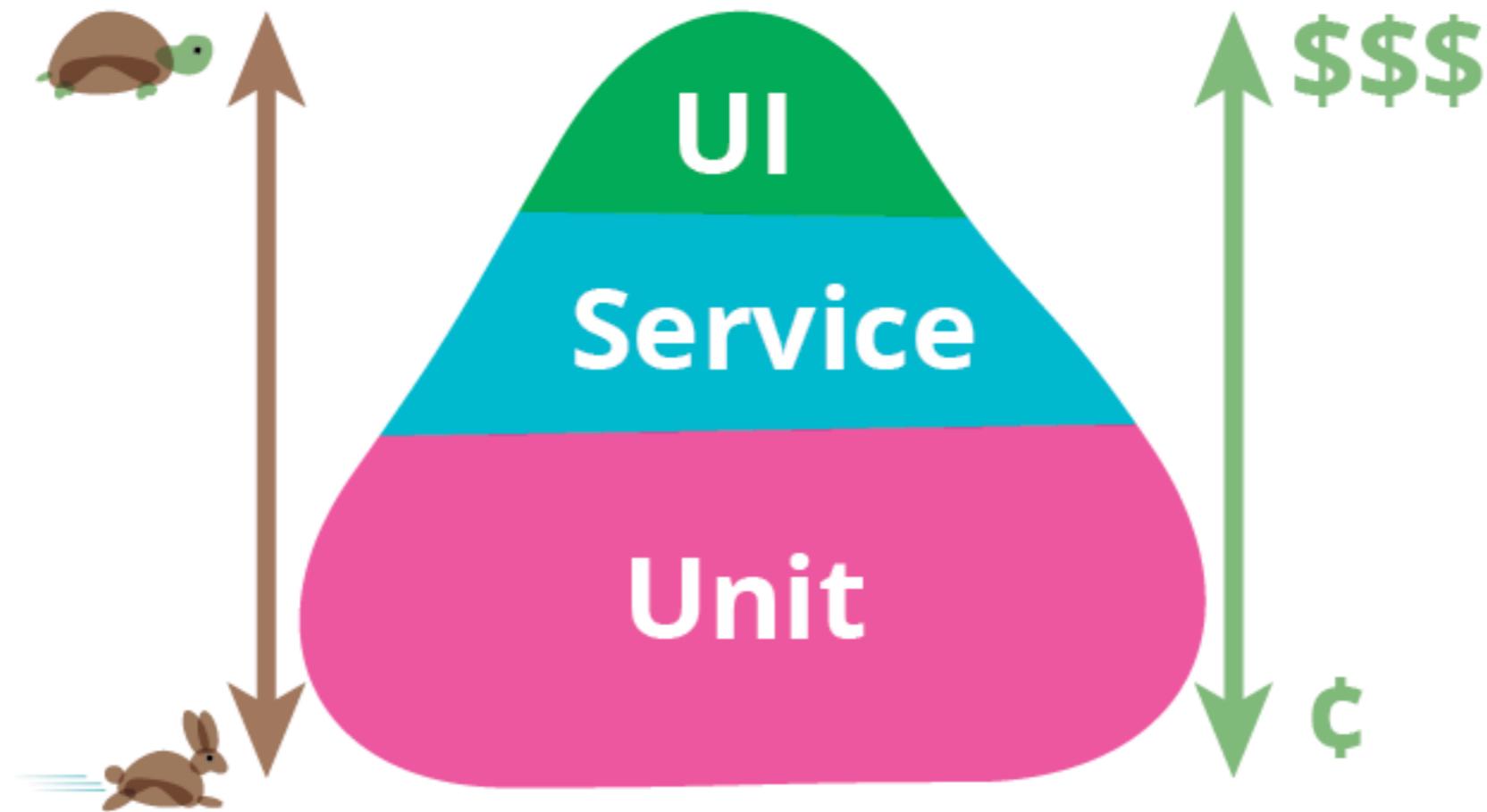
# Testing Pyramid



# Testing Pyramid

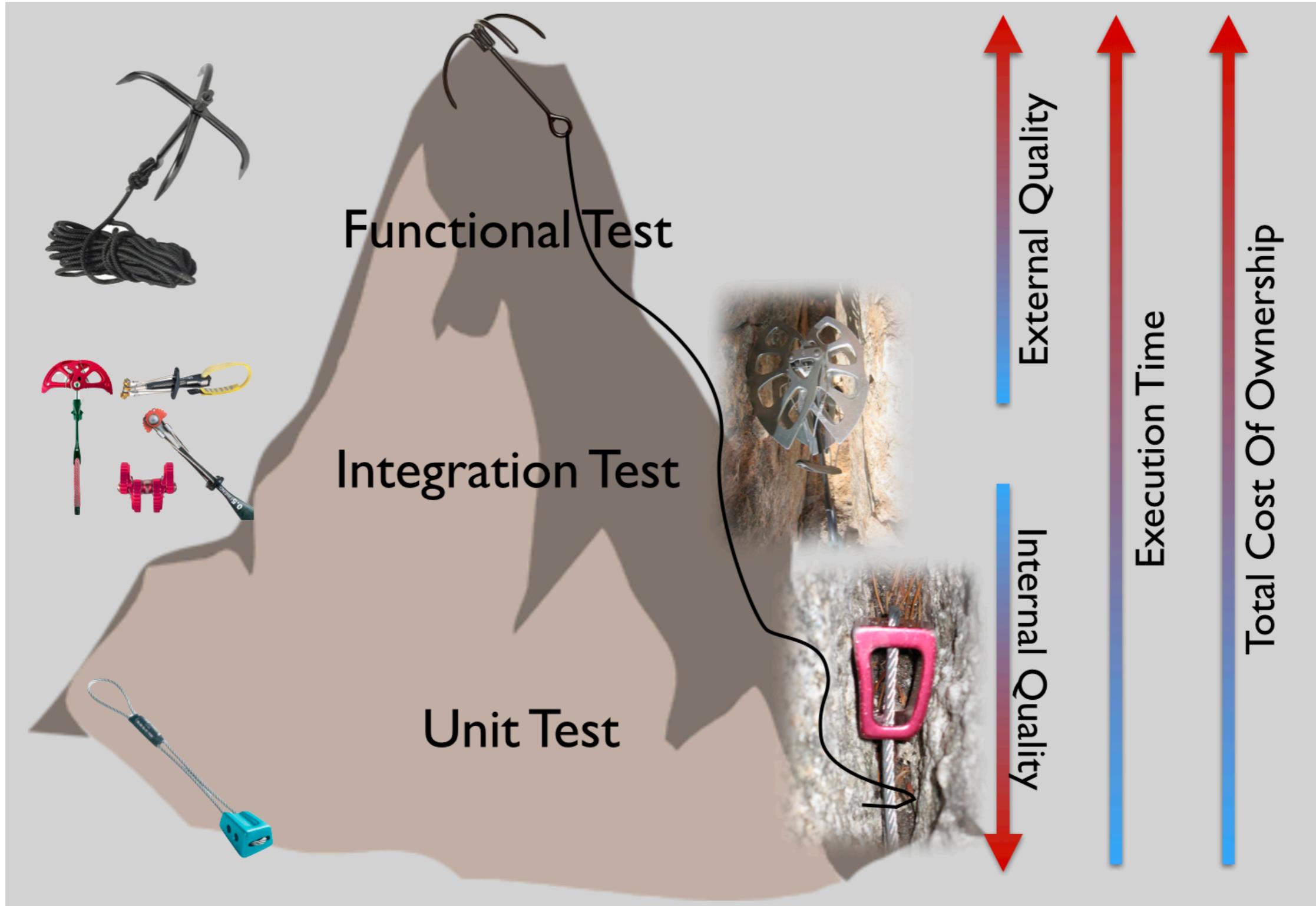


# Testing Pyramid



<https://martinfowler.com/bliki/TestPyramid.html>

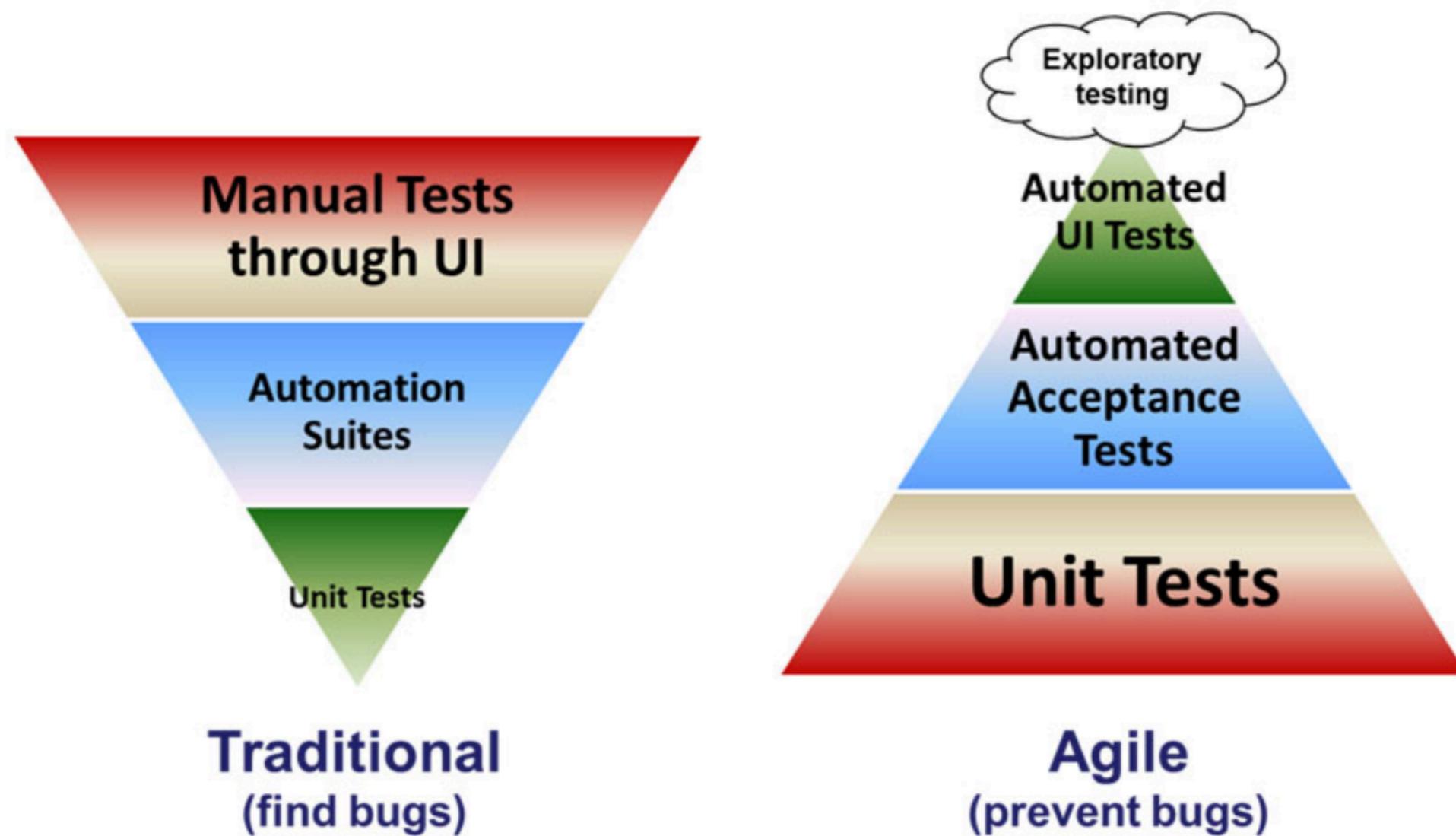




<https://less.works/less/technical-excellence/unit-testing.html>



# Find vs Prevent



<https://martinfowler.com/bliki/TestPyramid.html>



# Mind-set switch

**Instead of**

We are here to **find bug**

We are here to **ensure requirement are met**

We are here to **break the software**



# Mind-set switch

**Instead of**

We are here to **find bug**

We are here to **ensure requirement are met**

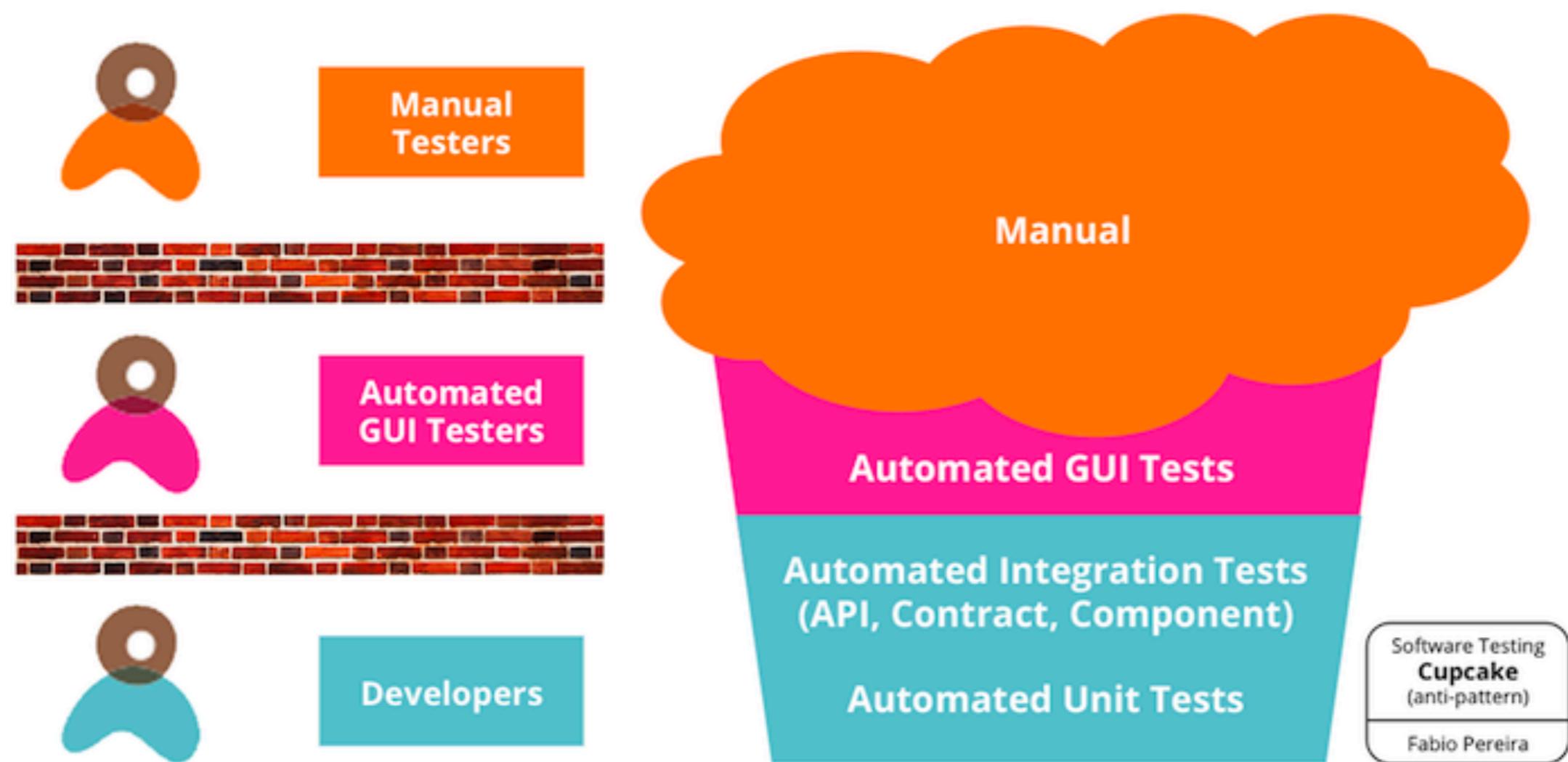
We are here to **break the software**

**Think**

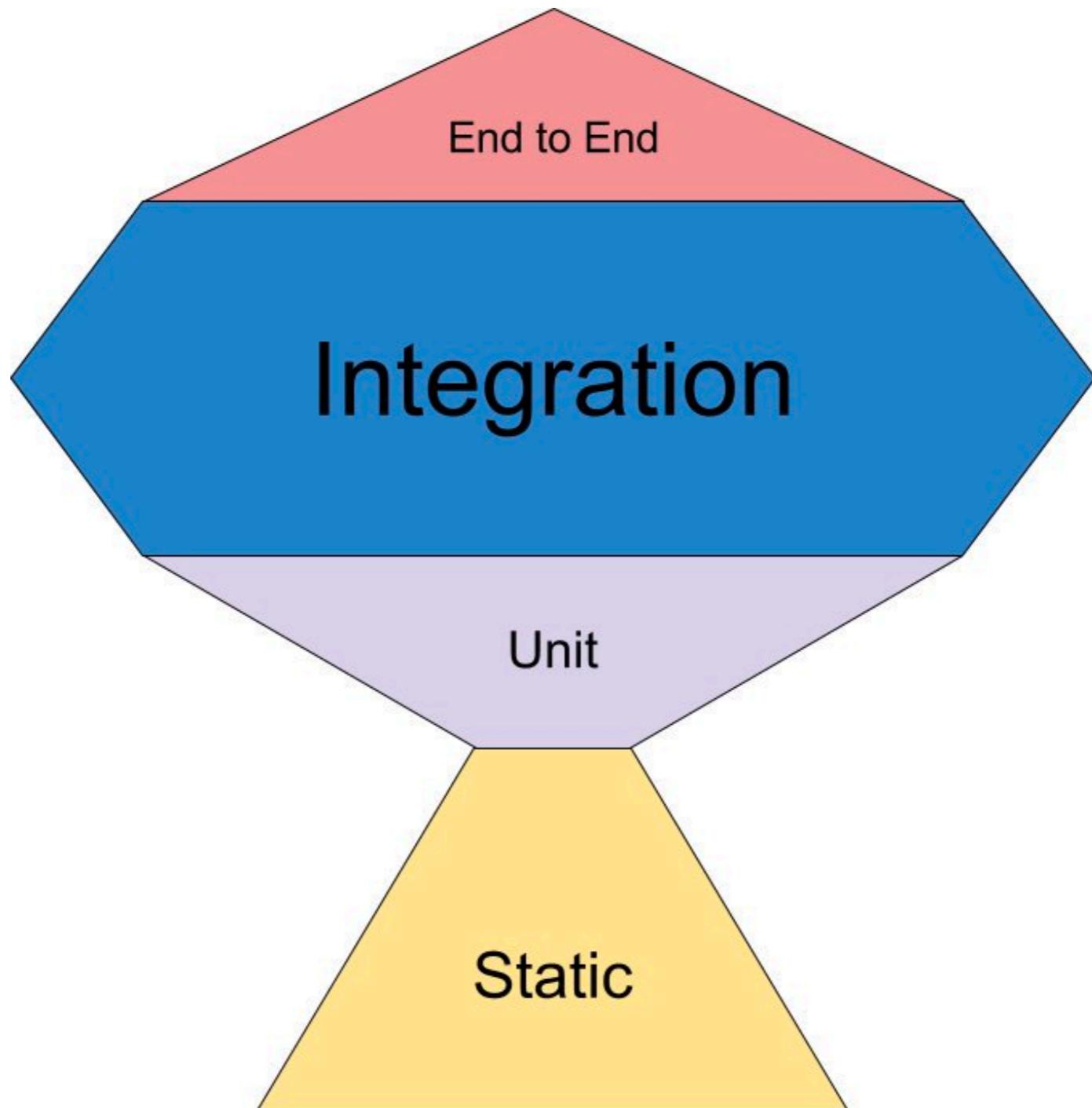
What can I do to help deliver the software  
successfully !!



# Cupcake testing



# Trophy testing



# Workshop Test Pyramid



# Test login

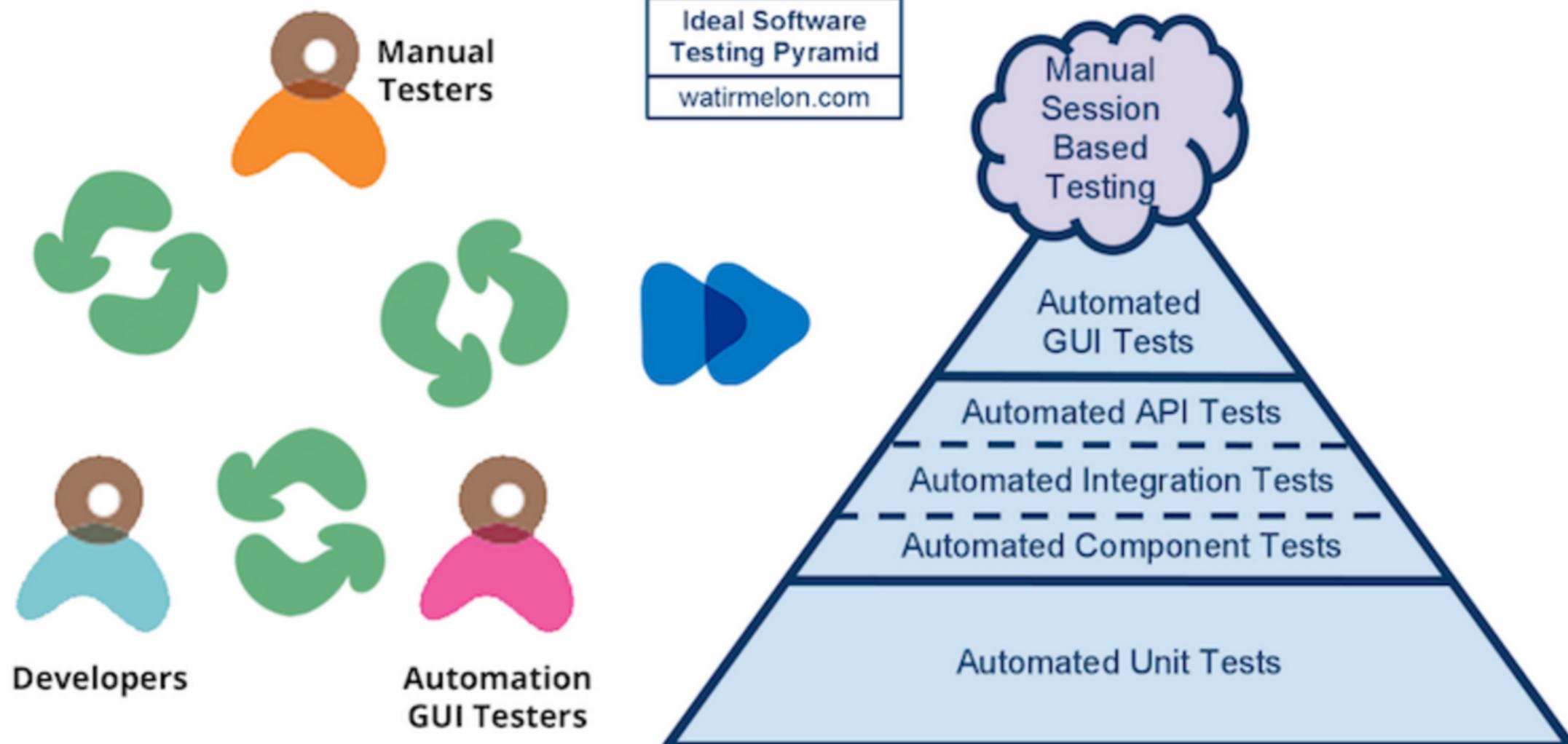
User name	Password	Expected result	Comment
Somkiat	PasswordSomKiat	Access to system as Somkiat	Valid
Som kiat	PasswordSomKiat	Error	Space in user
Somkiat	Password SomKiat	Error	Space in password
Somkiat	1234	Error	Password too short



# Workshop

1. Draw a test pyramid
2. Identify tests at each level
3. Think about value (business and customer)
4. What tests should be manual ?





# **Understand What and How to test ?**

**Discussion is very important**



# Remember !!

Test pyramid is a tool  
To talk about automation tasks  
How to prioritise and help to do automation ?  
Way to make **visible to the whole team**



# Where to start ?



What are the **bigest**  
**obstacles** ?

