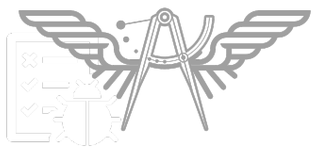
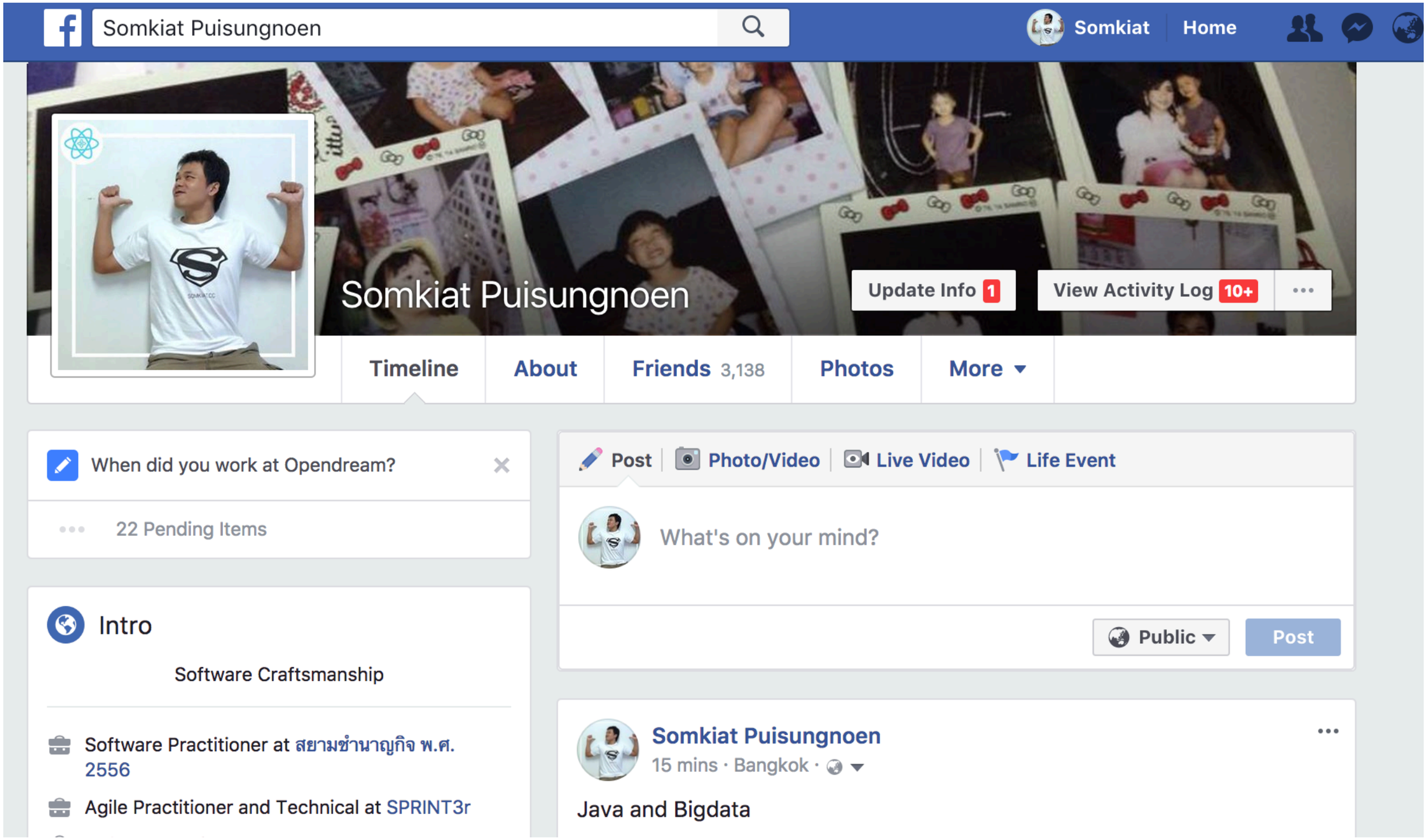
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Automated Testing For iOS app with Swift





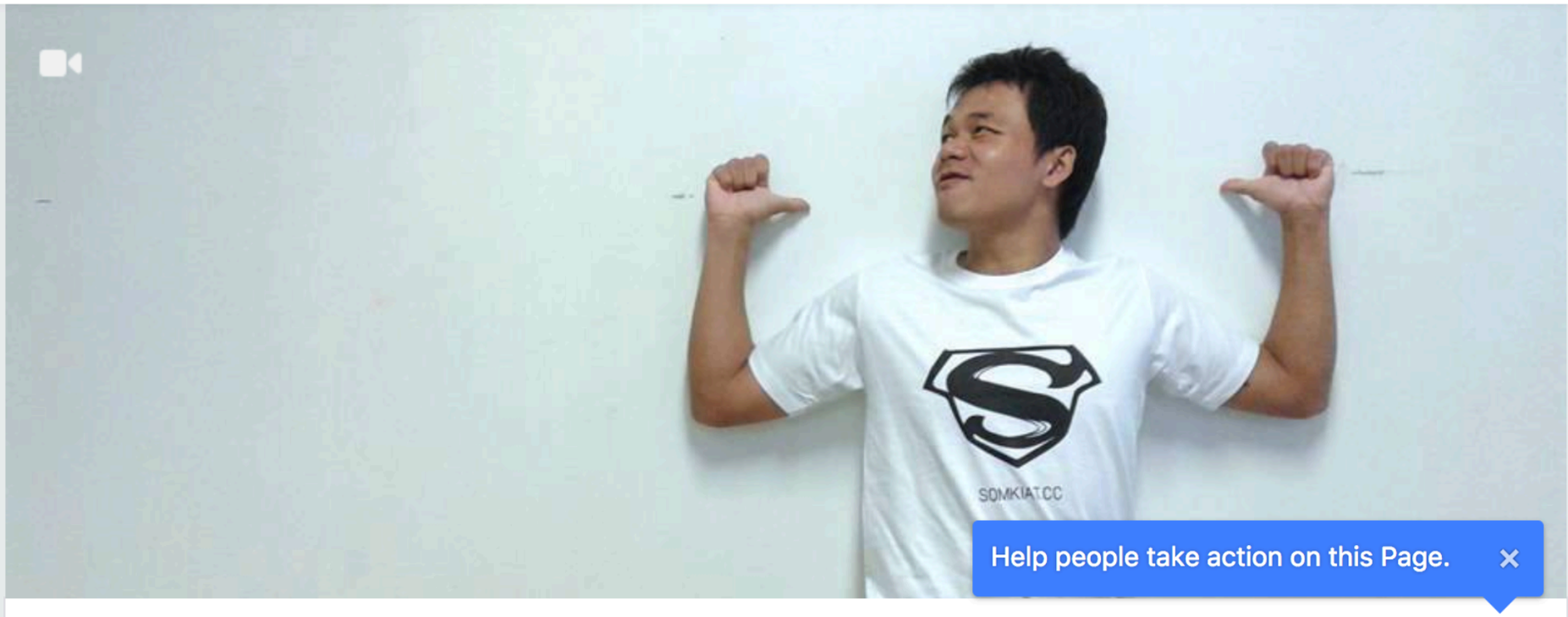
somkiat.cc

Somkiat | Home

Page | Messages | Notifications 3 | Insights | Publishing Tools | Settings | Help

somkiat.cc

@somkiat.cc



Home

Posts

Videos

Photos

Help people take action on this Page.

Liked | Following | Share | ...

+ Add a Button

Agenda

- Introduction of testing
- Why we need to test ?
- Types of tests
- Testing pyramid concept
- iOS app testing
- Workshop (step-by-step)

Agenda

- UI testing
- UI testing workshop
- Code coverage
- Dependency Injection
- Testable application
- CI/CD process and framework

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

Edsger Dykstra, 1970, Notes on Structured Programming

Why we need to test ?

Help you to **catch bugs**
Develop features **faster**
Enforce **modularity** of your project

Why we need to test ?

Confidence
Verification
Learning

But,

It’s <u>take time</u> to learning and practice !!

Testing can be done by Manually or Automated

Why automation ?

Faster regression testing
Easier refactoring
Continuous Integration

Let’s start

Goals

How to **THINK** when and where you should test ?

What you need to know ?