**Time/Tools/System/People**



# What should be careful ?

- Automating end-to-end tests
- User Interface are slow
- Working with database
- Working with external system
- Automating every paths



# Testing Quadrant



# Testing Quadrants

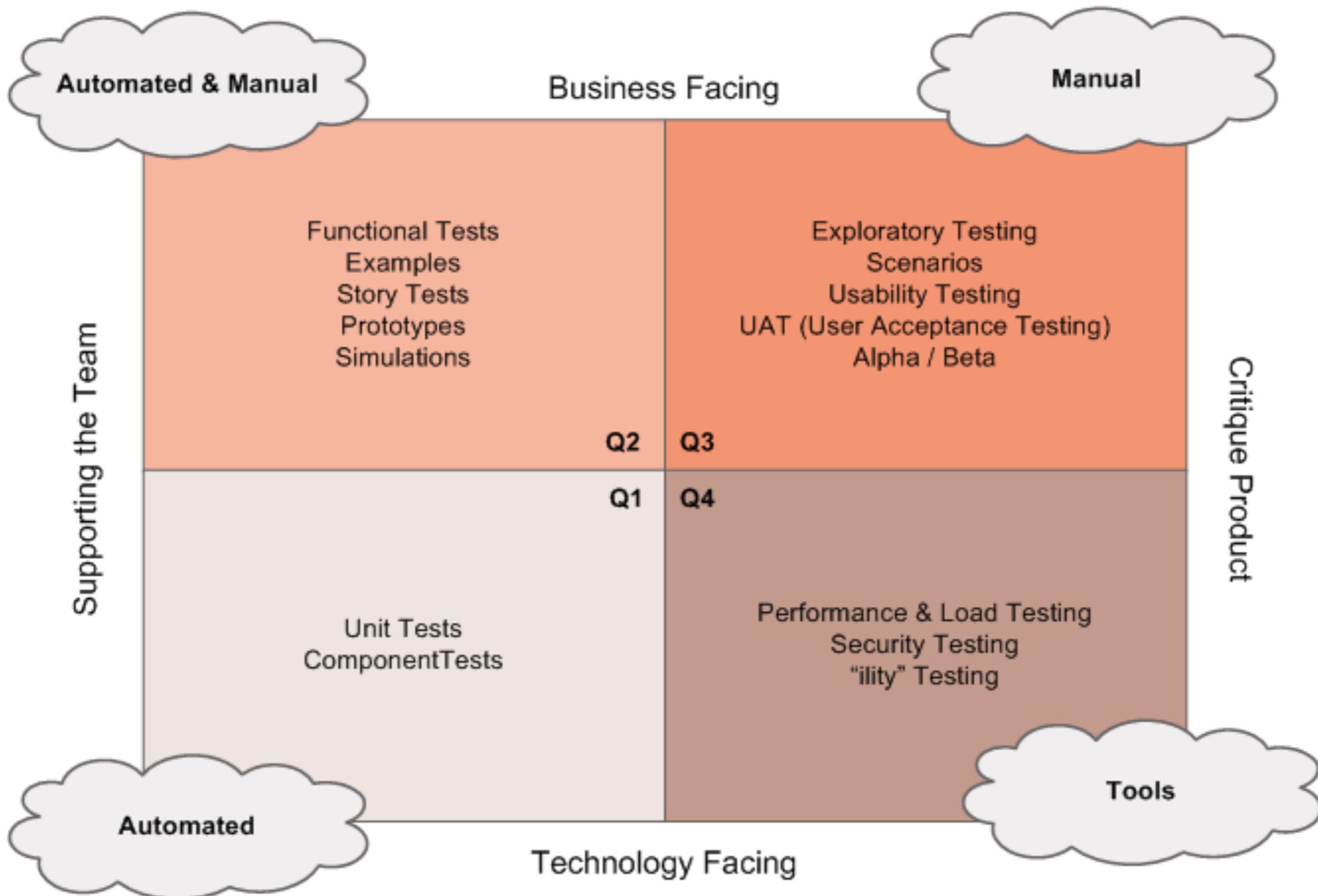
Scope of tests

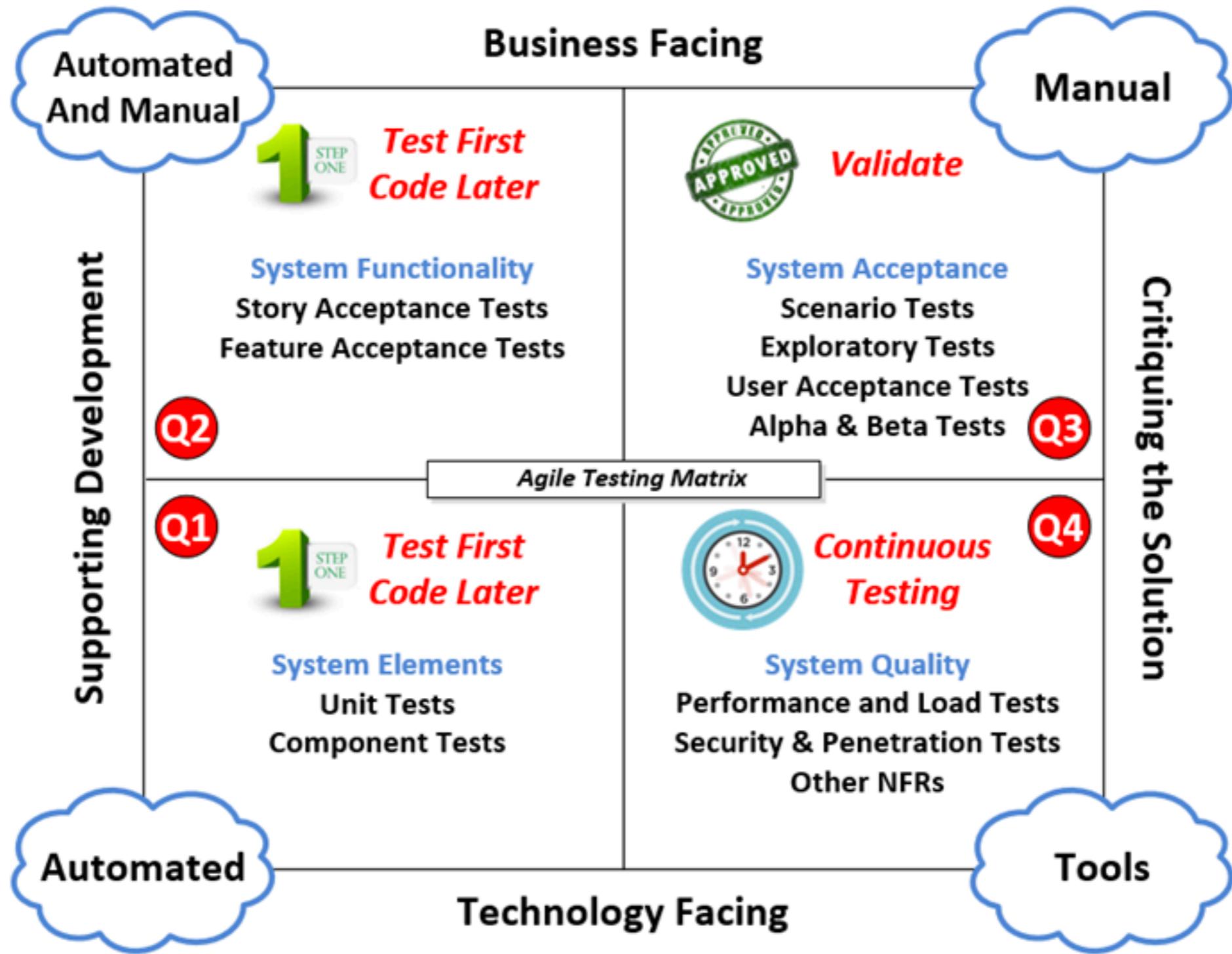
Classify your tests

Visible your tests

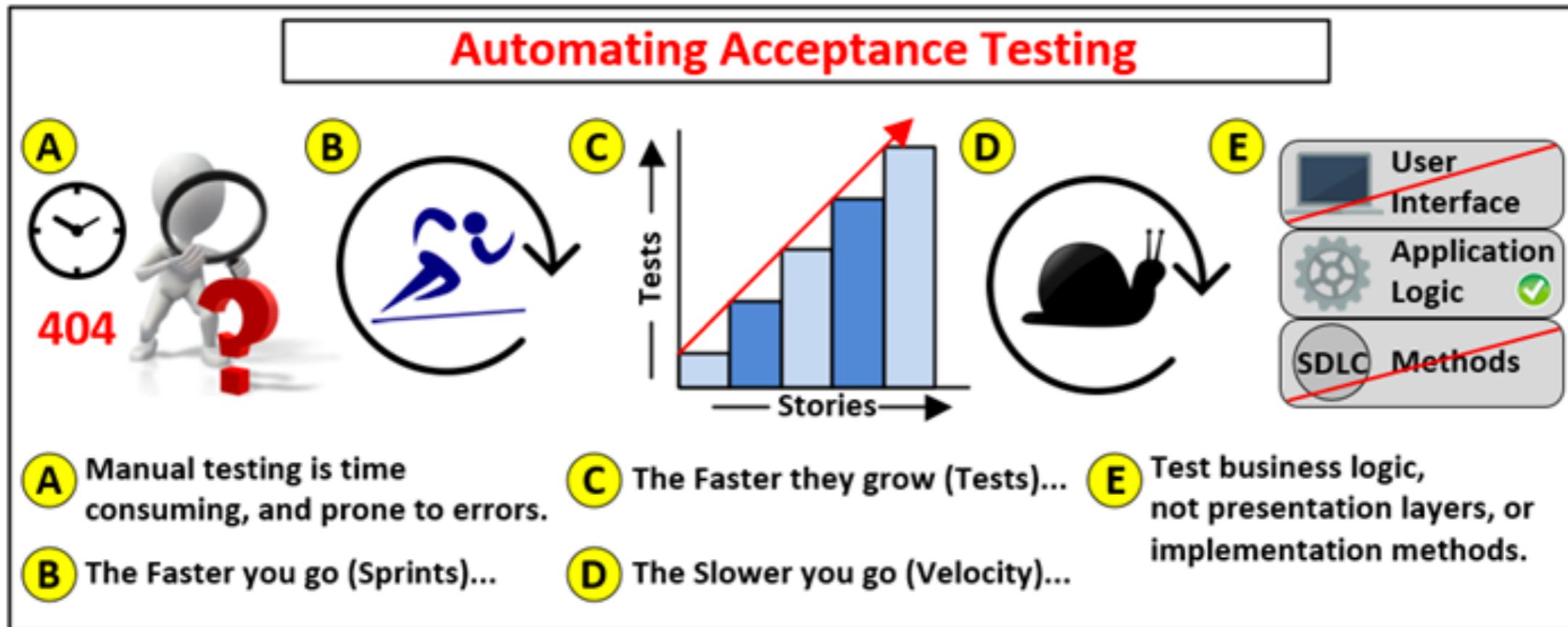


## Agile Testing Quadrants





# Test automation is essential

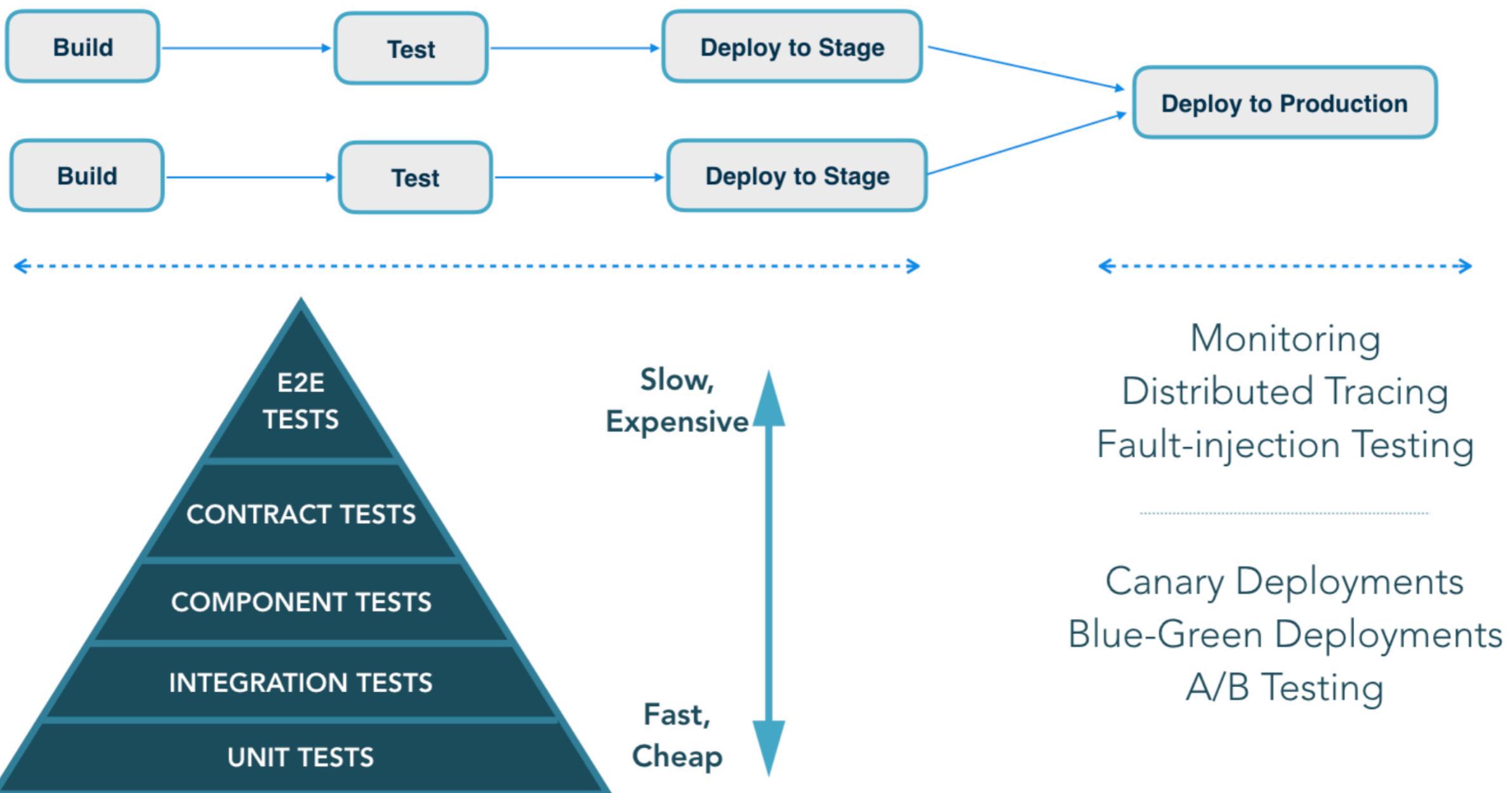


# Test strategy

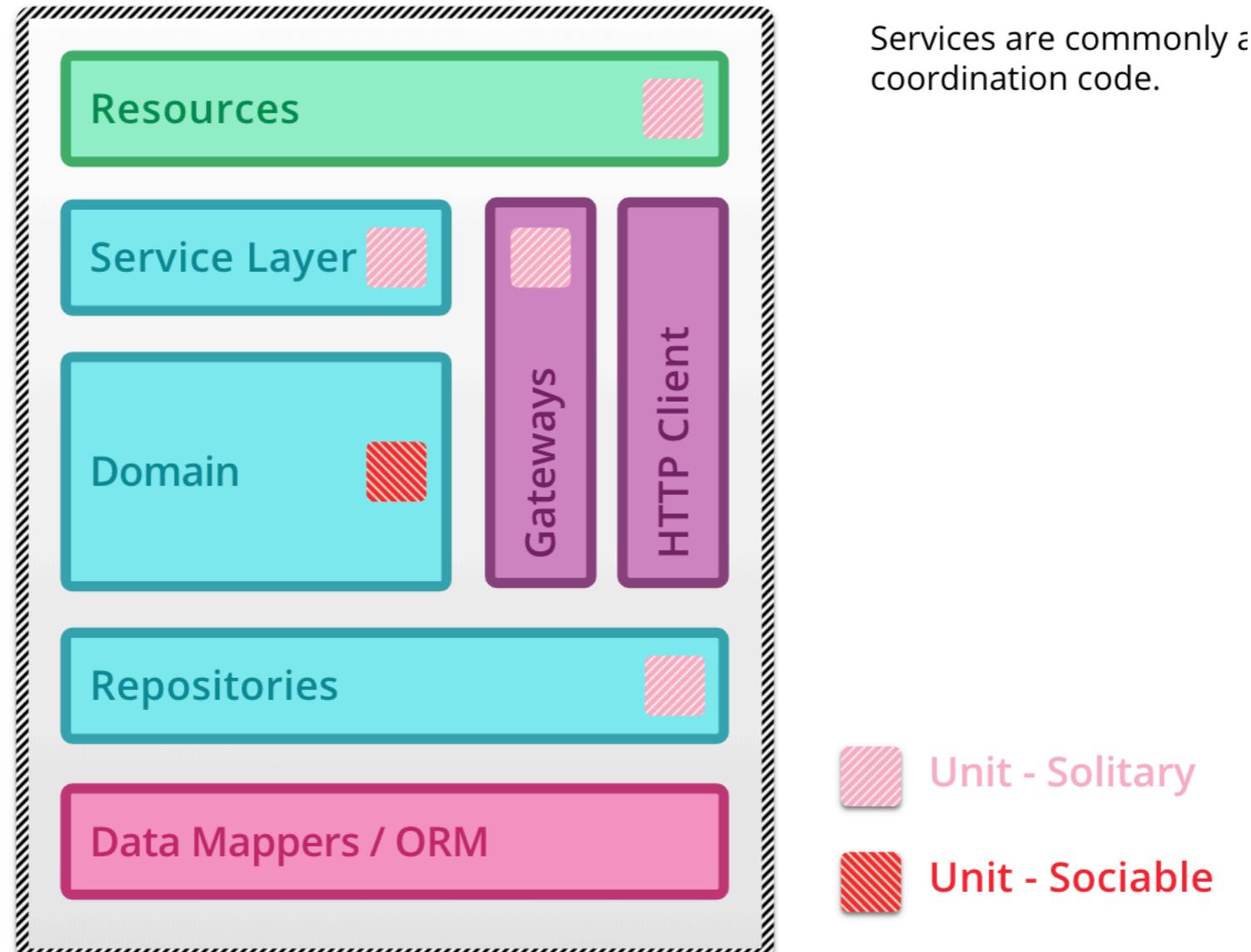
<https://martinfowler.com/articles/microservice-testing/>



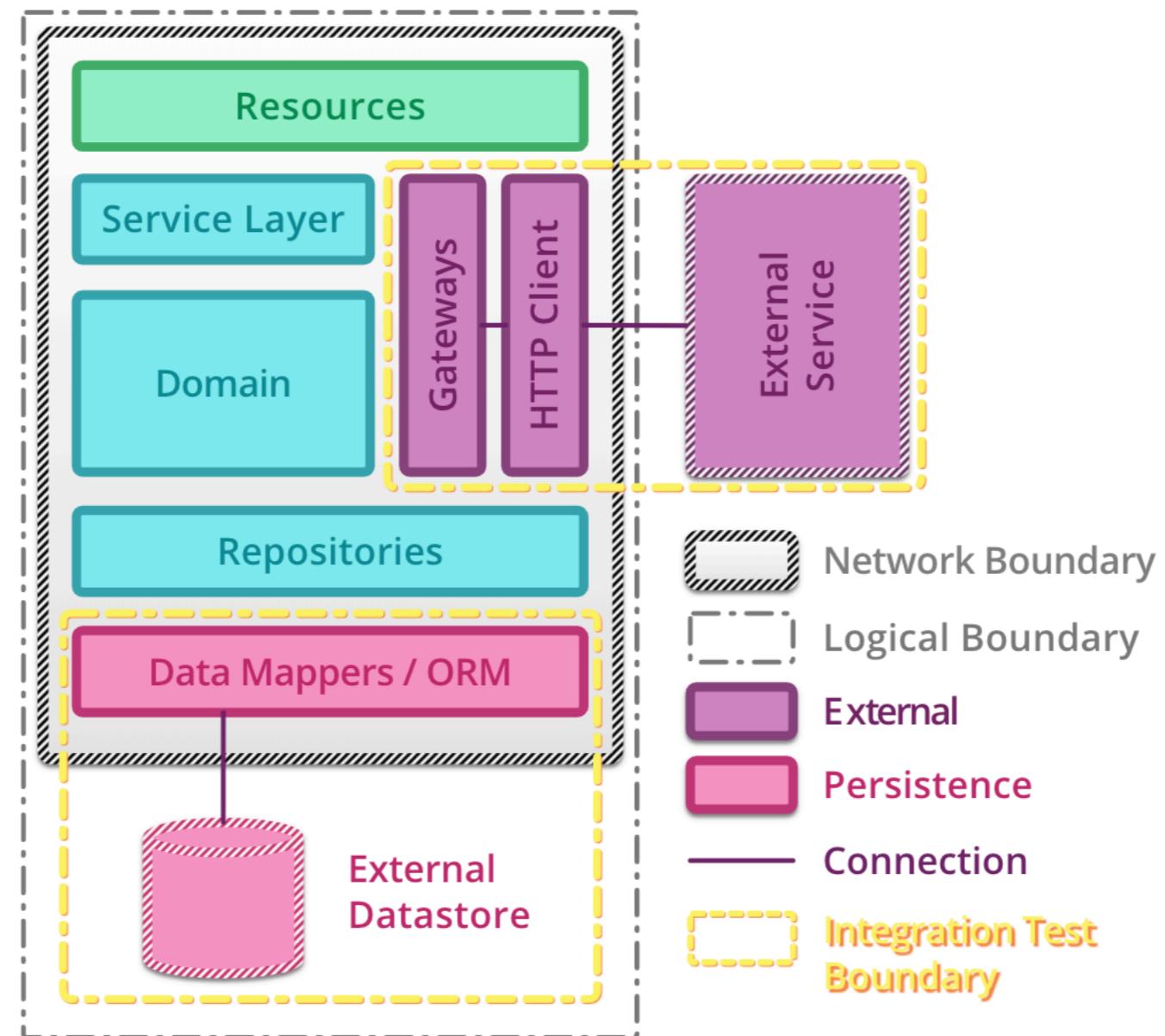
# Test strategy



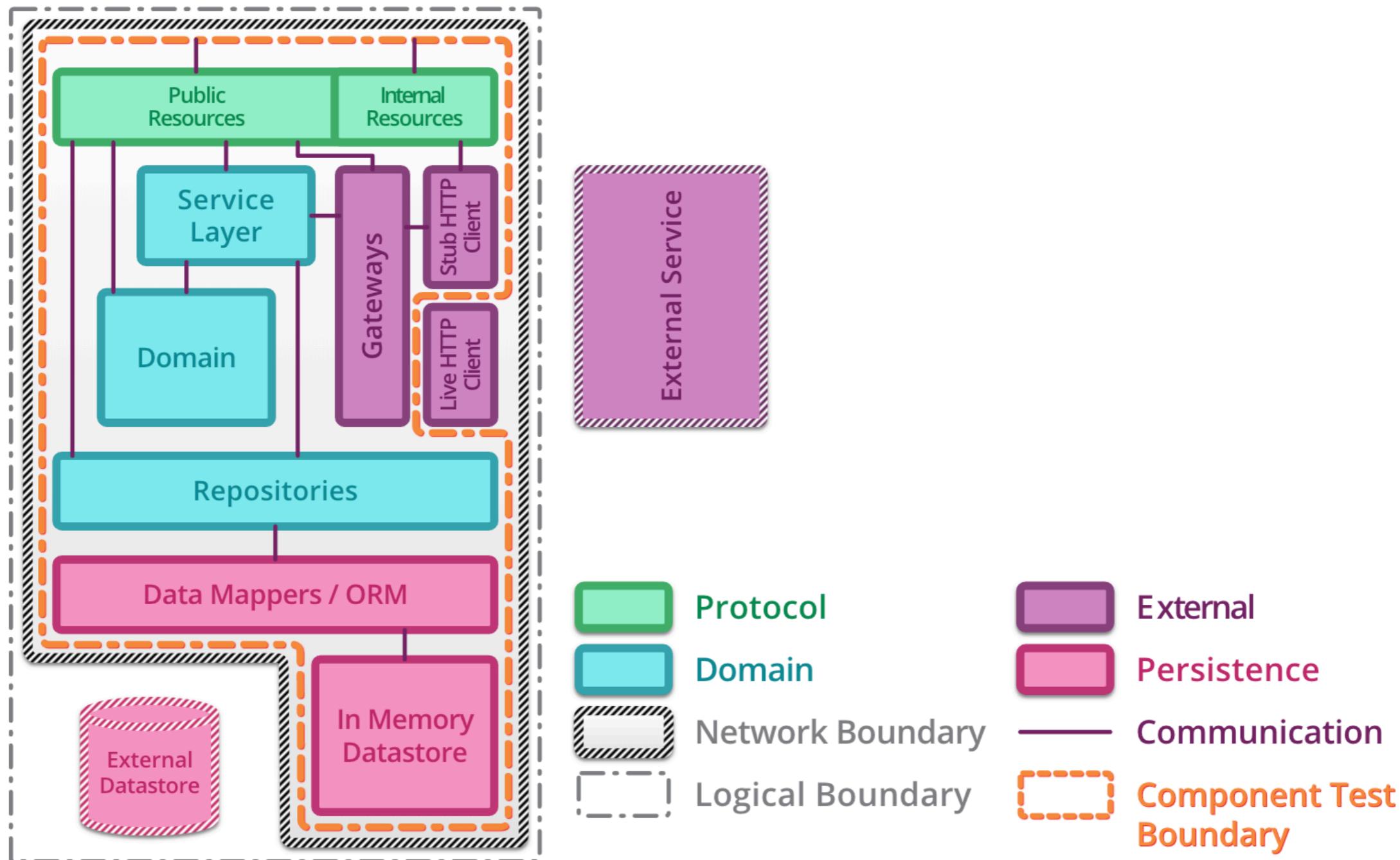
# Unit testing



# Integration testing



# Component testing



# Test design



# Economy of Test Design

Easy to understand

Easy to maintain

Readable by the business

One purpose per test

Re-runnable



# Economy of Test Design

Easy to understand

Easy to maintain

Readable by the business

One purpose per test

Re-runnable

Poor test practices reduce the benefits



# Respect your tests

Don't ignore it if it fails

Fix the code or fix the test

**100% of regression tests must pass all the time**

Always refactor or improve



# Test data !!

Avoid database/external system access

Setup/ tear down test data

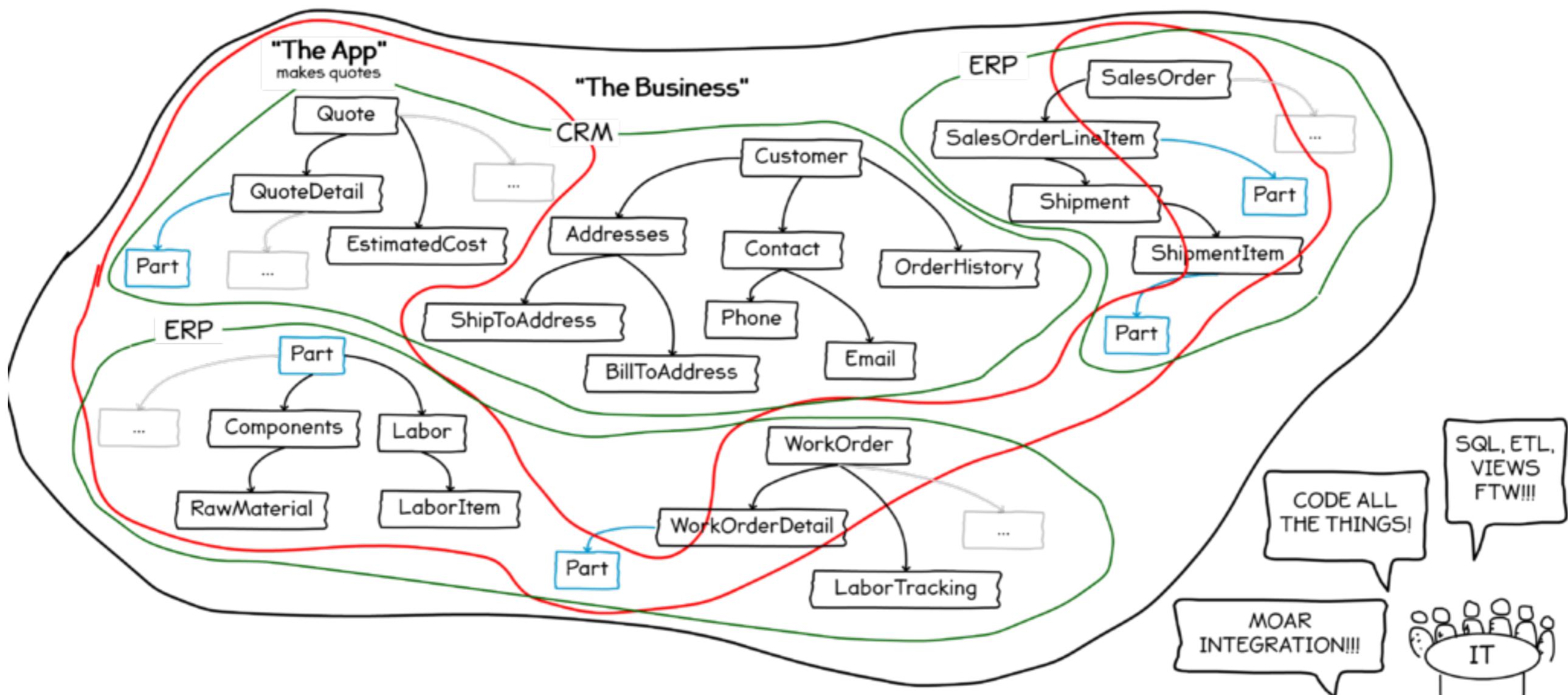
**Use production-like data**

Need to control your data test



# System impacts !!

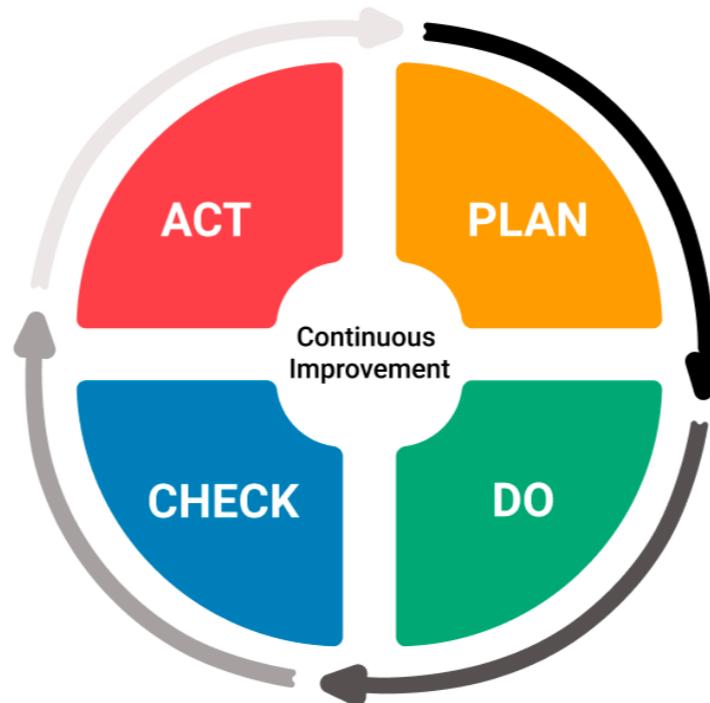
# Know your systems



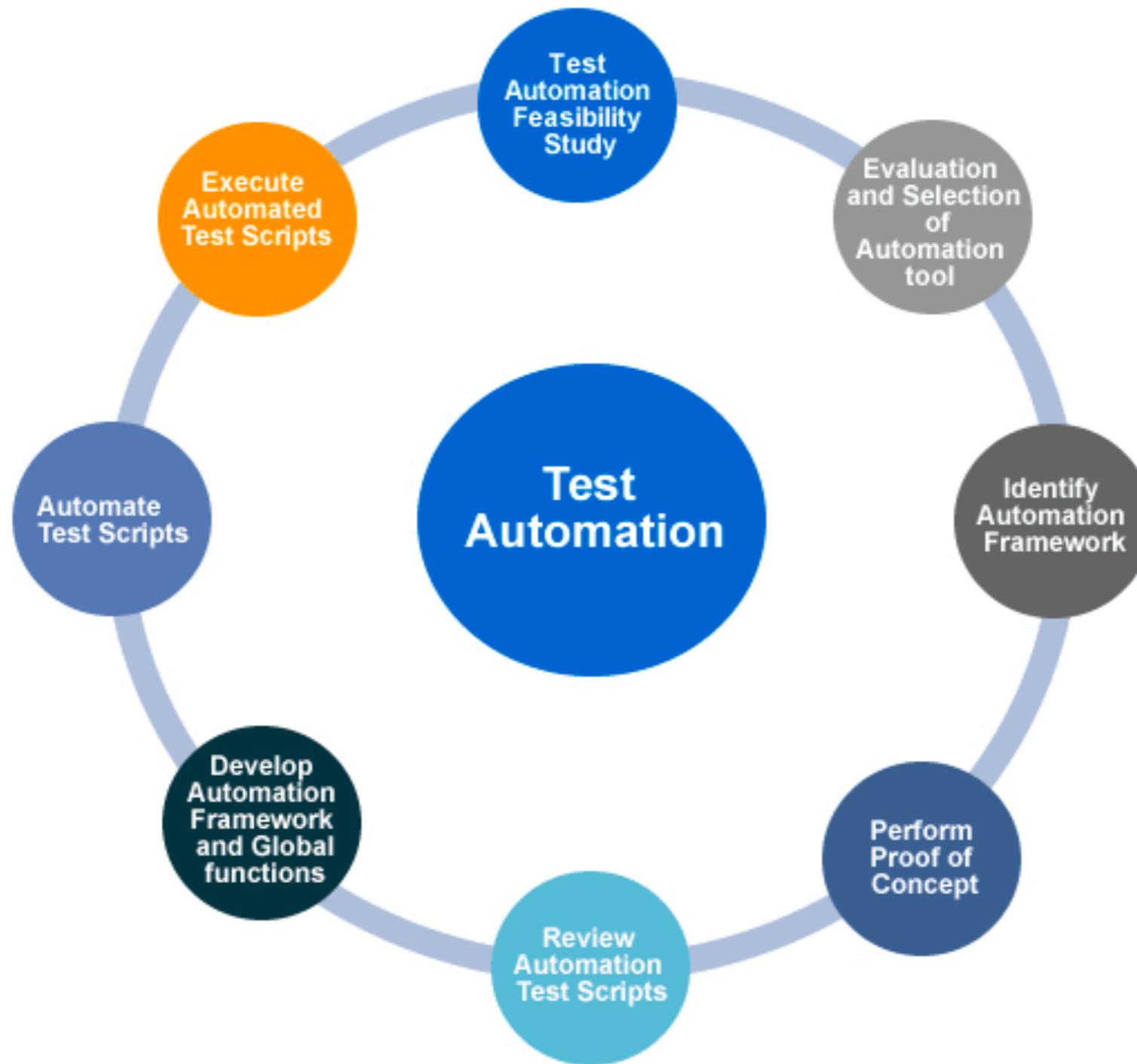
# Automation feedback

Easier over time ?

Time spent on maintenance ?  
Test find regression bugs ?



# Test Automation selection



# Manage automated tests

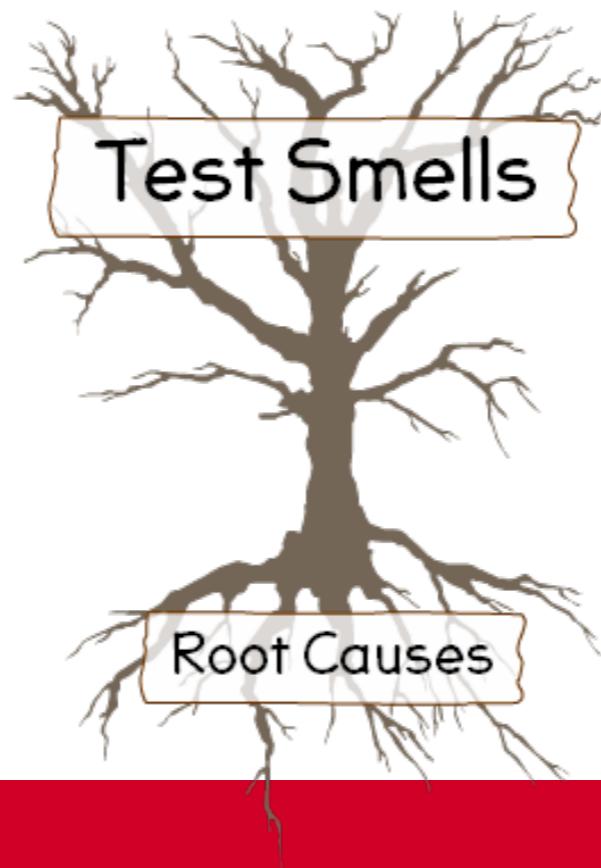
Use source control

**Continuous integration** to run tests on every change

Keep all tests passed

Analyse failures tests

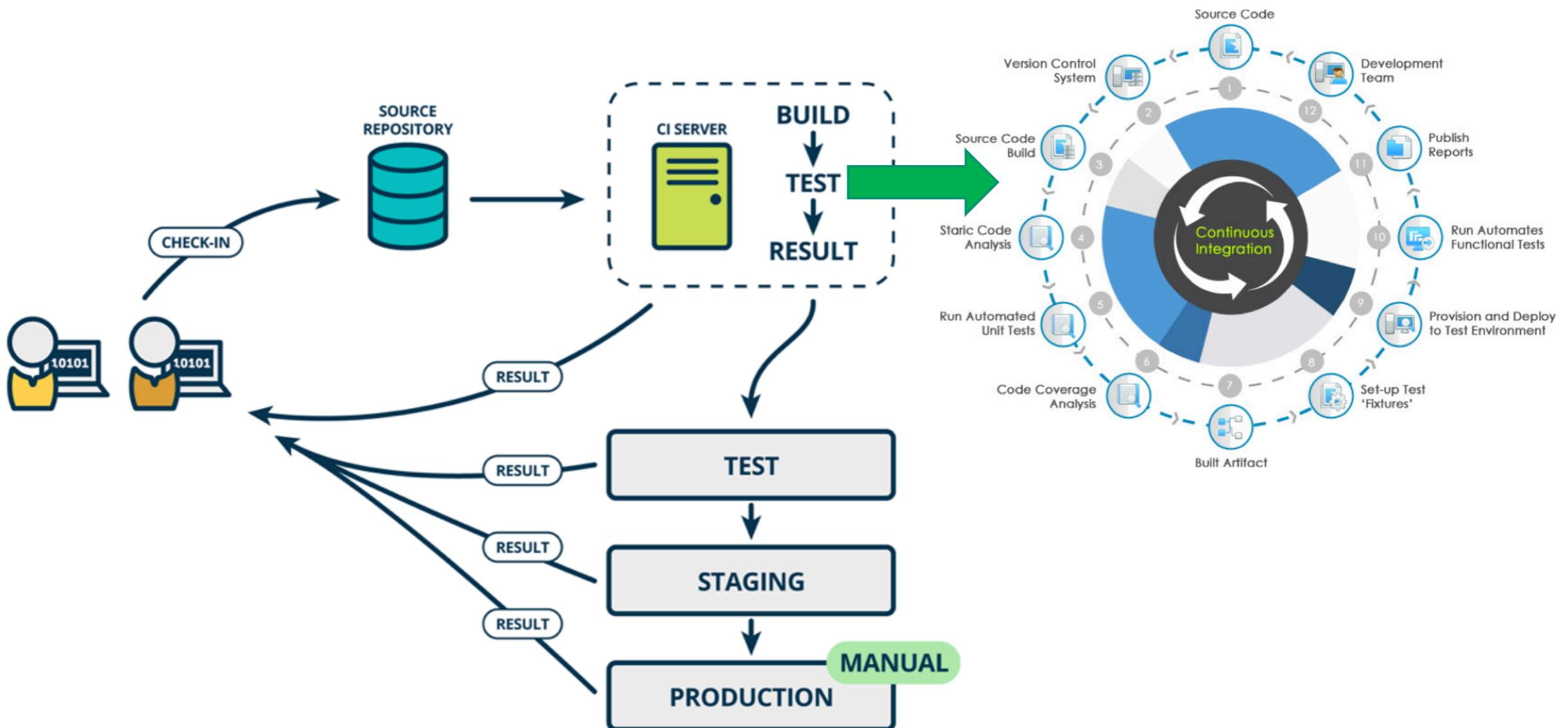
Create test standard



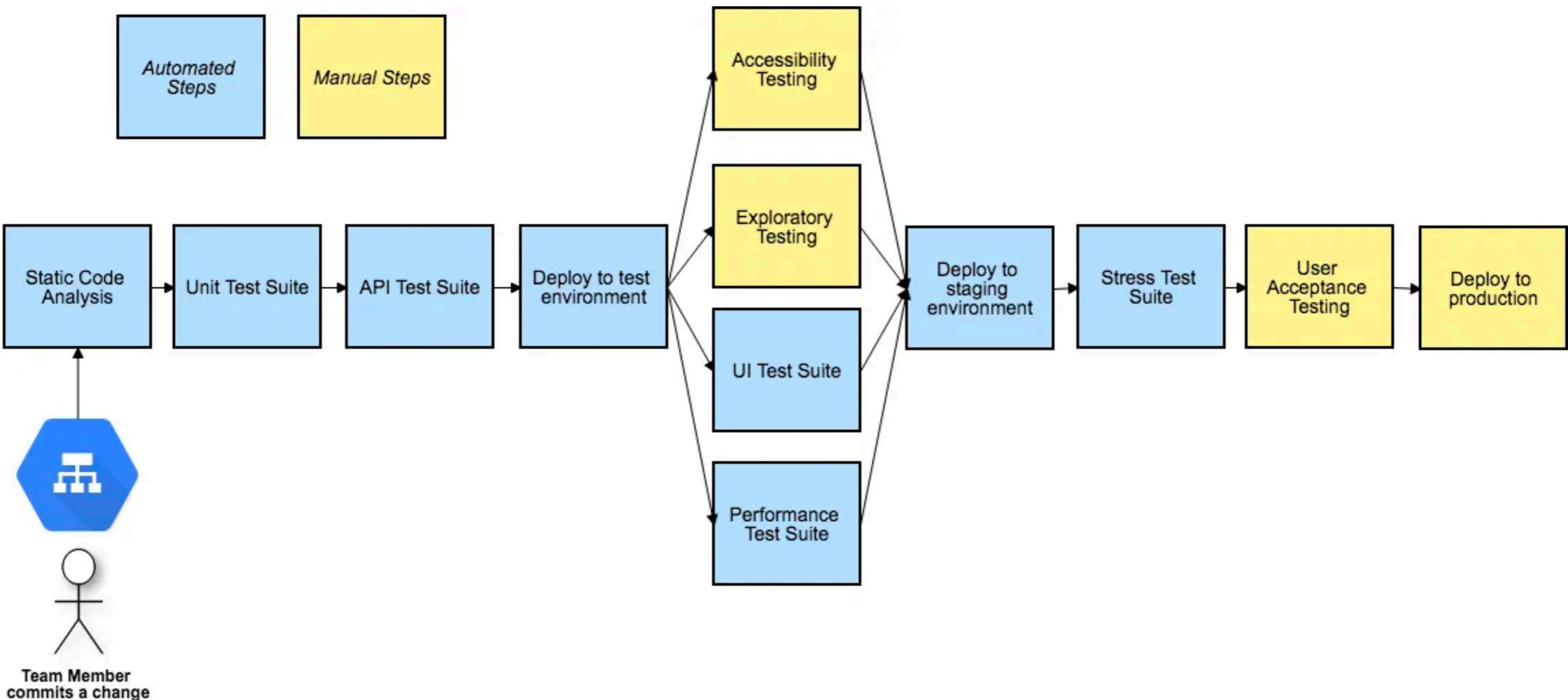
# Continuous integration



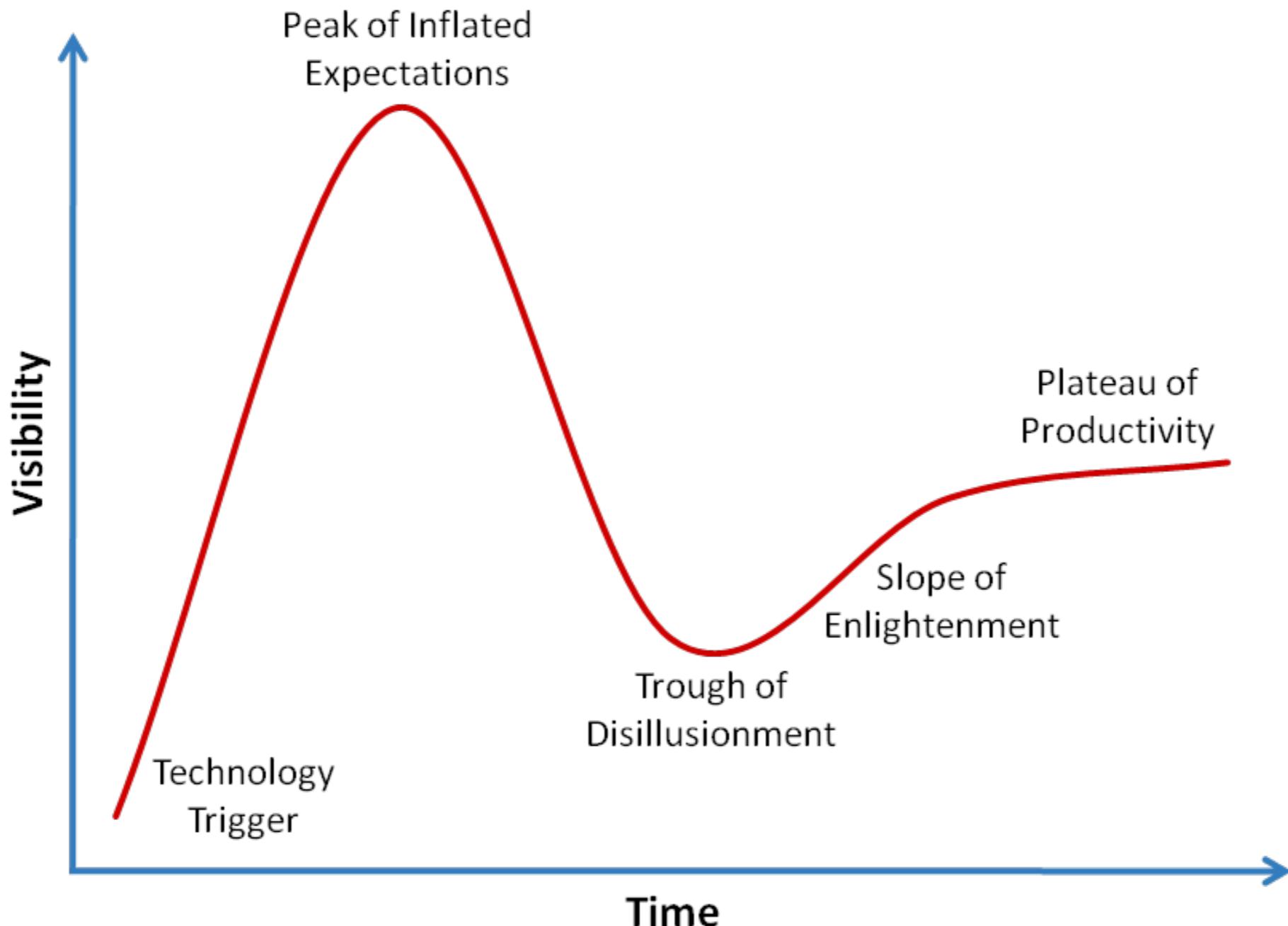
# Continuous integration



# Design your pipeline/process



# Hype cycle



# Start with simple



# Use **feedback** to improve



# Automate Unit Tests



# What Unit test ?

Software programs written to exercise other software programs with **specific preconditions** and verify the **expected behaviours**

Usually written in the **same programming language**

Test code

Production  
code



# What Unit test ?

Small and test only limited line of code functionality

Run very fast (100+ within a few seconds)

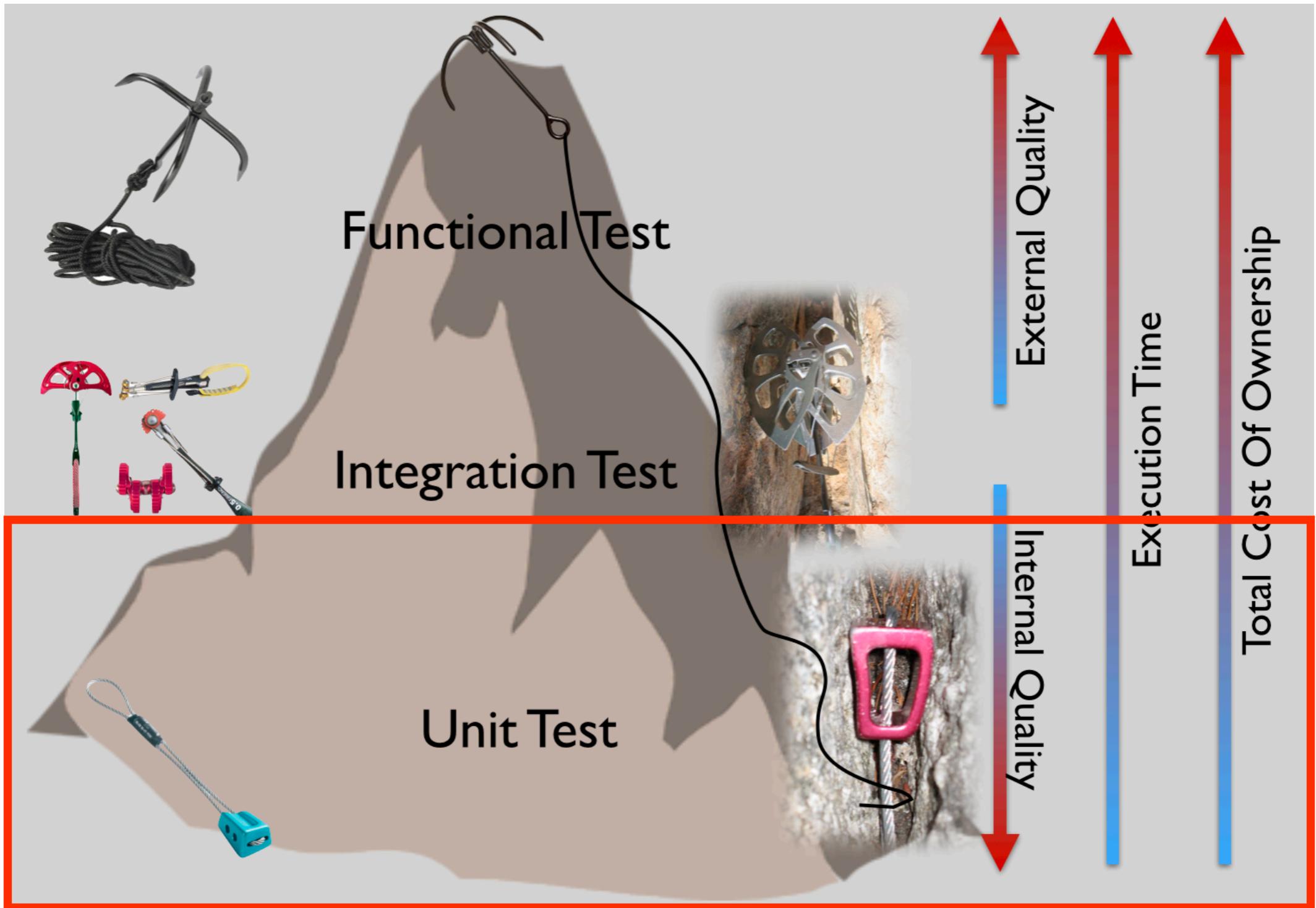


# Golden rule of Unit test

Each unit test case should be  
very limited in scope

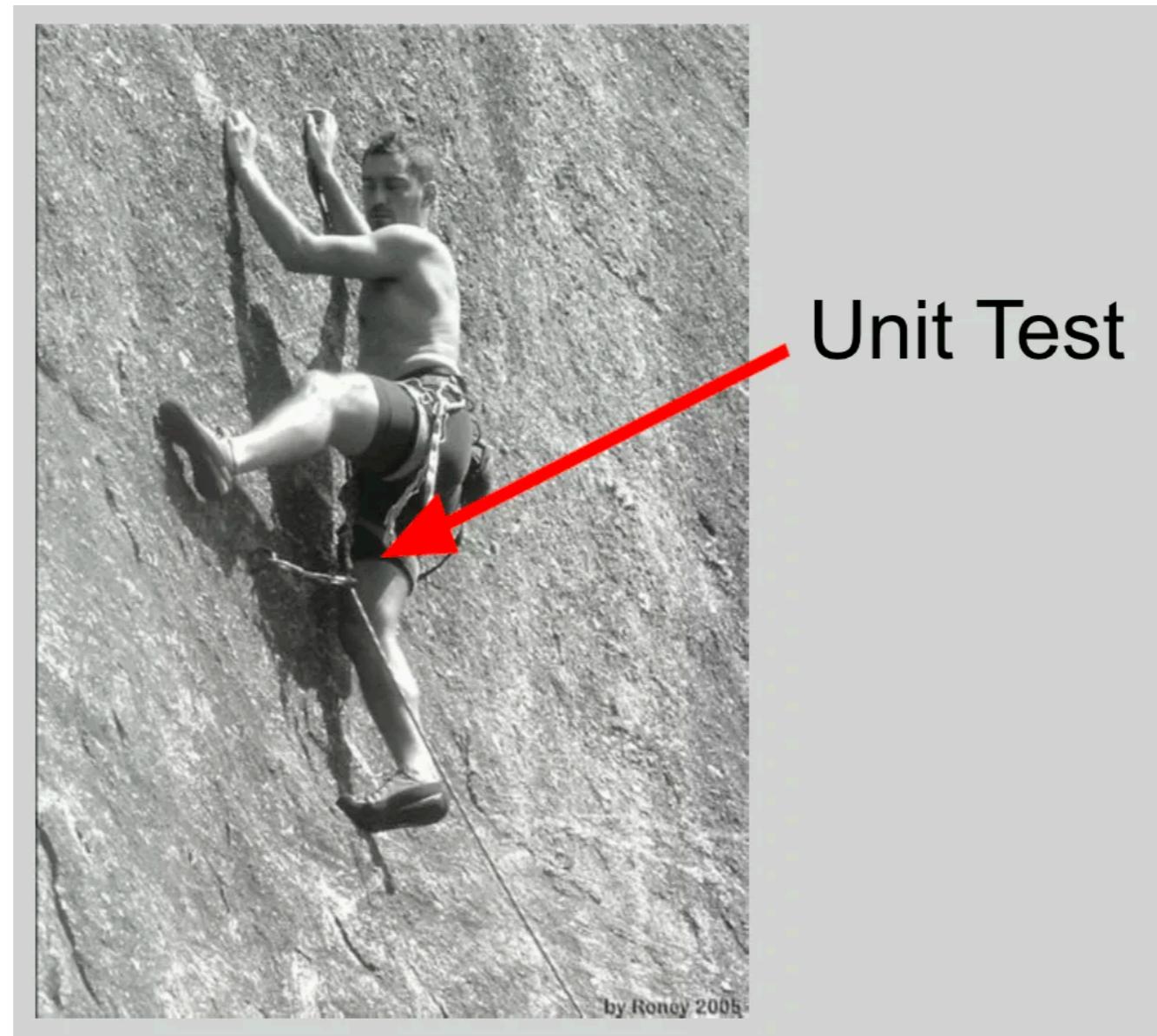


# Why Unit test ?



# Why Unit test ?

Protect what we have implemented than to find any defects



# Purpose of Unit test

Facilitates changes  
Simplifies integration  
Documentation  
Design tool



# Good Unit Tests (F.I.R.S.T)

Fast  
Independent/Isolate  
Repeat  
Self-validate  
Thorough/Timely



**We need testable code and  
easy to test**

**Fast feedback**



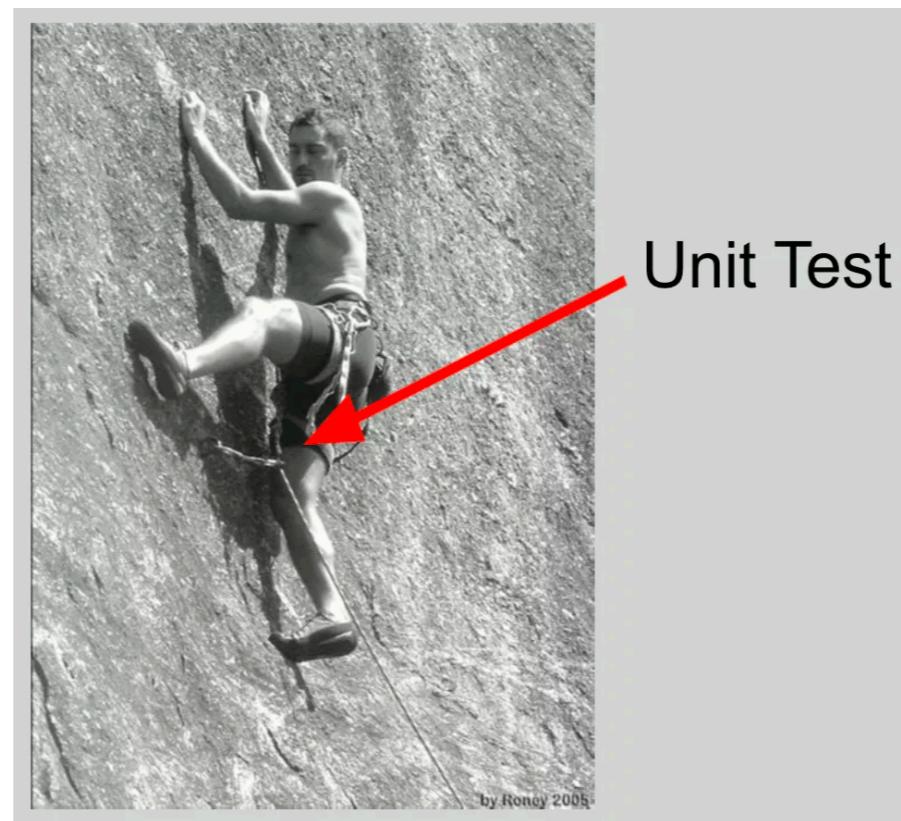
# Misconception of Unit test

Not as important as the production code

Done by testing engineers

Write unit test without changing production code

Add unit test later



# Maximize test accuracy

Table of error types		Functionality is	
		Correct	Broken
Test result	Test passes	Correct inference (true negatives)	Type II error (false negative)
	Test fails	Type I error (false positive)	Correct inference (true positives)