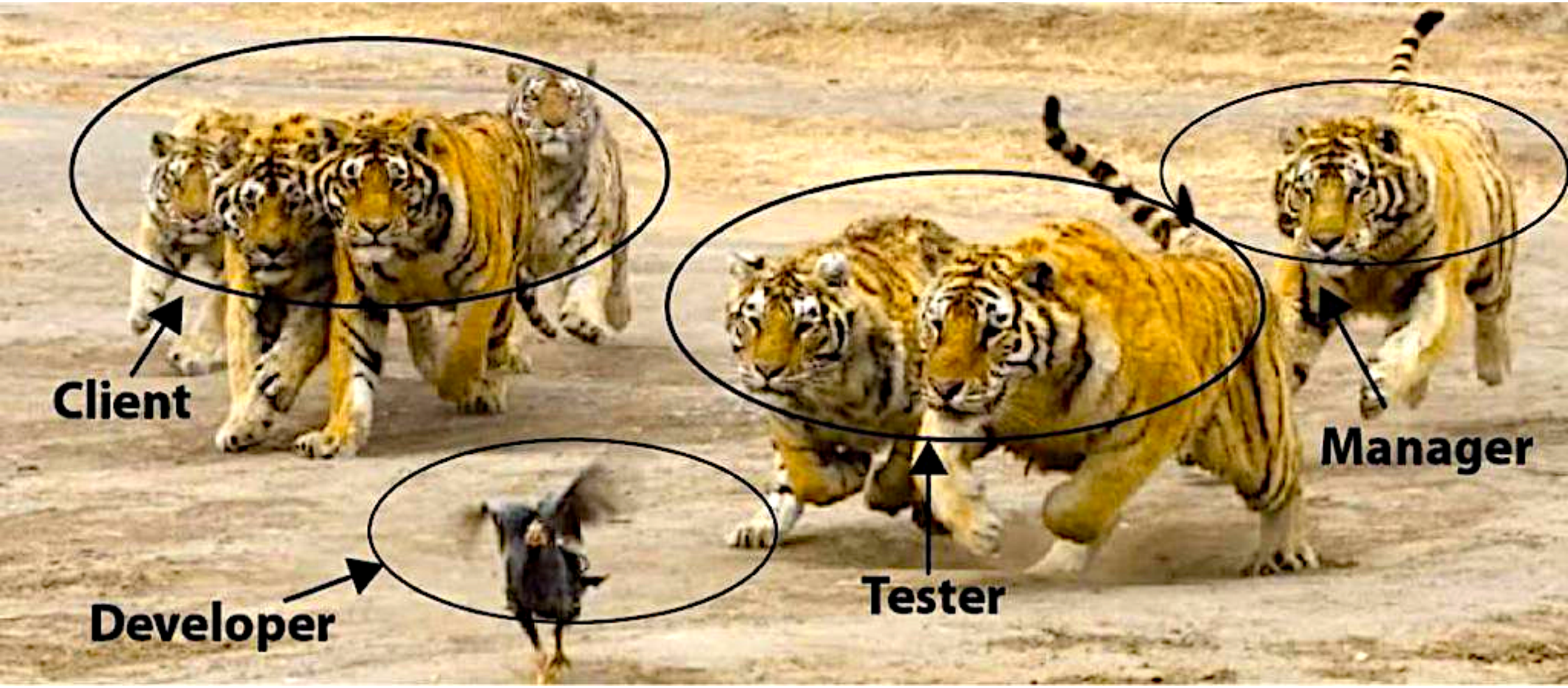
Swift

XCode

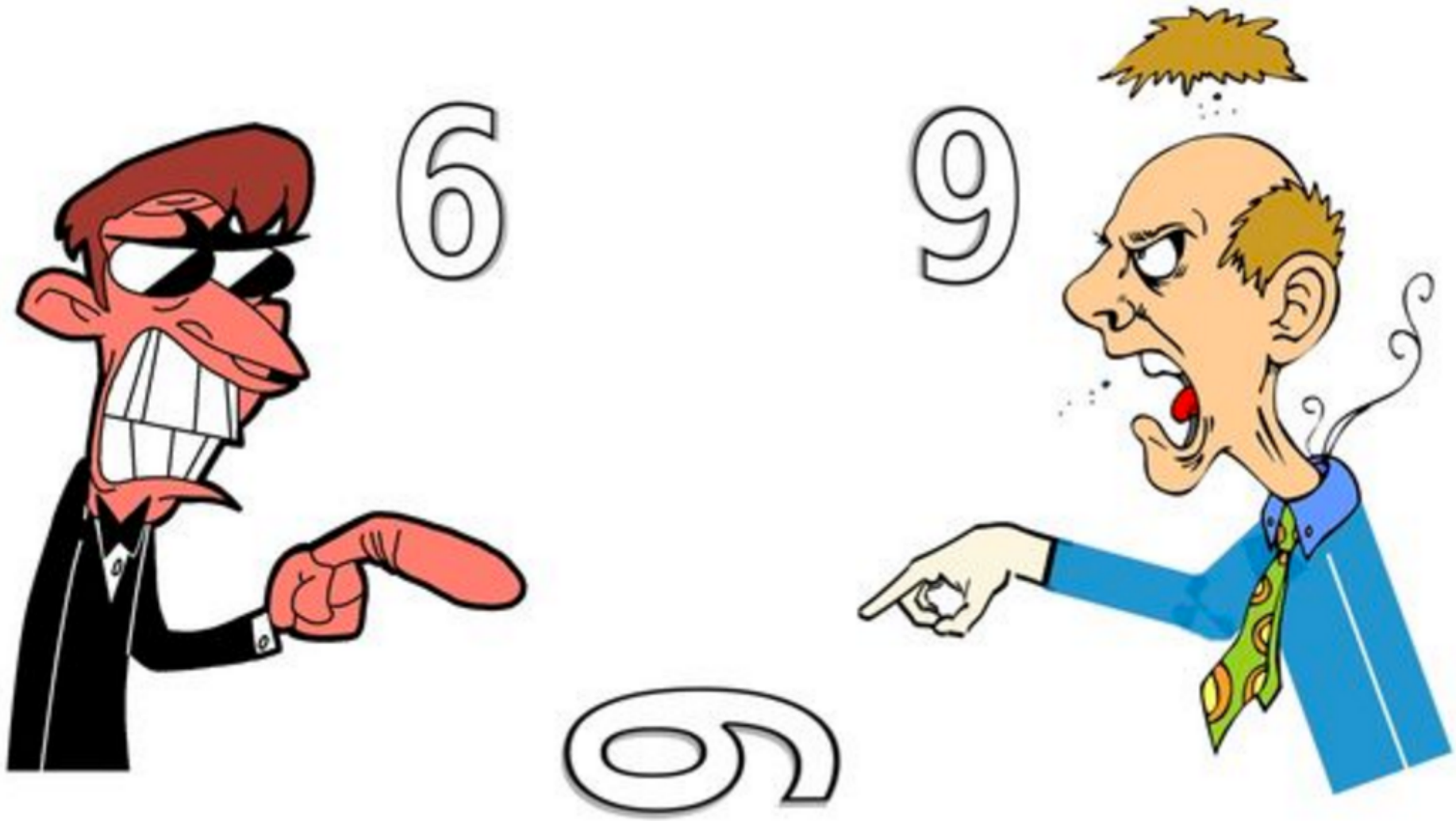
XCTest

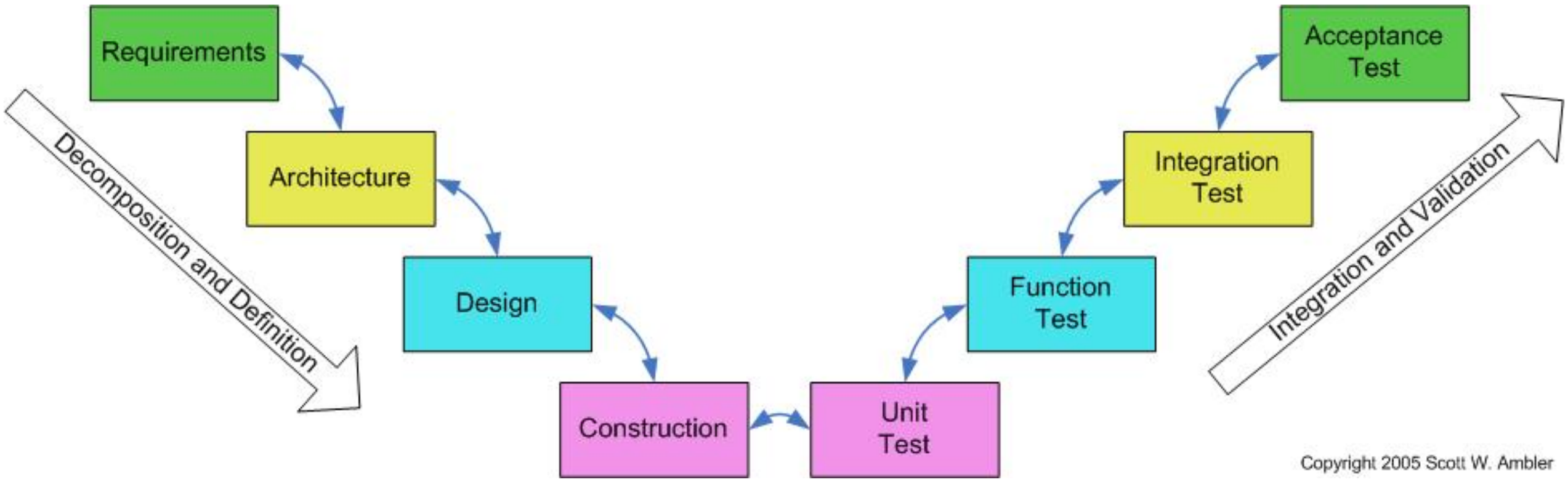
XCUITest

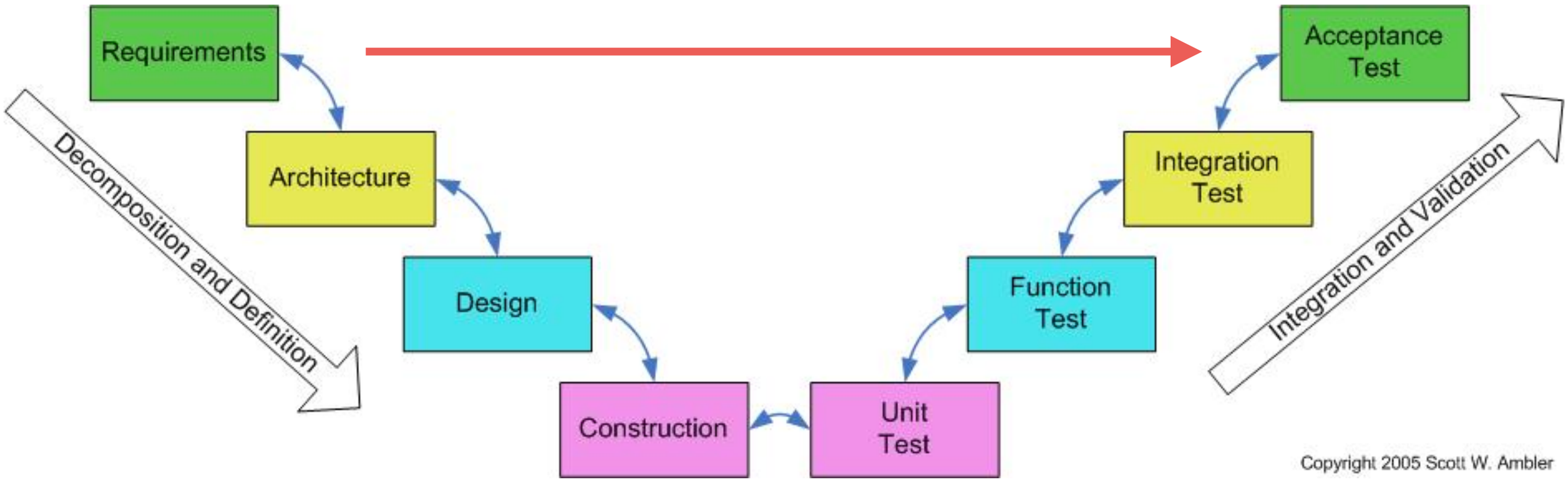
All about Testing

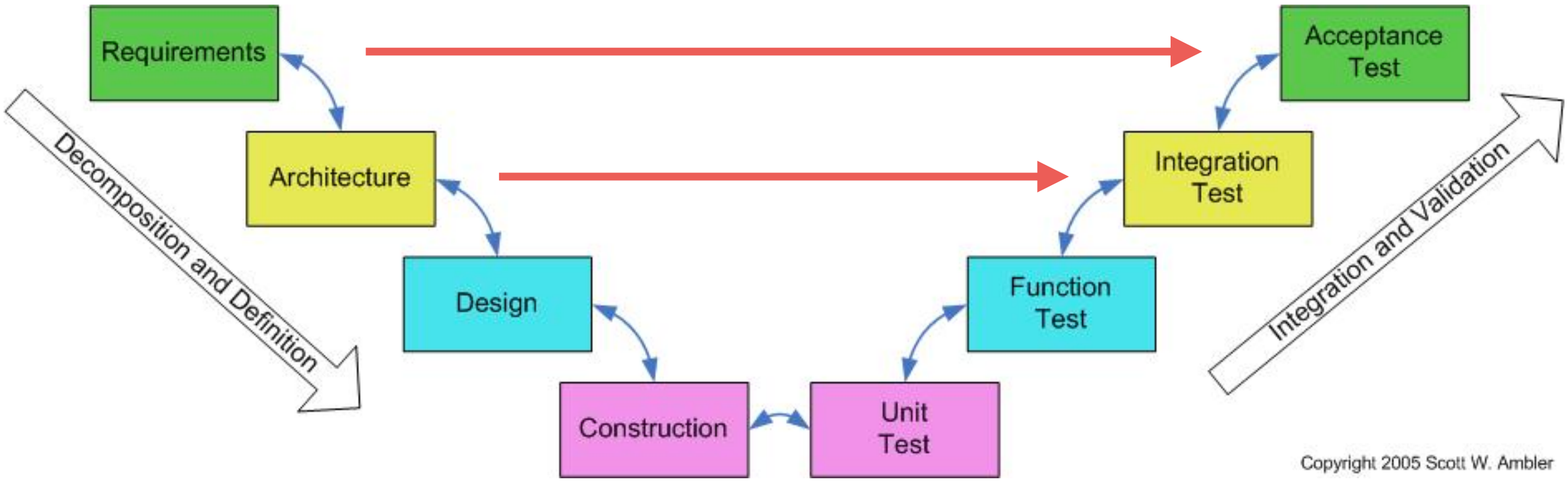


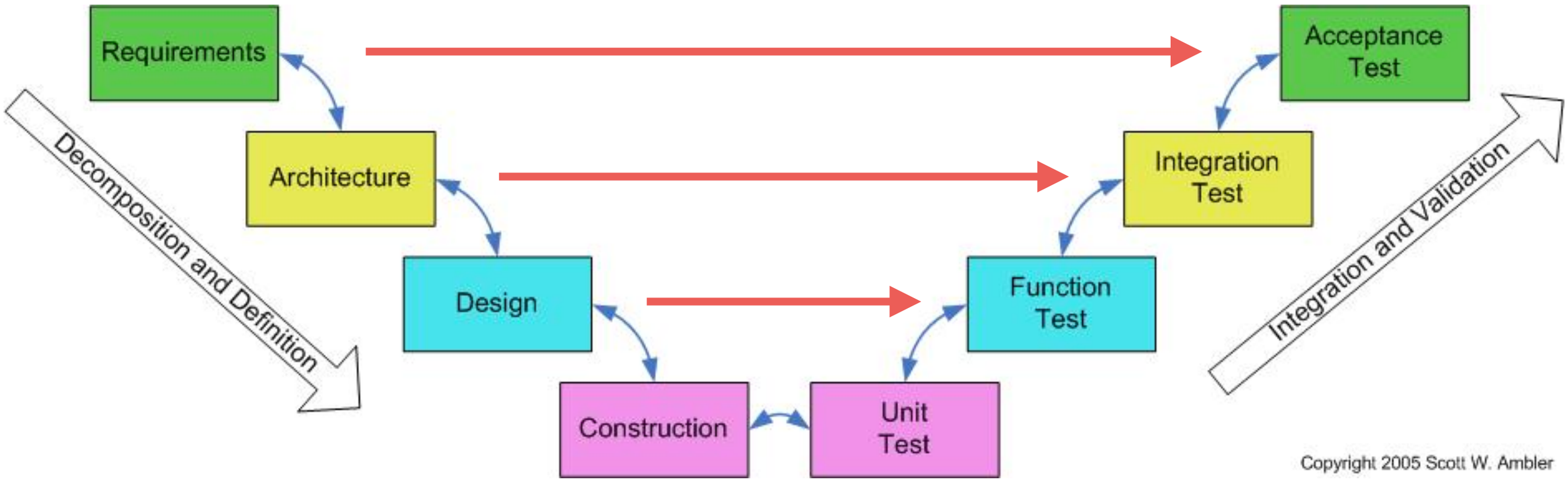
Developer vs Tester

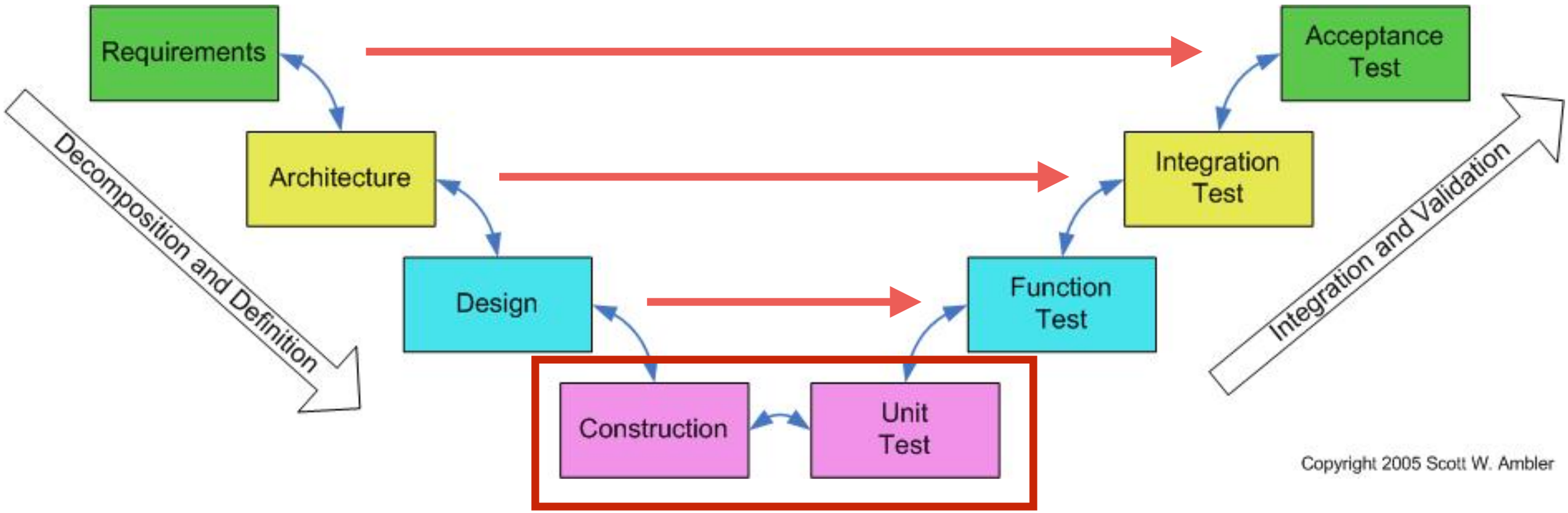






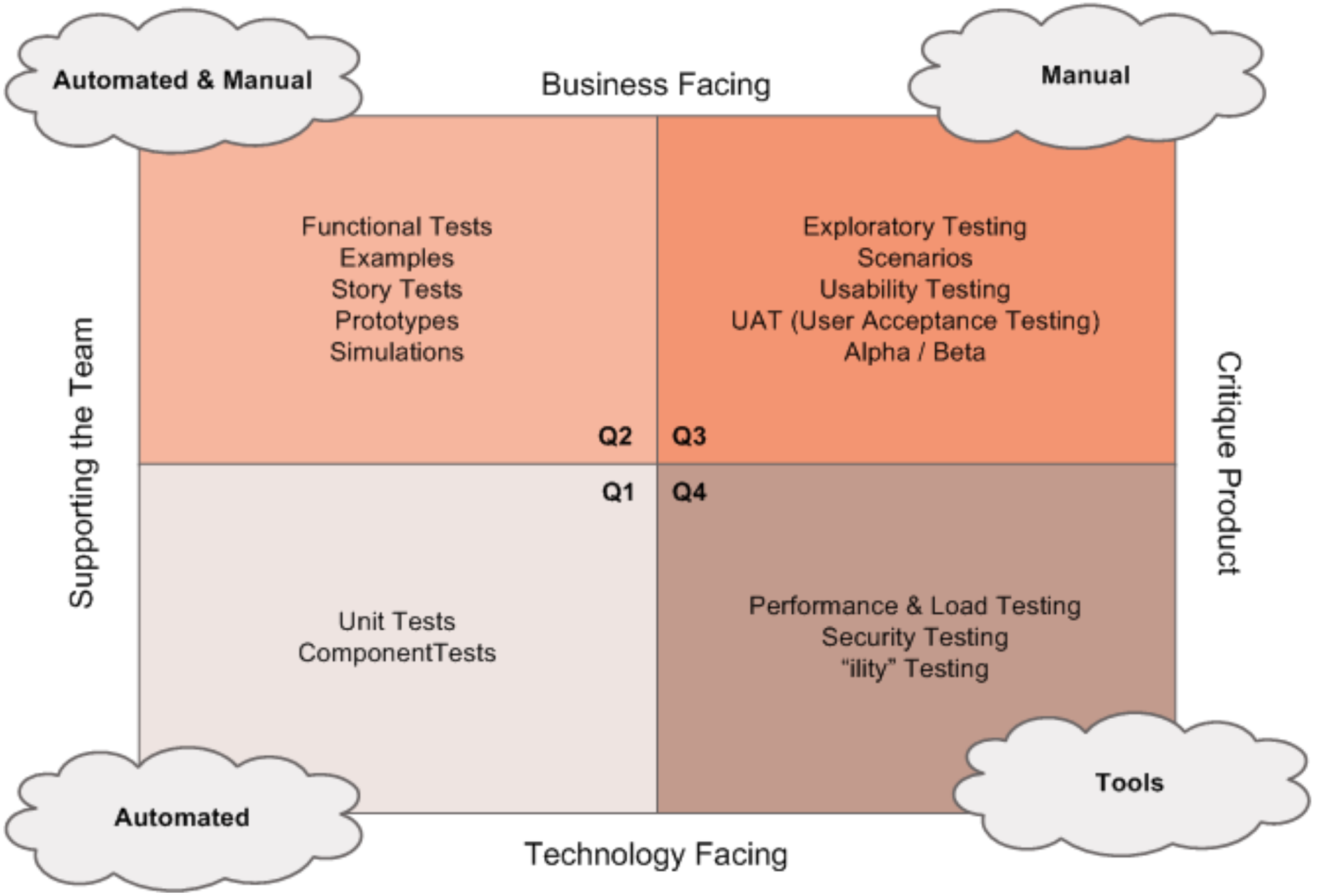






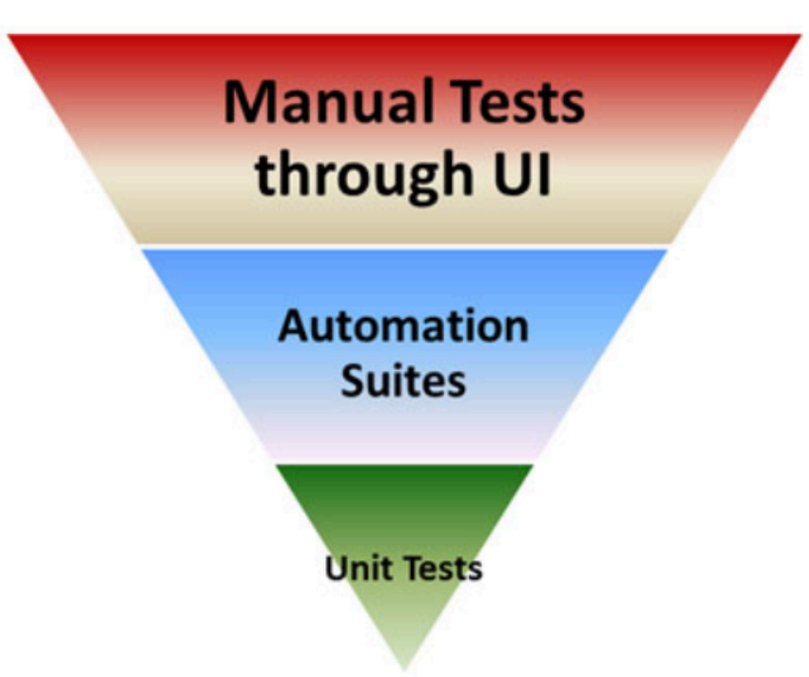
Type of testing

Agile Testing Quadrants

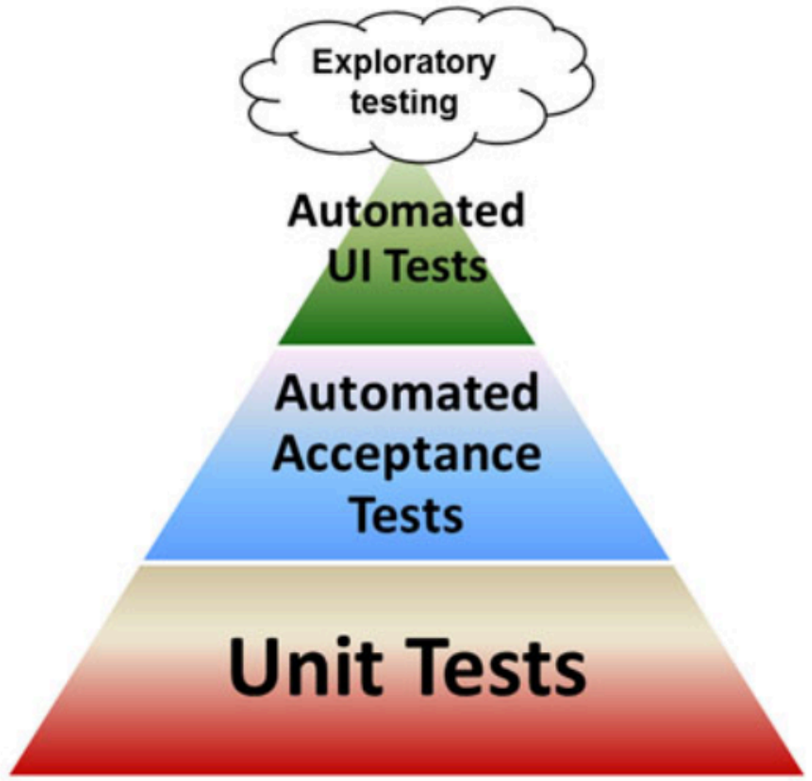


Testing pyramid

Testing Pyramid

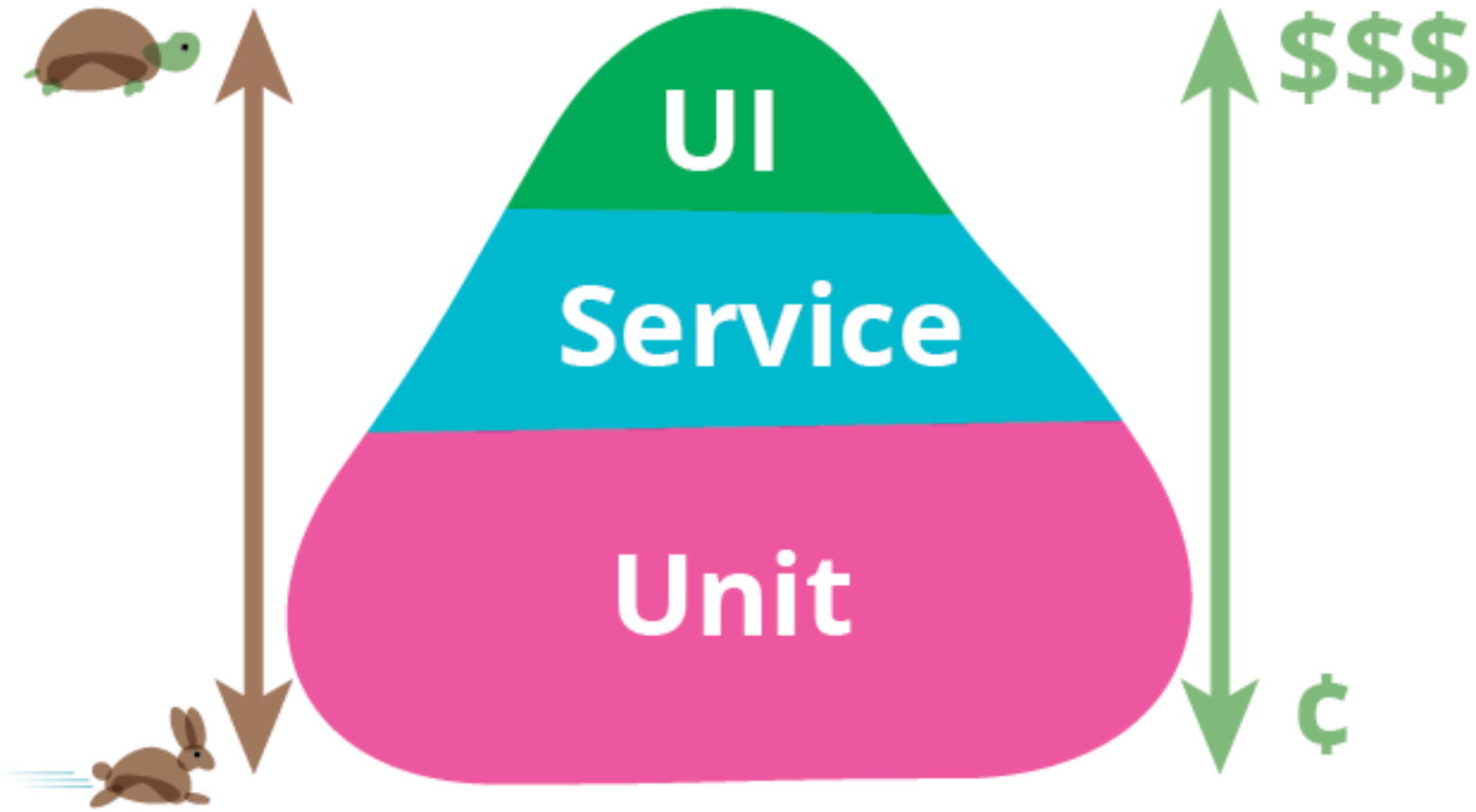


Traditional
(find bugs)



Agile
(prevent bugs)

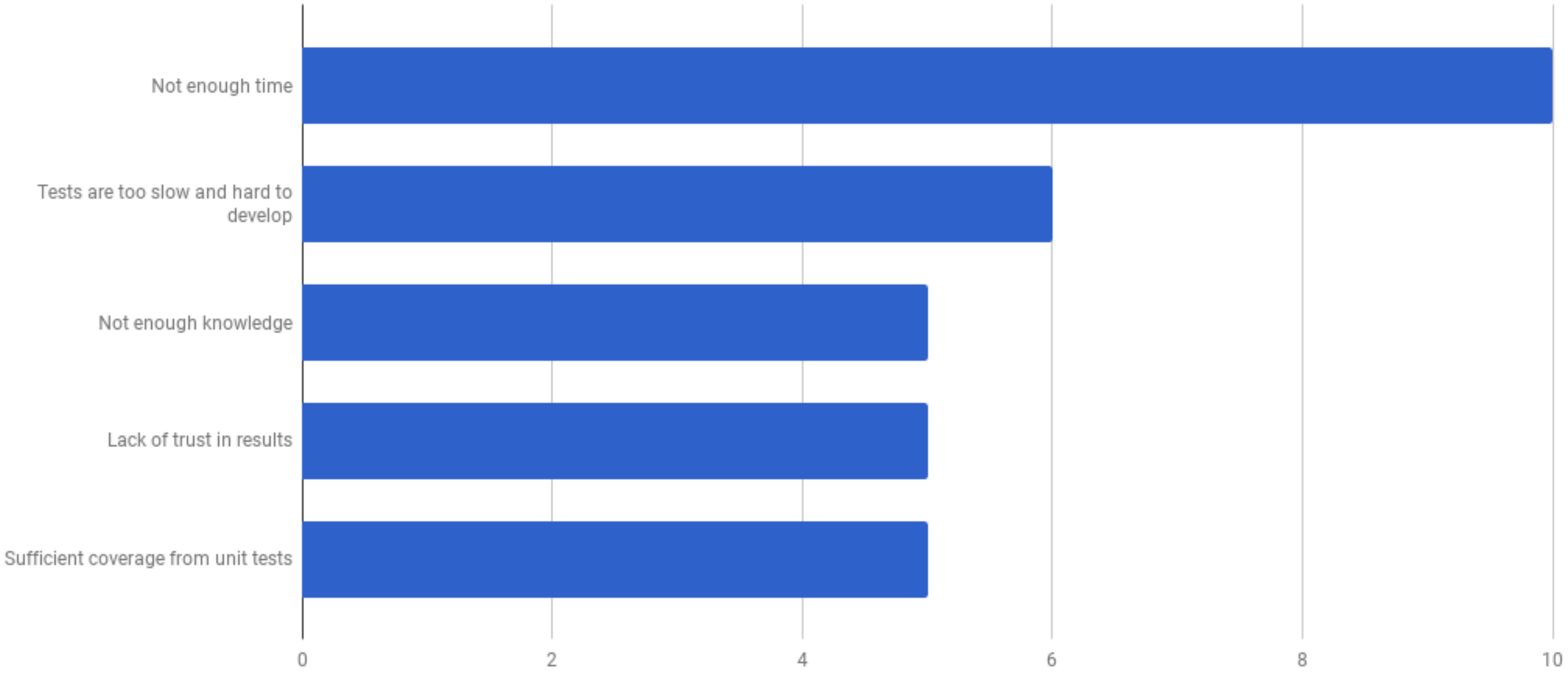
Testing Pyramid



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

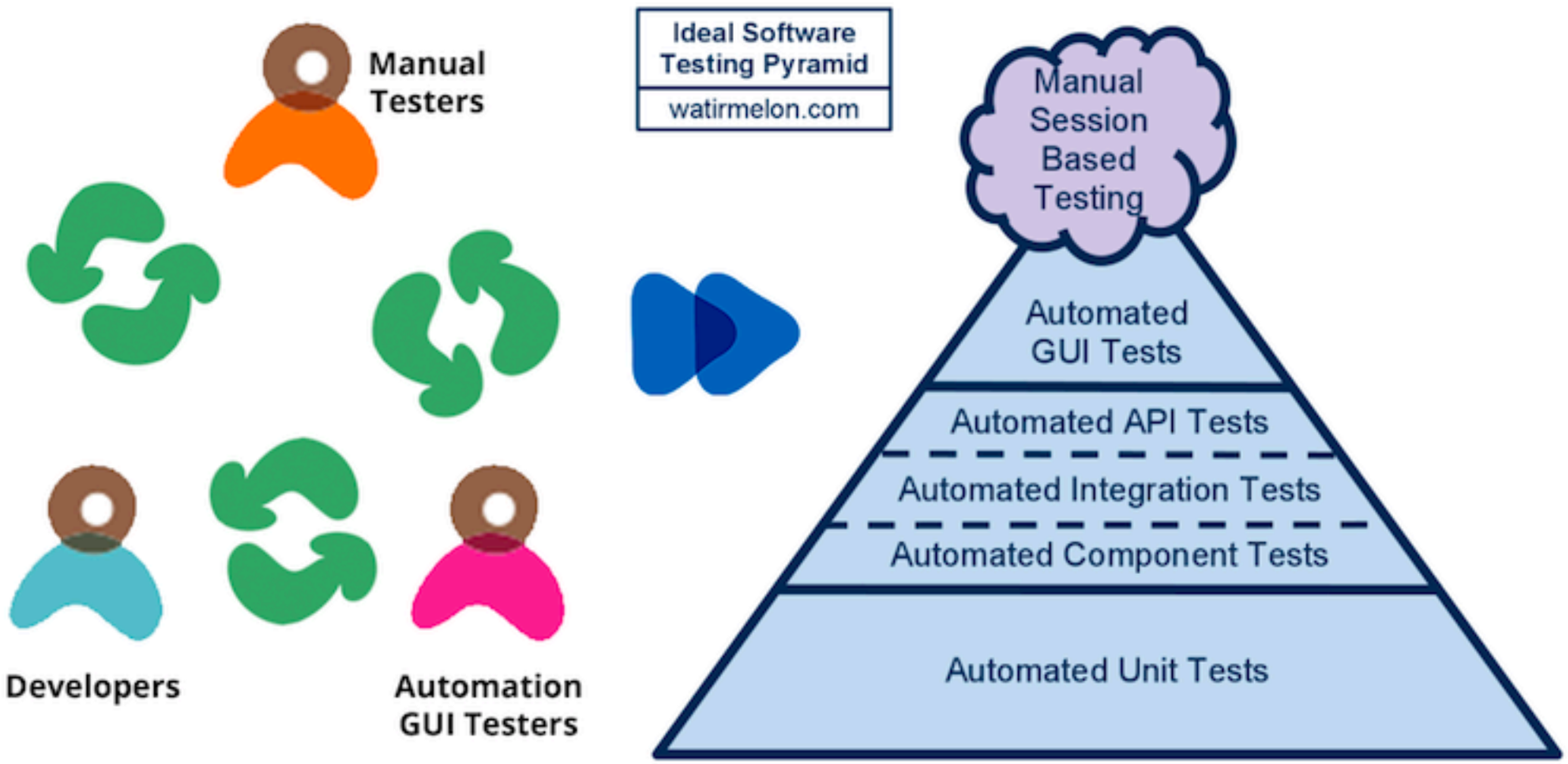
Testing not a phase
Testing is activity

Why not write automated tests ?