**Resistance to refactoring**

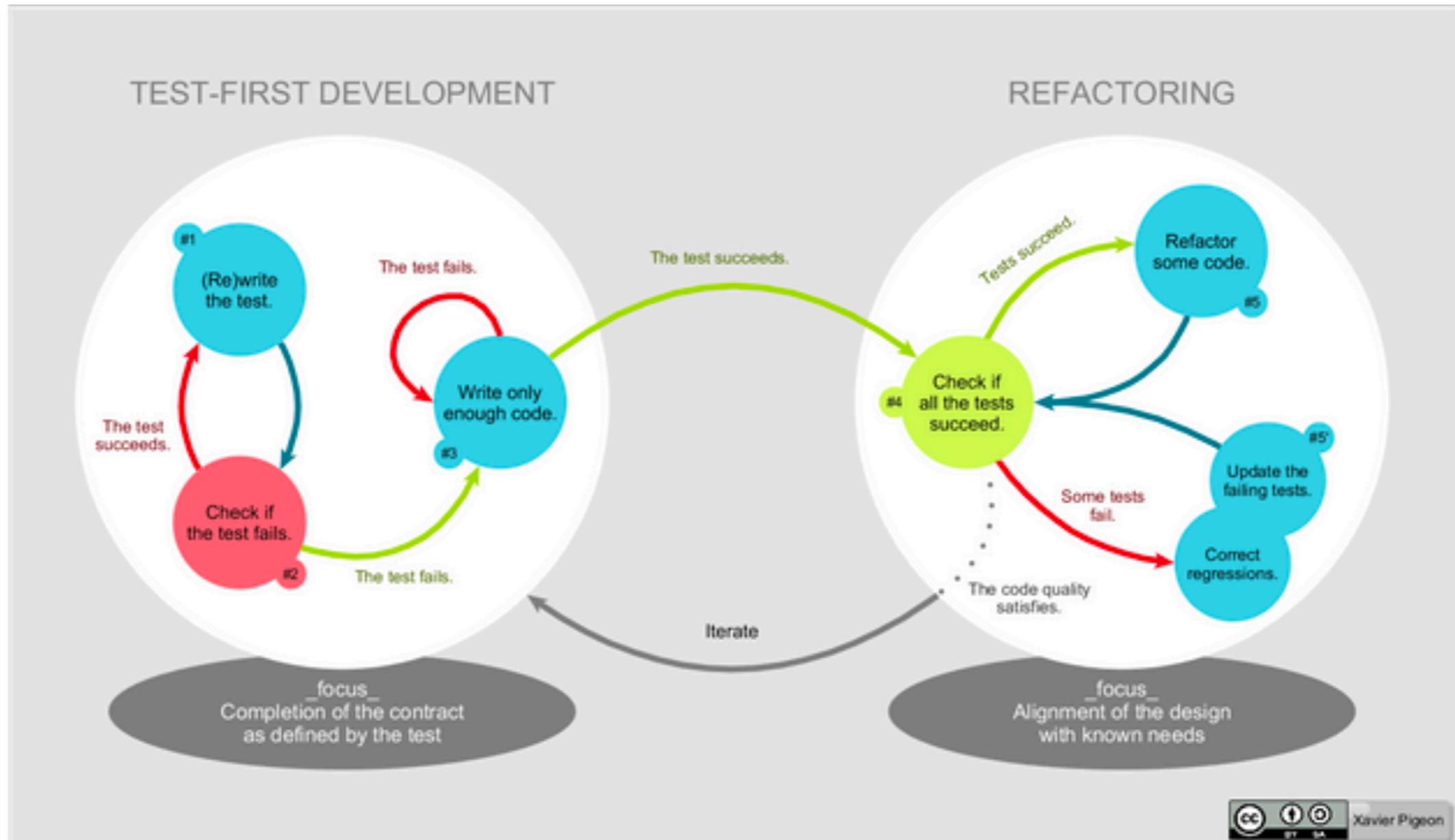
**Protection against regressions**



# Let's start with Test-First



# Test-First





**TDD**

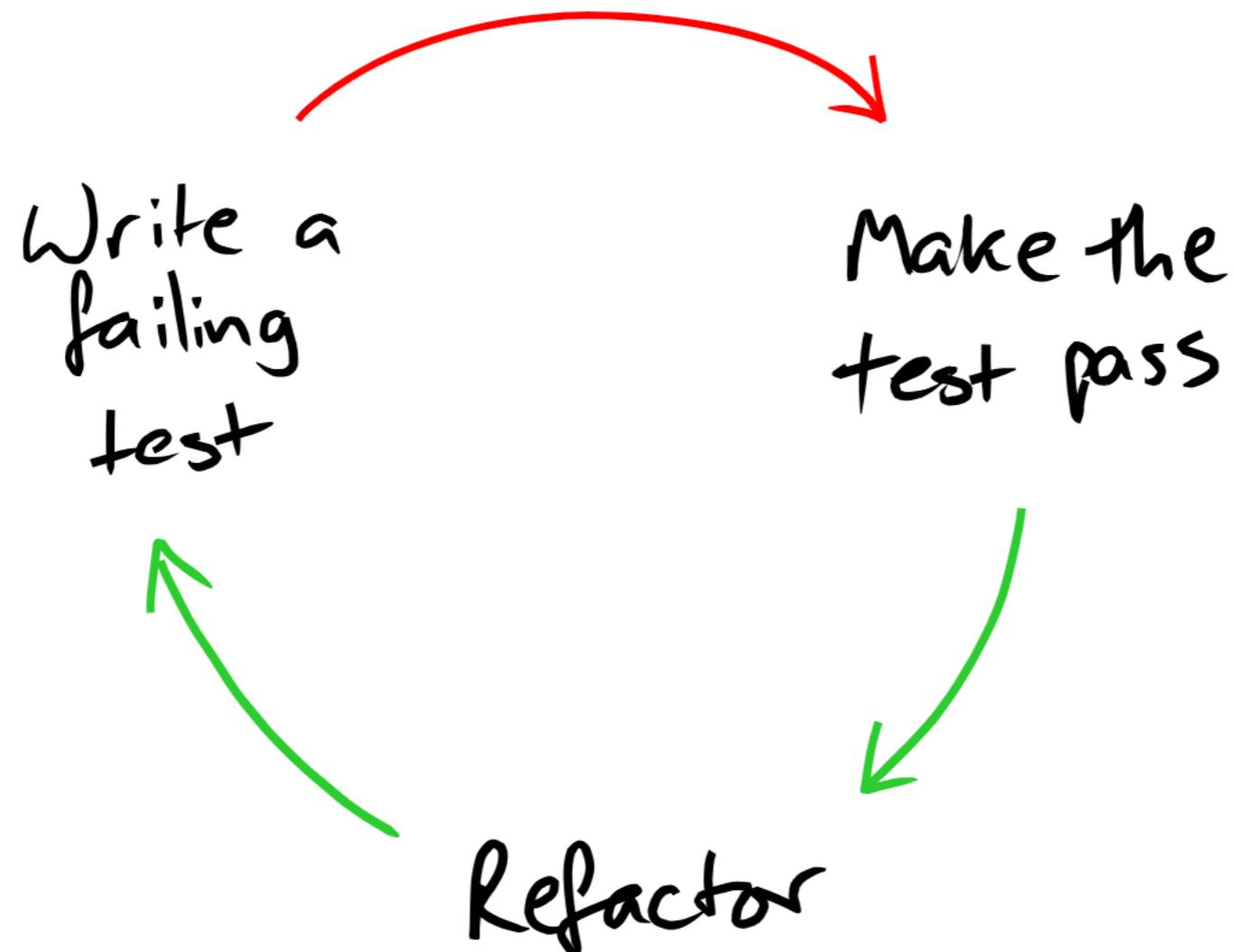
**ALL CODE IS GUILTY  
UNTIL PROVEN INNOCENT**



# TDD == ทำดีดี



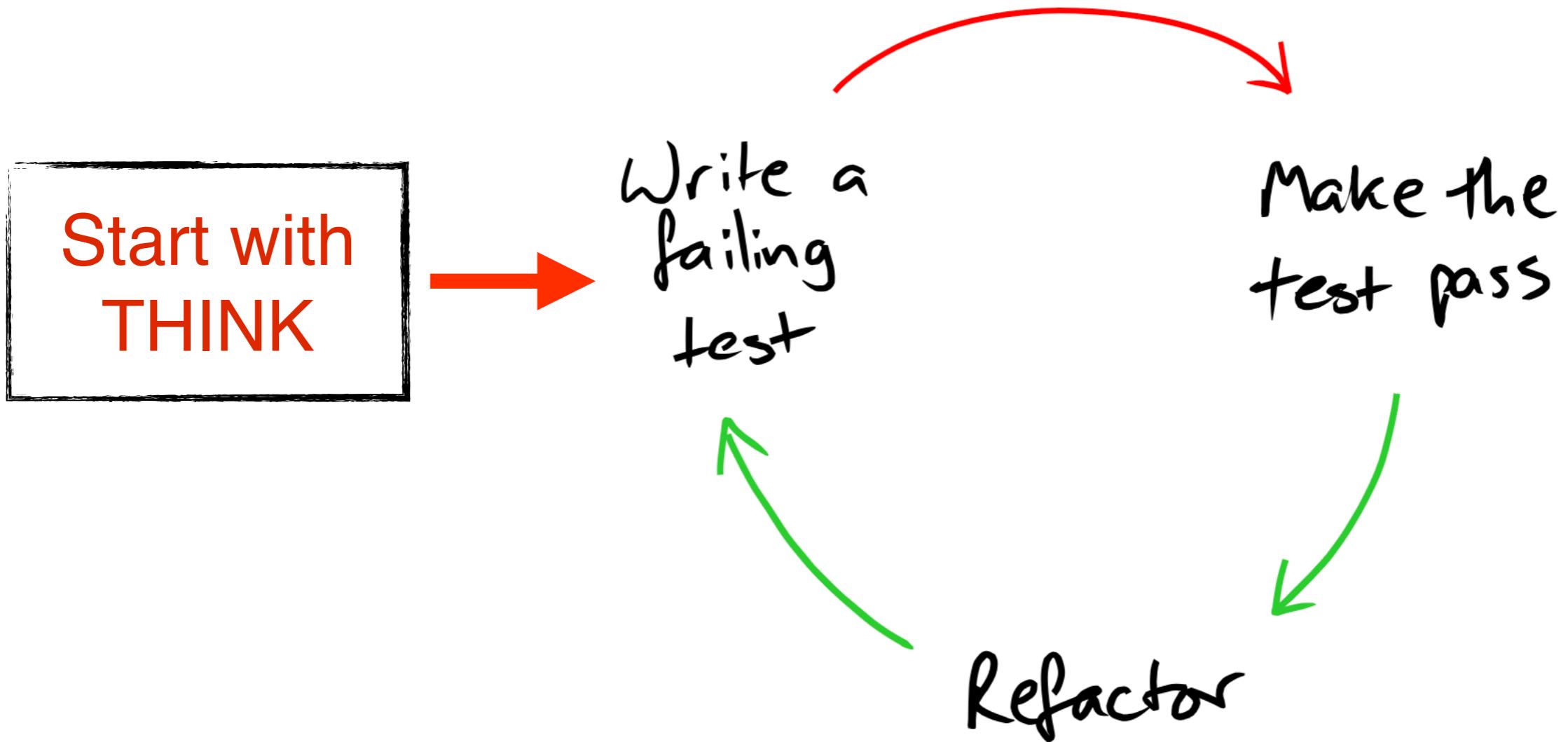
# TDD Cycle



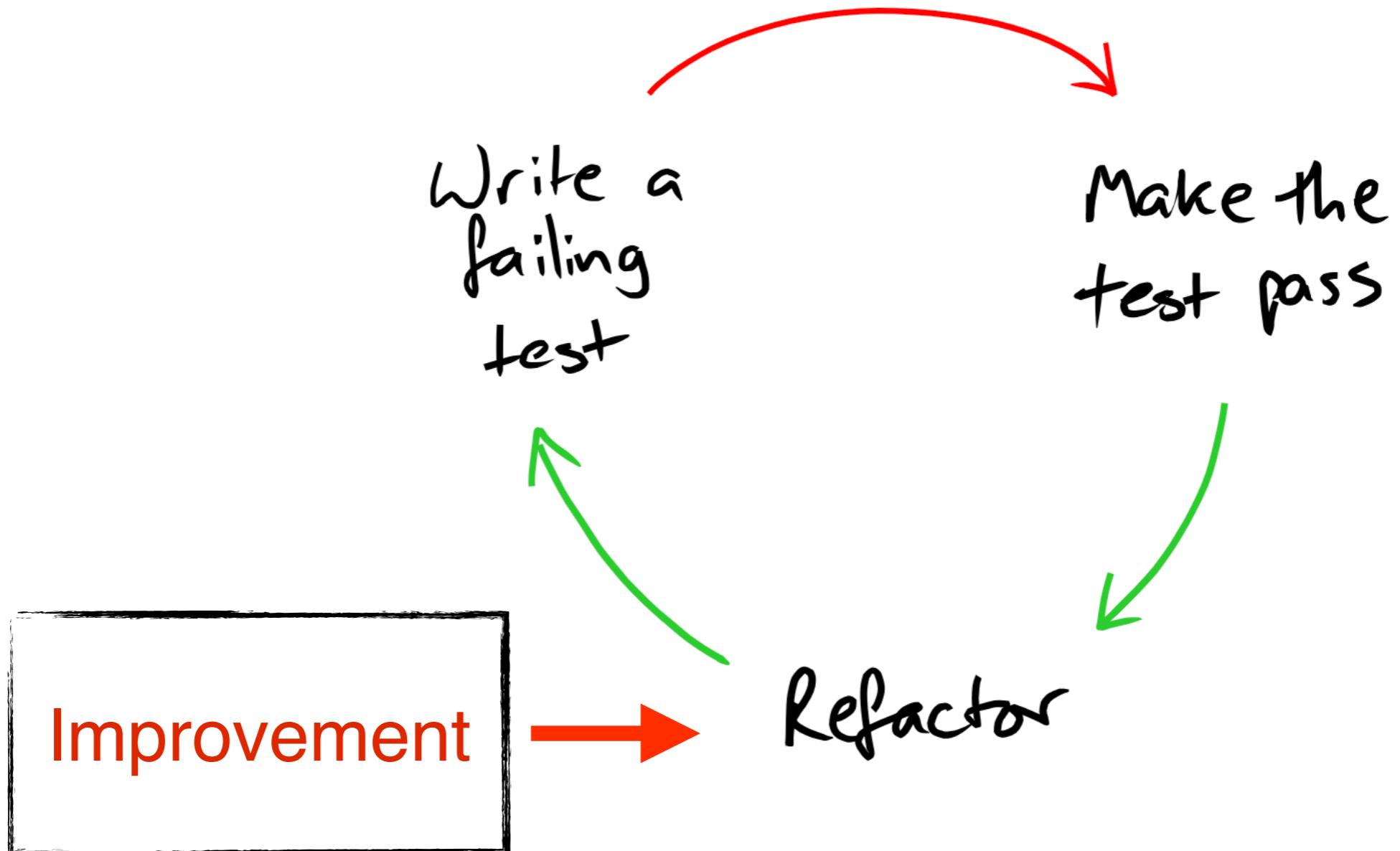
# Red Green Refactor



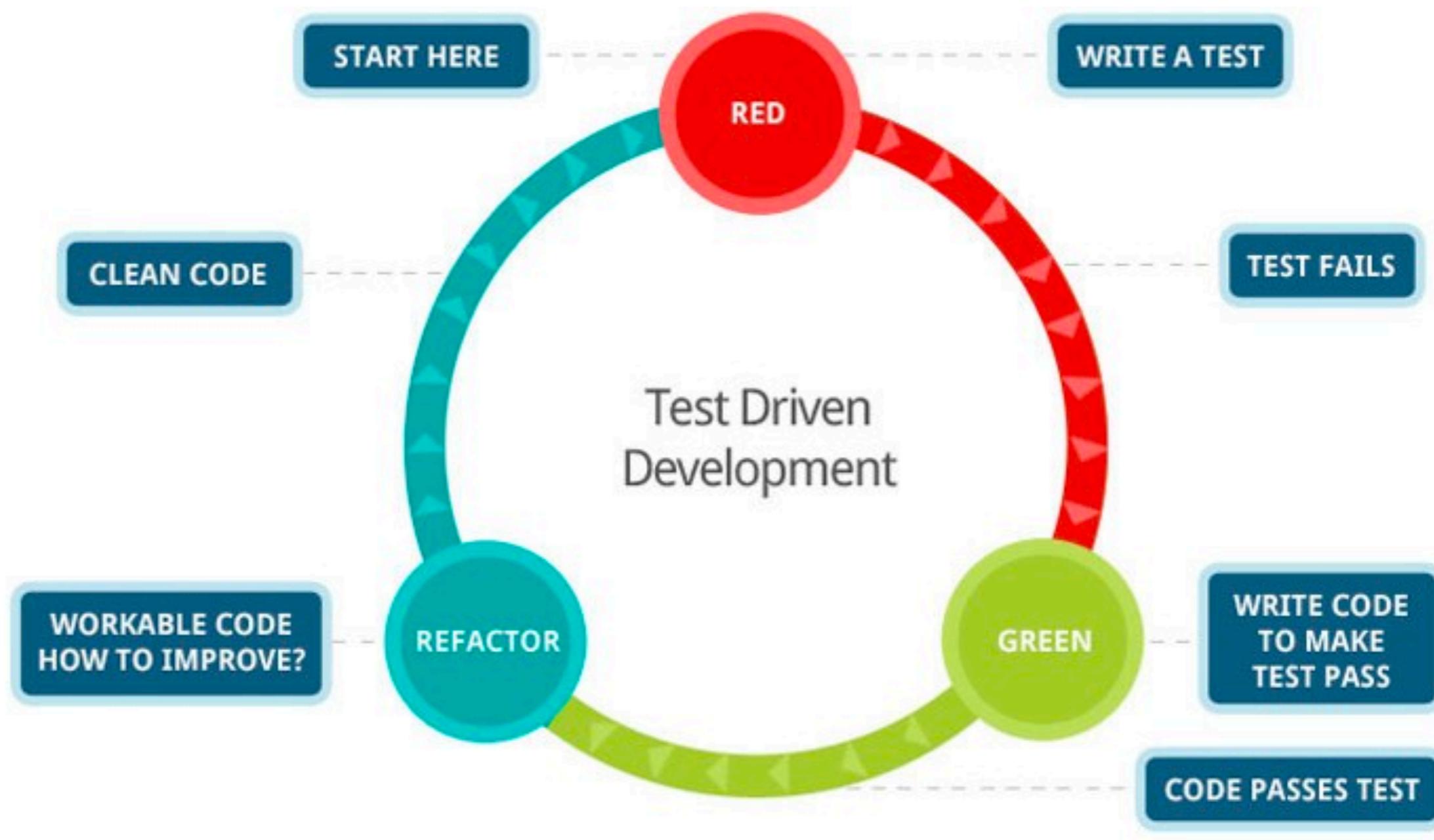
# Improve TDD Cycle



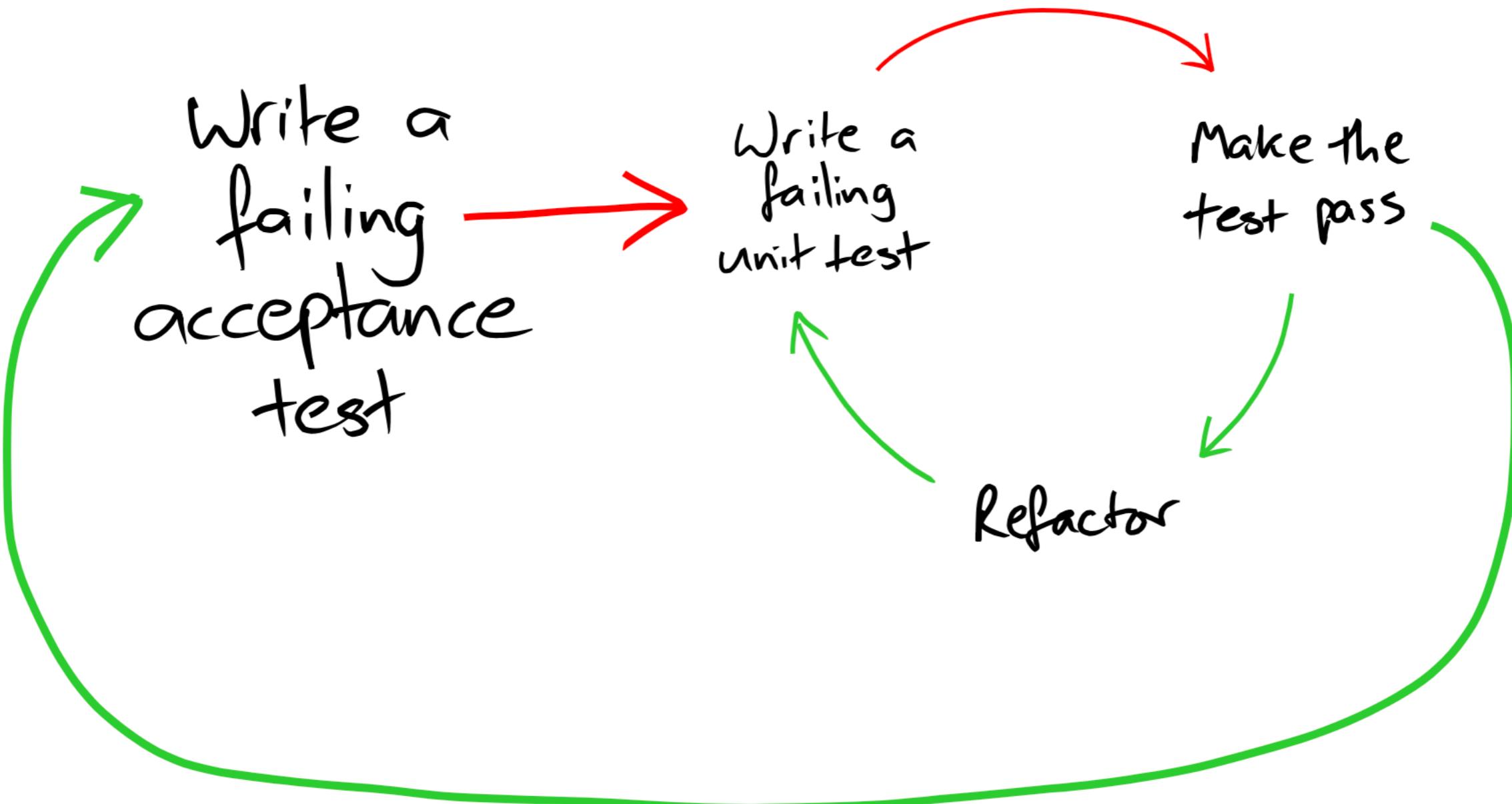
# Improve TDD Cycle



# Improve TDD Cycle



# Acceptance tests



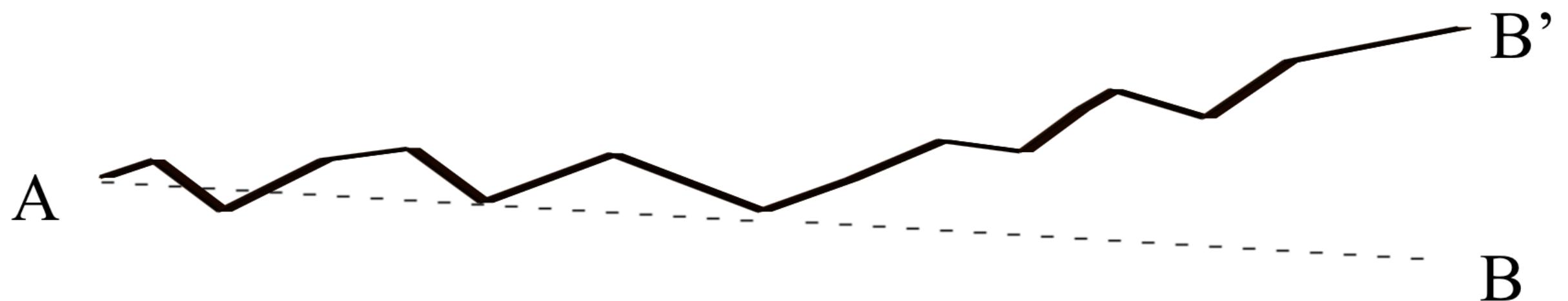
# The Golden rule of TDD

“Never write a line of code  
without a failing test”

- *Kent Beck* -



# Small Step



# Learning Testing Tools

jUnit The logo consists of the word "jUnit" in a gray sans-serif font followed by a large number "5". The number is white with a black outline, set against a circular background split into red on the left and green on the right.