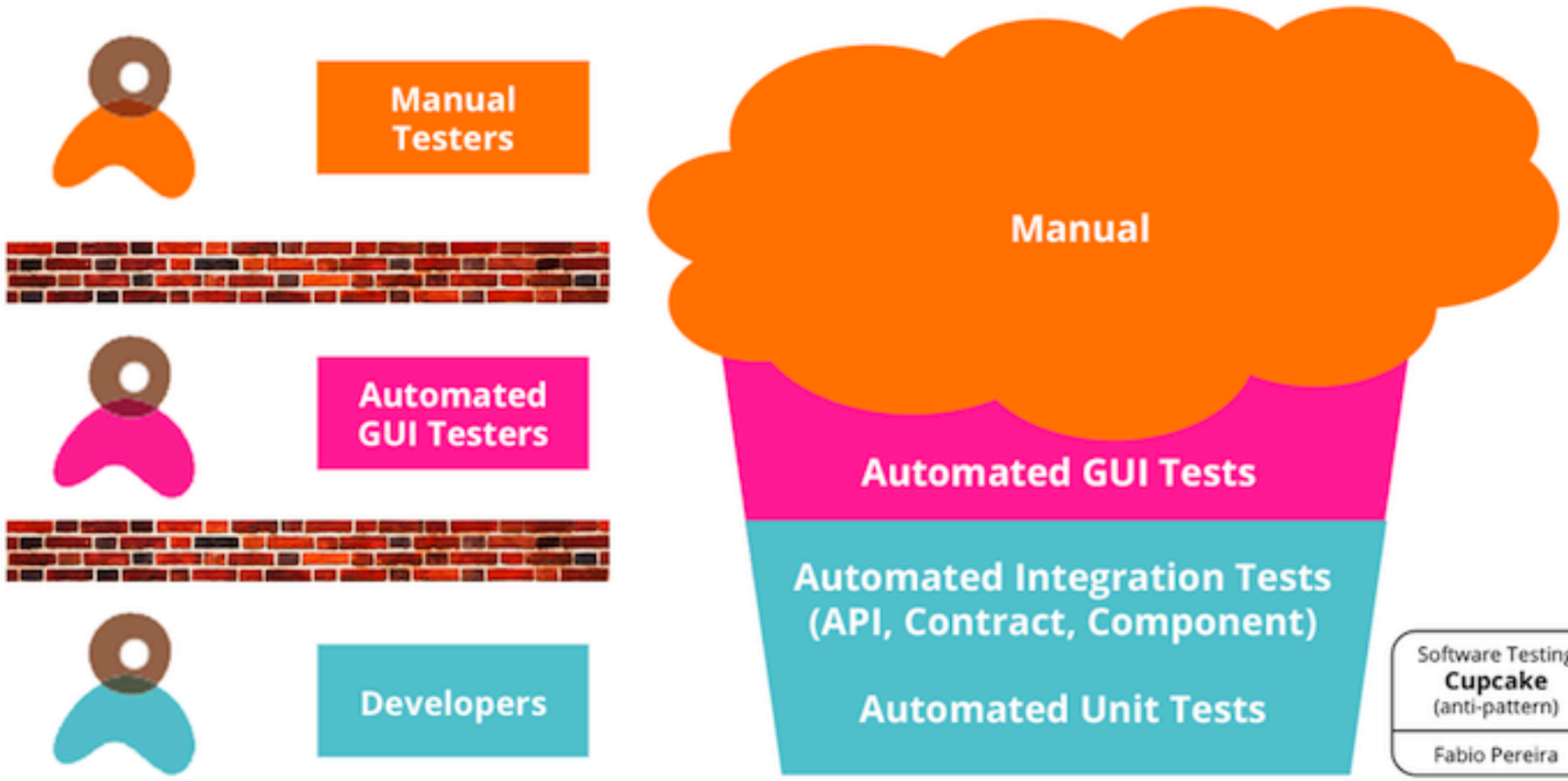


https://slack.engineering/android-ui-automation-part-1-building-trust-de3deb1c5995

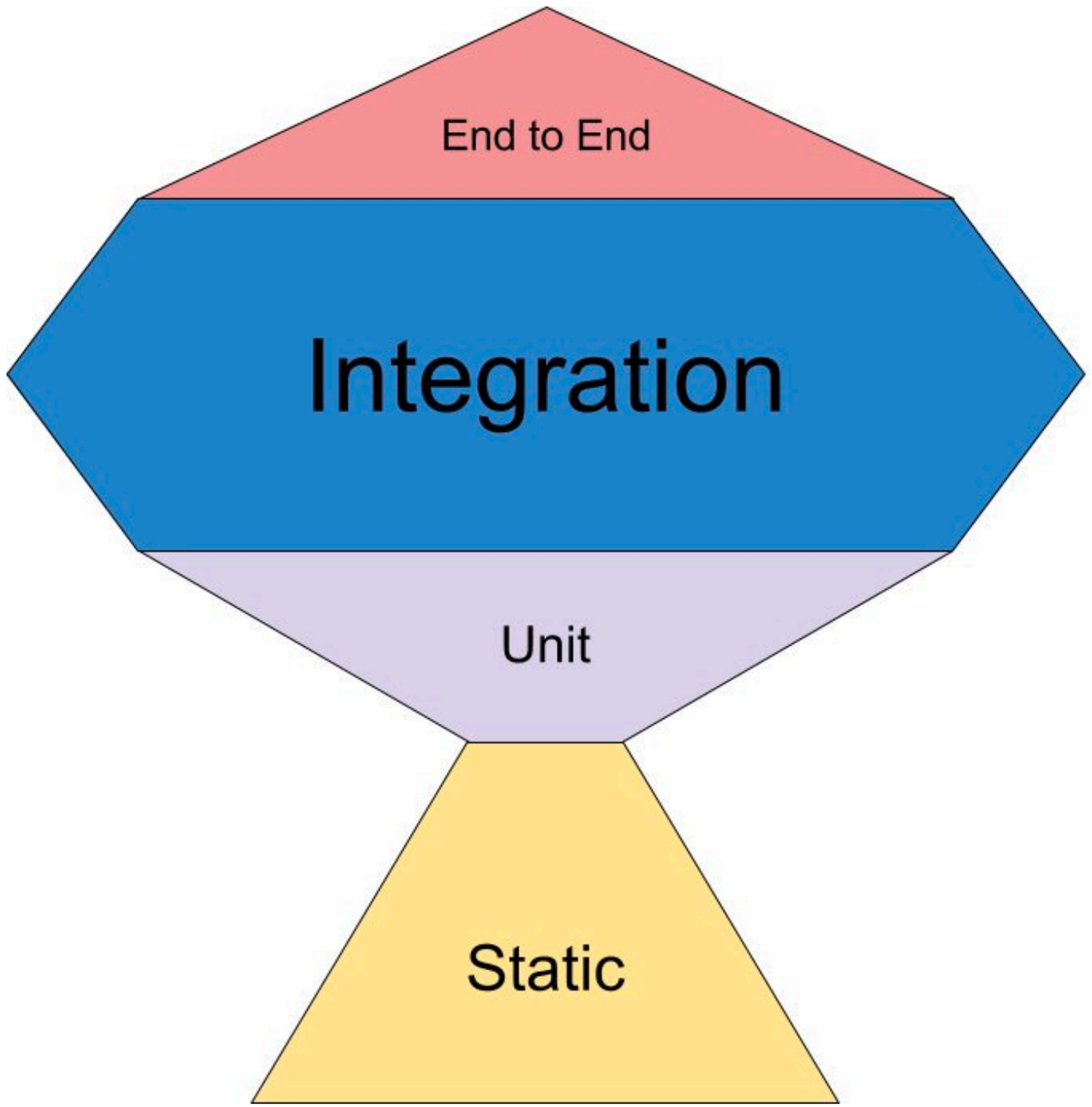
Pyramid testing



Cupcake testing



Trophy testing



Should we write our tests

Before

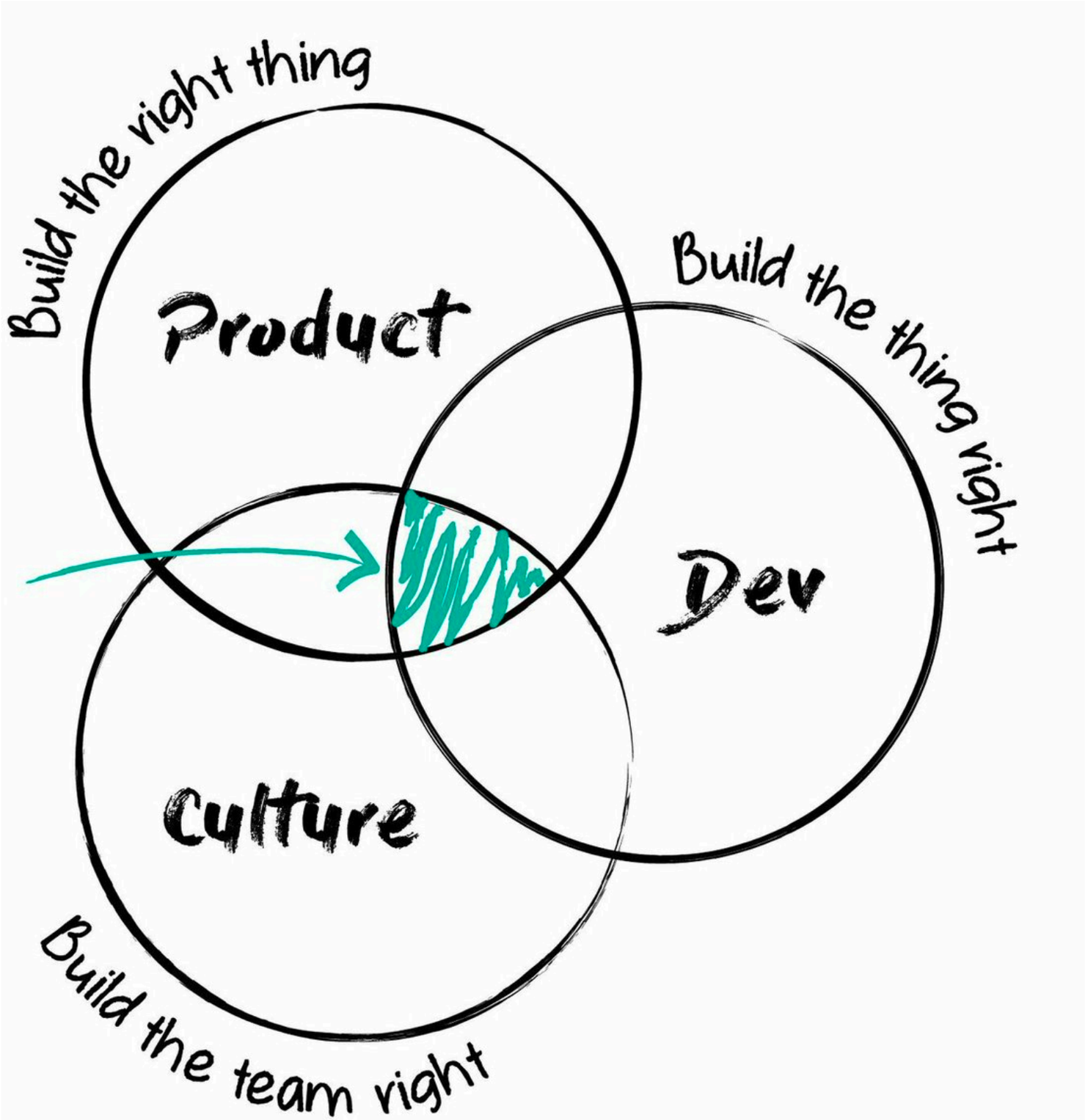
or

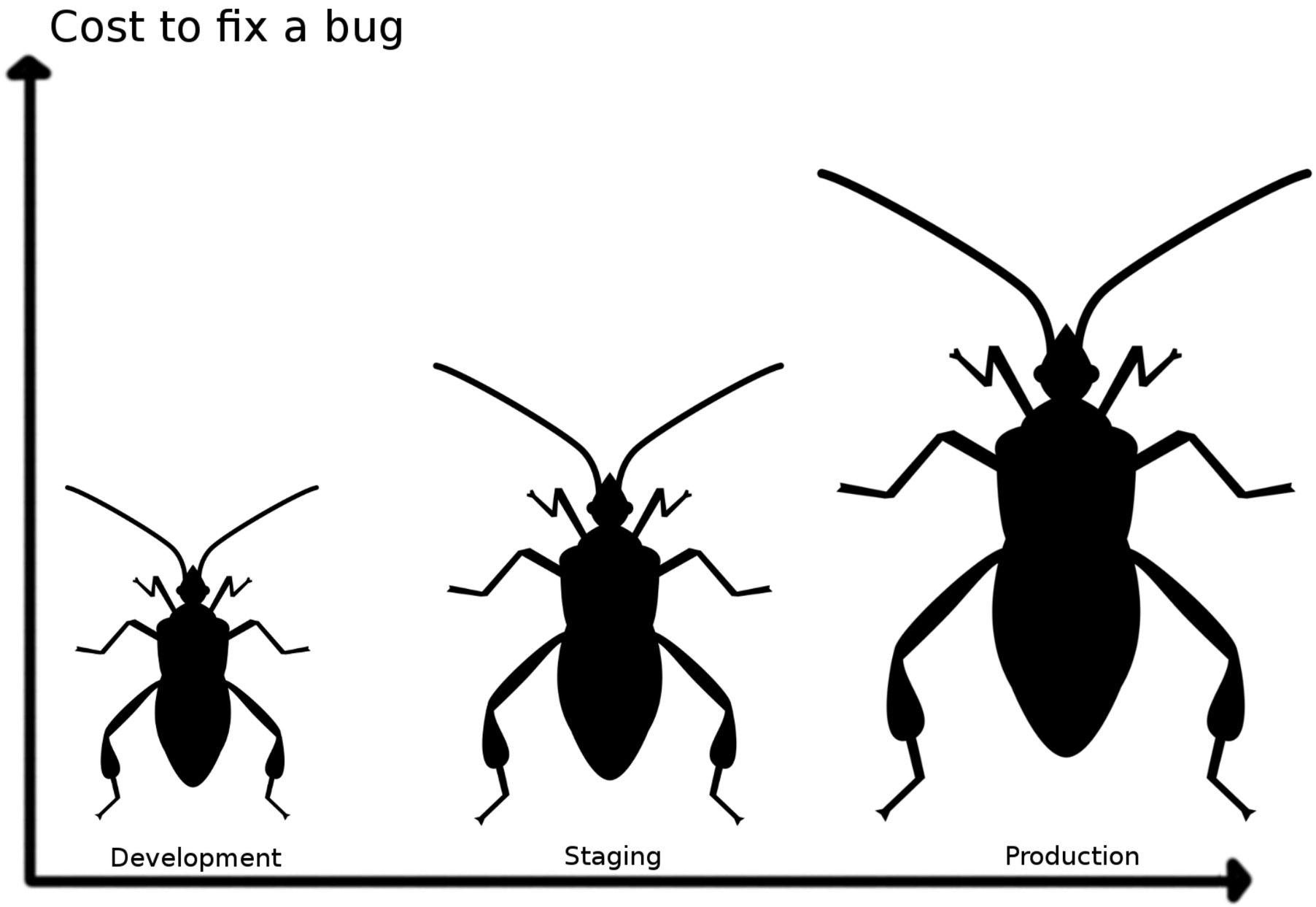
After

implementation ?

After is bad ?

Laziness
Lack of discipline
Compromise





🐛 Fixing Bugs 🐜

Bug

Find Code

Write Test

Fix Test

SPECIFICATION BY EXAMPLE

CONTINUOUS INTEGRATION

CONTINUOUS DELIVERY

TEST AUTOMATION

TECHNICAL EXCELLENCE

ARCHITECTURE & DESIGN

ACCEPTANCE TESTING

CODE

CLEAN CODE

THINKING ABOUT TESTING

TEST-DRIVEN DEVELOPMENT

UNIT TESTING

https://less.works/less/technical-excellence/index.html

SPECIFICATION BY EXAMPLE

CONTINUOUS INTEGRATION

CONTINUOUS DELIVERY

TEST AUTOMATION

TECHNICAL EXCELLENCE

ACCEPTANCE TESTING

ARCHITECTURE & DESIGN

CODE

CLEAN CODE

THINKING ABOUT TESTING

TEST-DRIVEN DEVELOPMENT

UNIT TESTING

https://less.works/less/technical-excellence/index.html

SPECIFICATION BY EXAMPLE

CONTINUOUS INTEGRATION

CONTINUOUS DELIVERY

TEST AUTOMATION

TECHNICAL EXCELLENCE

ARCHITECTURE & DESIGN

ACCEPTANCE TESTING

CODE

CLEAN CODE

THINKING ABOUT TESTING

TEST-DRIVEN DEVELOPMENT

UNIT TESTING

https://less.works/less/technical-excellence/index.html

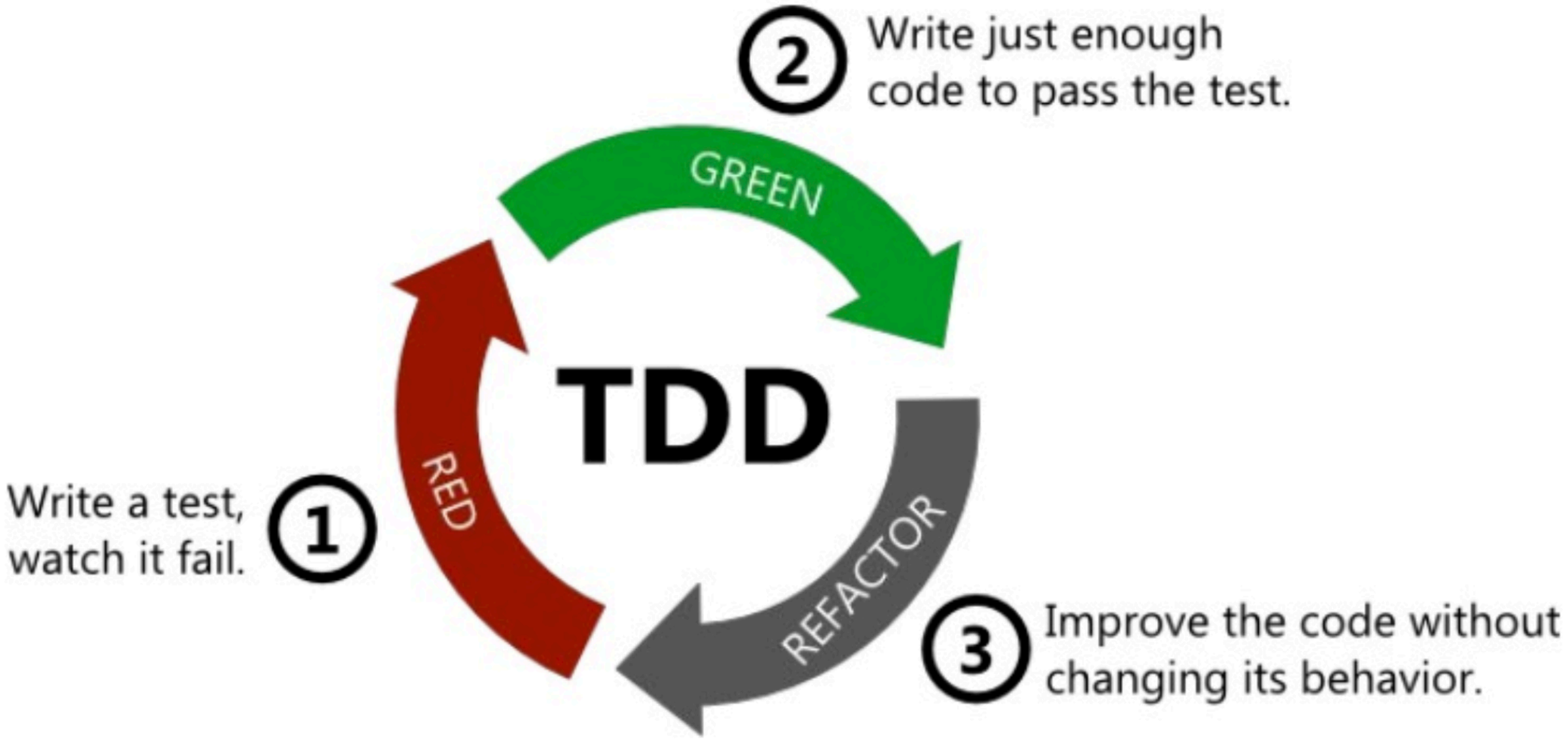
TDD

ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT

TDD

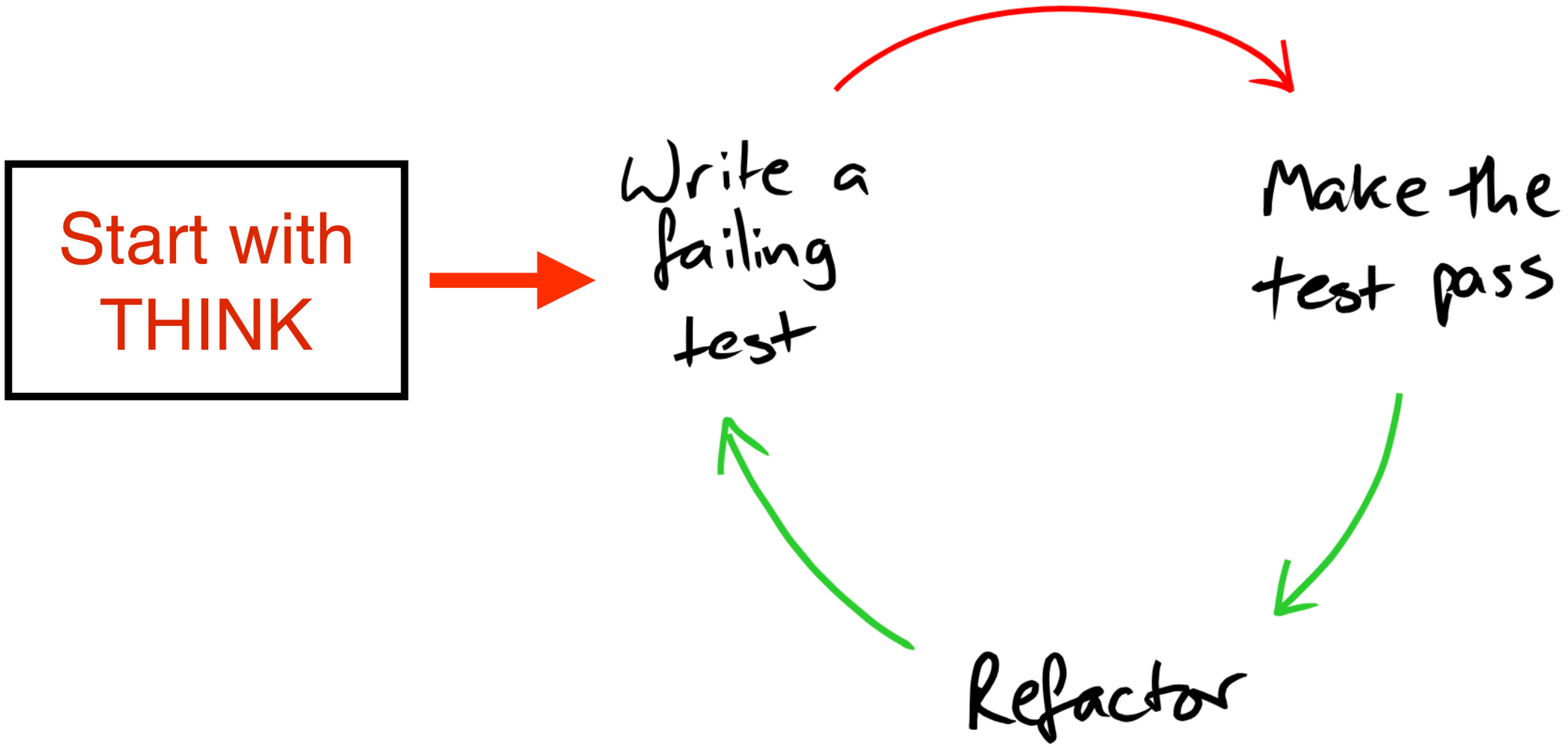
Test-Driven-Development
Test-Driven Design

TDD Cycle

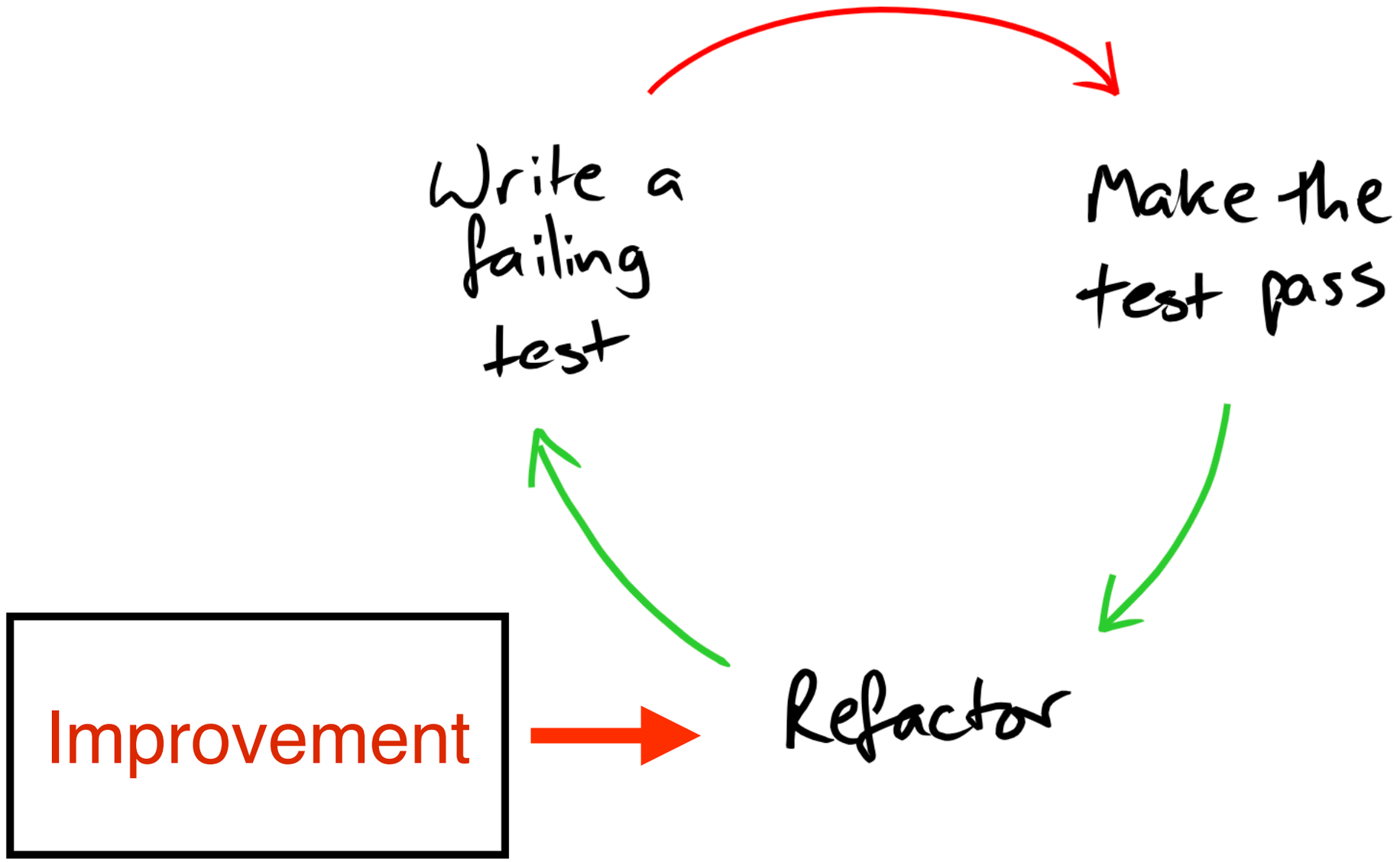


Red Green Refactor

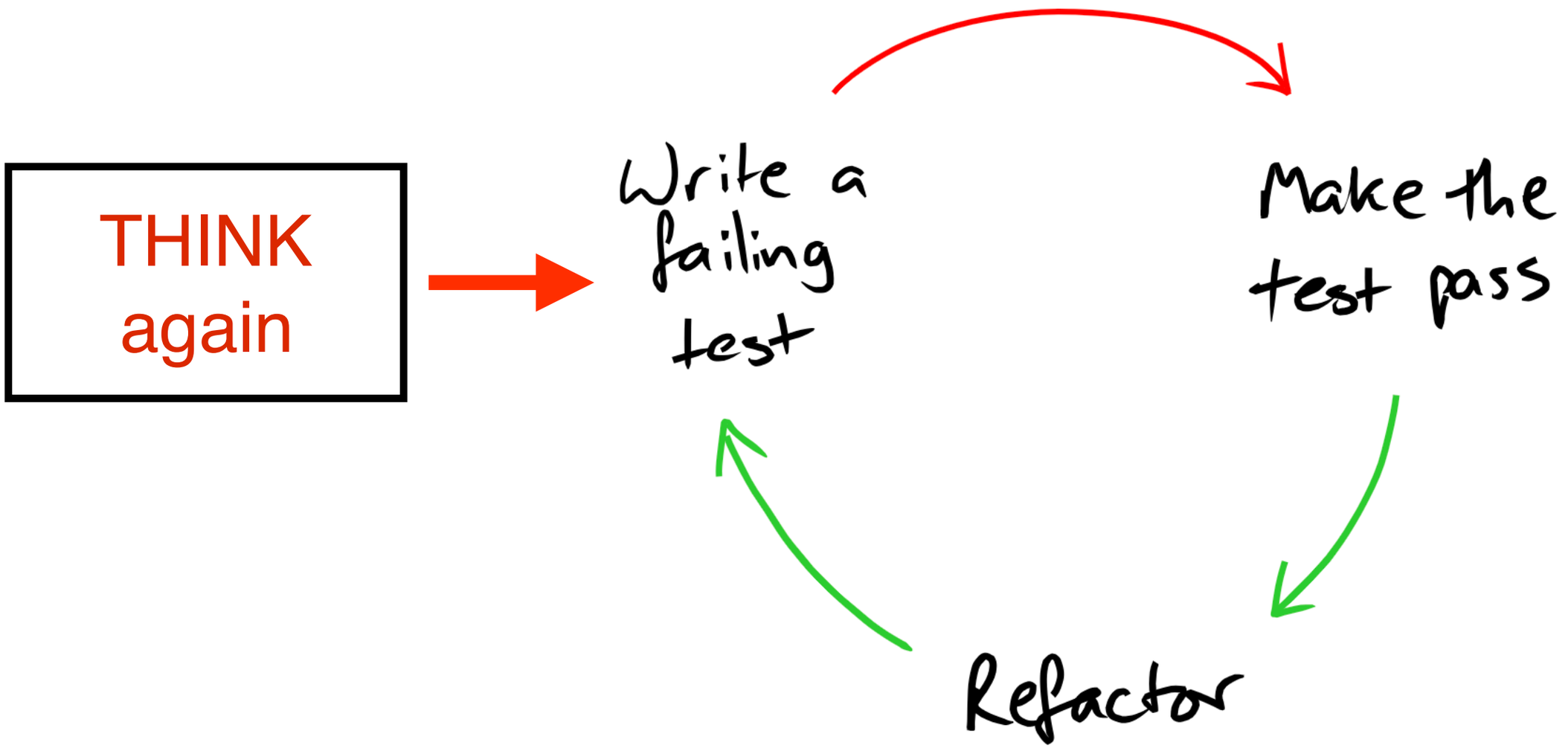
Improve TDD Cycle



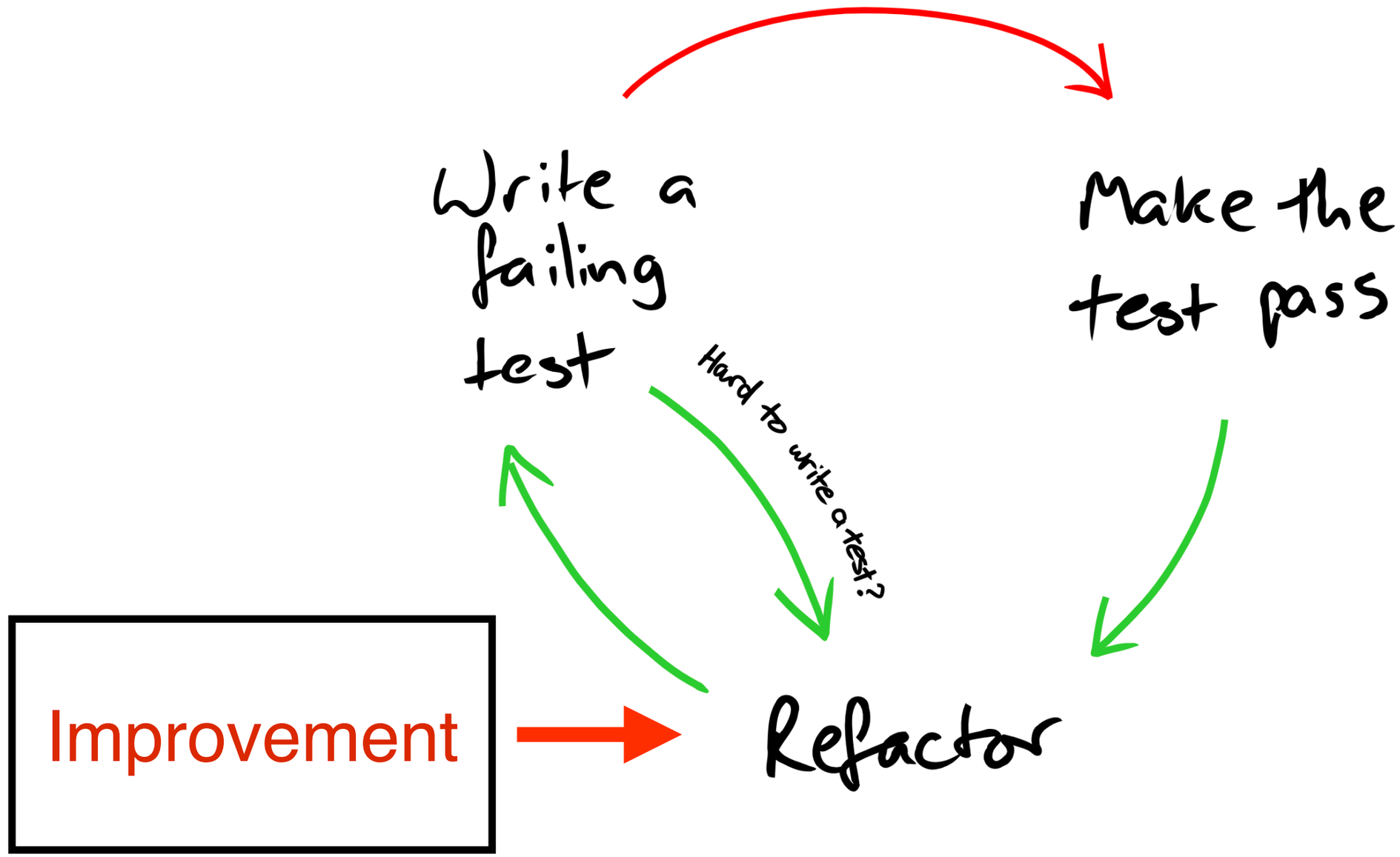
Improve TDD Cycle



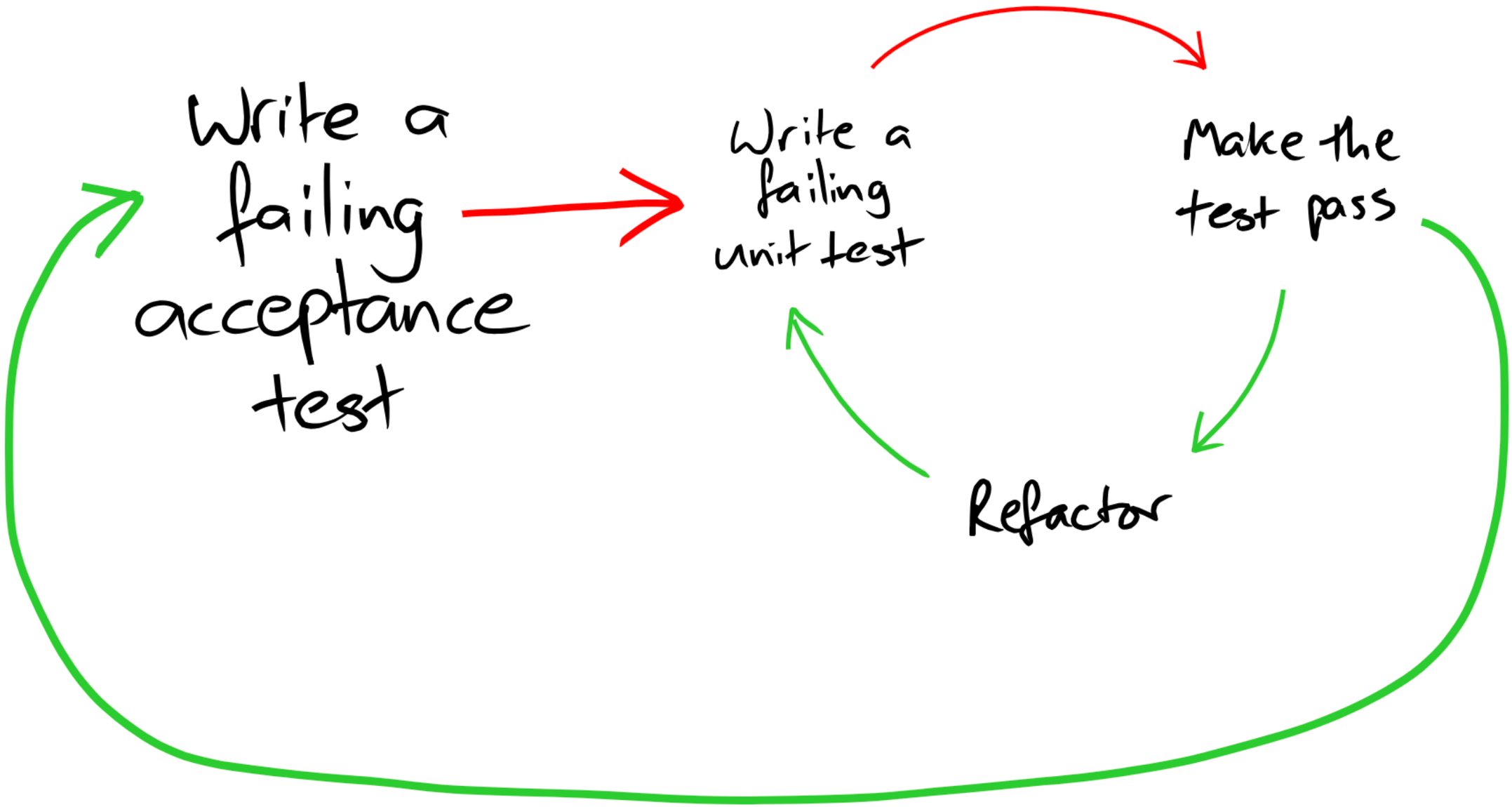
Write test again



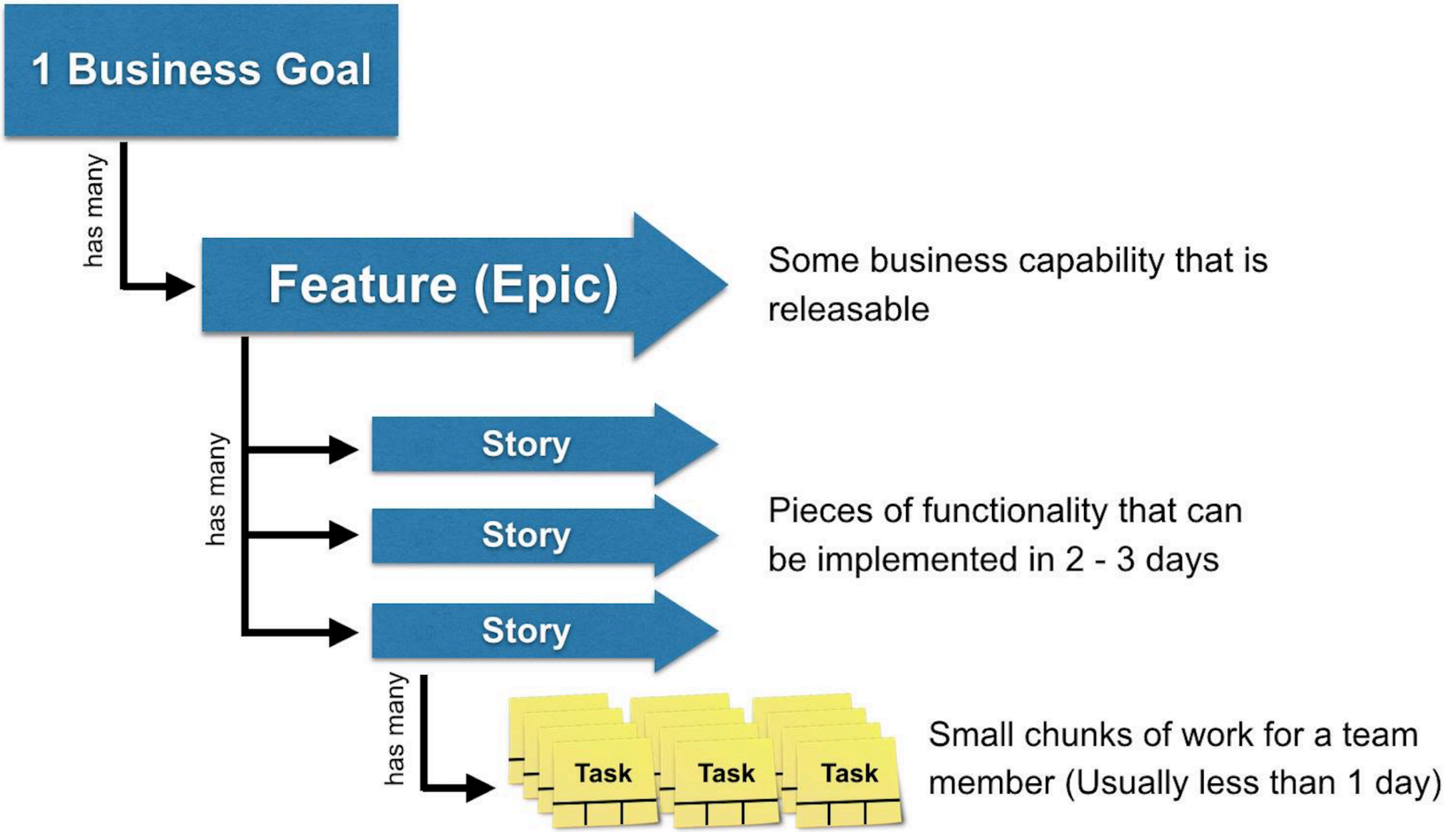
When hard to write test ?



Acceptance tests



Start with Business Goal to Tasks



Source: The Whole Team Approach to Agile Testing by Janet Gregory and Lisa Crispin

The Golden rule of TDD

"Never write a line of code without a failing test"

- Kent Beck -

Discipline #1

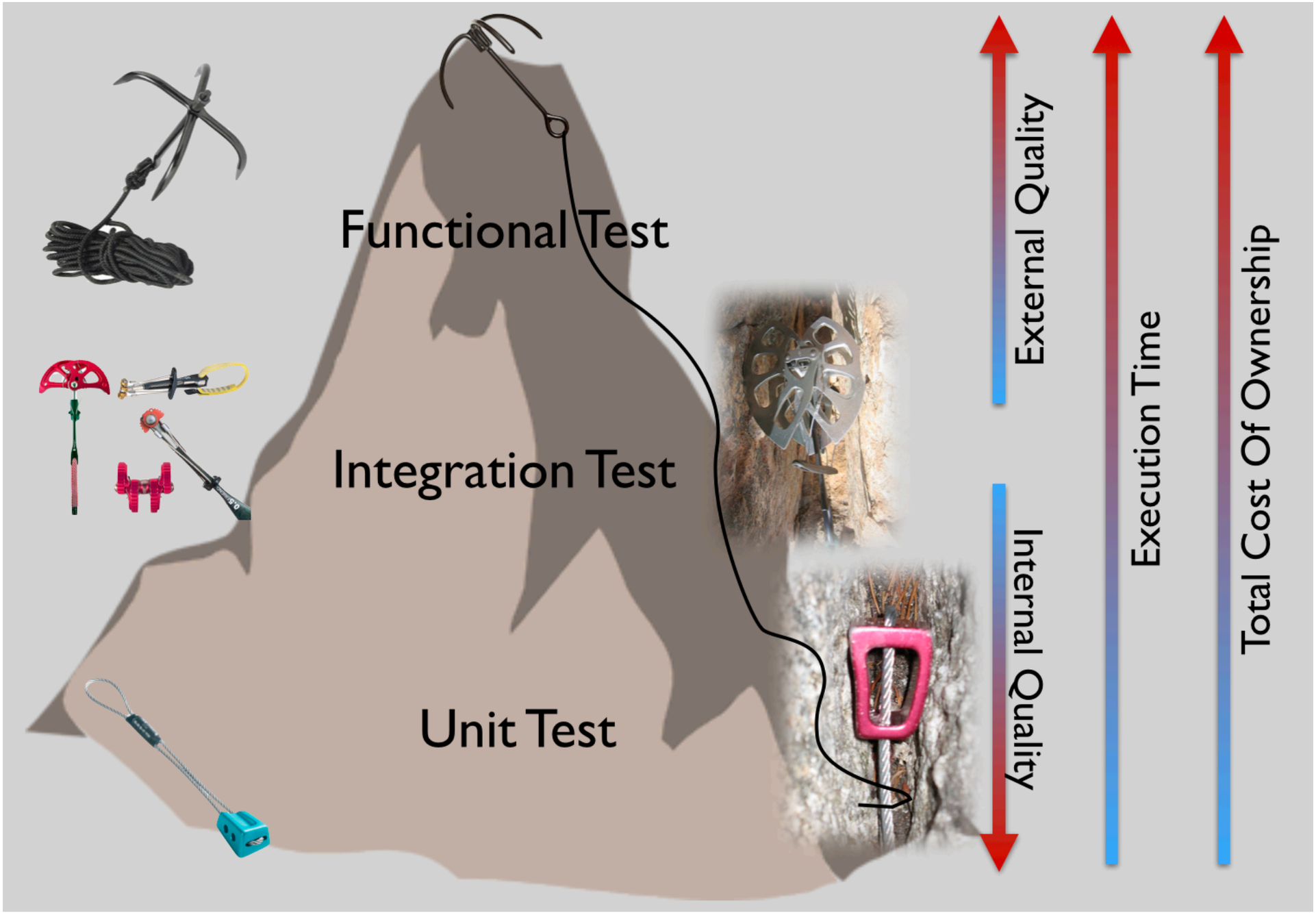
You are not allowed to write production code until you have first written a failing unit test.