<https://junit.org/junit5/>



# Build tools



# Configure Gradle

```
dependencies {  
    testImplementation(platform('org.junit:junit-bom:5.7.0'))  
    testImplementation('org.junit.jupiter:junit-jupiter')  
}  
  
test {  
    useJUnitPlatform()  
    testLogging {  
        events "passed", "skipped", "failed"  
    }  
}
```



# JUnit 5 :: Test structure

## Using @Test and @DisplayName

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

public class HelloTest {

    @Test
    @DisplayName("สวัสดี JUnit 5")
    public void testcase01() {

    }

}
```



# Run test in command-line

\$gradlew clean test

*Open report in /build/reports/tests/test/index.html*

## Class HelloTest

[all](#) > [default-package](#) > HelloTest

1  
tests

0  
failures

0  
ignored

0.012s  
duration

100%  
successful

### Tests

Test	Method name	Duration	Result
สวัสดิ์ JUnit 5	testcase01()	0.012s	passed



# Organize your tests

How to make your tests visually **consistent** ?  
Keeping tests maintainable by testing behavior  
**The importance of test naming**  
**Using test's life cycle**



# Good test structure

1. Setup the test data
2. Call your method under test
3. Assert that the expected results are returns



# Good test structure

AAA (Arrange Act Assert)

Given When Then from BDD style

<https://xp123.com/articles/3a-arrange-act-assert/>

<https://www.martinfowler.com/bliki/GivenWhenThen.html>



# Good test structure

## Using Arrange-Act-Assert

```
public class HelloTest {  
  
    @Test  
    @DisplayName("สวัสดี JUnit 5")  
    public void testcase01() {  
        // Arrange  
        Hello hello = new Hello();  
        // Act  
        String actualResult = hello.say("demo");  
        // Assert  
        assertEquals("Hello demo", actualResult);  
    }  
  
}
```



# Assertion in JUnit 5

assertEquals/NotEquals

assertArrayEquals

assertTrue/False

assertNull/NotNull

assertSame/NotSame

assertThrows/DoesNotThrow

assertAll



# Workshop

Input	Expected result
[1, 5]	1,2,3,4,5
[1, 5)	1,2,3,4
(1, 5]	2,3,4,5
(1, 5)	2,3,4

<http://codingdojo.org/kata/Range/>



# Parameterized testing

## Manage data for testing

```
public class DemoParameterizedTest {  
  
    @ParameterizedTest(name = "sayHi with {0} is {1}")  
    @CsvSource({  
        "user 01,      Hello user 01",  
        "user 02,      Hello user 02",  
    })  
    public void sayHi(String username, String expectedResult) {  
        assertEquals(expectedResult, process(username));  
    }  
  
}
```

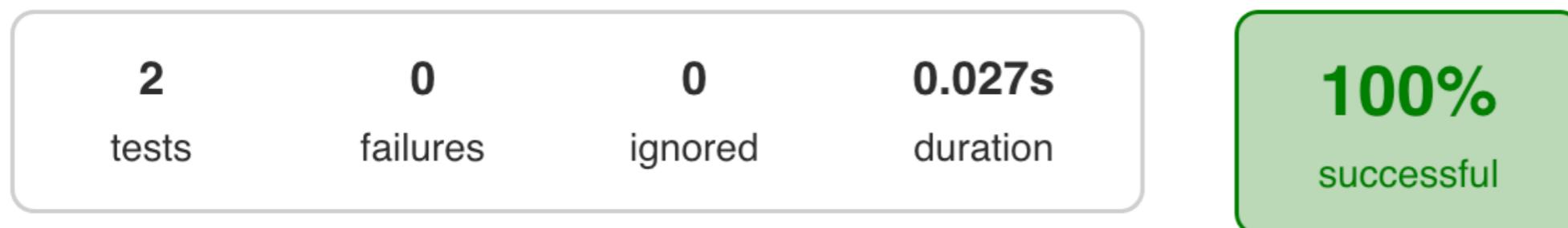


# Parameterized testing

Manage data for testing

## Class DemoParameterizedTest

[all](#) > [default-package](#) > DemoParameterizedTest



### Tests

Test	Method name	Duration	Result
sayHi with user 01 is Hello user 01	sayHi(String, String)[1]	0.026s	passed
sayHi with user 02 is Hello user 02	sayHi(String, String)[2]	0.001s	passed



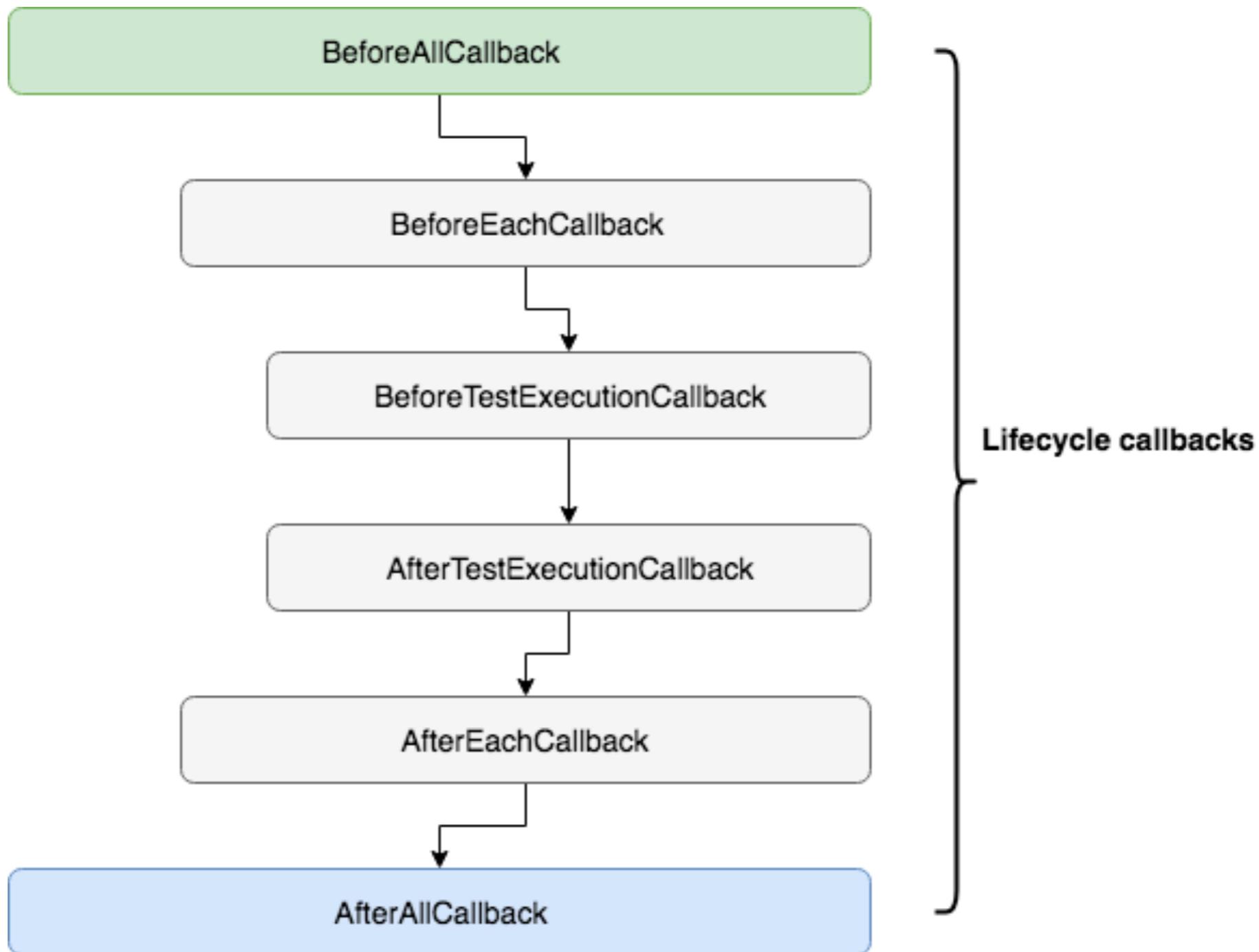
# Working with Exceptions

## Using assertThrows/DoesNotThrow

```
@Test  
 @DisplayName("throws SomeException when called hi")  
 void throwsExceptionWhenCalledHi() {  
  
     Hello hello = new Hello();  
     SomeException exception =  
         assertThrows(SomeException.class, () ->  
             hello.say("error"))  
     );  
  
     assertEquals(exception.getMessage(), "Some exception");  
 }  
}
```



# Testing life-cycle



# Testing life-cycle

```
public class DemoLifecycleTest {  
  
    @BeforeAll  
    public static void setup() {  
        System.out.println("Call setup");  
    }  
    @AfterAll  
    public static void teardown() {  
        System.out.println("Call teardown");  
    }  
    @BeforeEach  
    public void setupBeforeTest() {  
        System.out.println("Call setupBeforeTest");  
    }  
    @AfterEach  
    public void teardownAfterTest() {  
        System.out.println("Call teardownAfterTest");  
    }  
}
```



# Code coverage



**"Code coverage can show the high risk areas in a program, but never the risk-free."**

Paul Reilly, 2018, Kotlin TDD with Code Coverage



# Code coverage

A tool to measure how much of your code is covered by tests that break down into classes, methods and lines.



# Code coverage

## Hello.java

```
1. public class Hello {  
2.     public String say(String demo) {  
3.         if(demo == null) {  
4.             throw new SomeException();  
5.         }  
6.         return "Hello " + demo;  
7.     }  
8.  
9. }
```



# Code coverage

But 100% of code coverage **does not mean** that your code is 100% correct



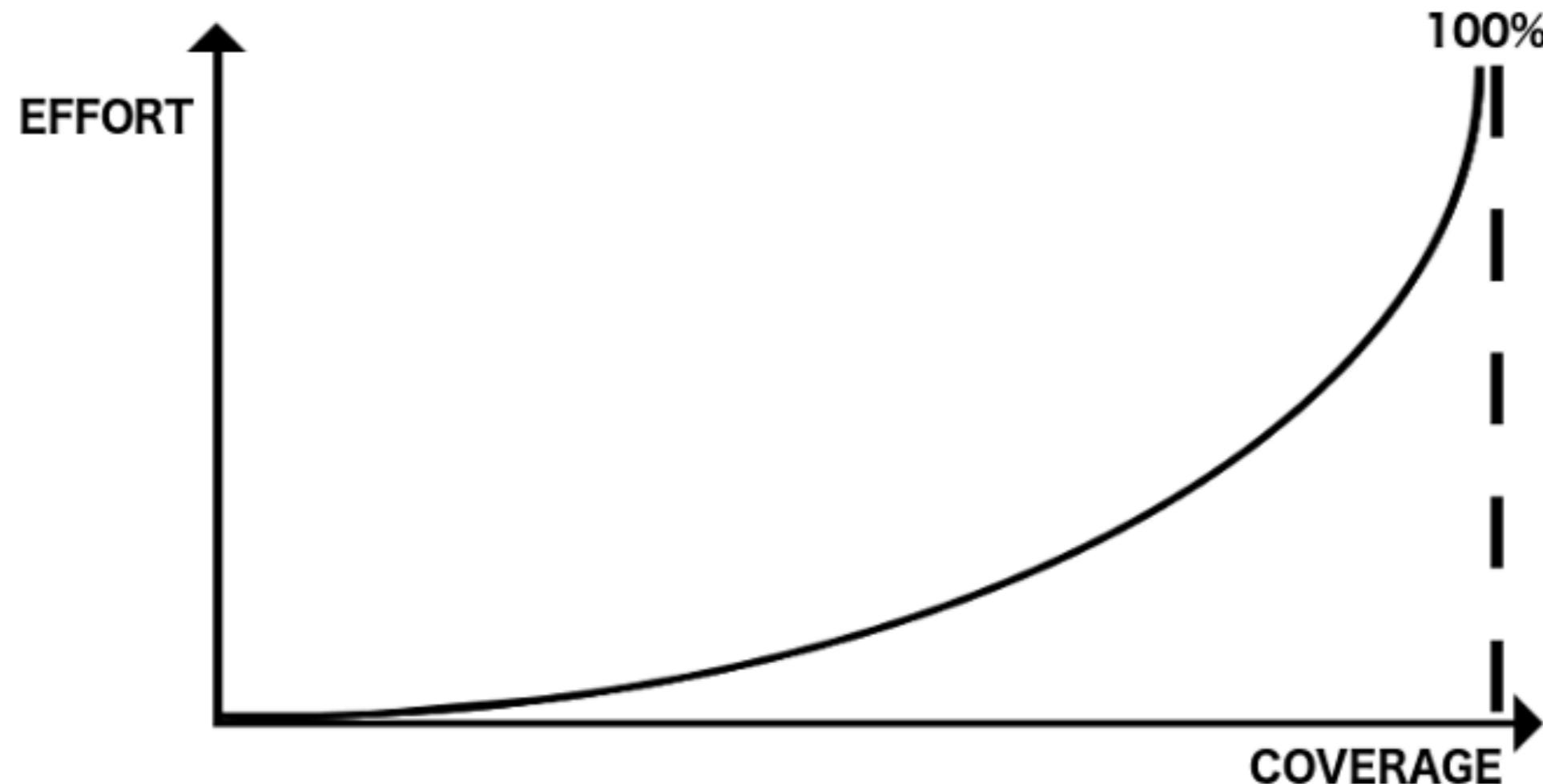
# Code coverage

Powerful tool to improve the quality of your code

**Code coverage != quality of tests**



# Code coverage 100% ?



# Using Jacoco

## JaCoCo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
<a href="#">org.jacoco.examples</a>	58%	64%	24	53	97	193	19	38	6	12		
<a href="#">org.jacoco.core</a>	97%	93%	107	1,388	115	3,347	21	720	2	139		
<a href="#">org.jacoco.agent.rt</a>	77%	84%	31	121	62	310	21	74	7	20		
<a href="#">jacoco-maven-plugin</a>	90%	81%	35	183	44	407	8	110	0	19		
<a href="#">org.jacoco.cli</a>	97%	100%	4	109	10	275	4	74	0	20		
<a href="#">org.jacoco.report</a>	99%	99%	4	572	2	1,345	1	371	0	64		
<a href="#">org.jacoco.ant</a>	98%	99%	4	163	8	429	3	111	0	19		
<a href="#">org.jacoco.agent</a>	86%	75%	2	10	3	27	0	6	0	1		
Total	1,355 of 27,352	95%	143 of 2,125	93%	211	2,599	341	6,333	77	1,504	15	294

<https://www.eclemma.org/jacoco/>



# Configure Jacoco with Gradle

Edit file build.gradle