Discipline #2

You are not allowed to write more of a unit test than is sufficient to fail, and not compiling is failing.

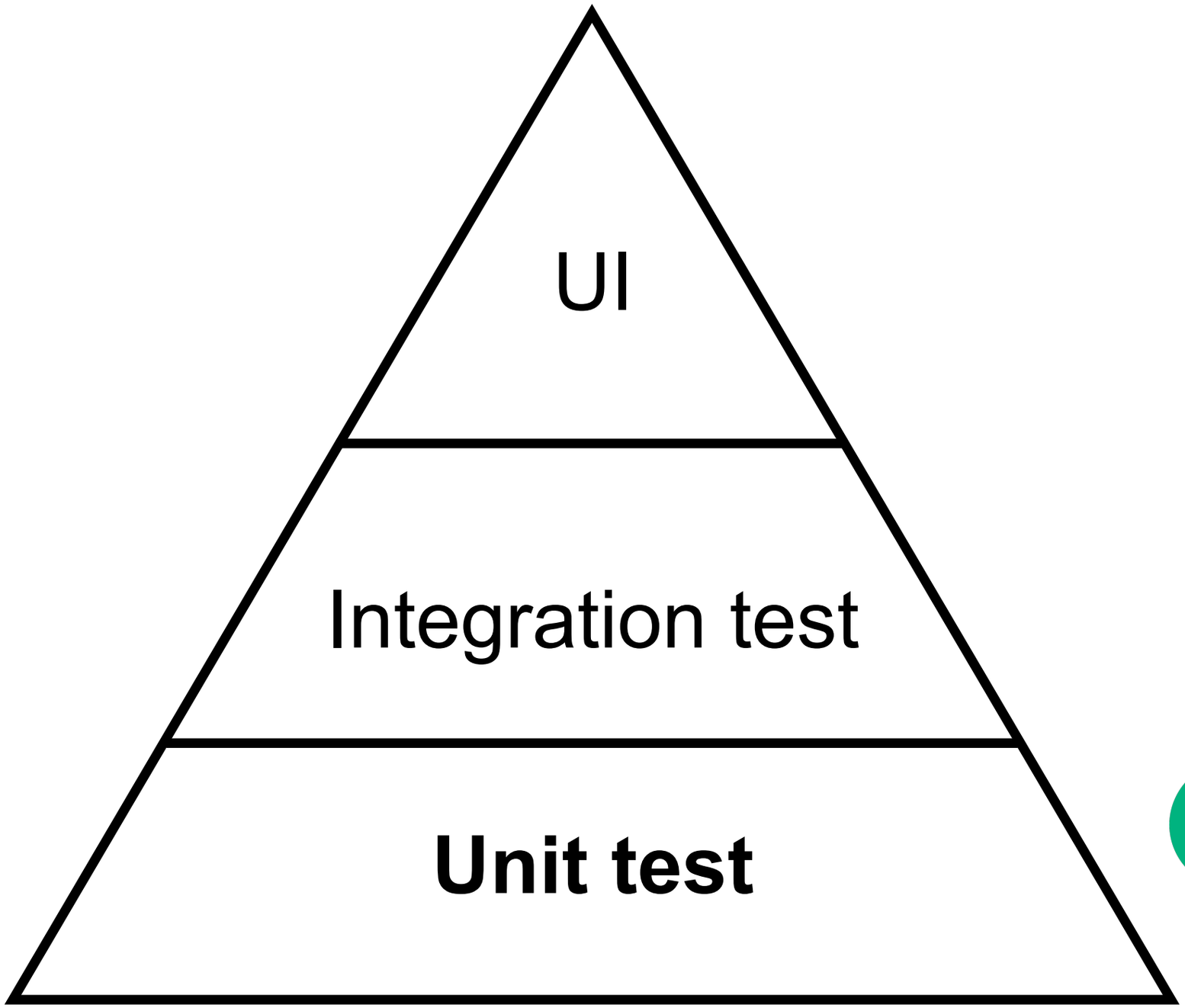
Discipline #3

You are not allowed to write more production code than is sufficient to pass the currently failing test.



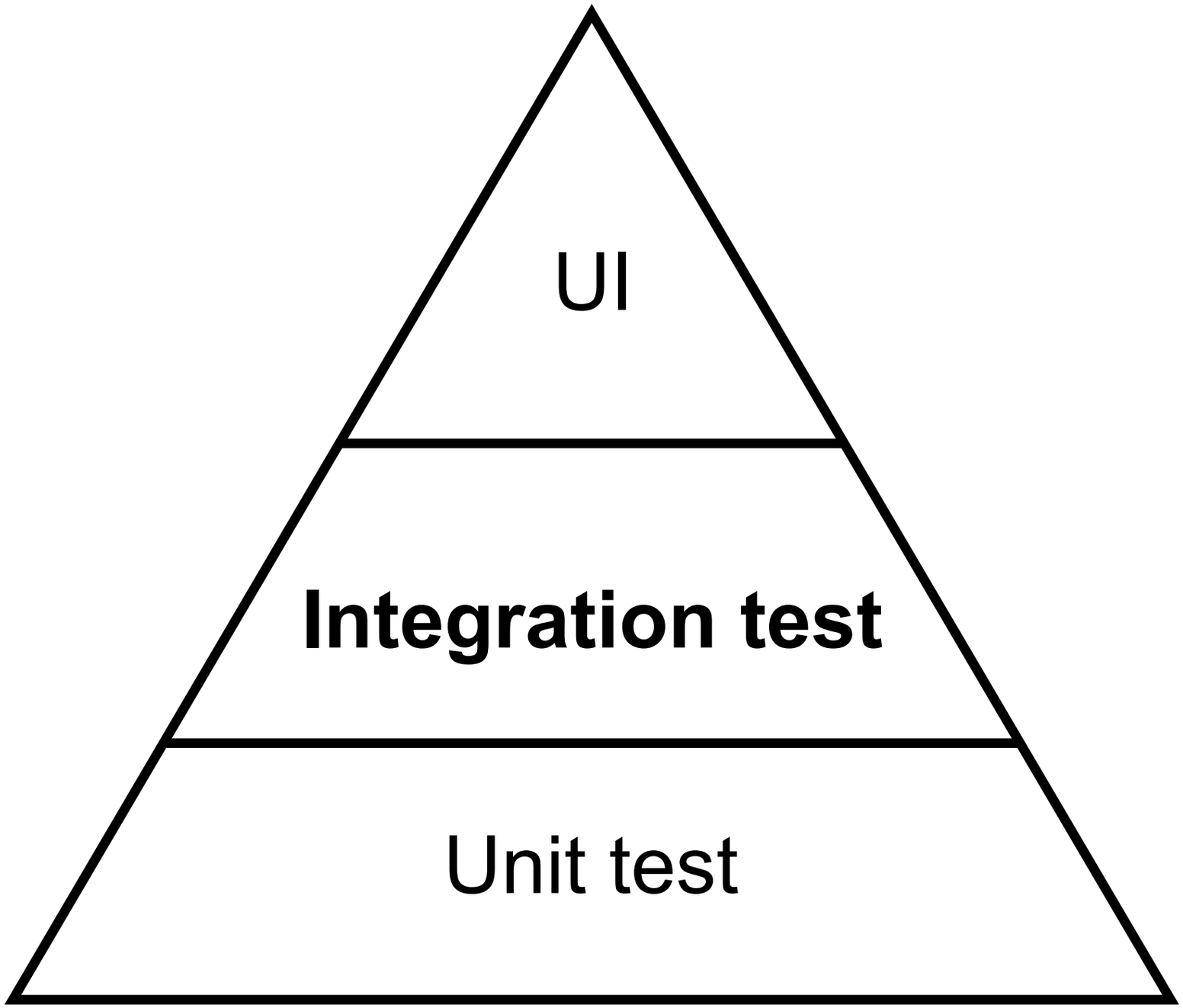
https://less.works/jp/less/technical-excellence/unit-testing.html

Testing in iOS app ?

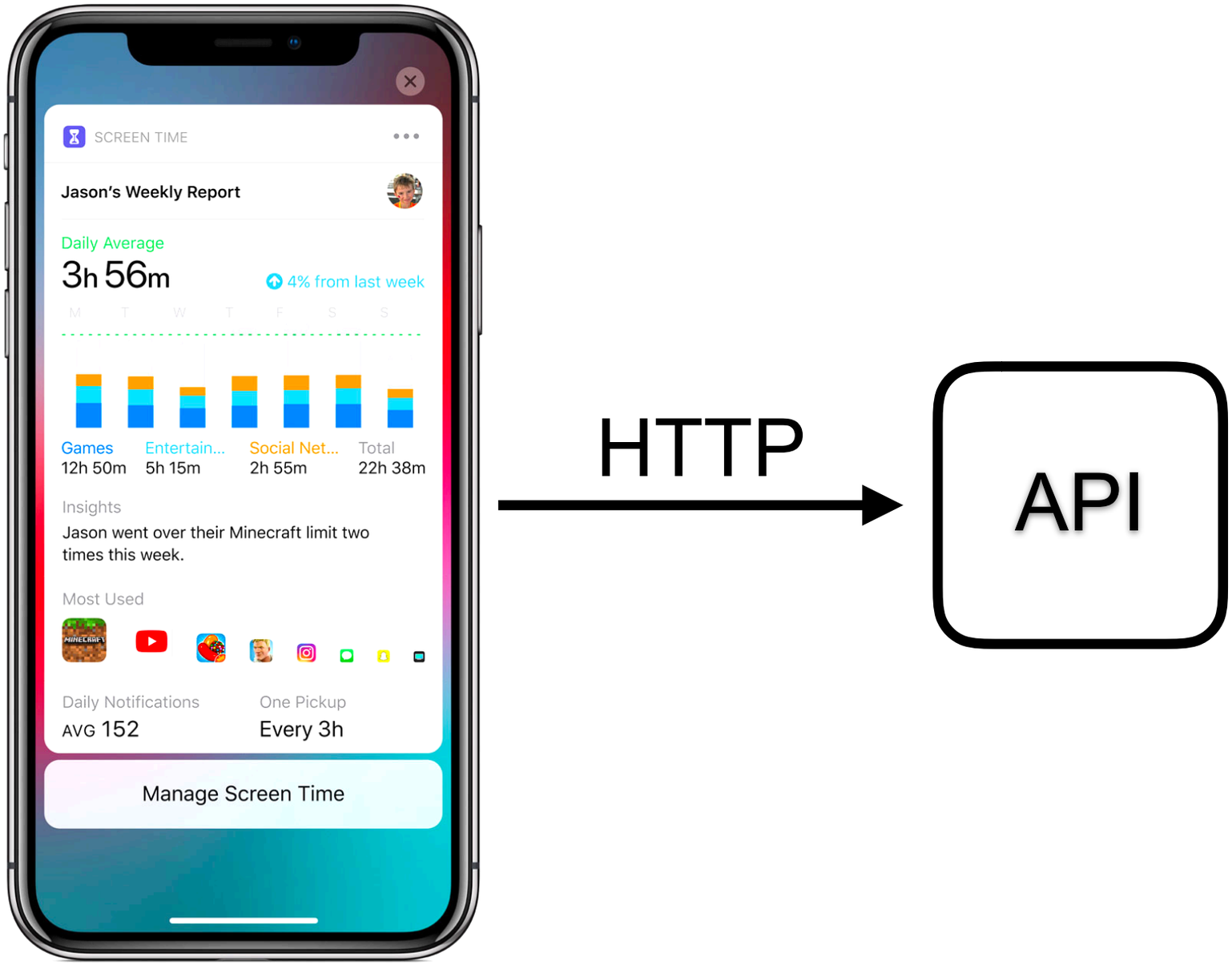


Nimble

Unit XCTest



Integration test ?

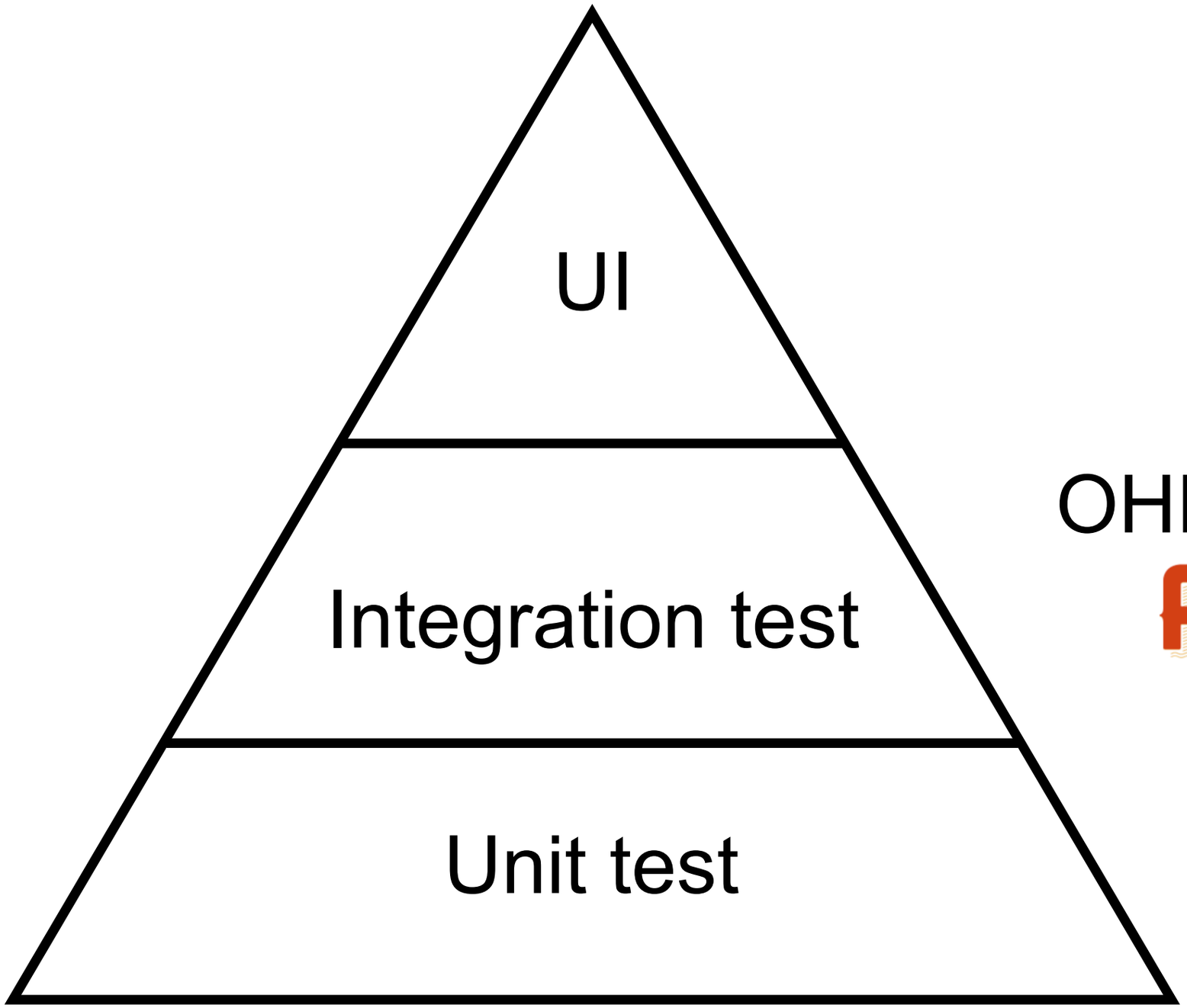


Integration test ?

Network request
UserDefaults
Core Data
Singleton objects

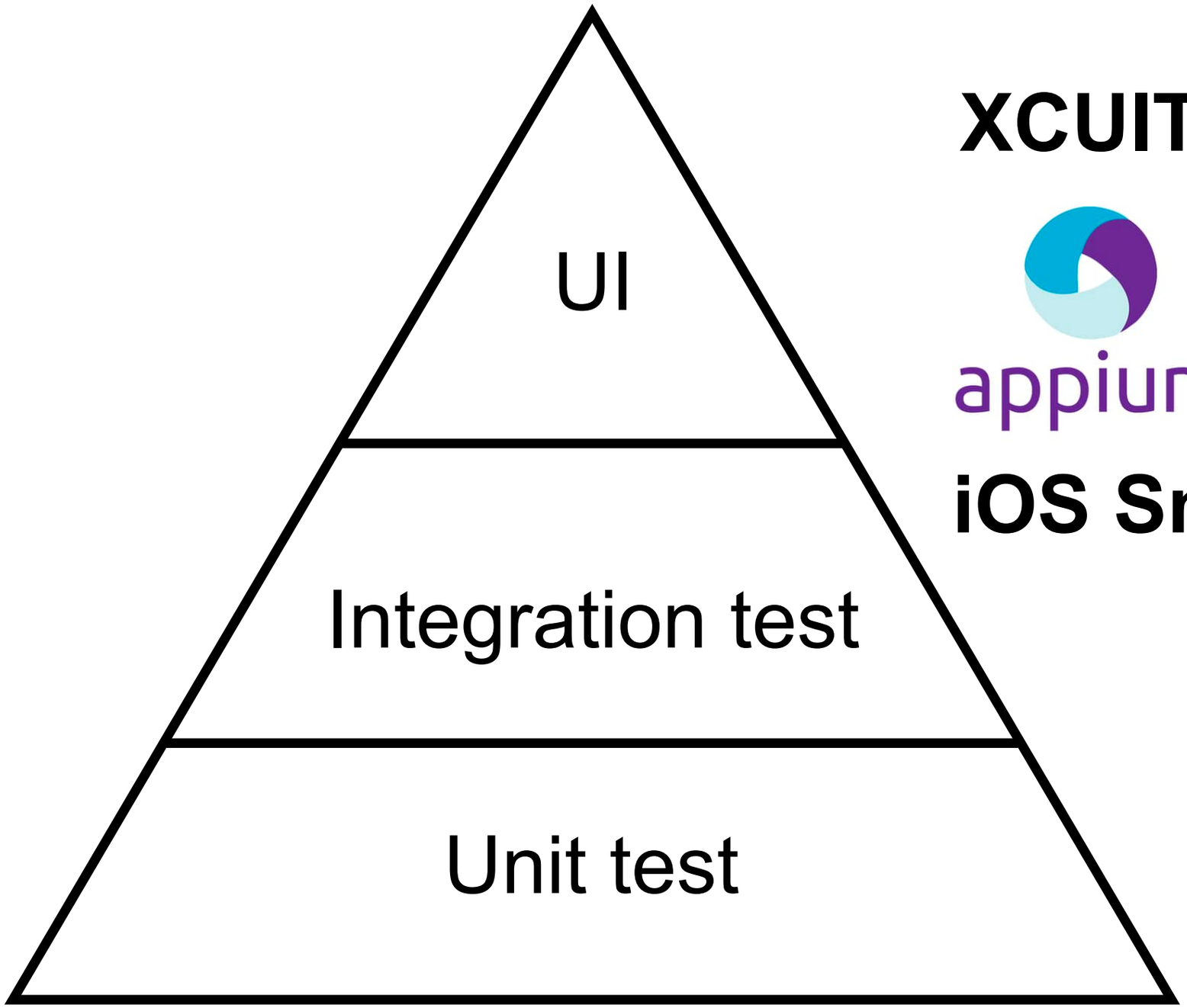
Singleton object ?

NSFileManager
NSApplication
UIApplication
etc …



OHHTTPStubs with

ALAMOFIRE



XCUITest

appium

KIF

Calaba.sh

iOS Snapshot test

Swift monkey test

All tests need device/ simulator !!

Start with Unit testing

Make the test pass !!

Fake or hard code
Triangulation
Professional code

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

Fast

Independent/Isolate

Repeat

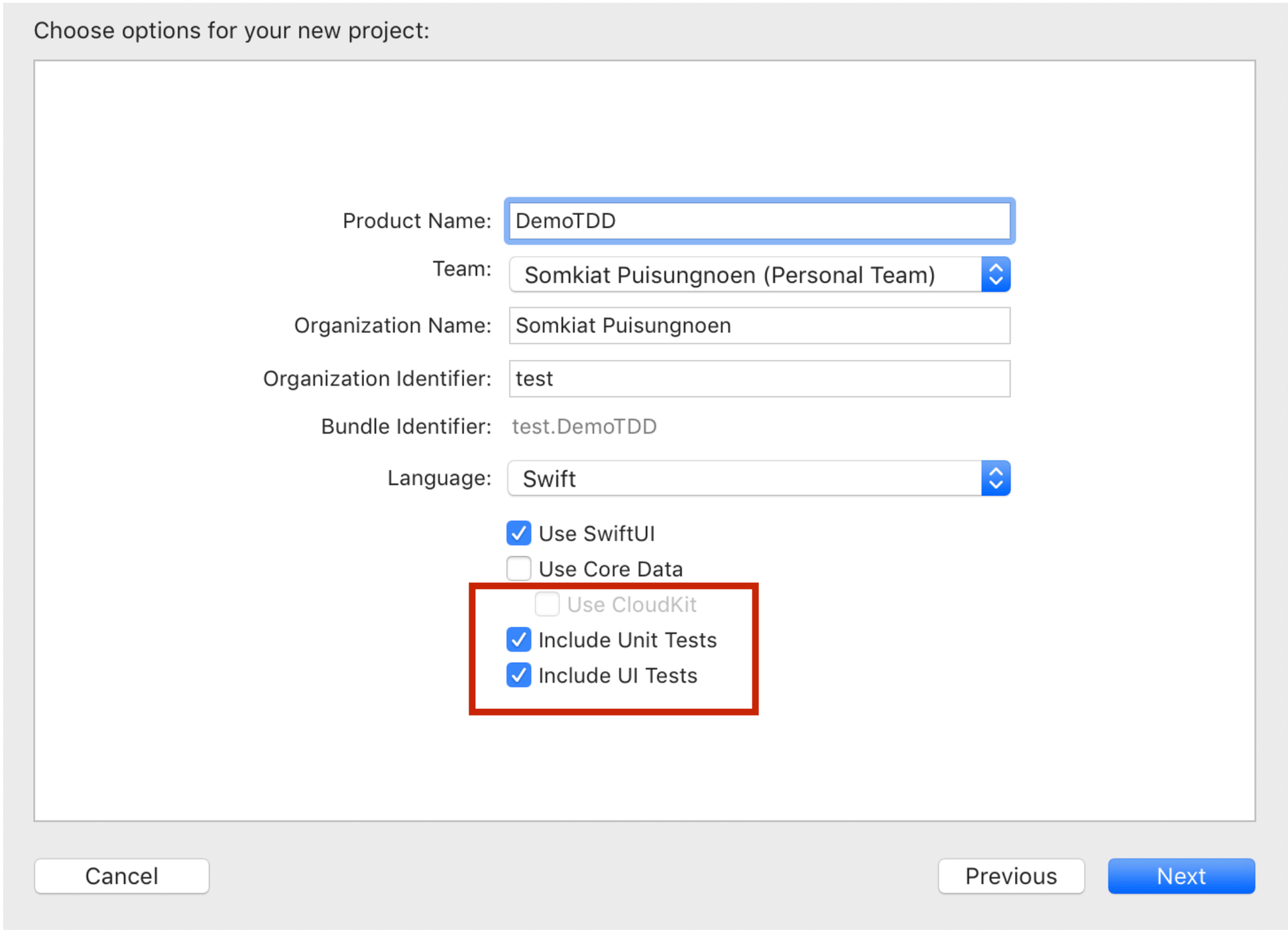
Self-validate

Thorough/Timely

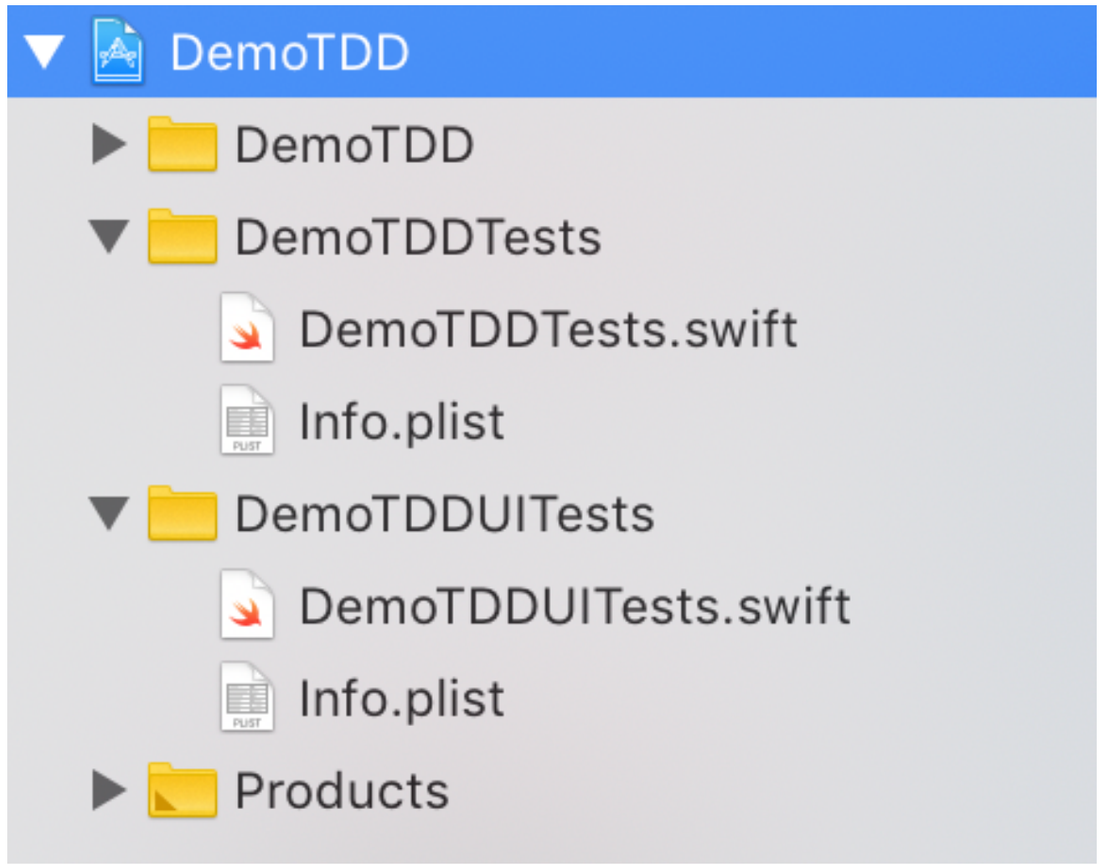
Not Unit test if …

Talk to database
Communicate across the network
Touch a file system
Can’t run the same time as any of other tests
Do special things to your environment to run it

Create test in Xcode



Provide Unit and UI tests



XCTest

Testing framework provided by Xcode
Allows test to be run directly from source code
Support both Swift and Objective C

https://developer.apple.com/documentation/xctest

XCTest provides …

Create test case subclasses (XCTestCase)
Create test methods (start with test)
Assertions (XCTAssert)

Unit testing with XCTest

```swift
import XCTest
@testable import DemoTDD

class DemoTDDTests: XCTestCase {

    override func setUp() {
        // Put setup code here. This method is called before the invocation of
            each test method in the class.
    }

    override func tearDown() {
        // Put teardown code here. This method is called after the invocation
            of each test method in the class.
    }

    func testExample() {
        // This is an example of a functional test case.
        // Use XCTAssert and related functions to verify your tests produce
            the correct results.
    }

    func testPerformanceExample() {
        // This is an example of a performance test case.
        self.measure {
            // Put the code you want to measure the time of here.
        }
    }

}
```

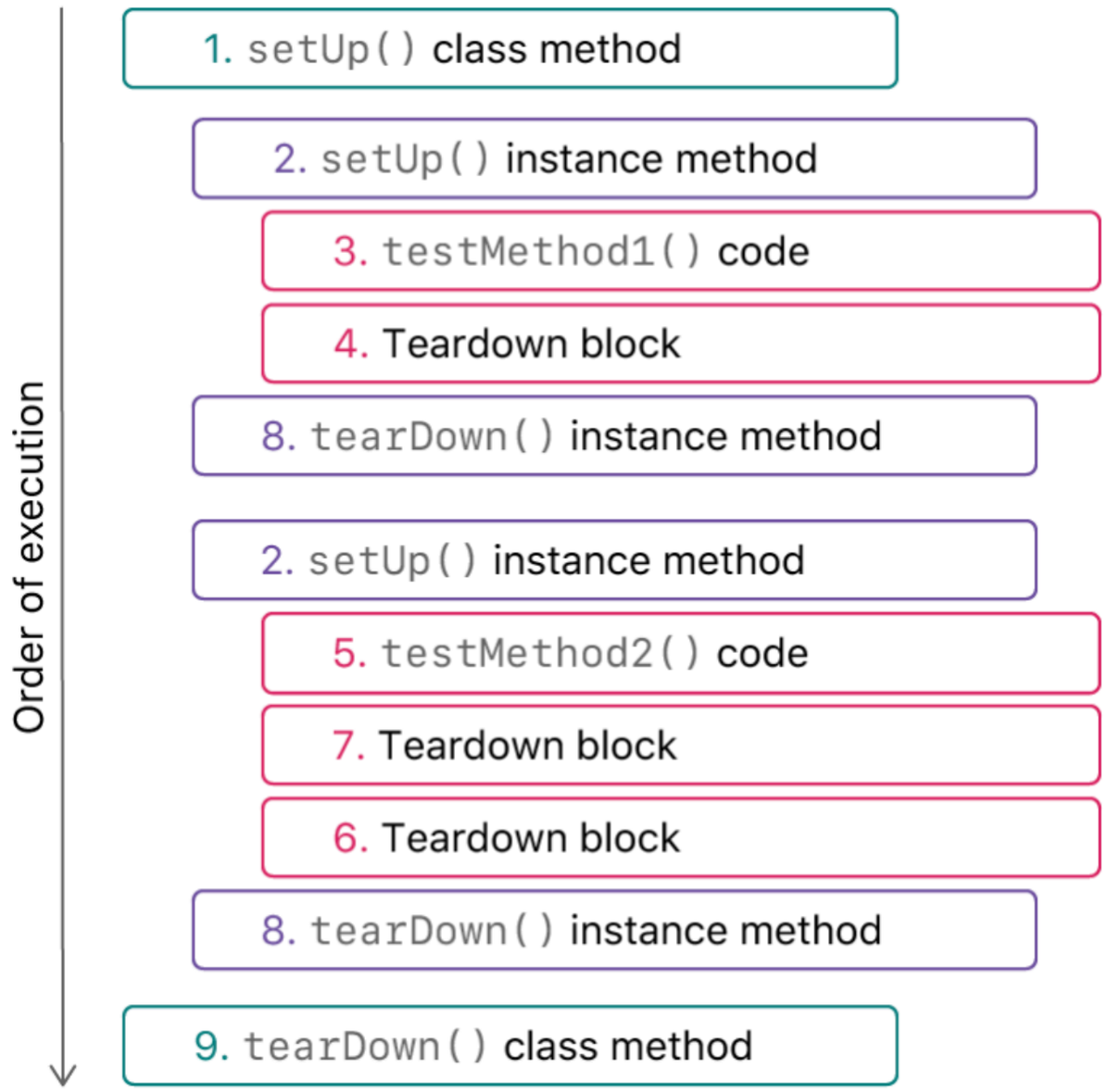
1 Import XCTest

Unit testing with XCTest

2 Tests life cycles

```
import XCTest
@testable import DemoTDD

class DemoTDDTests: XCTestCase {

    override func setUp() {
        // Put setup code here. This method is called before the invocation of
            each test method in the class.
    }

    override func tearDown() {
        // Put teardown code here. This method is called after the invocation
            of each test method in the class.
    }

    func testExample() {
        // This is an example of a functional test case.
        // Use XCTAssert and related functions to verify your tests produce
            the correct results.
    }

    func testPerformanceExample() {
        // This is an example of a performance test case.
        self.measure {
            // Put the code you want to measure the time of here.
        }
    }

}
```

Test Life Cycle



https://developer.apple.com/documentation/xctest/xctestcase/understanding_setup_and_teardown_for_test_methods

Unit testing with XCTest

```
import XCTest
@testable import DemoTDD

class DemoTDDTests: XCTestCase {

    override func setUp() {
        // Put setup code here. This method is called before the invocation of
            each test method in the class.
    }

    override func tearDown() {
        // Put teardown code here. This method is called after the invocation
            of each test method in the class.
    }

    func testExample() {
        // This is an example of a functional test case.
        // Use XCTAssert and related functions to verify your tests produce
            the correct results.
    }

    func testPerformanceExample() {
        // This is an example of a performance test case.
        self.measure {
            // Put the code you want to measure the time of here.
        }
    }

}
```

3 First test case

Unit testing with XCTest

```
import XCTest
@testable import DemoTDD

class DemoTDDTests: XCTestCase {

    override func setUp() {
        // Put setup code here. This method is called before the invocation of
            each test method in the class.
    }

    override func tearDown() {
        // Put teardown code here. This method is called after the invocation
            of each test method in the class.
    }

    func testExample() {
        // This is an example of a functional test case.
        // Use XCTAssert and related functions to verify your tests produce
            the correct results.
    }

    func testPerformanceExample() {
        // This is an example of a performance test case.
        self.measure {
            // Put the code you want to measure the time of here.
        }
    }

}
```

4 Performance test

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://www.martinfowler.com/bliki/GivenWhenThen.html
https://xp123.com/articles/3a-arrange-act-assert/

Good test structure

```
func testSayHiWithUp1_should_return_Hello_up1() {
    // Arrange
    let greeting = Greeting()

    // Act
    let actualResult = greeting.sayHi(name: "up1")

    // Assert
    XCTAssertEqual("Hello up1", actualResult)
}
```

Name of test case

```
func testSayHiWithUp1_should_return_Hello_up1() {
    // Arrange
    let greeting = Greeting()

    // Act
    let actualResult = greeting.sayHi(name: "up1")

    // Assert
    XCTAssertEqual("Hello up1", actualResult)
}
```

Name of test case

MethodName_StateUnderTest_Expected

MethodName_Expected_StateUnderTest

Feature to be tested

Should_Expected_When_StateUnderTest

When_StateUnderTest_Should_Expected

Given_Precondition_When_StateUnderTest_Then_Expected

Design your code

```
func testSayHiWithUp1_should_return_Hello_up1() {
    // Arrange
    let greeting = Greeting()

    // Act
    let actualResult = greeting.sayHi(name: "up1")

    // Assert
    XCTAssertEqual("Hello up1", actualResult)
}
```

Verification with expected result

```
func testSayHiWithUp1_should_return_Hello_up1() {
    // Arrange
    let greeting = Greeting()

    // Act
    let actualResult = greeting.sayHi(name: "up1")

    // Assert
    XCTAssertEqual("Hello up1", actualResult)
}
```

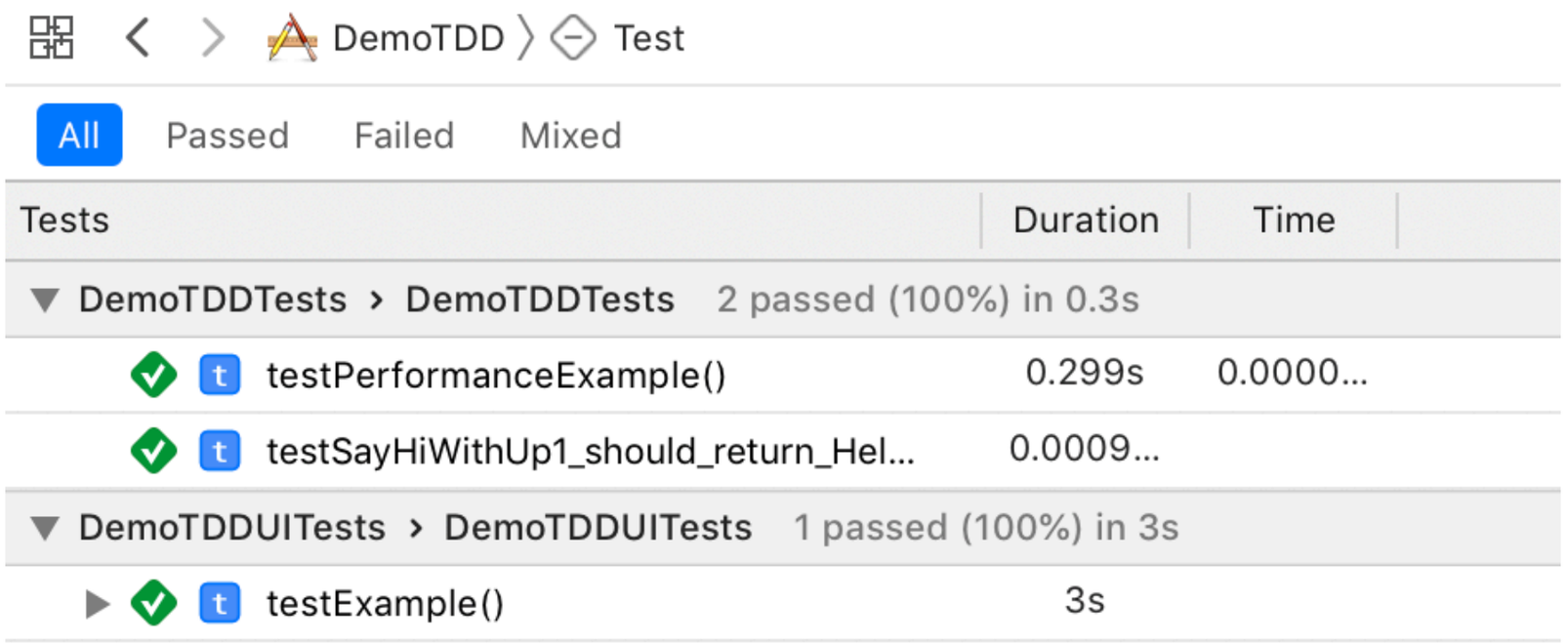
Assertion in XCTest

Method	Description
XCTAssertEqual	Testing for equality
XCTAssertNotEqual	Testing for inequality
XCTAssertTrue	Testing if a condition is True
XCTAssertFalse	Testing if a condition is False
XCTAssertThrowsError	Testing for Errors
XCTAssertNoThrow	Testing for Errors

https://developer.apple.com/documentation/xctest

Make your test pass !!

Run your test and see result



Write a second test case

Workshop

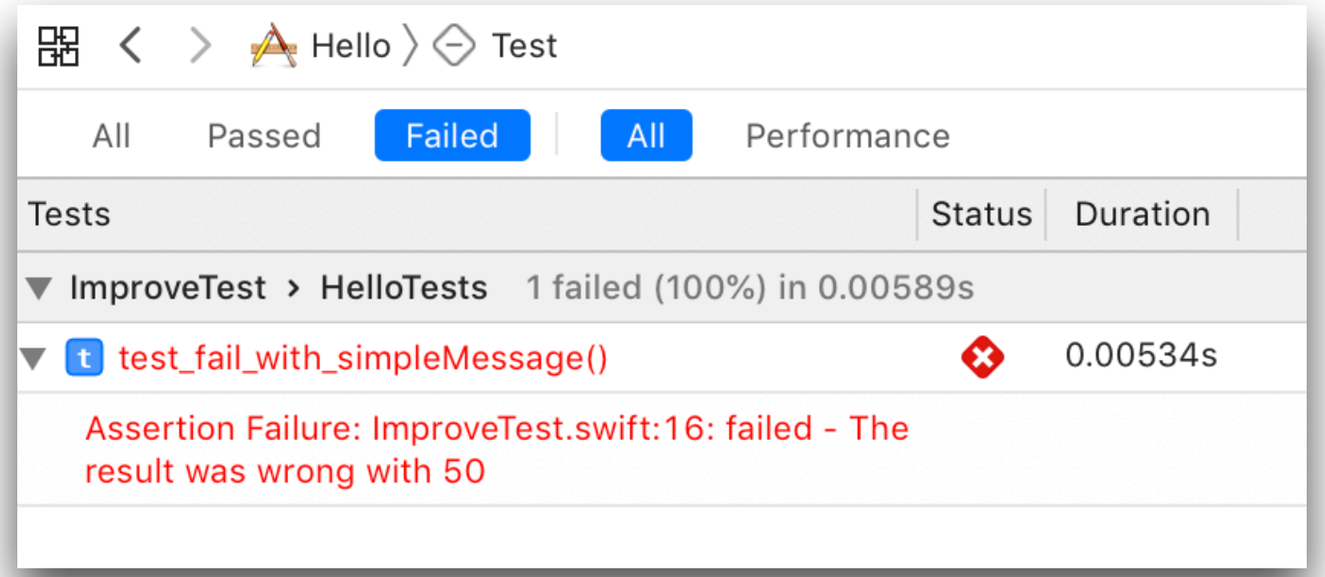
Input	Expected result
[1, 5]	1,2,3,4,5
[1, 5)	1,2,3,4
(1, 5]	2,3,4,5
(1, 5)	2,3,4

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

Listen from your tests

Add a descriptive message

```
class ImproveTest: XCTestCase {

    func test_fail_with_simpleMessage() {
        let result = 50

        XCTFail("The result was wrong with \(result)")      failed - The result was wrong with 50
    }

}
```



Avoid conditional in tests

Eliminate branches from test code
Make it easy to understand

Avoid conditional in tests

Bad

```
func test_avoid_conditionalCode() {
    let success = false
    if !success {
        XCTFail()
    }
}
```

Good

```
func test_assertTrue() {
    let success = false
    XCTAssertTrue(success)
}
```

Describing objects upon failure

Description message tell us only What happen

But can’t tell us Why !!

Custom description of Struct

```
struct customStruct: CustomStringConvertible {
    let x: Int
    let y: Int

    var description: String {
        return ("\(x), \(y)")
    }
}

func test_assertNil_withSelfDescription() {
    let data = customStruct(x: 1, y: 2)
    XCTAssertNil(data)
}
```

Testing for equality

In XCTAssertEqual,
the argument order doesn't matter

```
func test_assertEqual() {
    let actual = "actual"
    XCTAssertEqual(actual, "expect")
}
```

XCTAssertEqual failed: ("actual") is not equal to ("expect")

Need to improve failure message

Format (actual, expected)

Easy to understand
Communicate to your team

```
func test_improve_assertEqual() {
    XCTAssertEqual("expect", "actual")
}
```

Testing with double and float !!

```
func test_floatingPoint() {
    let result = 0.1 + 0.2
    XCTAssertEqual(0.3, result)
```

XCTAssertEqual failed: ("0.3") is not equal to ("0.30000000000000004")

Using accuracy

```
func test_floatingPoint_withAccuracy() {
    let result = 0.1 + 0.2
    XCTAssertEqual(0.3, result, accuracy: 0.0001)
}
```

Choose the right assertion

Each assertion have **goals** and **purposes**

Fail when something other than expected

Document how the SUT is suppose to behave

What happen when test failed ?

It’s enough to report
What the actual result ?
How differ from expected result ?
Choose assertion that closest to what yo want

XCTAssert (15)

Method	Description
XCTAssertEqual	Assert 2 values are equal
XCTAssertNotEqual	Assert 2 values are not equal
XCTAssertTrue	Asserts that an expression is true
XCTAssertFalse	Asserts that an expression is false
XCTAssertThrowsError	Testing for Errors
XCTAssertNoThrow	Testing for Errors
XCTAssertNil	Asserts that an optional value is nil
XCTFail	Fails the current test

https://developer.apple.com/documentation/xctest

Duplication in test code

```
class MyClassTest: XCTestCase {

    func test_methodOne() {
        let sut = MyClass()
        sut.methodOne()
    }

    func test_methodTwo() {
        let sut = MyClass()
        sut.methodTwo()
    }

}
```

Remove duplication in wrong way !!

Problem with global state !!

```
class MyClassTest: XCTestCase {

    private let sut = MyClass()

    func test_methodOne() {
        sut.methodOne()
    }

    func test_methodTwo() {
        sut.methodTwo()
    }

}
```

Remove duplication in wrong way !!