```
apply plugin: "jacoco"
jacoco {
    toolVersion = '0.7.9'
}

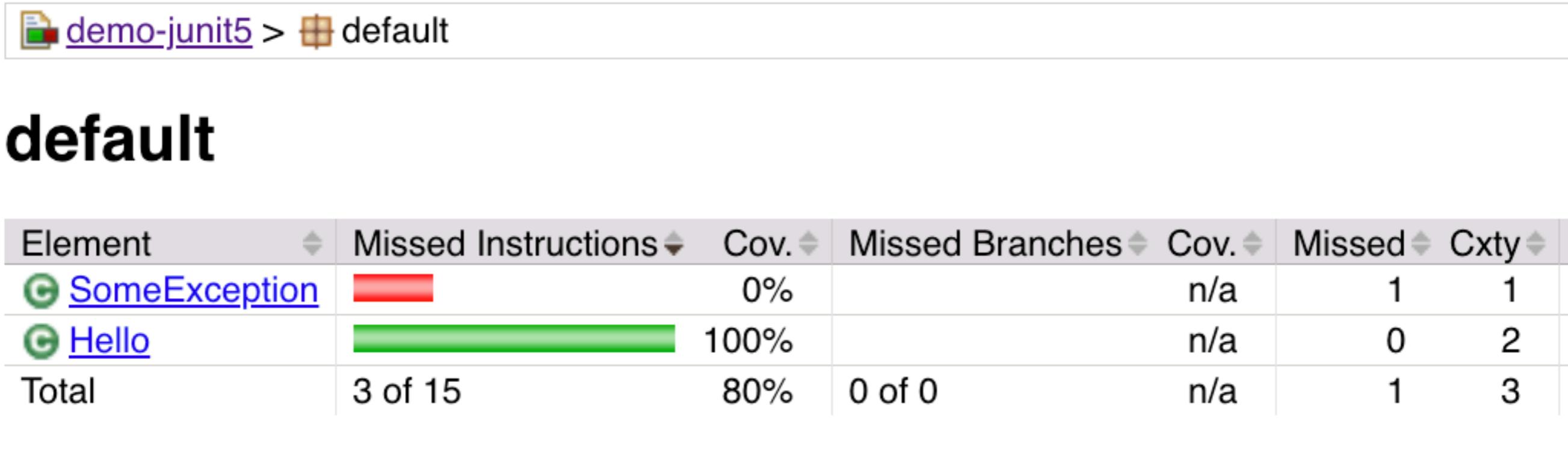
jacocoTestReport {
    reports {
        xml.enabled = true
        html.enabled = true
    }
}
```



# Run code coverage

\$gradlew jacocoTestReport

***Open report in build/reports/jacoco/test/html/index.html***



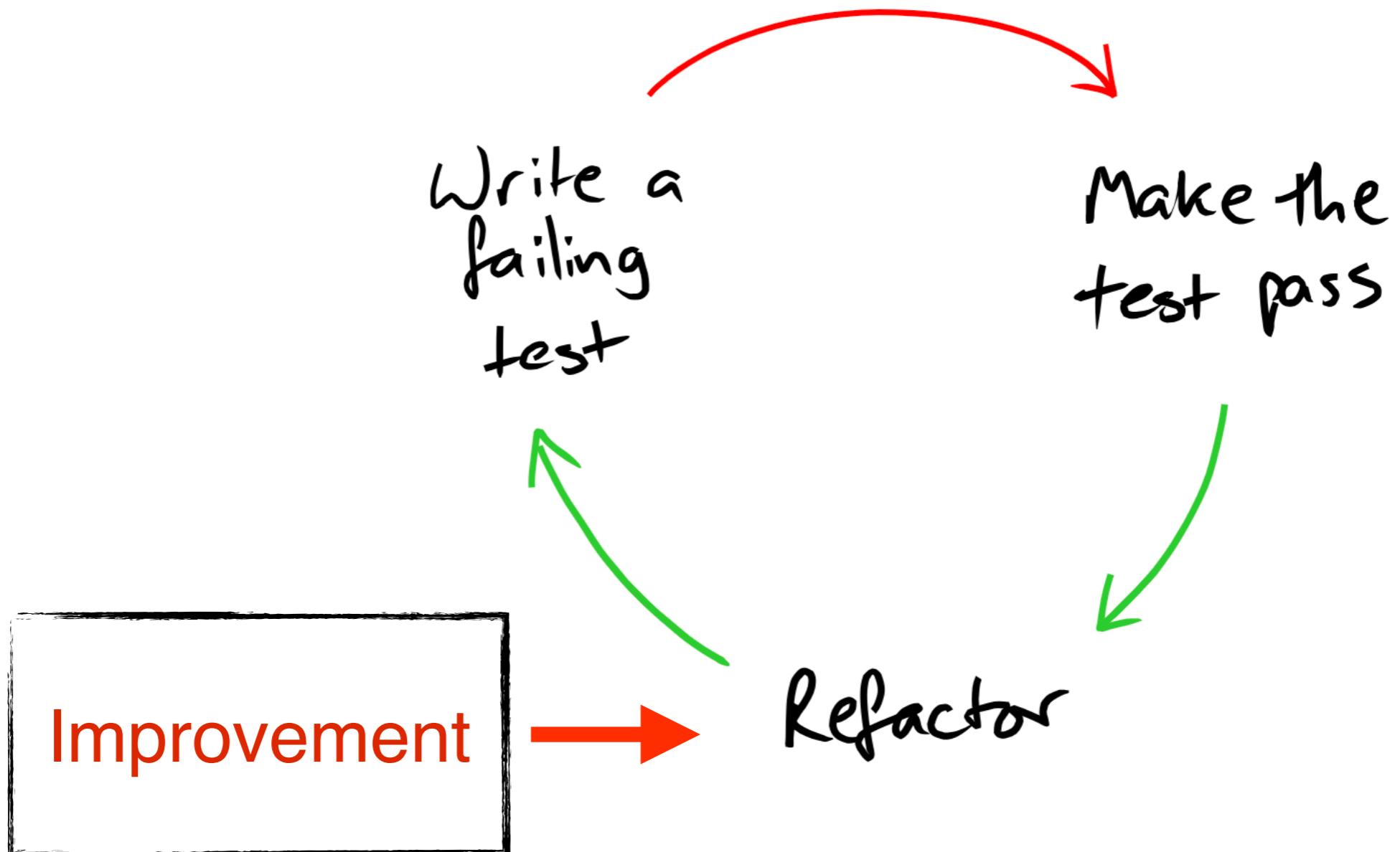
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty
 <a href="#">SomeException</a>		0%		n/a	1	1
 <a href="#">Hello</a>		100%		n/a	0	2
Total	3 of 15	80%	0 of 0	n/a	1	3

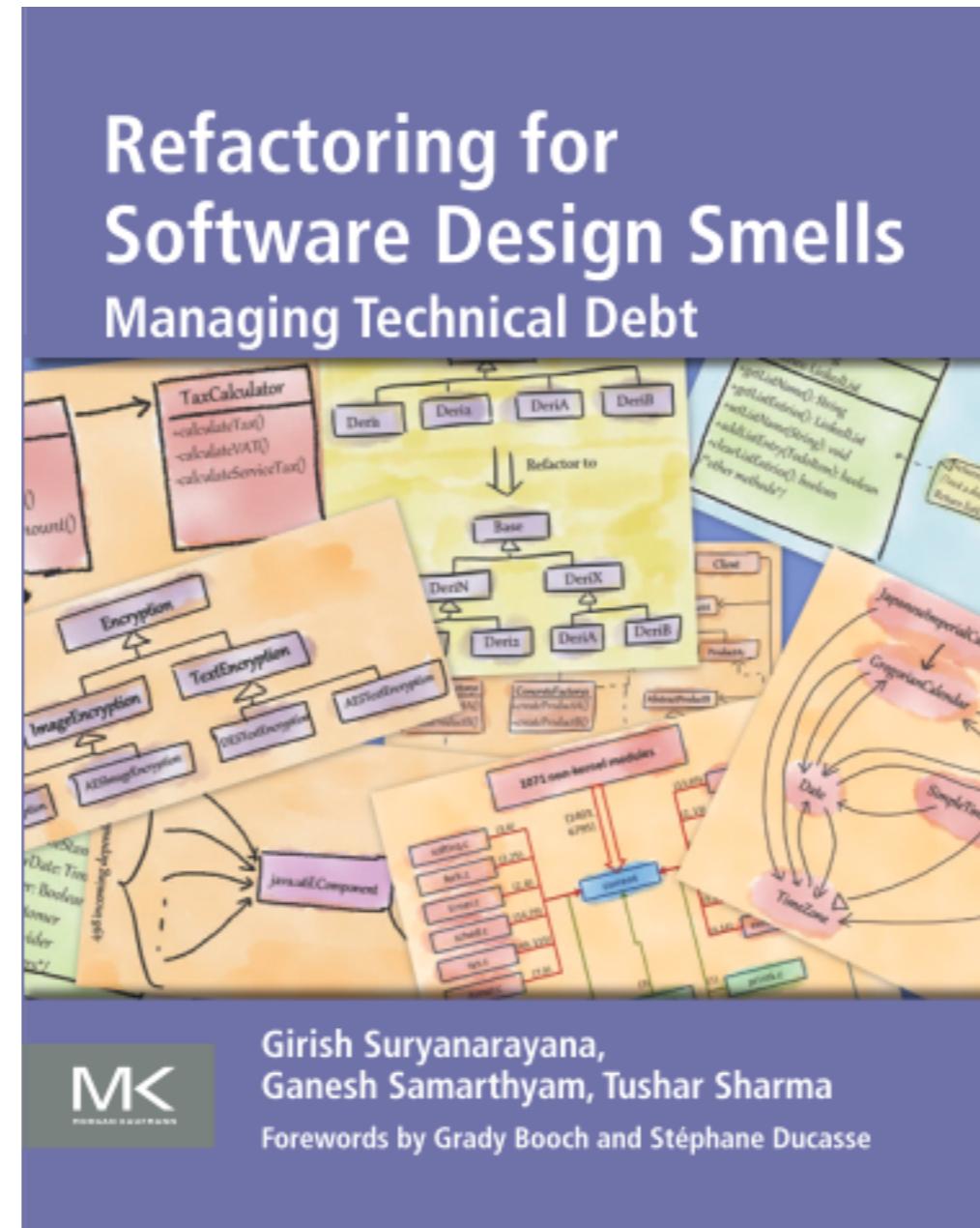


# Don't forgot Refactoring



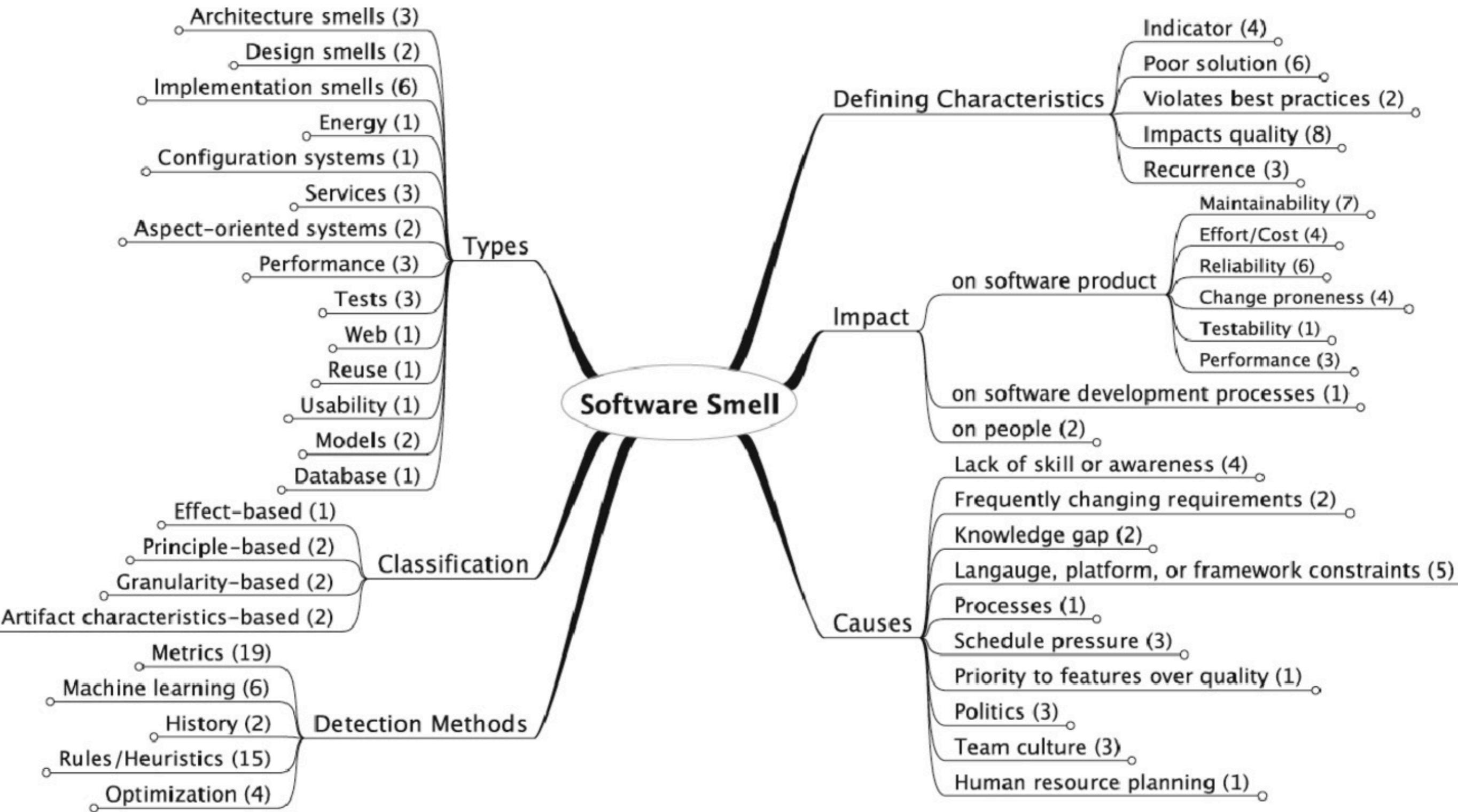
# Improve TDD Cycle





<http://www.tusharma.in/smells/>





<https://www.sciencedirect.com/science/article/pii/S0164121217303114>



# Code Smell



# Code Smells

- What? How can code "smell"??
- Well it doesn't have a nose... but it definitely can stink!



## Bloaters

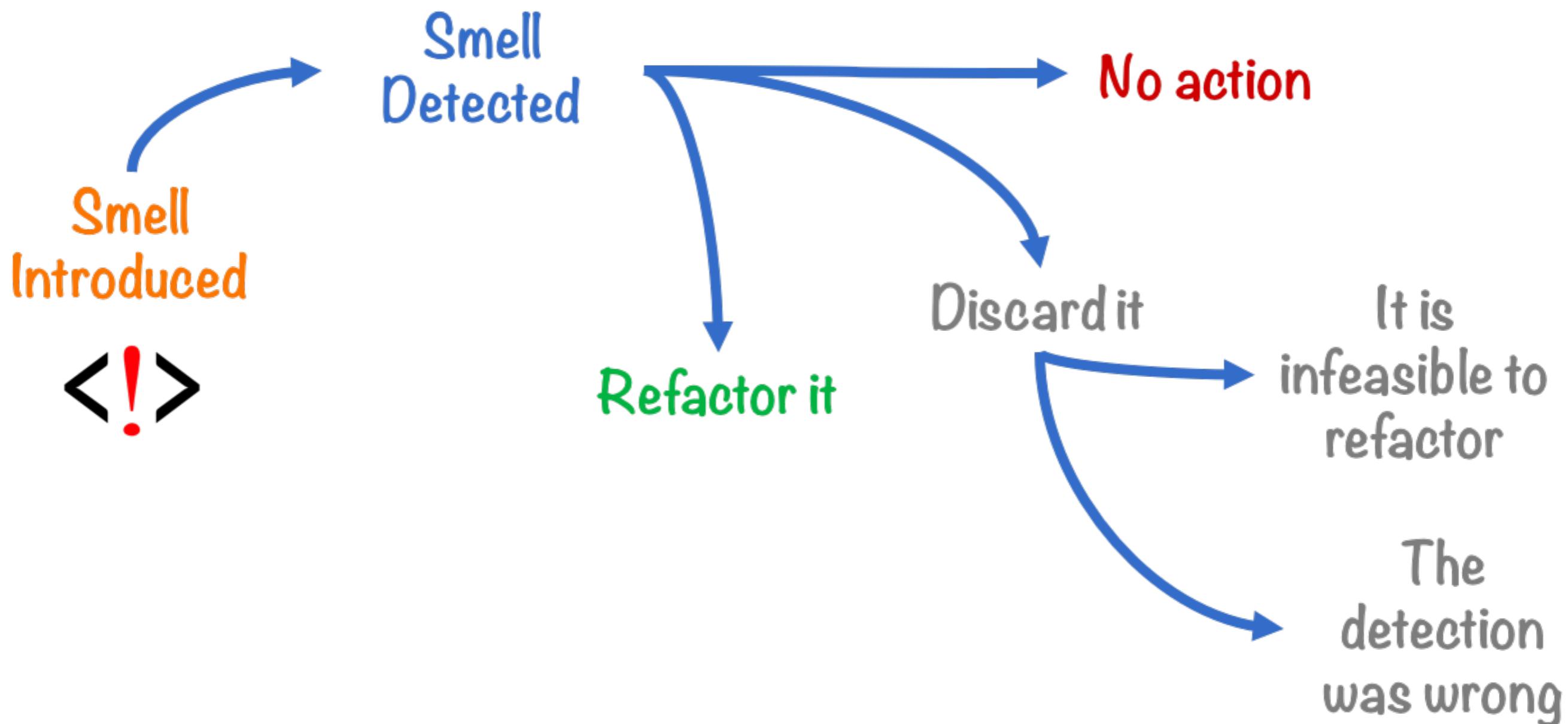
Bloaters are code, methods and classes that have increased to such gargantuan proportions that they are hard to work with. Usually these smells do not crop up right away, rather they accumulate over time as the program evolves (and especially when nobody makes an effort to eradicate them).

- Long Method
- Large Class
- Primitive Obsession
- Long Parameter List
- Data Clumps

<https://sourcemaking.com/refactoring/smells>



# Life-cycle of a smell



# Workshop



<https://codingdojo.org/kata/Tennis/>



# Test as Design



# Workshop



<http://codingdojo.org/kata/Potter/>



# S.O.L.I.D principles



# S.O.L.I.D



**SOLID**

Software development is not a Jenga game.



# Single Responsibility Principle



## Single Responsibility Principle

Just because you *can* doesn't mean you *should*.



# Open-Closed Principle



## Open-Closed Principle

Open-chest surgery isn't needed when putting on a coat.



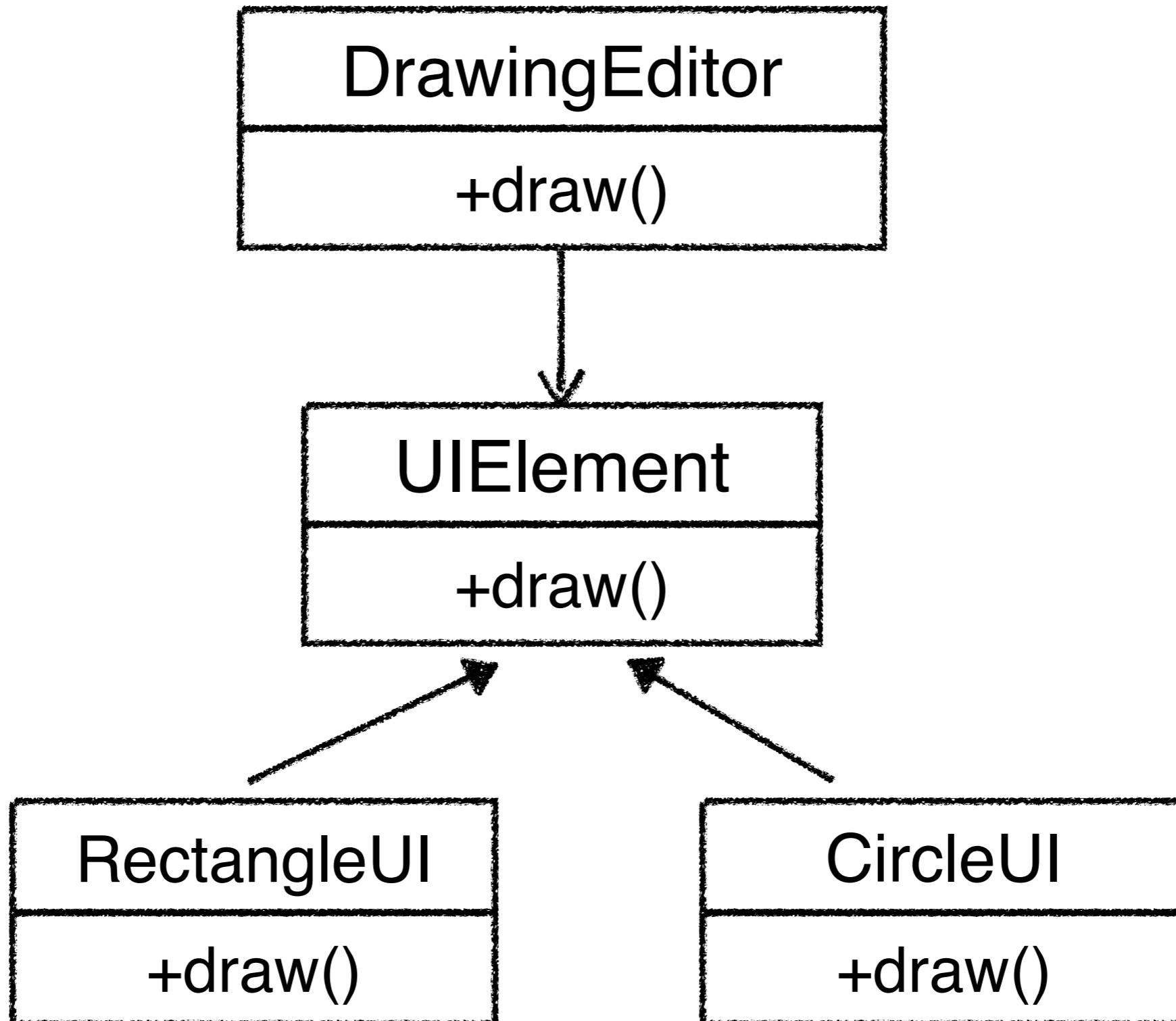
# Example

## DrawingEditor

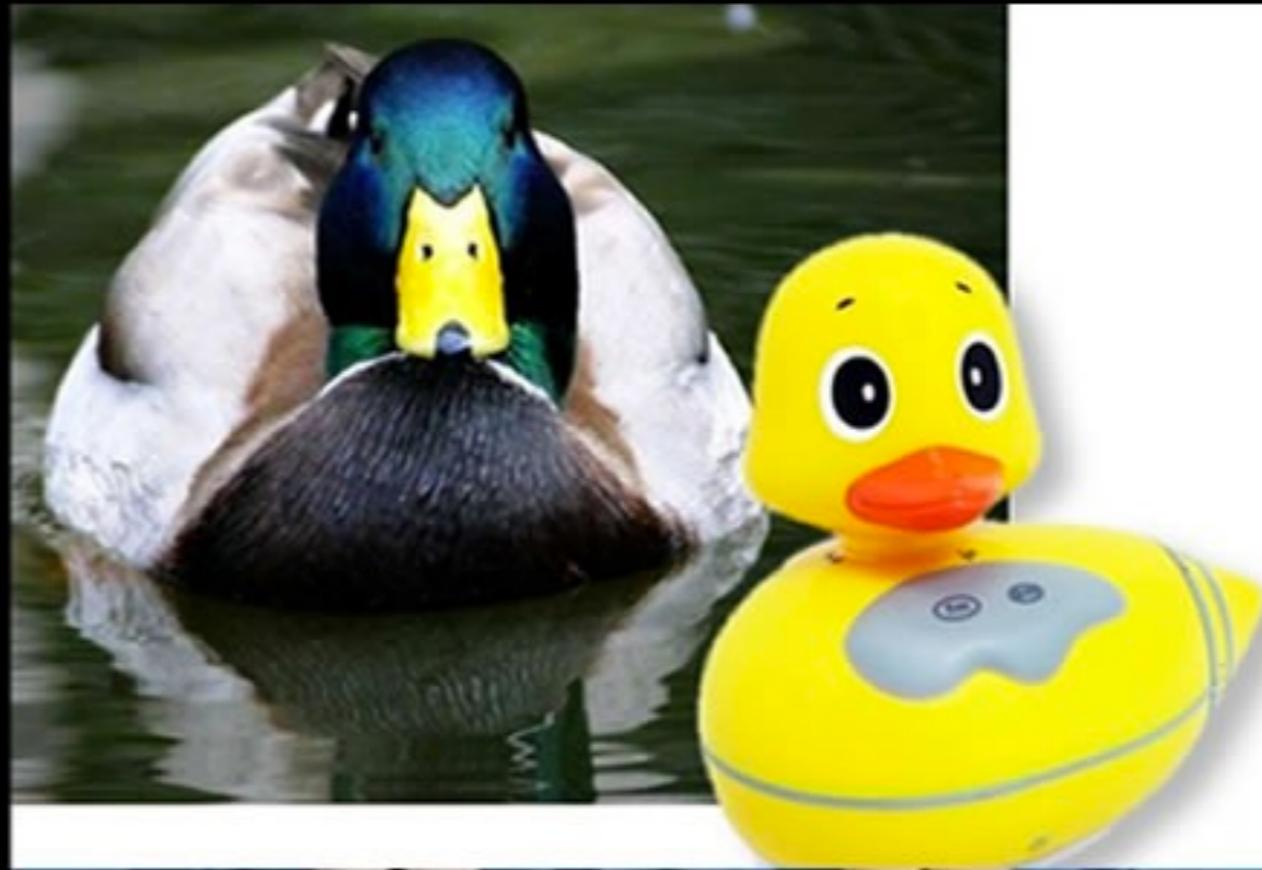
```
+draw()  
-drawCircle()  
-drawRectangle()
```



# Example



# Liskov Substitution Principle

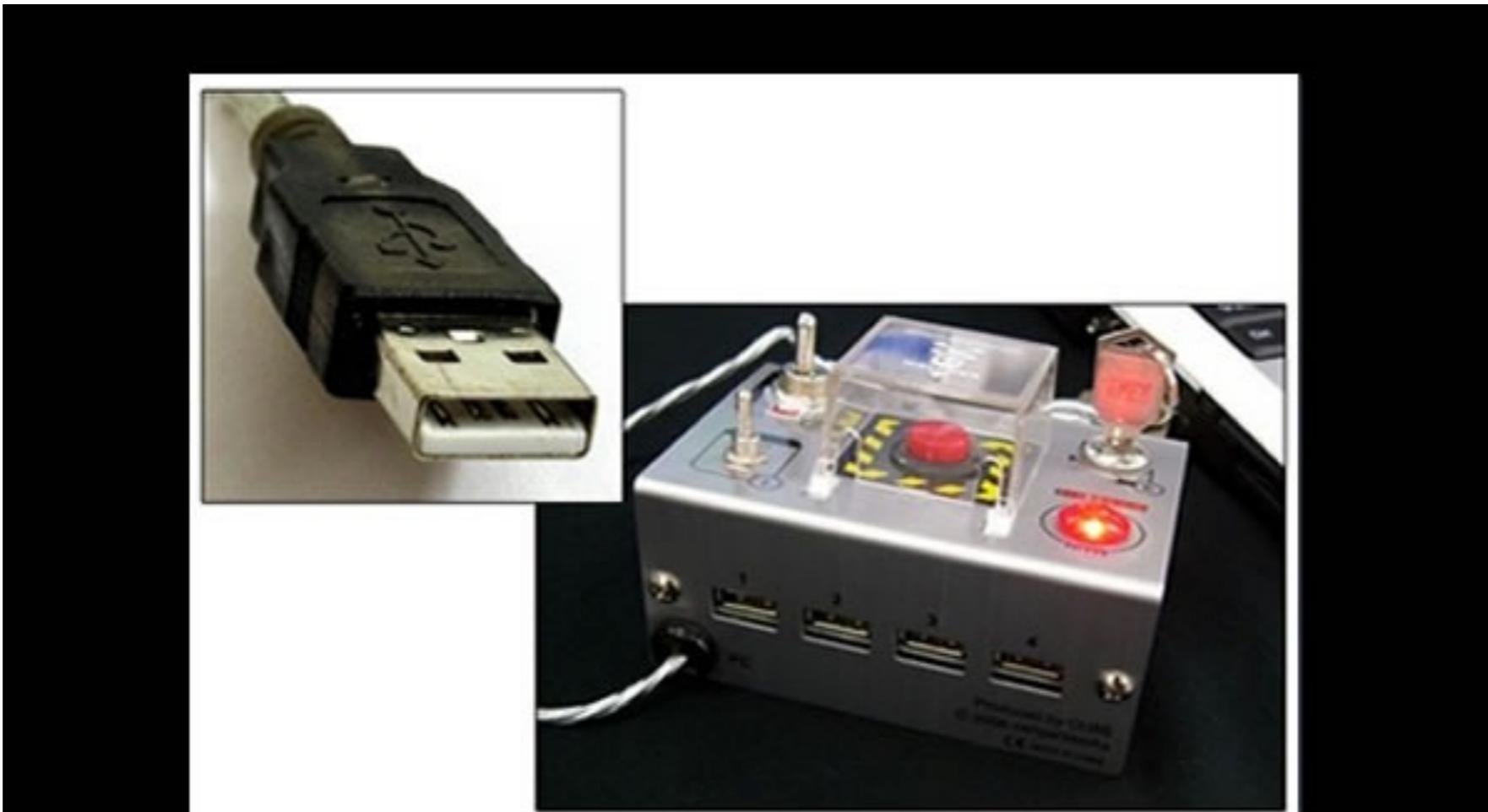


## Liskov Substitution Principle

If it looks like a duck and quacks like a duck but needs batteries,  
you probably have the wrong abstraction.



# Interface Segregation Principle

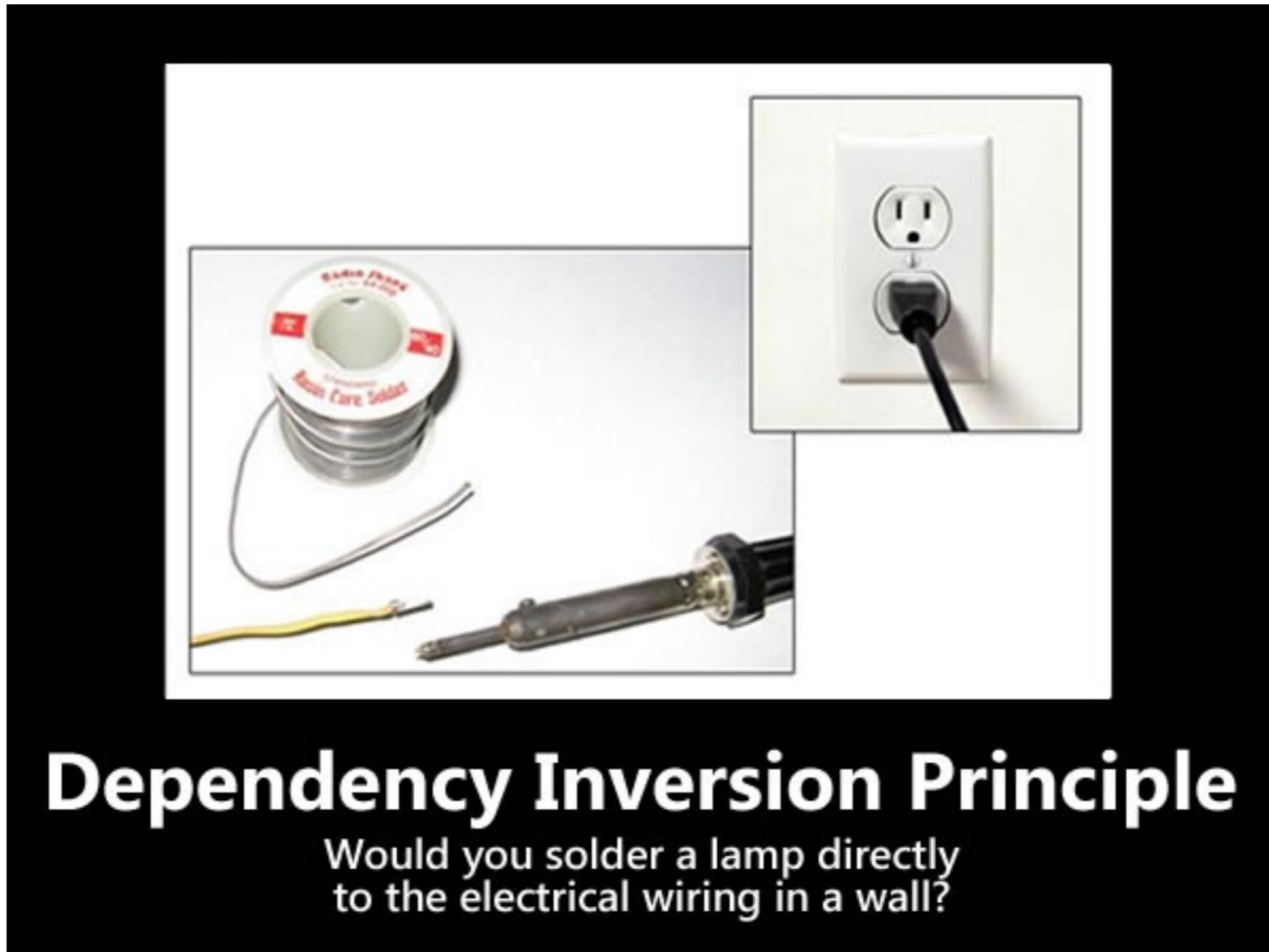


## Interface Segregation Principle

You want me to plug this in *where?*



# Dependency Inversion Principle



## Dependency Inversion Principle

Would you solder a lamp directly  
to the electrical wiring in a wall?