Instance of MyClass not destroy

```
Called init
Called init
Test Suite 'Selected tests' started at 2019-06-23 14:39:10.479
Test Suite 'HelloTests.xctest' started at 2019-06-23 14:39:10.479
Test Suite 'MyClassTest' started at 2019-06-23 14:39:10.479
Test Case '-[HelloTests.MyClassTest test_methodOne]' started.
Called 1
Test Case '-[HelloTests.MyClassTest test_methodOne]' passed (0.001 seconds).
Test Case '-[HelloTests.MyClassTest test_methodTwo]' started.
Called 2
Test Case '-[HelloTests.MyClassTest test_methodTwo]' passed (0.000 seconds).
```

Remove duplication

Using test life cycle

```
class MyClassTest: XCTestCase {

    private var sut: MyClass!

    override func setUp() {
        super.setUp()
        sut = MyClass()
    }

    override func tearDown() {
        sut = nil
        super.tearDown()
    }
```

Remove duplication

Using test life cycle

```
Test Case '-[HelloTests.MyClassTest test_methodOne]' started.
Called init
Called 1
Called deinit
Test Case '-[HelloTests.MyClassTest test_methodOne]' passed (0.001 seconds).
```

```
Test Case '-[HelloTests.MyClassTest test_methodTwo]' started.
Called init
Called 2
Called deinit
Test Case '-[HelloTests.MyClassTest test_methodTwo]' passed (0.000 seconds).
```

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

Class coverage
Method coverage
Line coverage
Branch coverage

Code coverage

```
1.  public class MyRange {
2.      public boolean startWithInclude(String input) {
3.          return input.startsWith("[");
4.      }
5.
6.      public boolean endWithInclude(String input) {
7.          if(input == null) {
8.              return false;
9.          }
10.         return input.endsWith("]");
11.     }
12.
13.     public boolean startWithInclude2(String input) {
14.         return input.startsWith("[");
15.     }
16. }
```

Code coverage

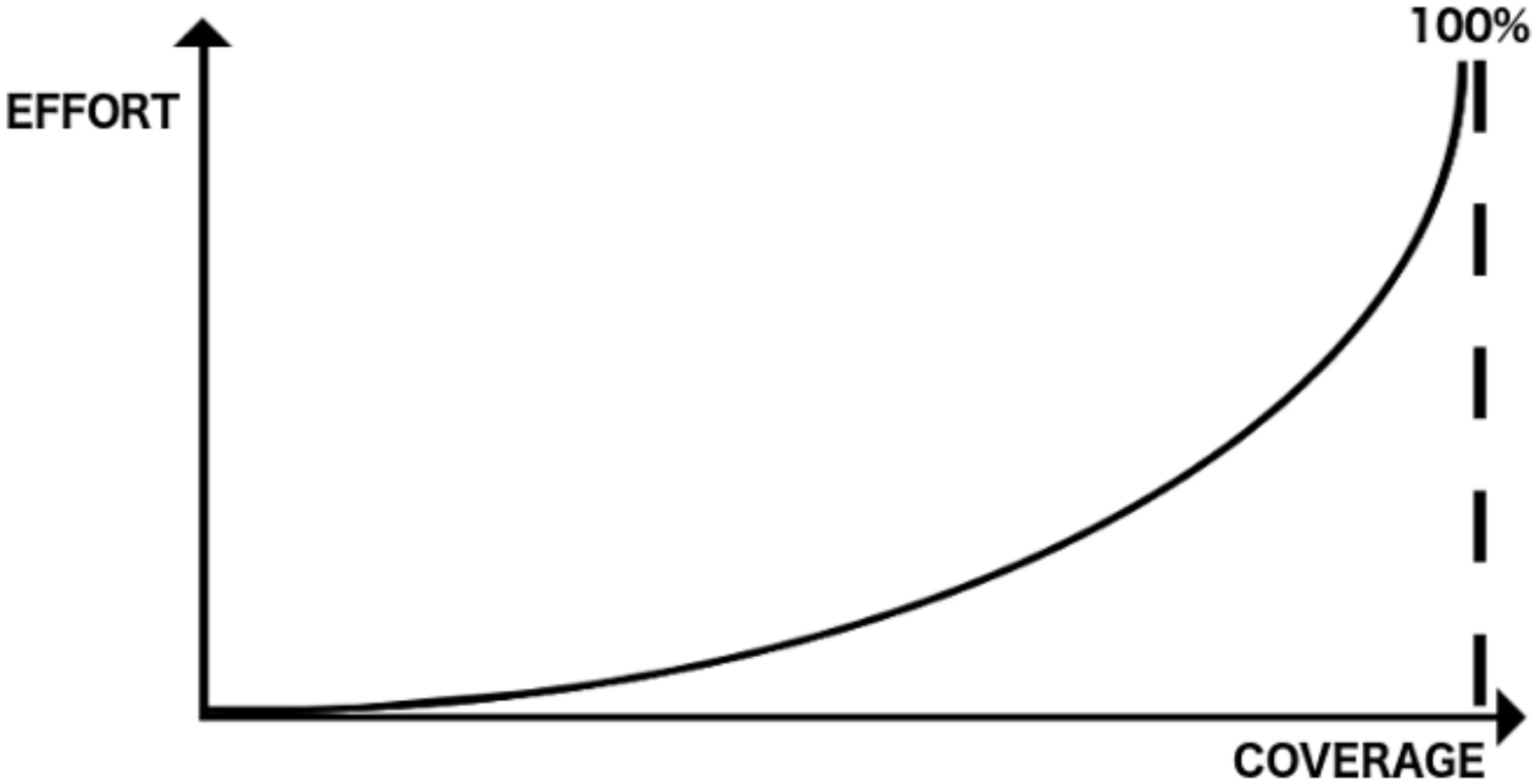
But 100% of code coverage **does not mean** that your code is 100% correct

Code coverage

Powerful tool to improve the quality of your code

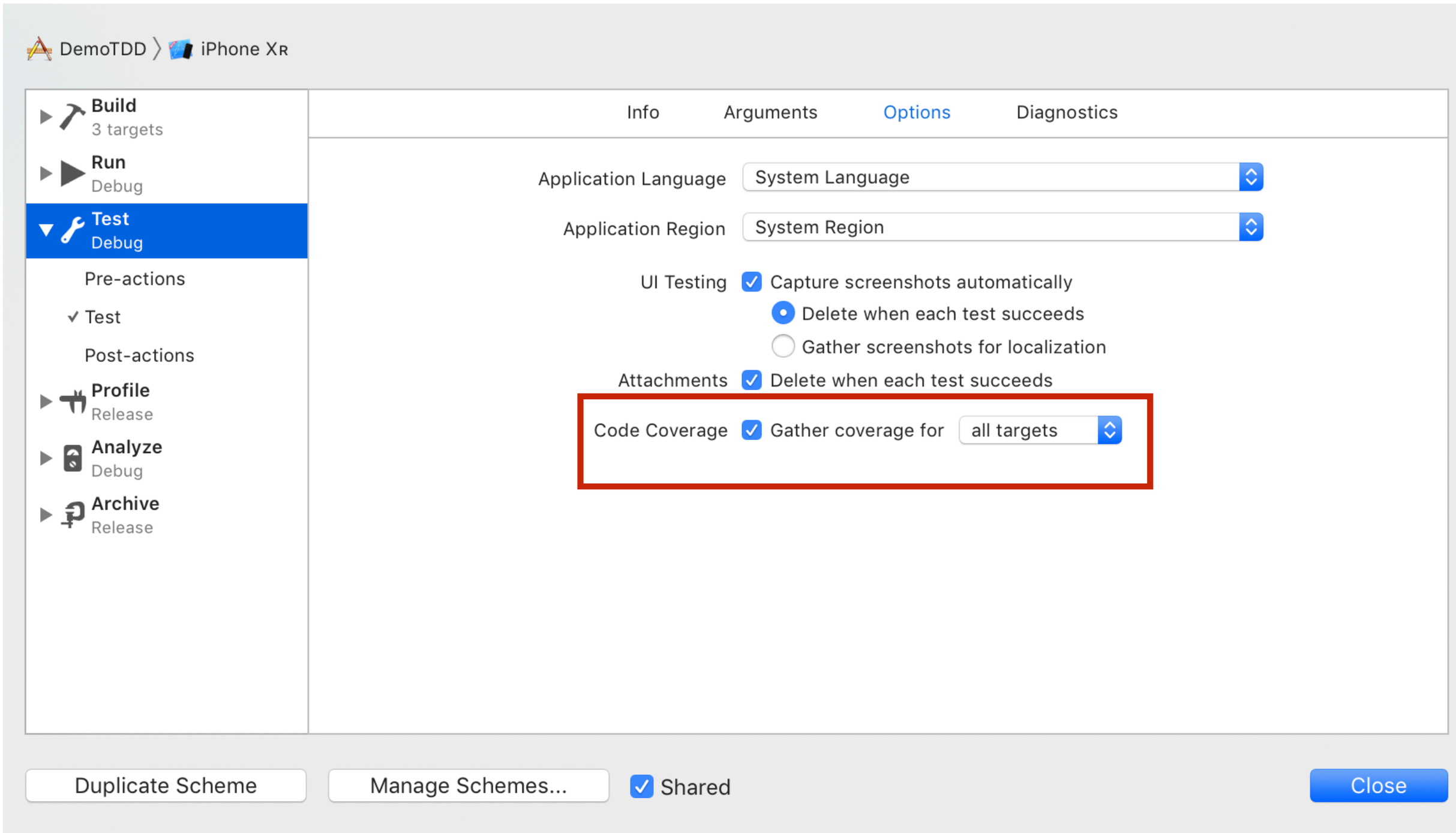
Code coverage != quality of tests

Code coverage 100% ?

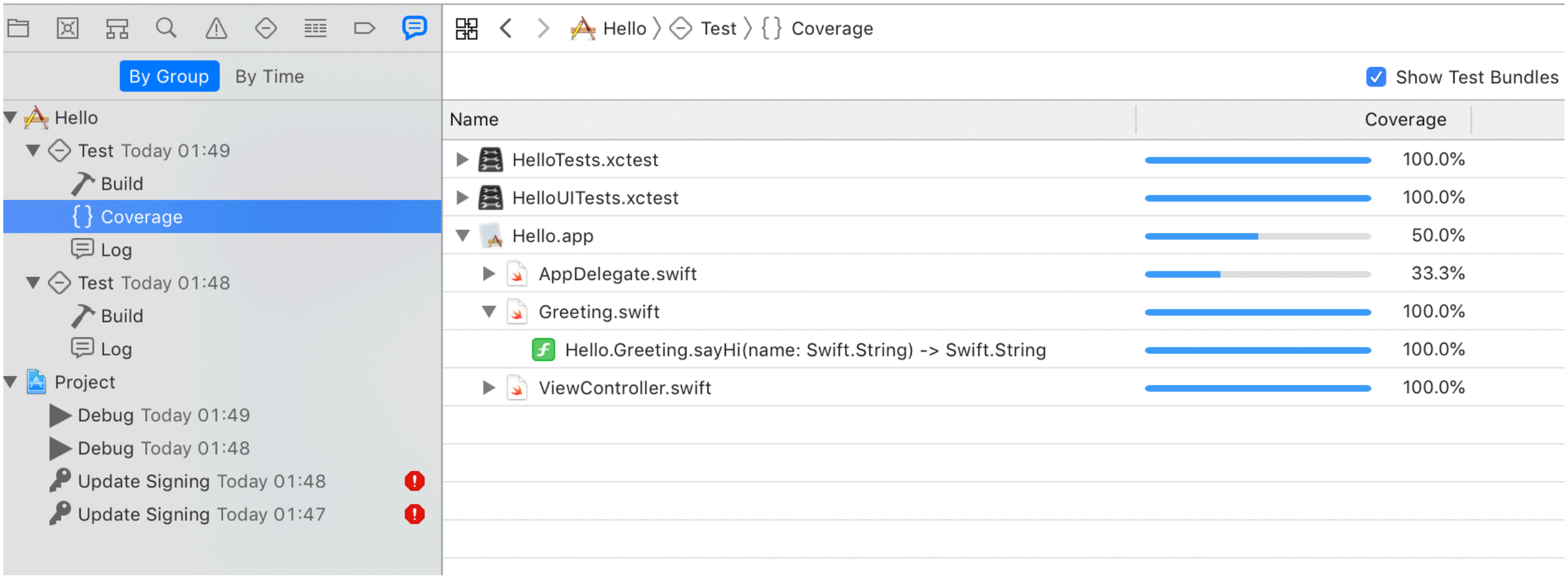


Code coverage in Xcode

Enable code coverage in Scheme



Code coverage report

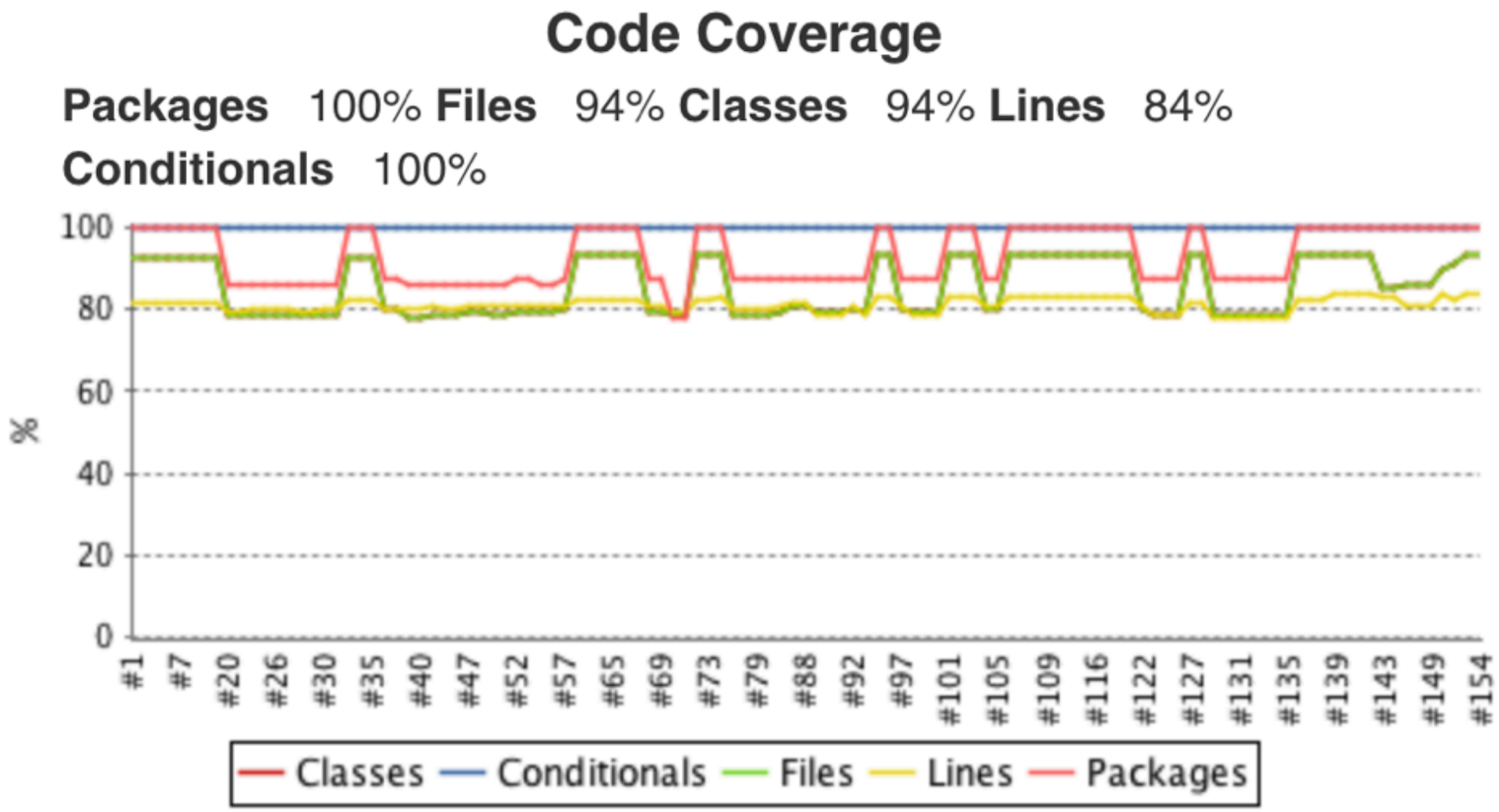


Code coverage in Editor

```
class MyClass {

    init() {                                    2
        print("Called init")
    }

    deinit {                                    0
        print("Called deinit")
    }

    func methodOne() {                          1
        print("Called 1")
    }

    func methodTwo() {                          1
        print("Called 2")
    }

}
```