# Manage your dependencies

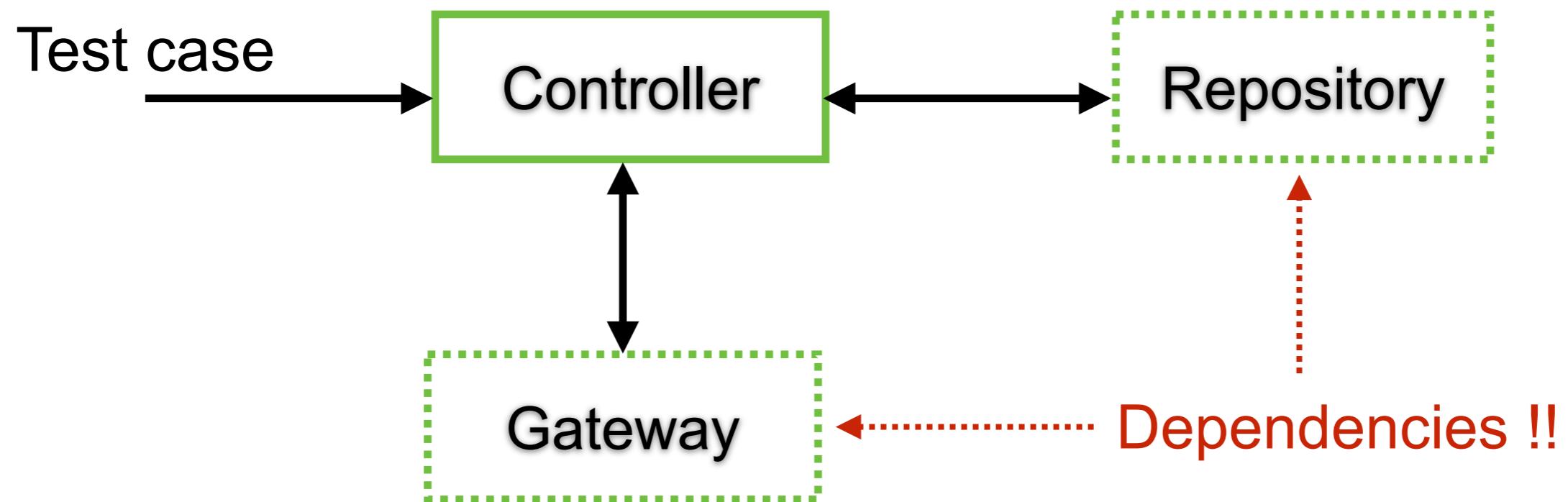


# Good Unit Tests (F.I.R.S.T)

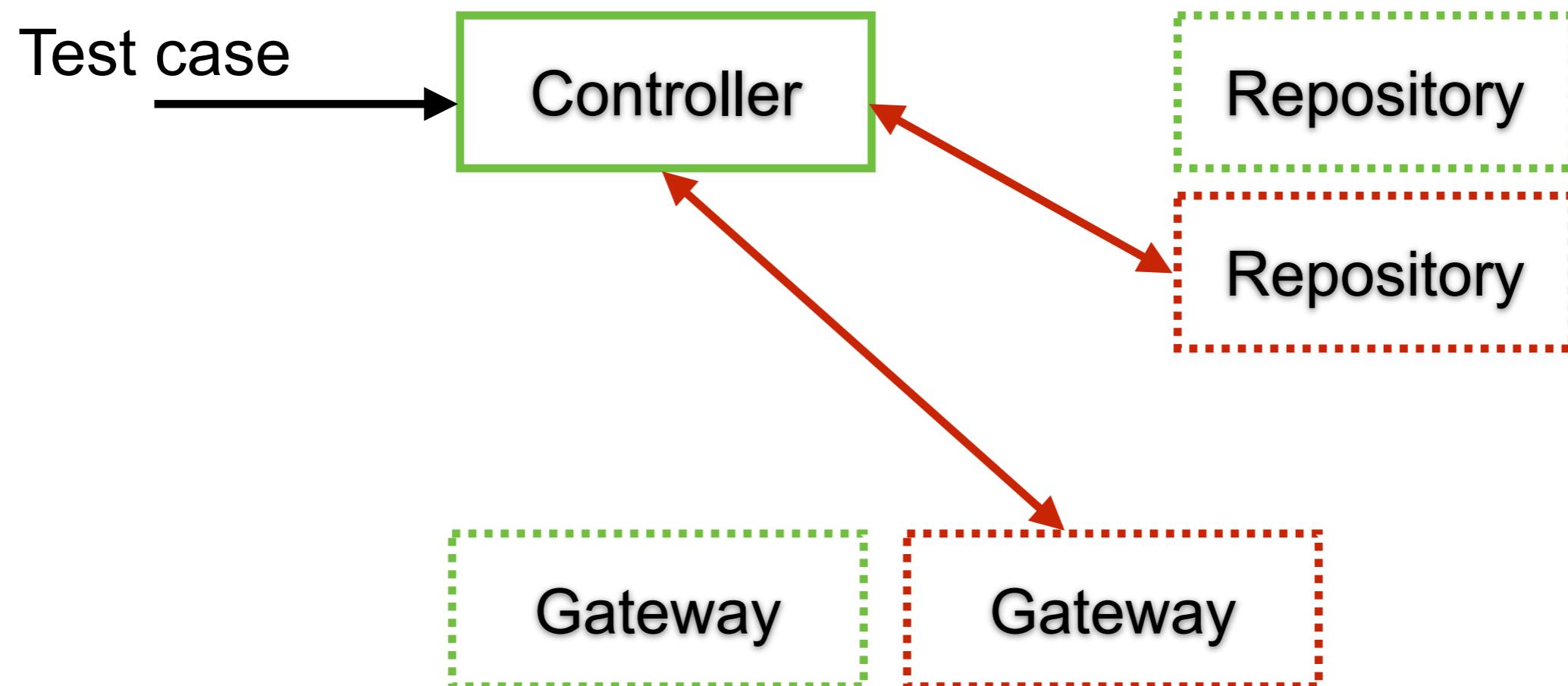
Fast  
Independent/Isolate  
Repeat  
**S**elf-validate  
Thorough/Timely



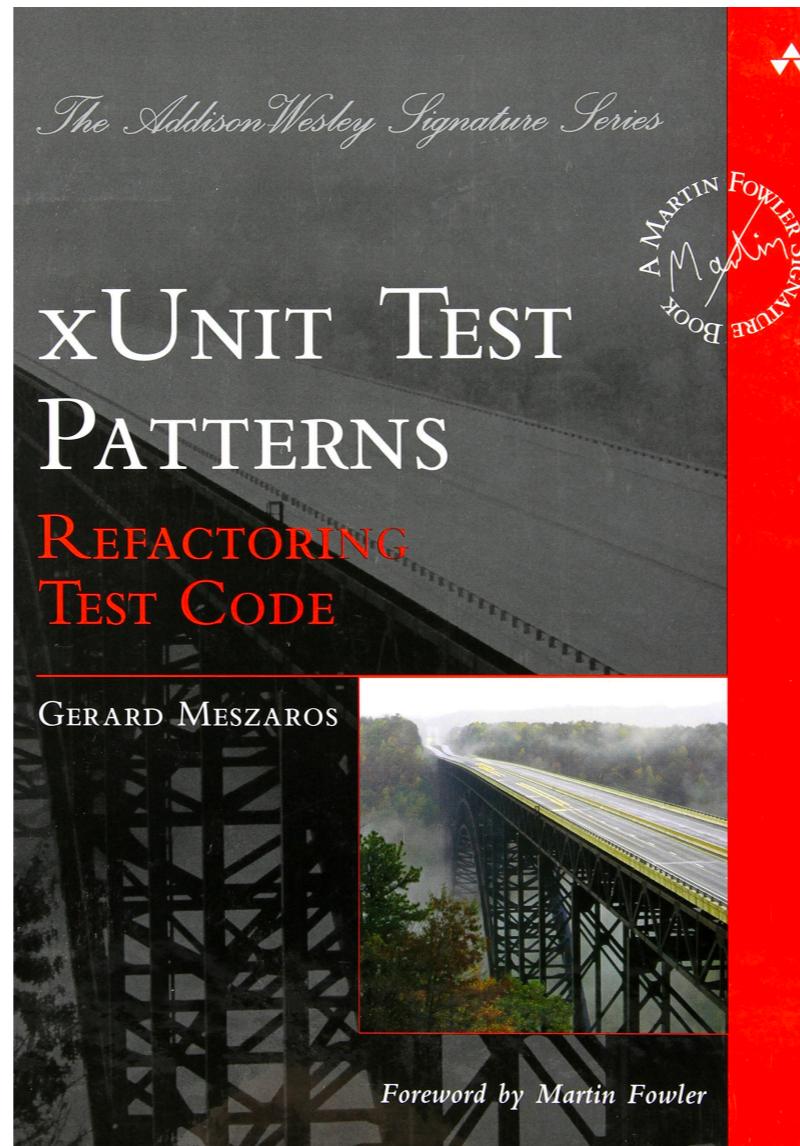
# How to test Controller ?



# Working with dependencies



# Test double ?

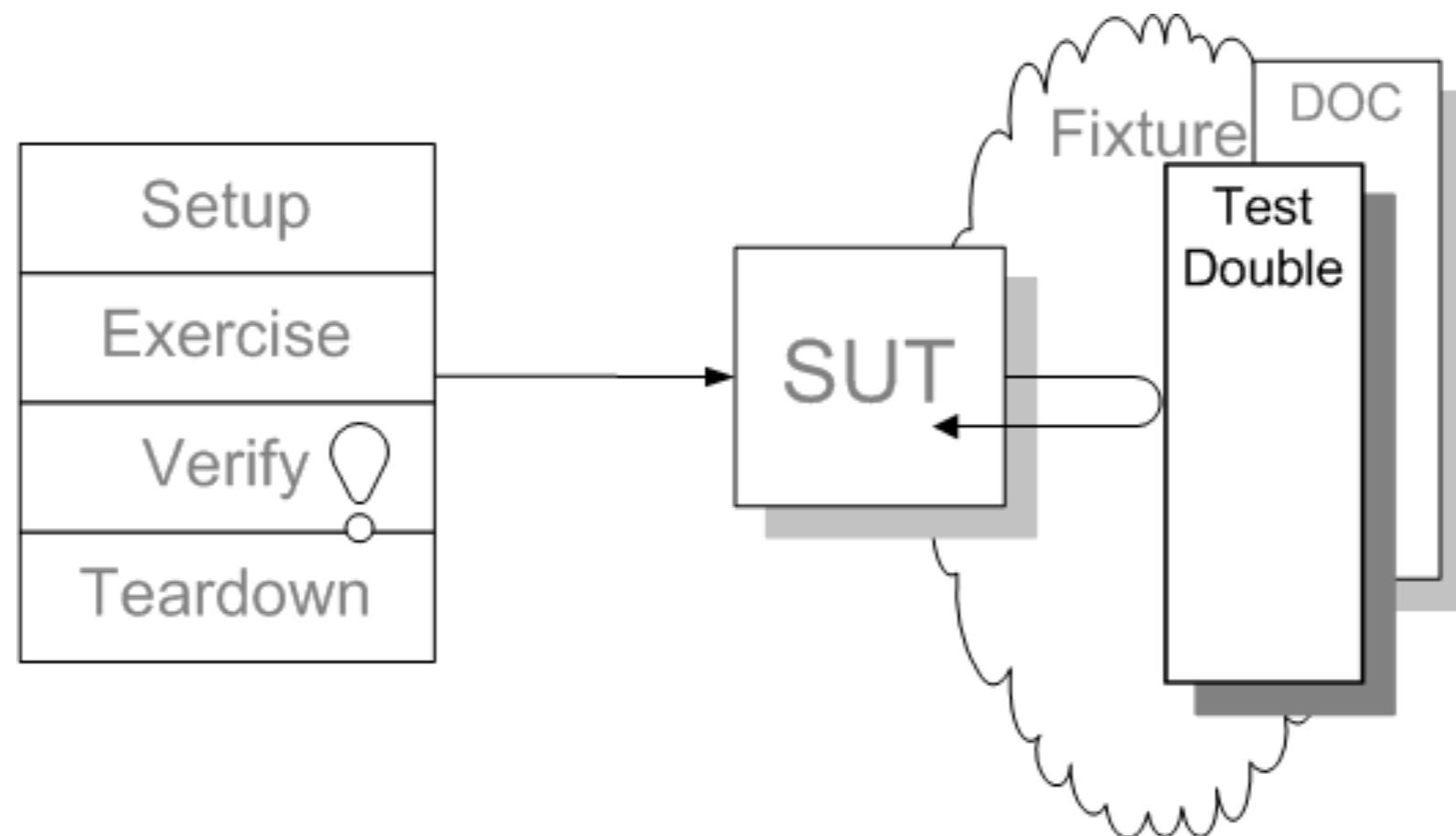


<http://xunitpatterns.com>



# Test double ?

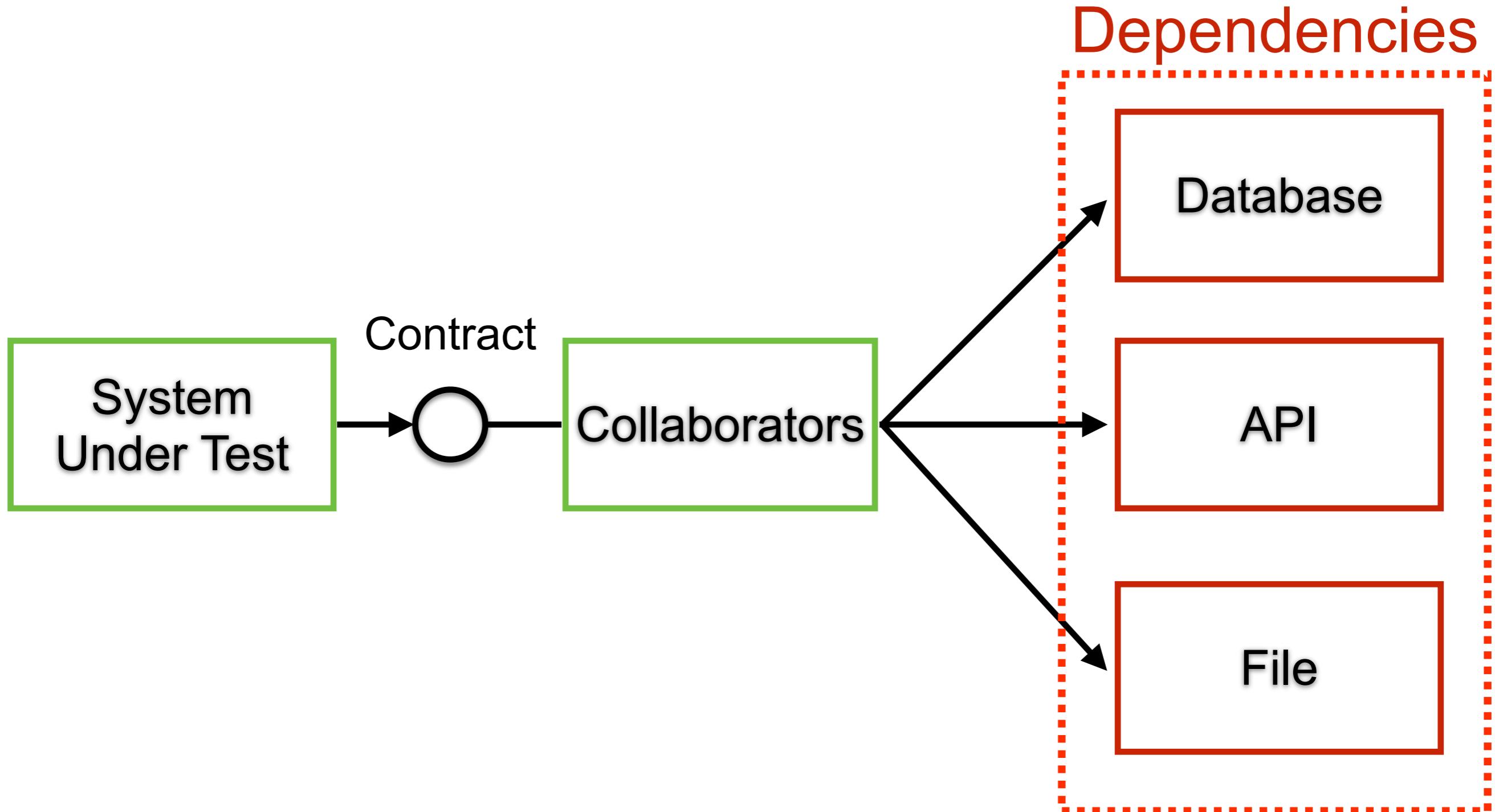
Stand-in for something that would be otherwise real  
in the execution of your program



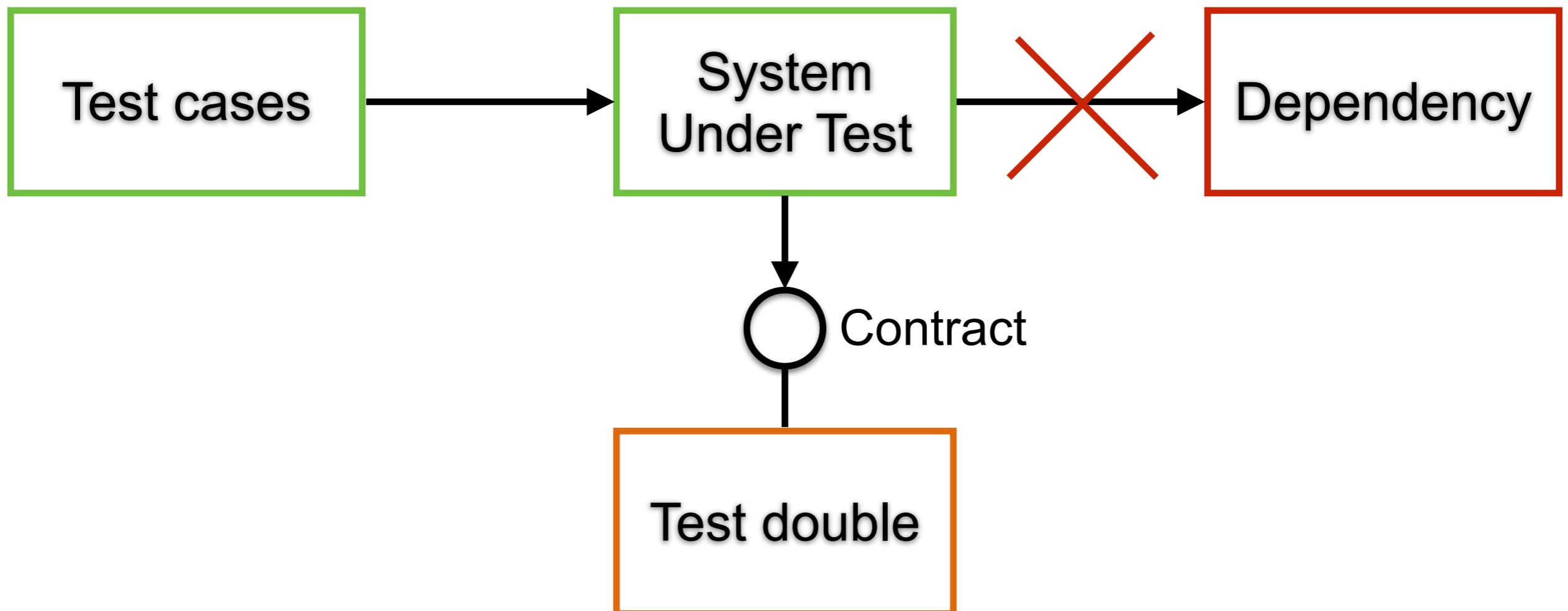
<http://xunitpatterns.com/Test%20Double.html>



# Real implementation



# With Test double



# Types of Test double

Dummy  
object

Stub

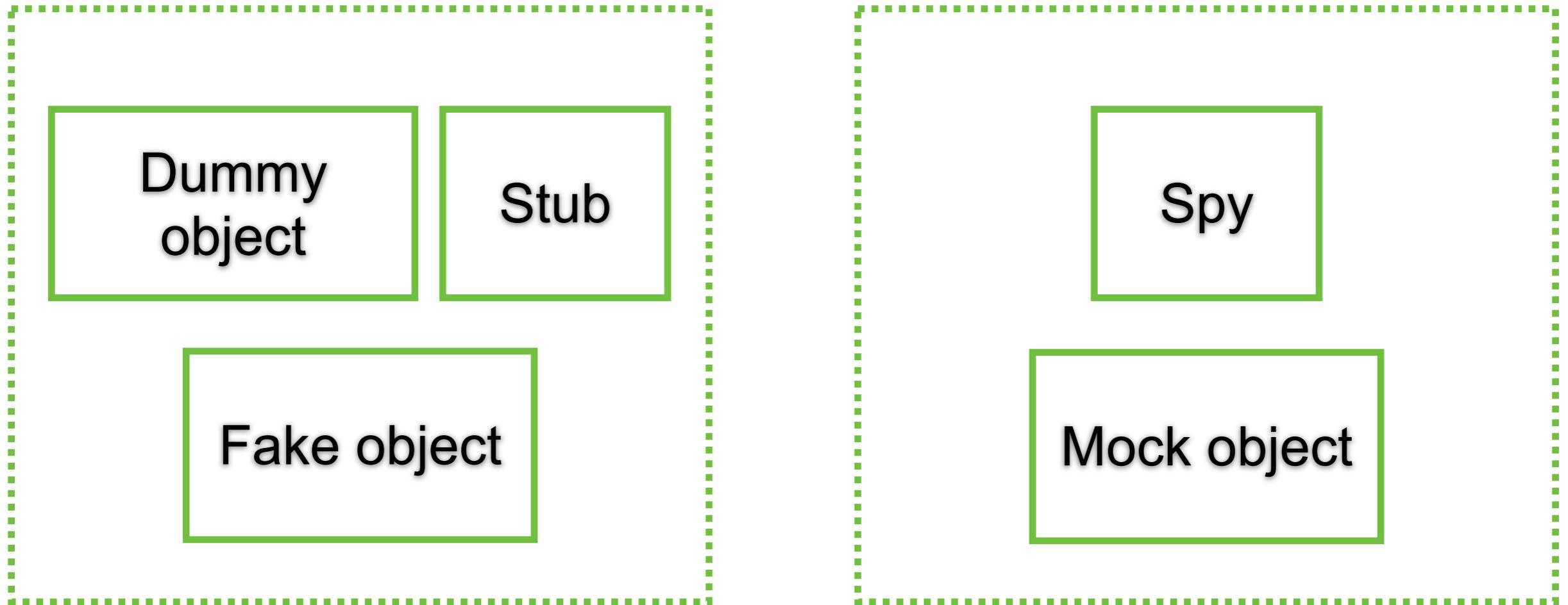
Spy

Mock object

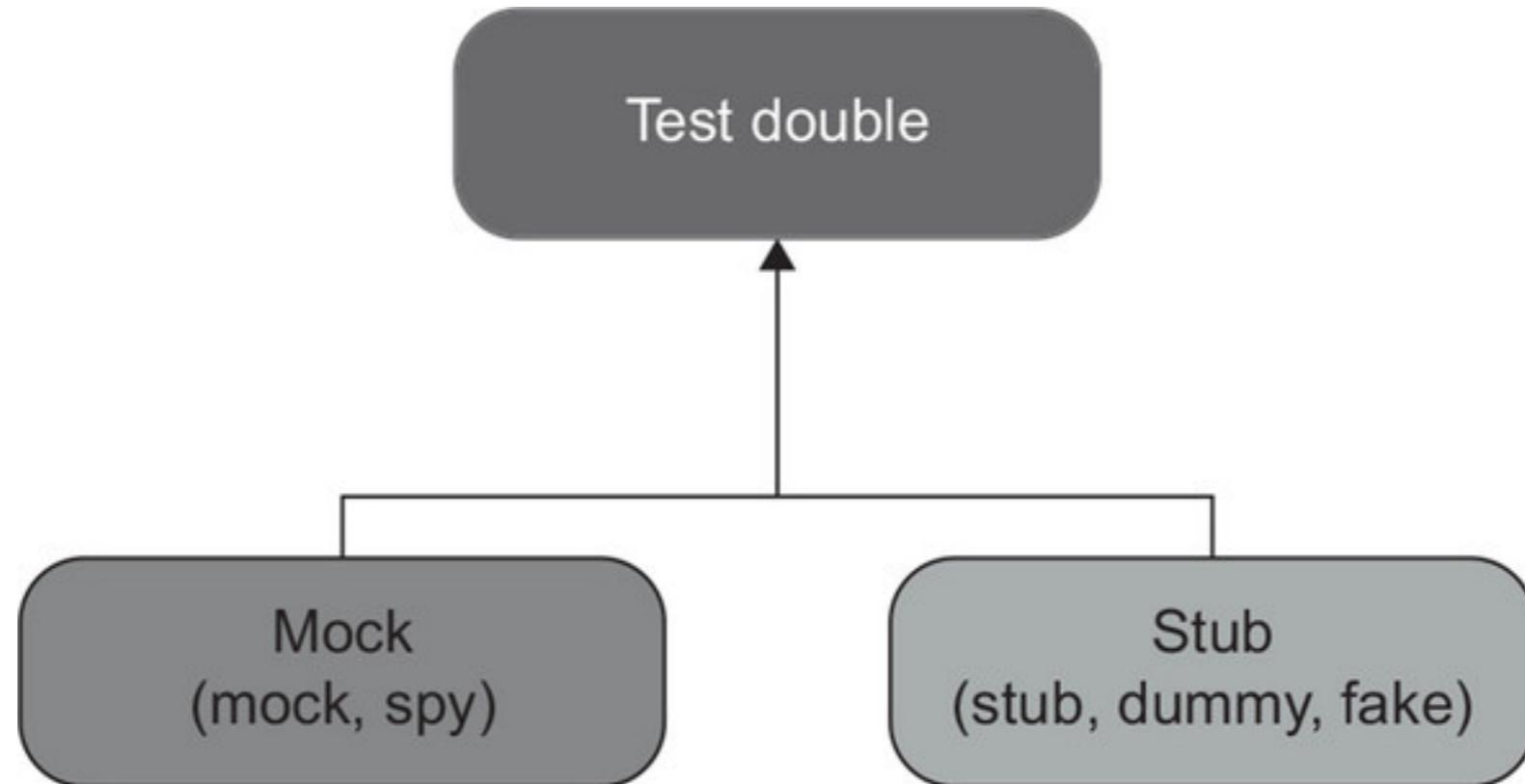
Fake object



# Types of Test double

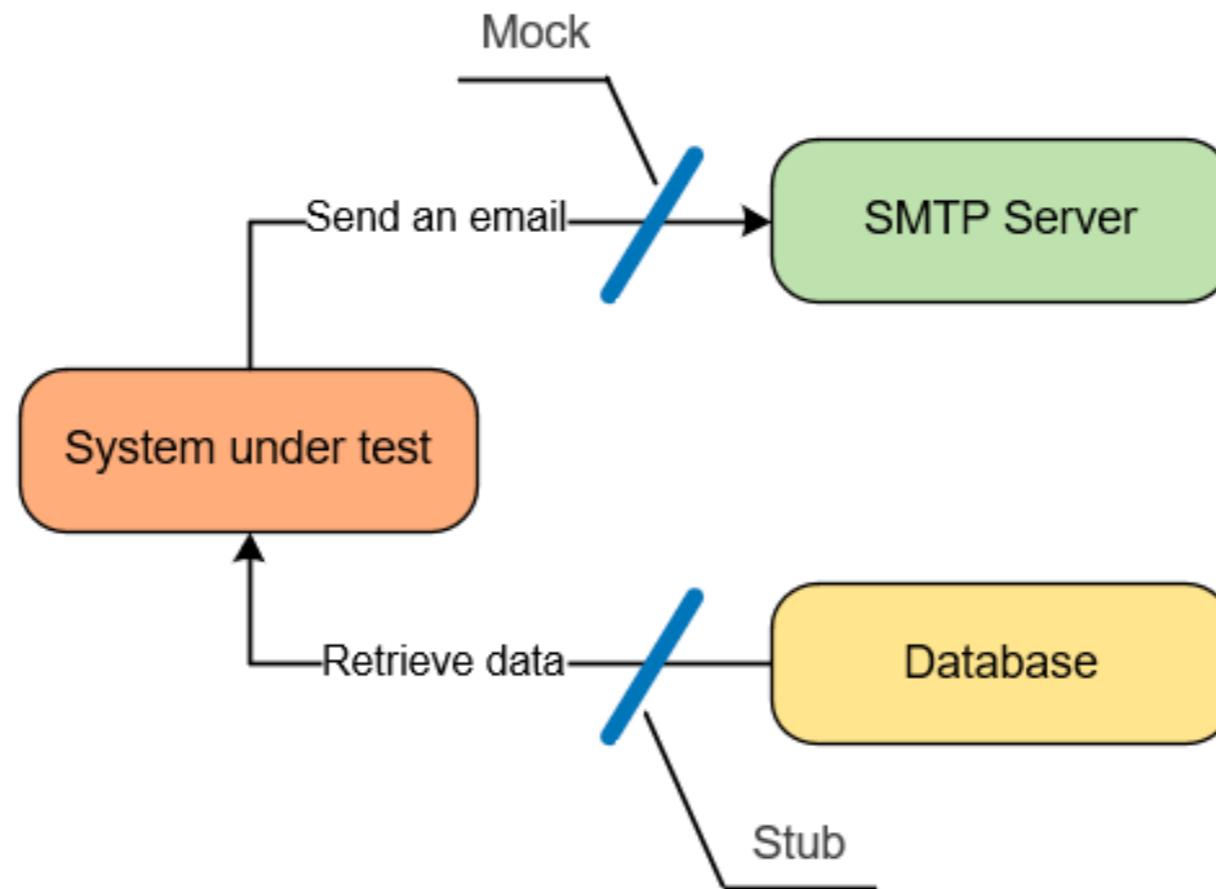


# Types of Test double



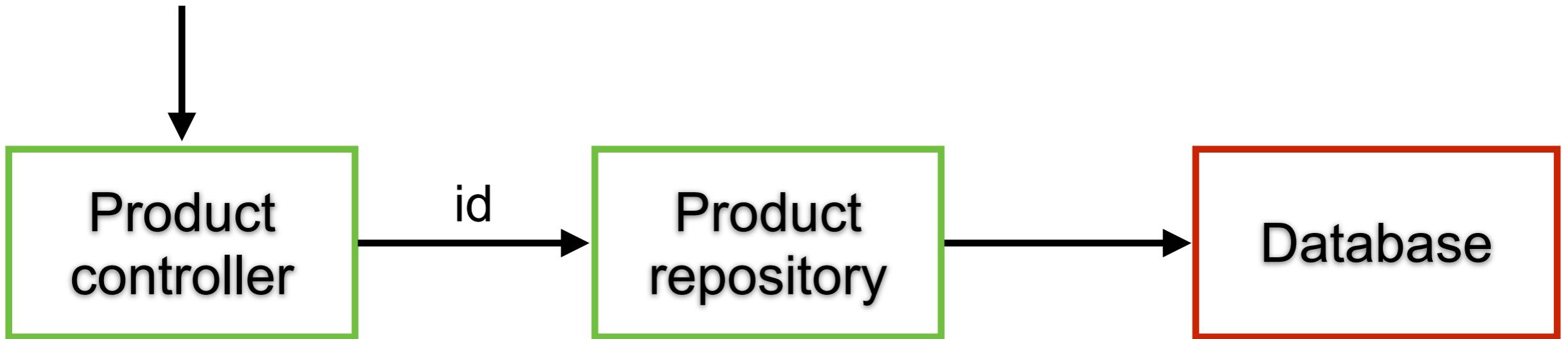
# Mock vs Stub

**Mock** for outgoing interaction  
**Stub** for incoming interaction



# Example

Get product detail by ID



# Fake

Working with implementation but take some shortcut

Not suitable for production

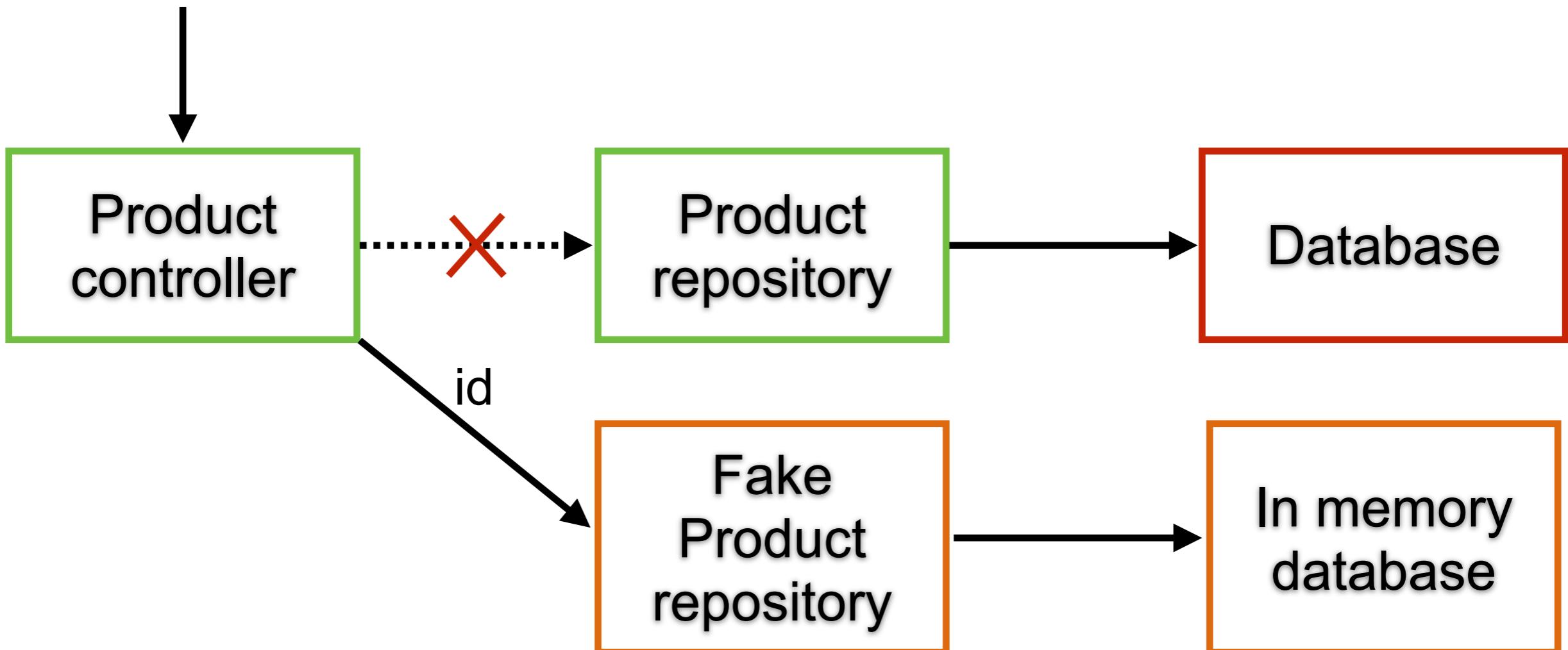
Use with read and write operations

*E.g. In-memory database, Fake API server*



# Fake

Get product detail by ID



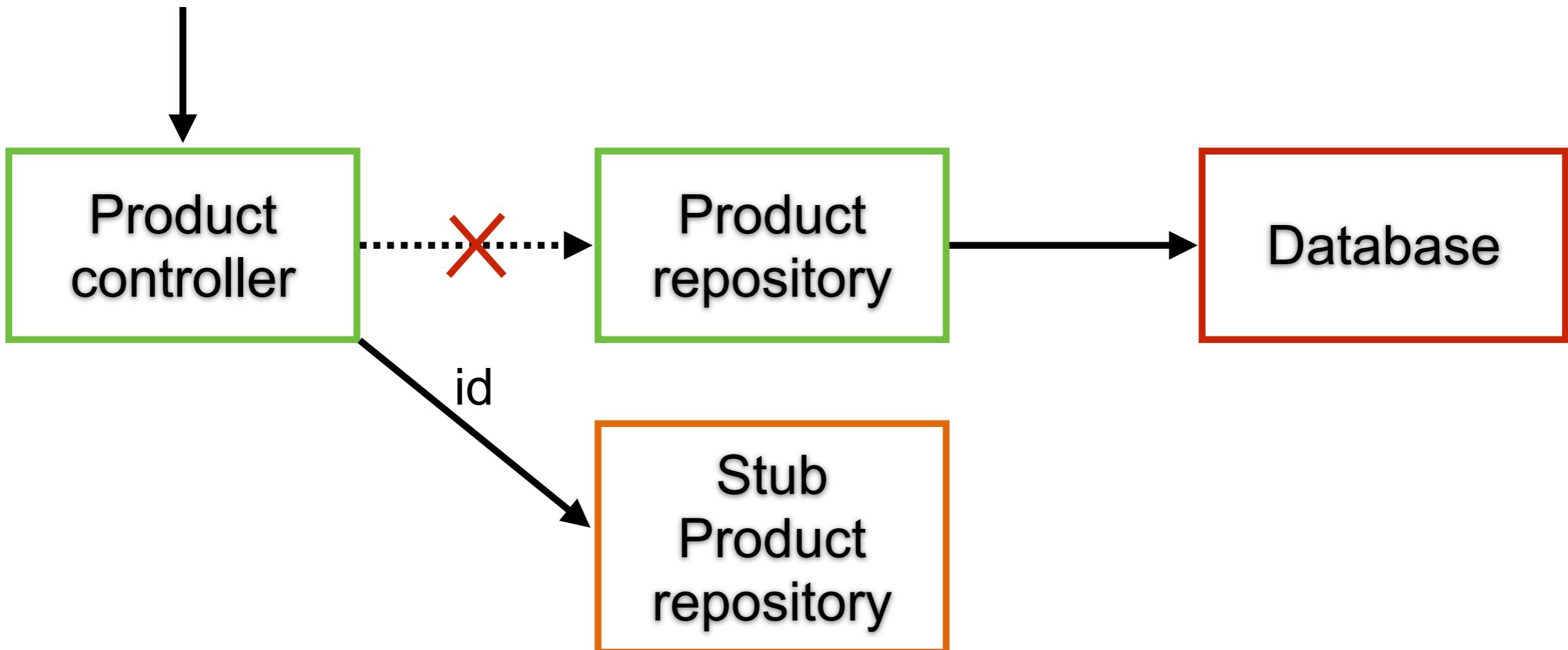
# Stub

Provide answers to calls made during the test  
A double with hardcoded return values



# Stub

Get product detail by ID



# Spy

Like stub

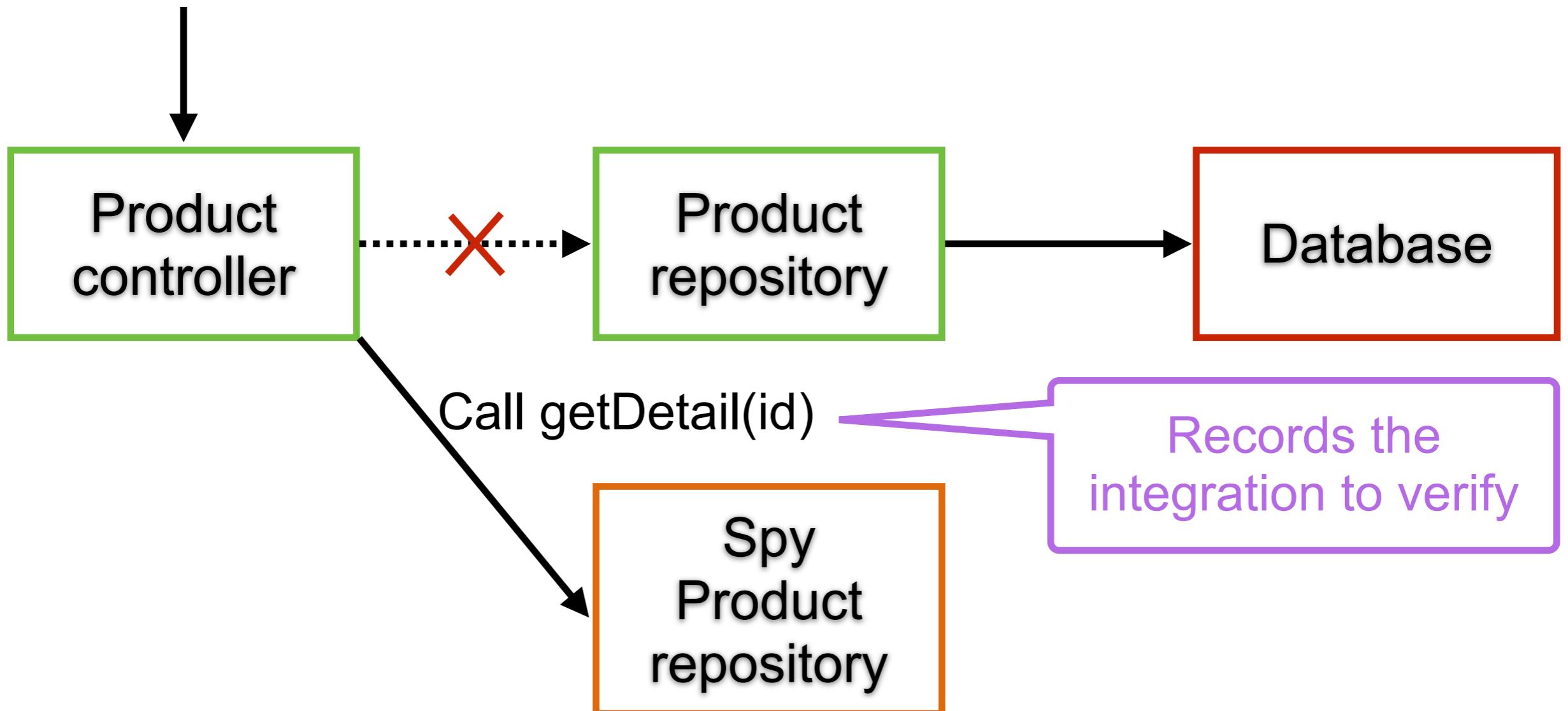
Record some information based on how its called

*E.g. how many message it was sent via email service*



# Spy

Get product detail by ID



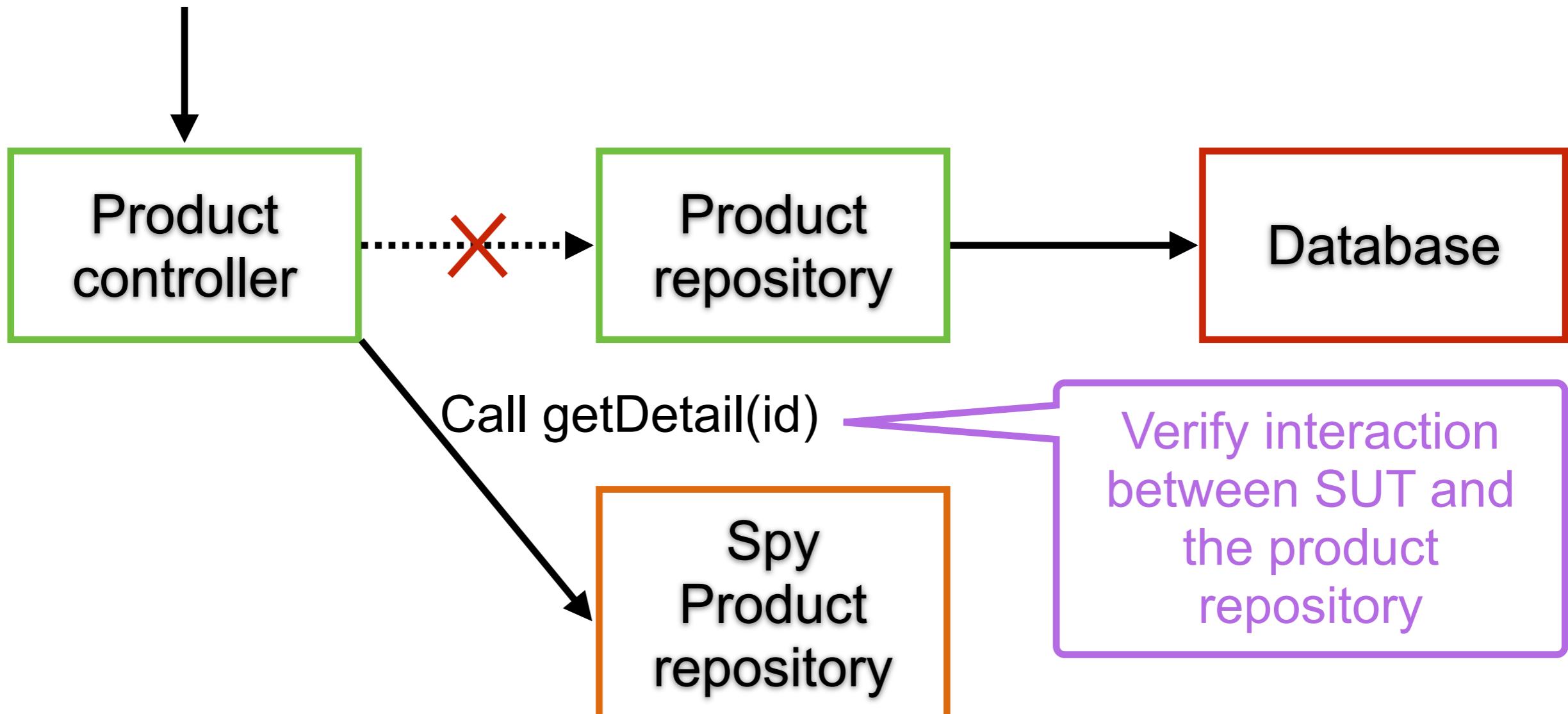
# Mock object

Pre-programmed with expectations with spec  
Mock object can throw an exception if receive a call  
that don't expect



# Mock object

Get product detail by ID

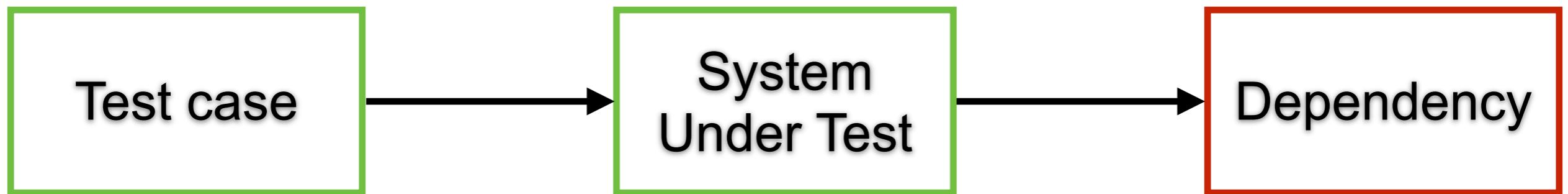


# Dummy object

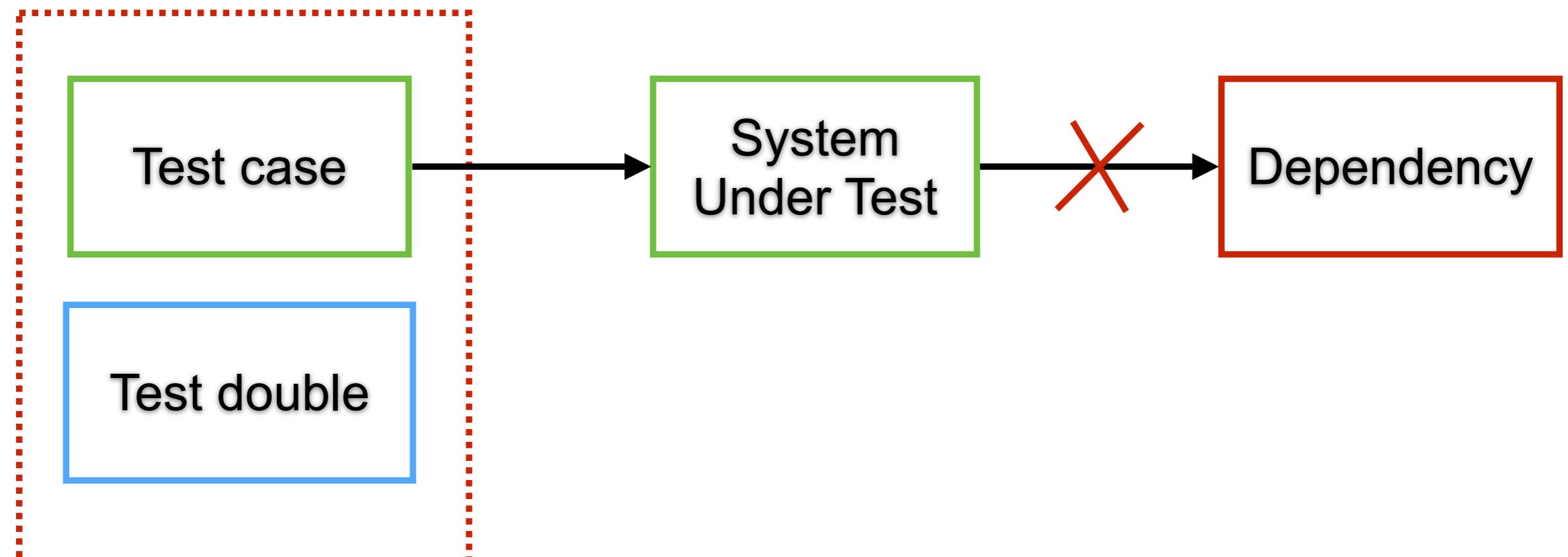
Passed around but never actually used  
Used to fill parameter lists



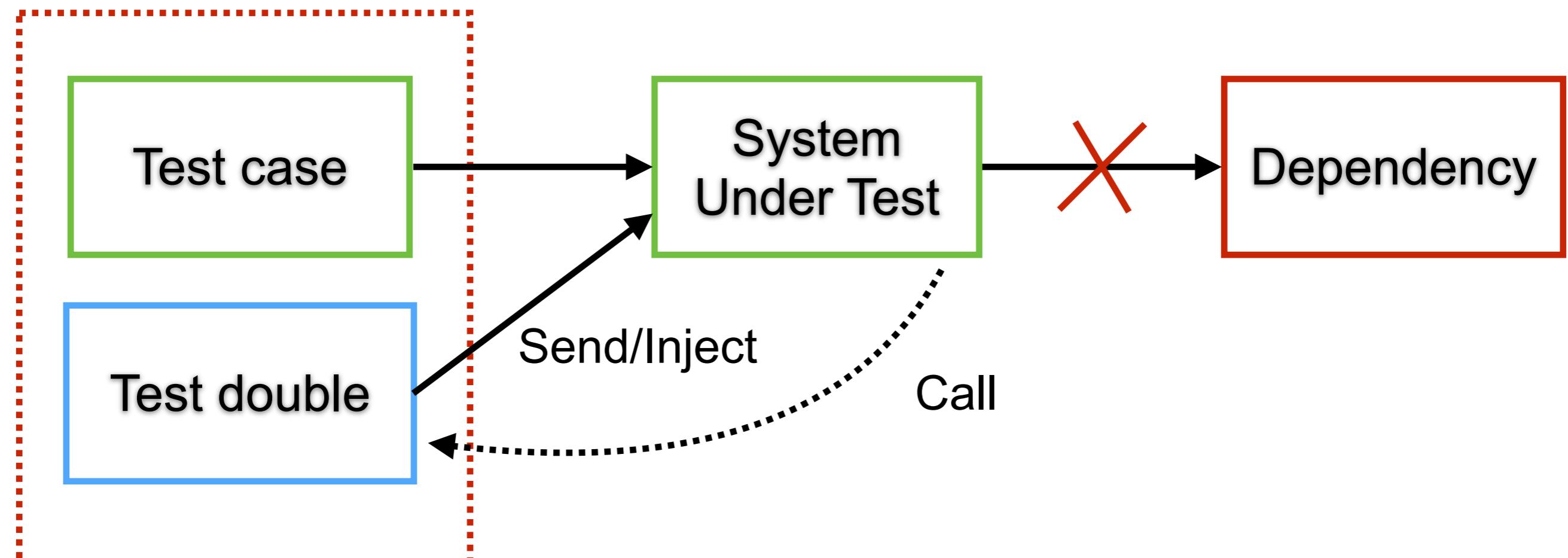
# Test double ?



# Create test double



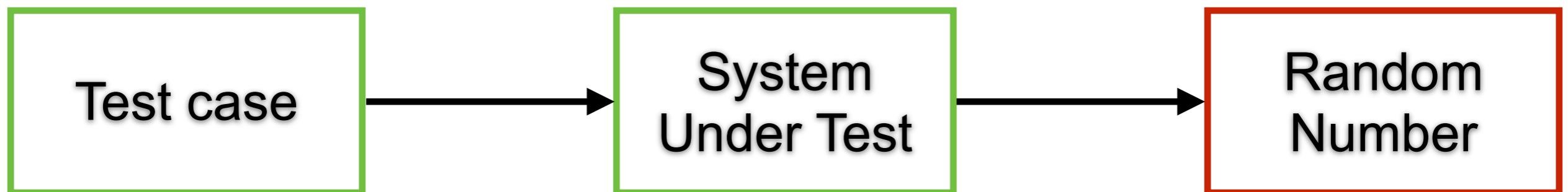
# Send/inject test double to SUT



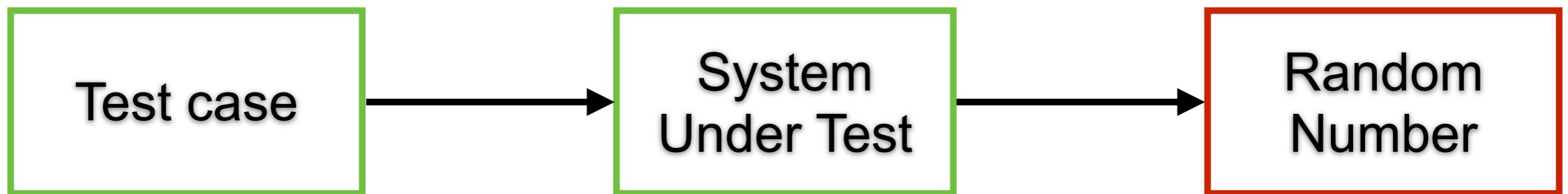
# Workshop



# Workshop



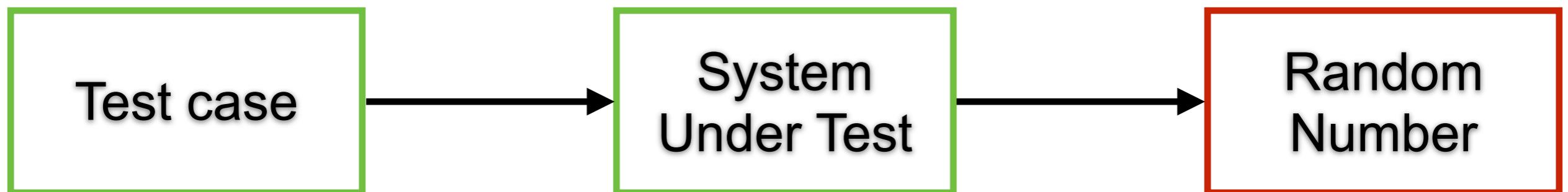
# Workshop



Test random number = 5 ?



# Workshop



Test random number must called 1 time ?



# Working with Mockito



<https://site.mockito.org/>



# Configure Mockito with Gradle

Edit file build.gradle

```
dependencies {  
    ...  
    testImplementation "org.mockito:mockito-core:3.6.0"  
    testImplementation "org.mockito:mockito-junit-jupiter:3.6.0"  
    ...  
}
```



# Using Mockito with JUnit 5

```
@ExtendWith(MockitoExtension.class)
public class DemoWithMockito {

    @Mock
    Random random;

    @Test
    public void usingMockito() {
        // Create stub
        when(random.nextInt(10))
            .thenReturn(5);
    }
}
```



# Workshop with Login process



# Test login

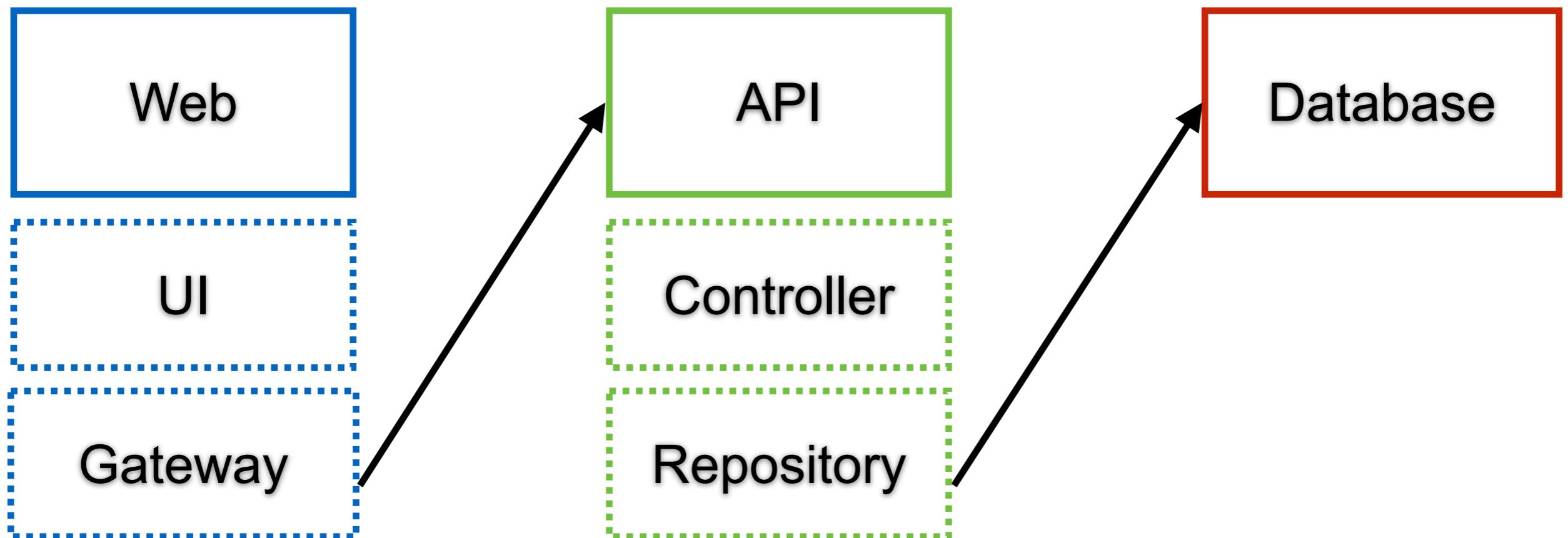
User name	Password	Expected result	Comment
Somkiat	PasswordSomKiat	Access to system as Somkiat	Valid
Som kiat	PasswordSomKiat	Error	Space in user
Somkiat	Password SomKiat	Error	Space in password
Somkiat	1234	Error	Password too short



# Workshop :: Login



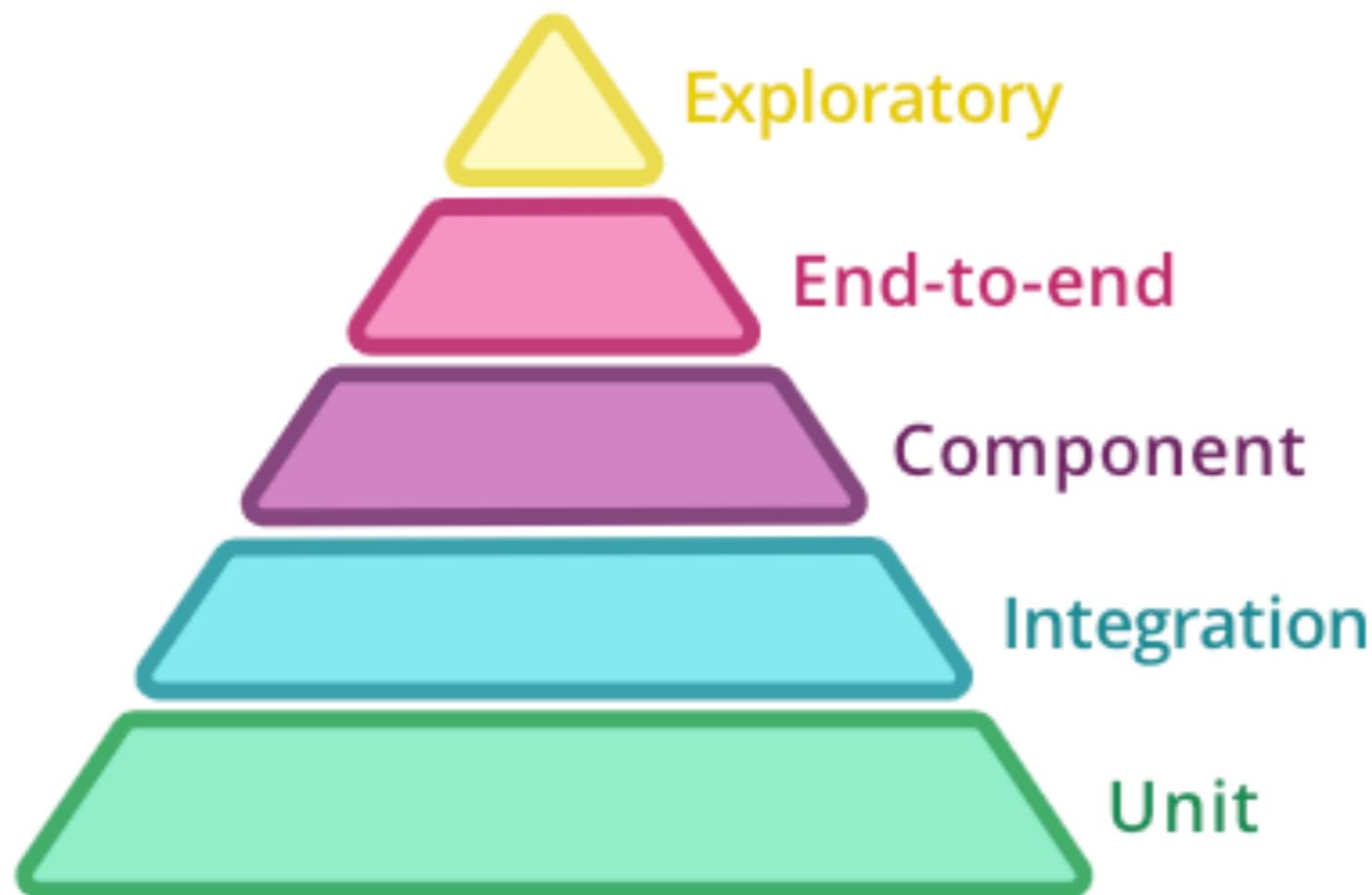
# Workshop :: Login



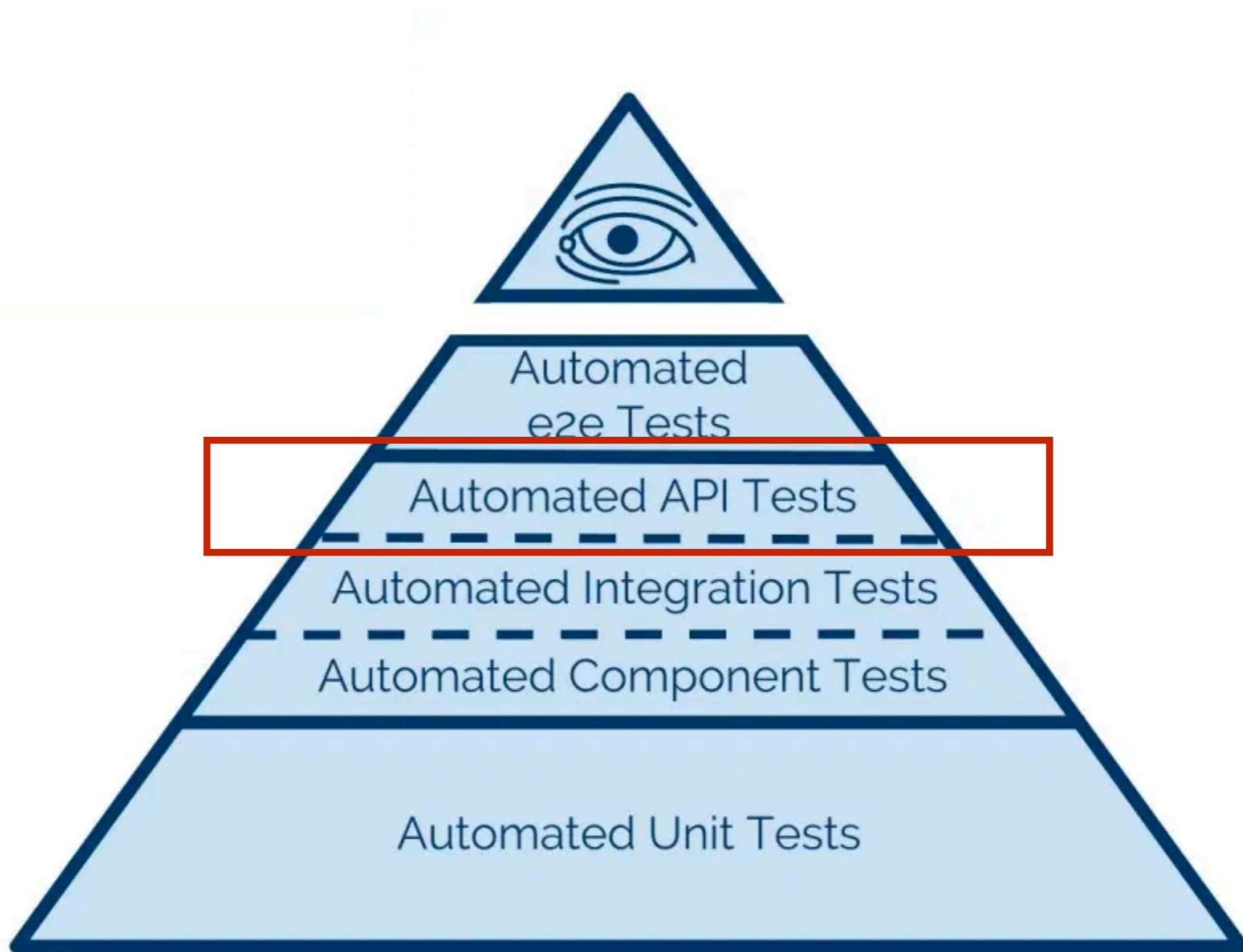
# What to test ?



# What to test ?



Somkiat	PasswordSomKiat	Access to system as Somkiat	Valid
---------	-----------------	-----------------------------	-------



# Code Example

<https://github.com/up1/demo-login-api>



# Books