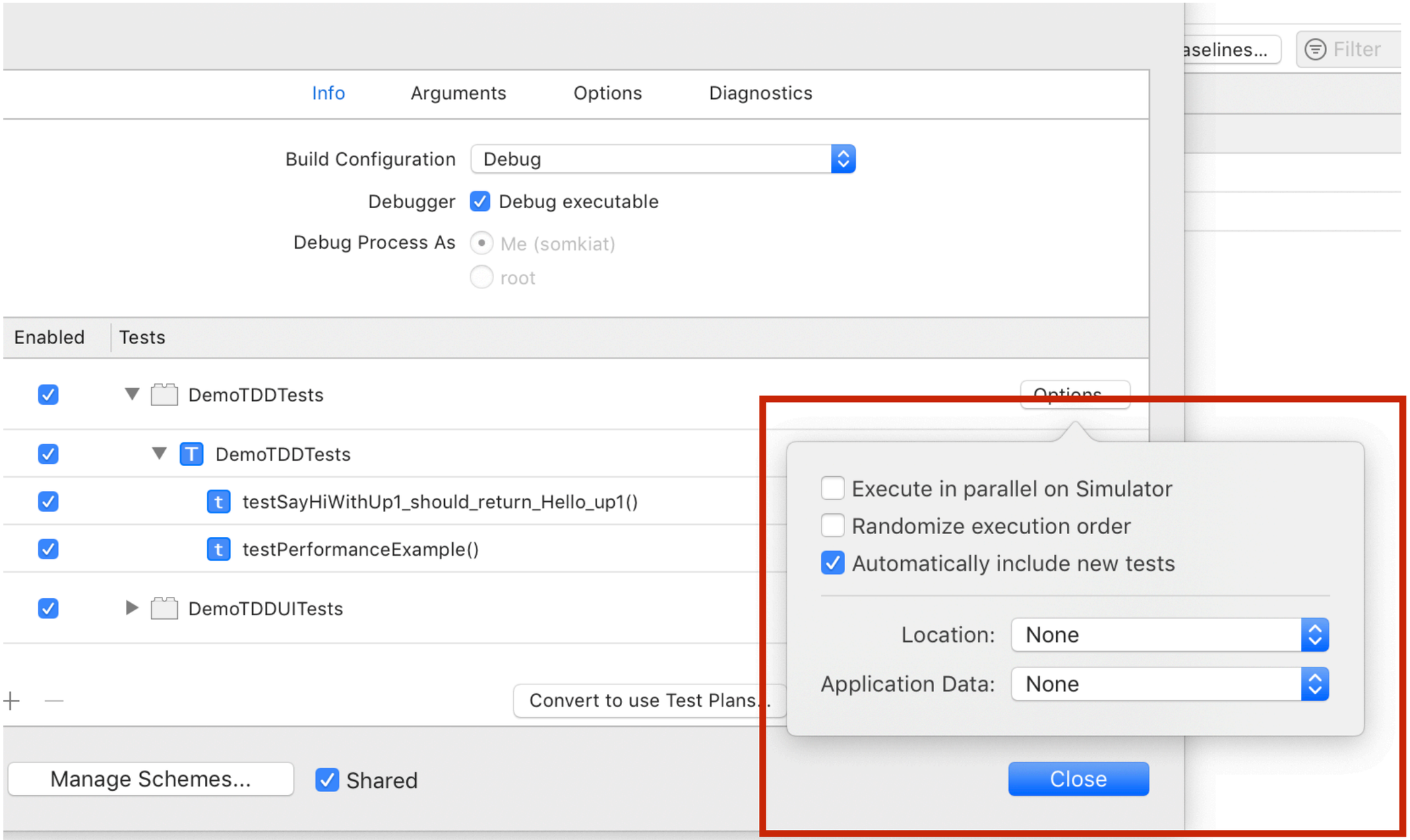
Code coverage report

Generate report from command line by **xcodebuild** and **xcrun**

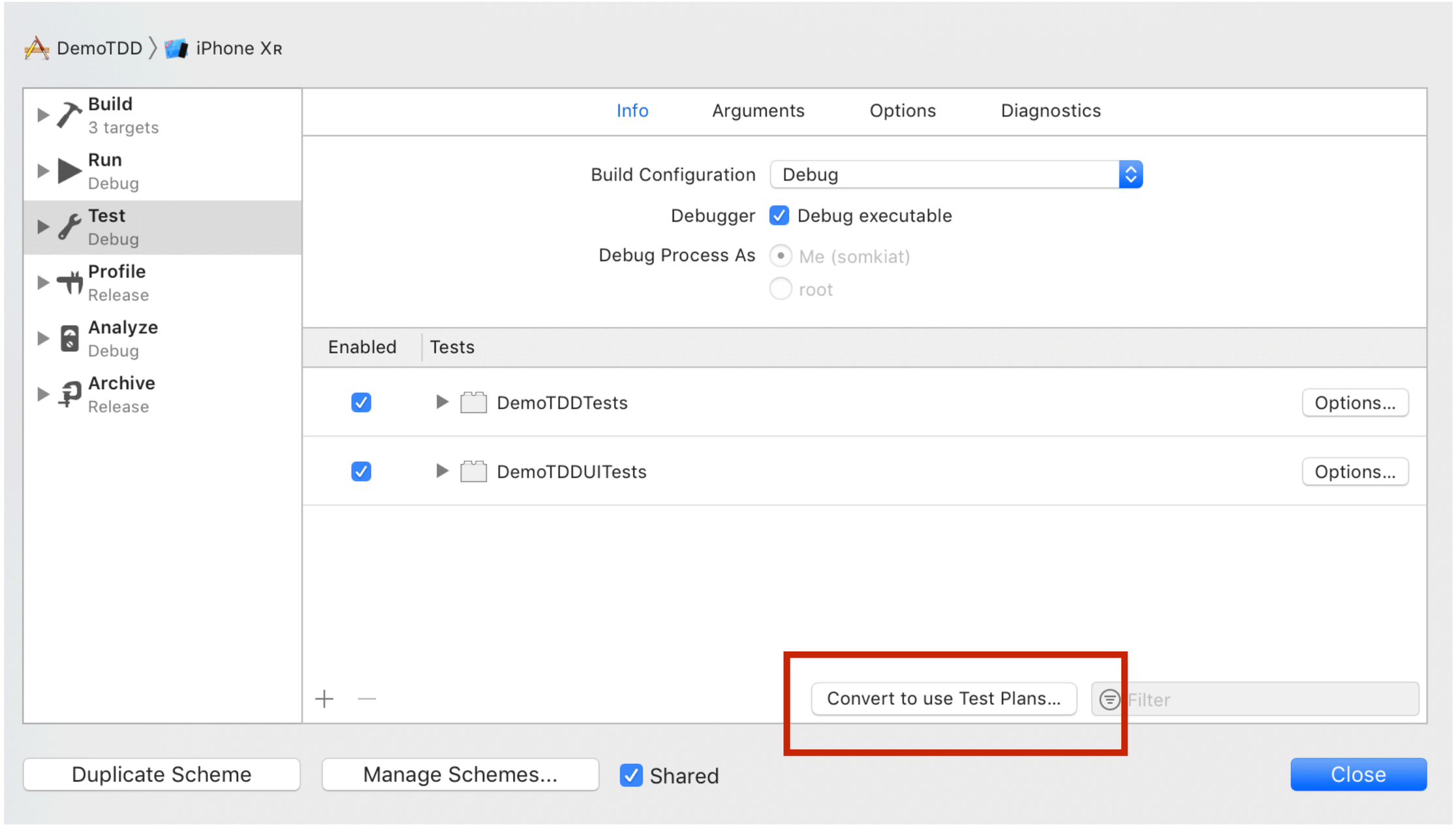


More test features in Xcode

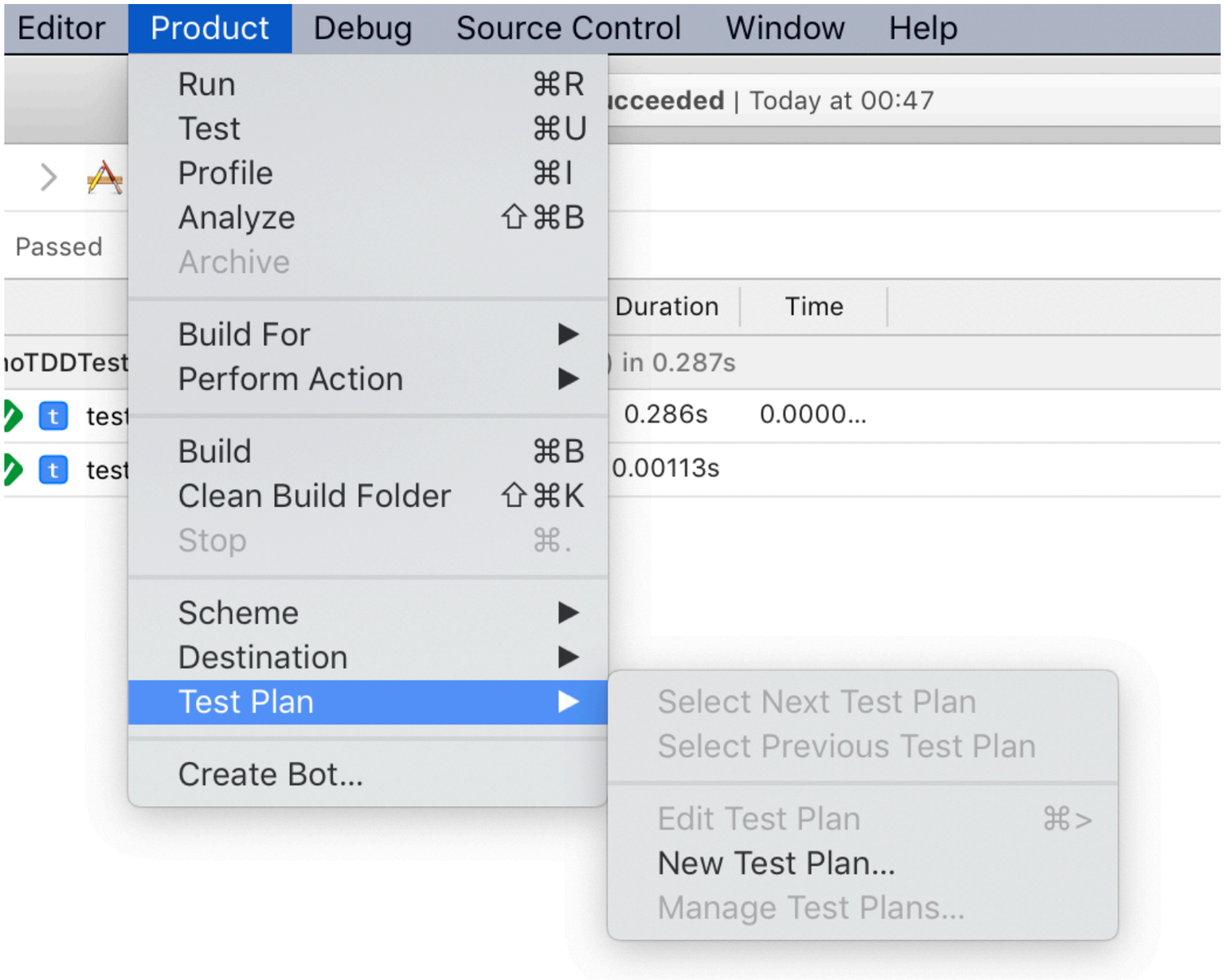
Run random and parallel



Test plan in Xcode 11



Test plan in Xcode 11



TDD in Iteration development

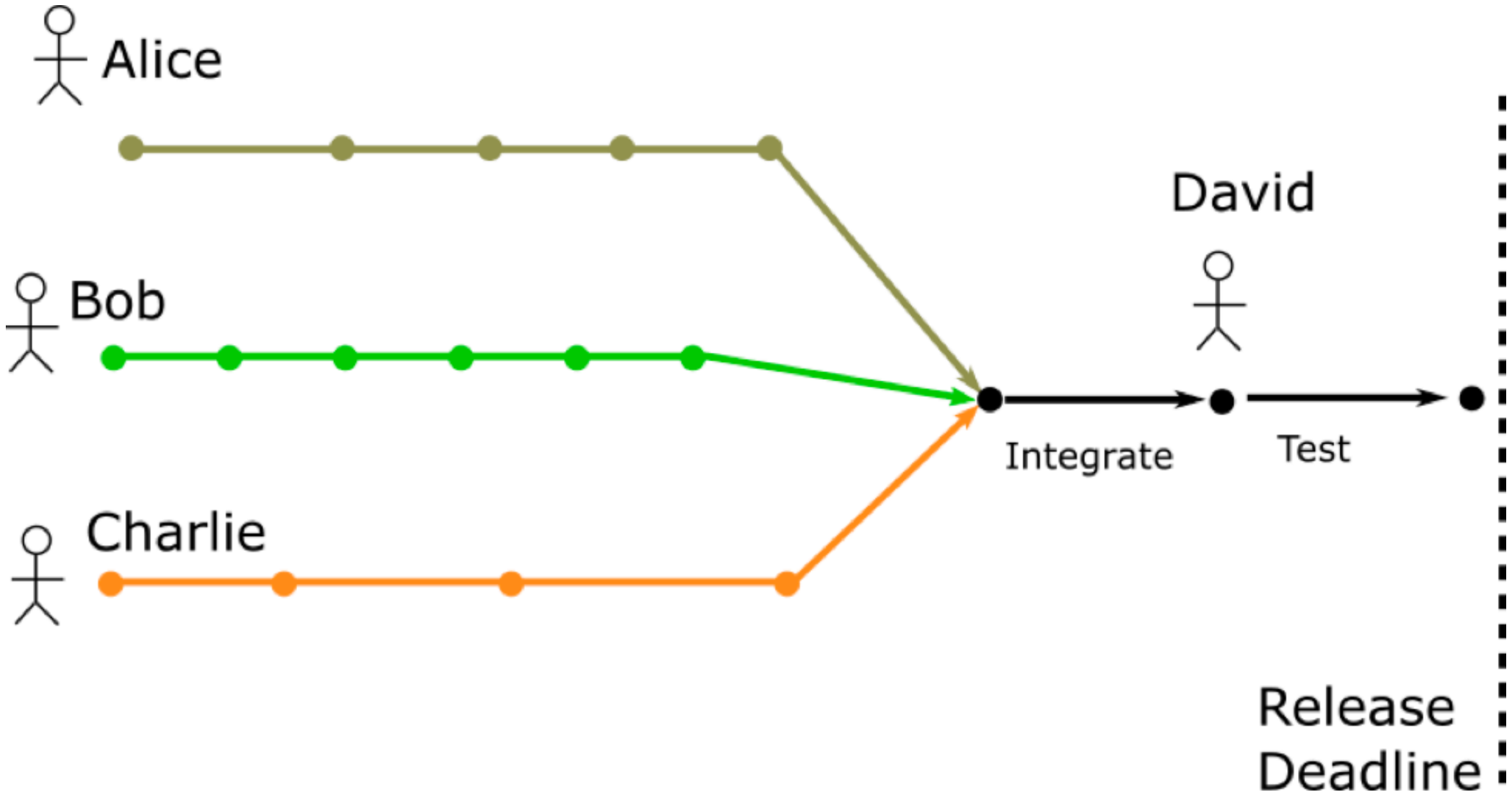
TDD in Iteration development

Iteration cycle 2-4 weeks

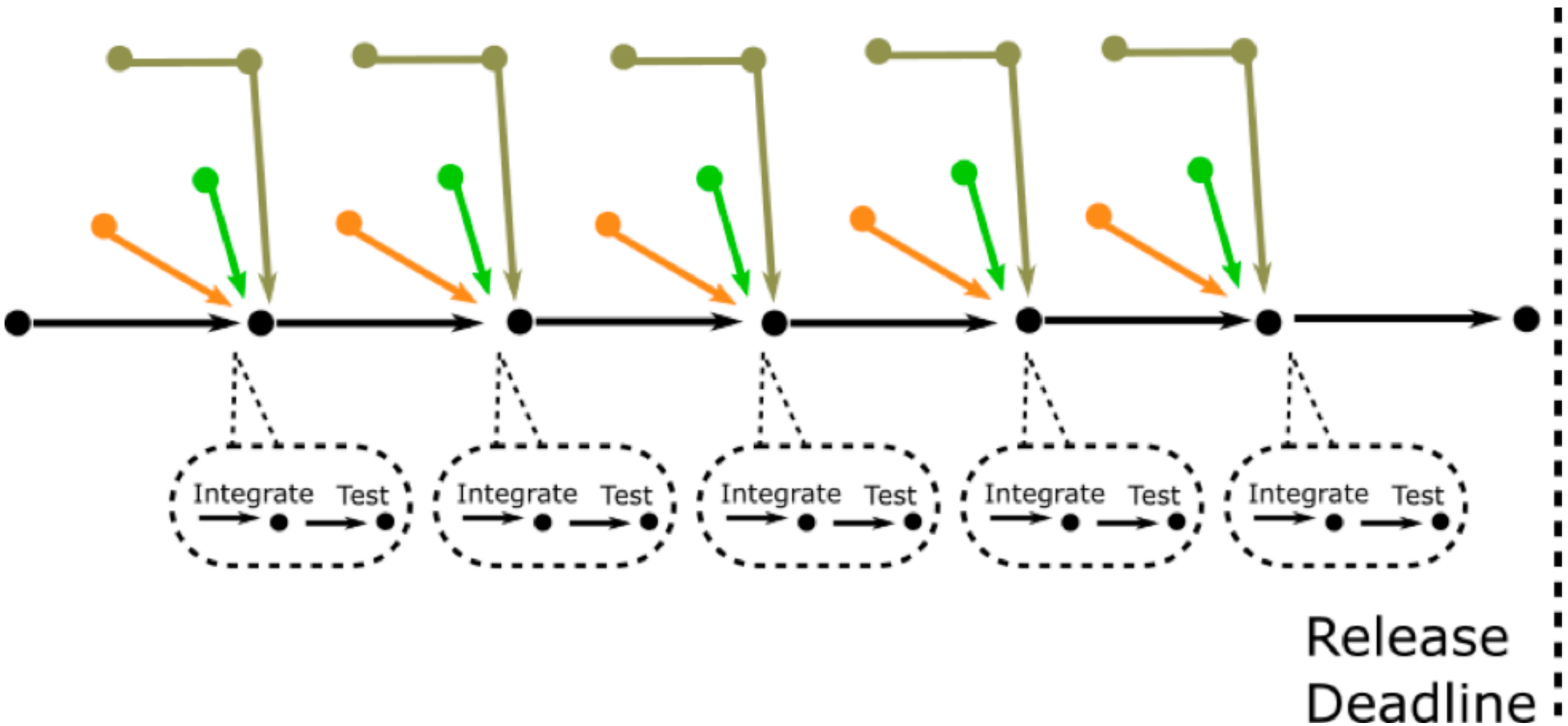
Continuous integration cycle
(multiple times a day)

TDD cycle in a few minutes (3-10 minutes)

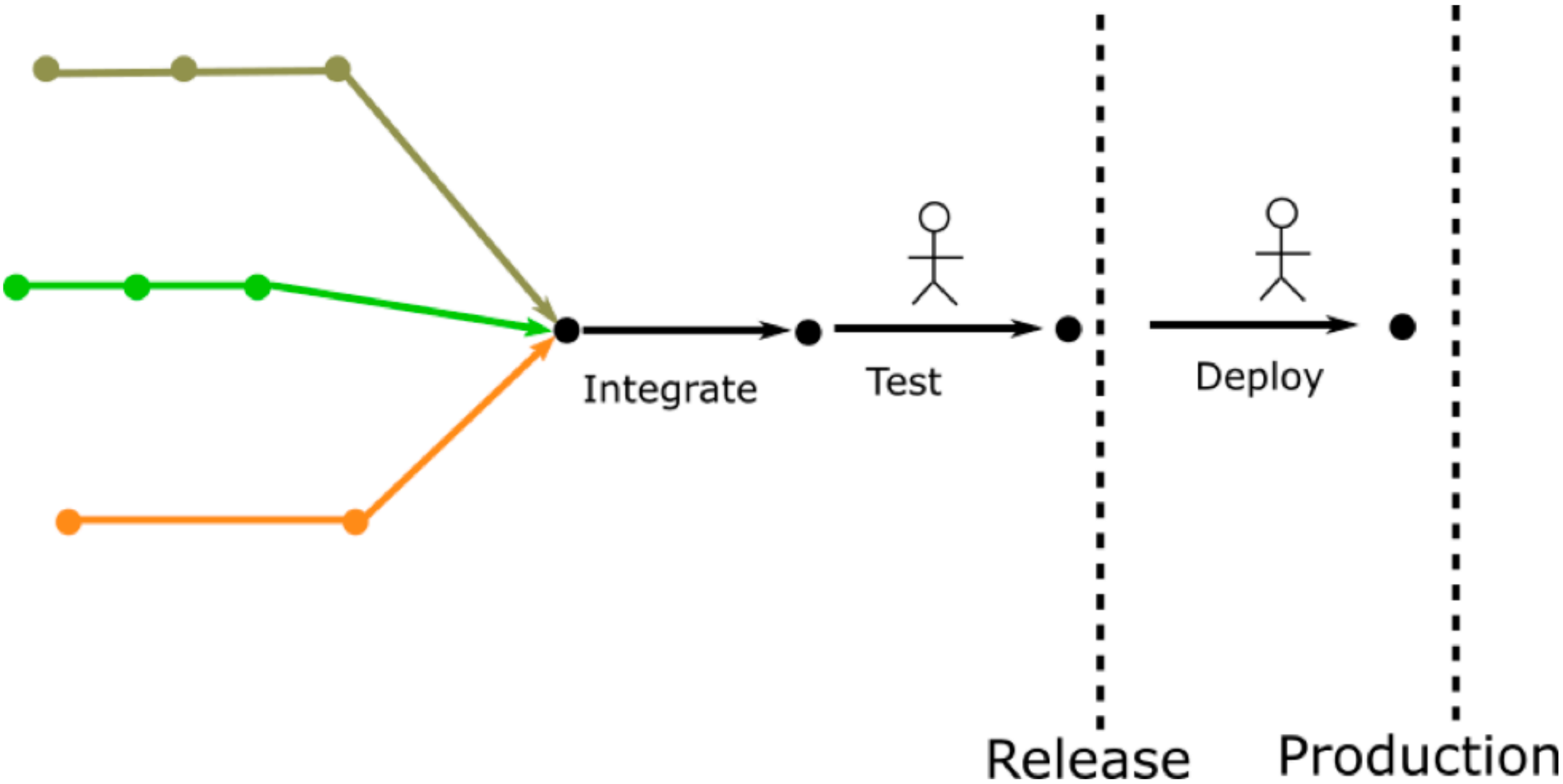
TDD in Iteration development



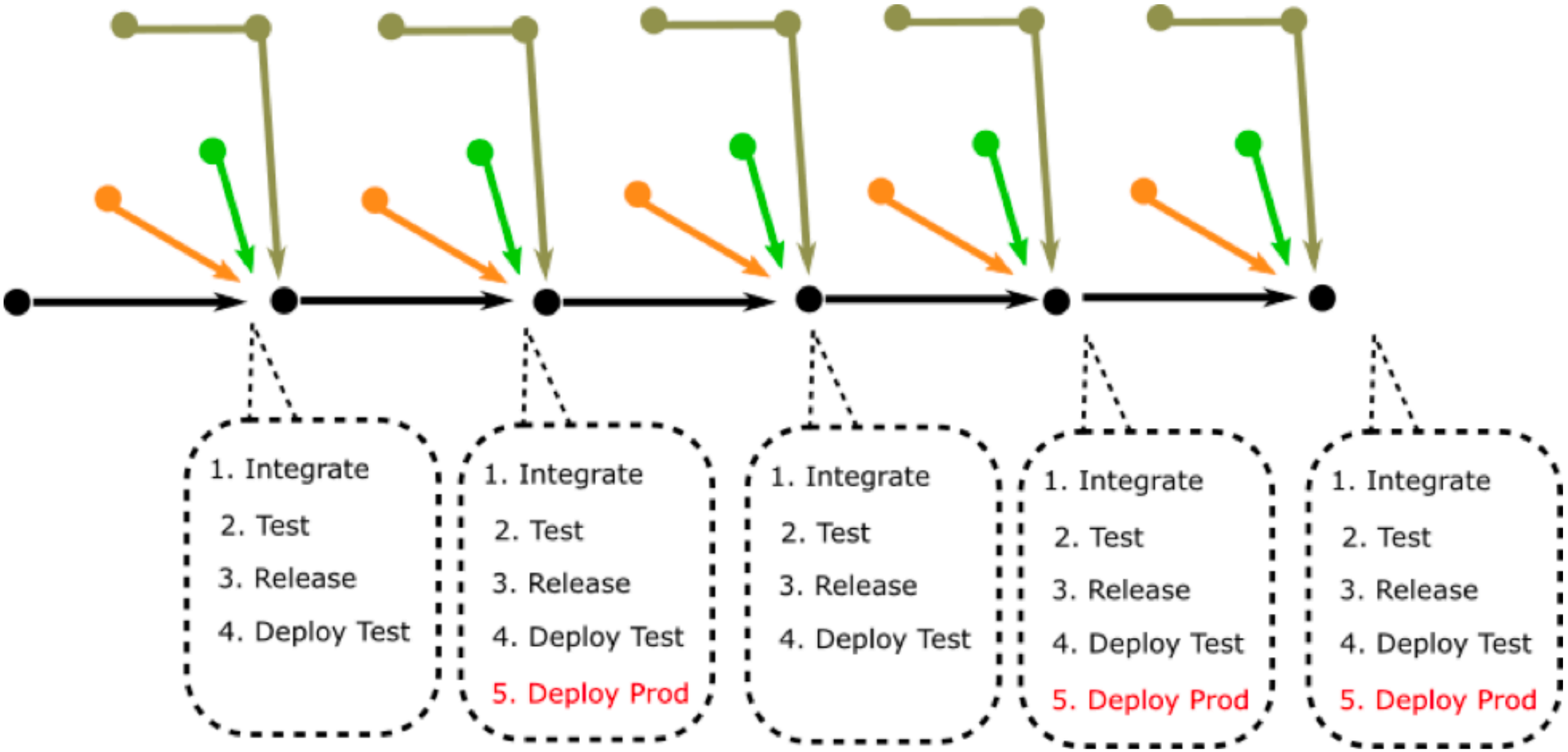
TDD in Iteration development



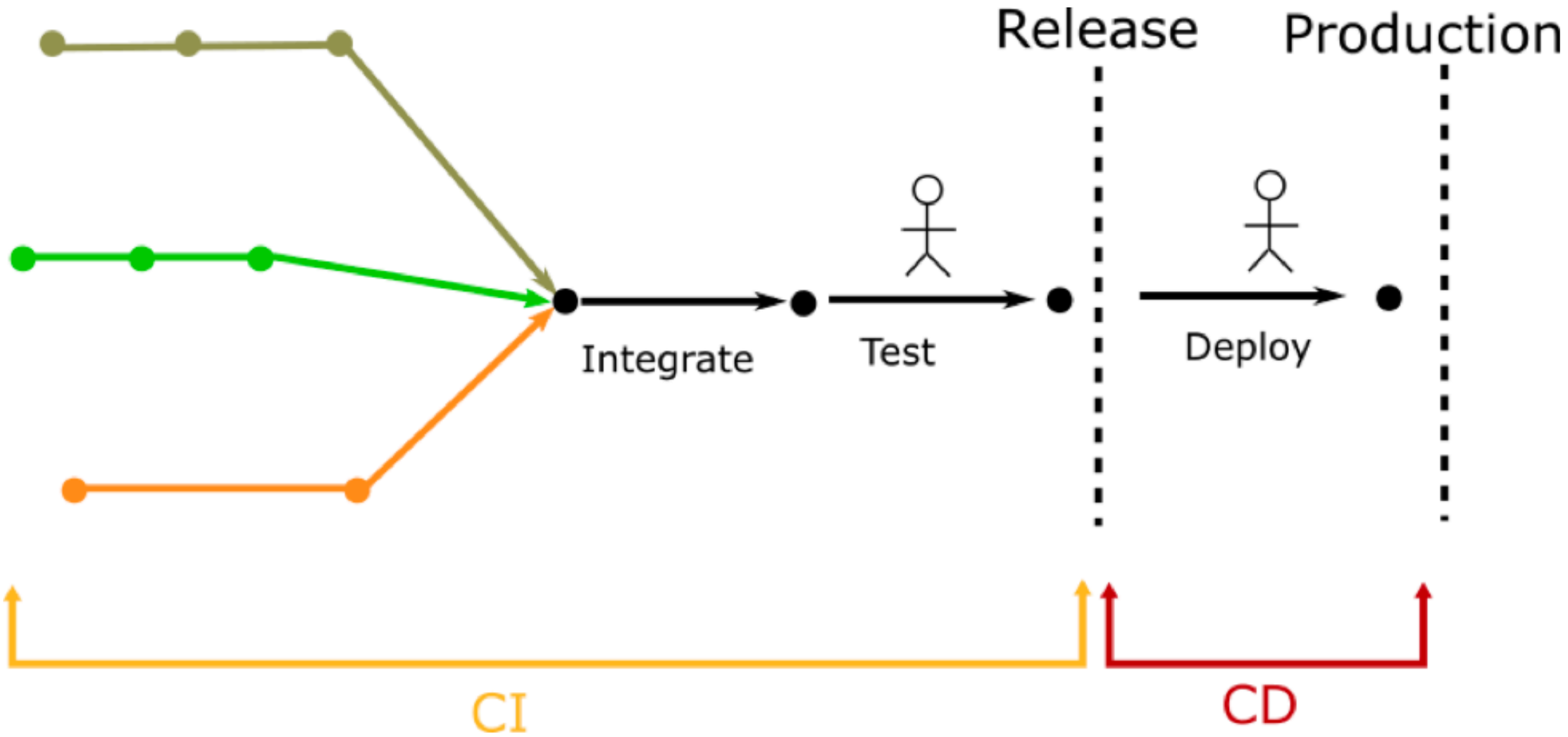
TDD in Iteration development



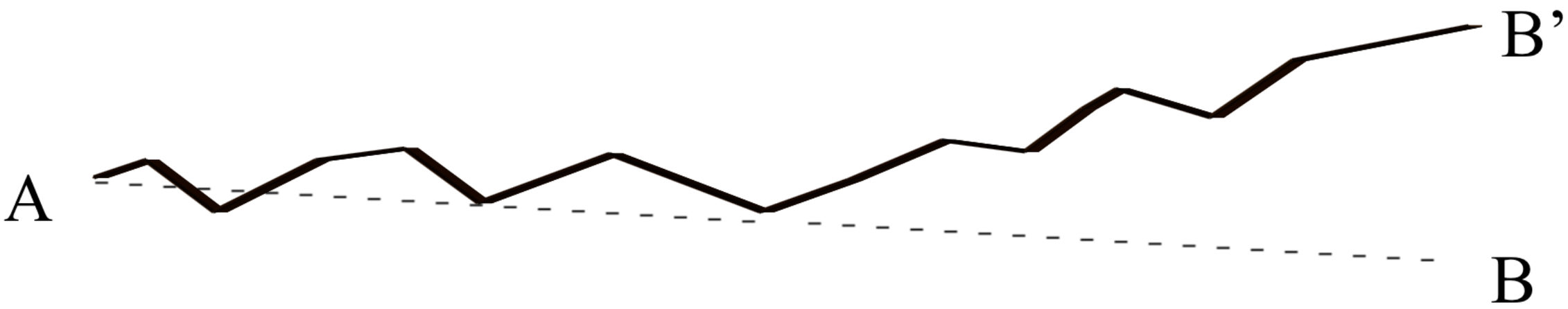
TDD in Iteration development



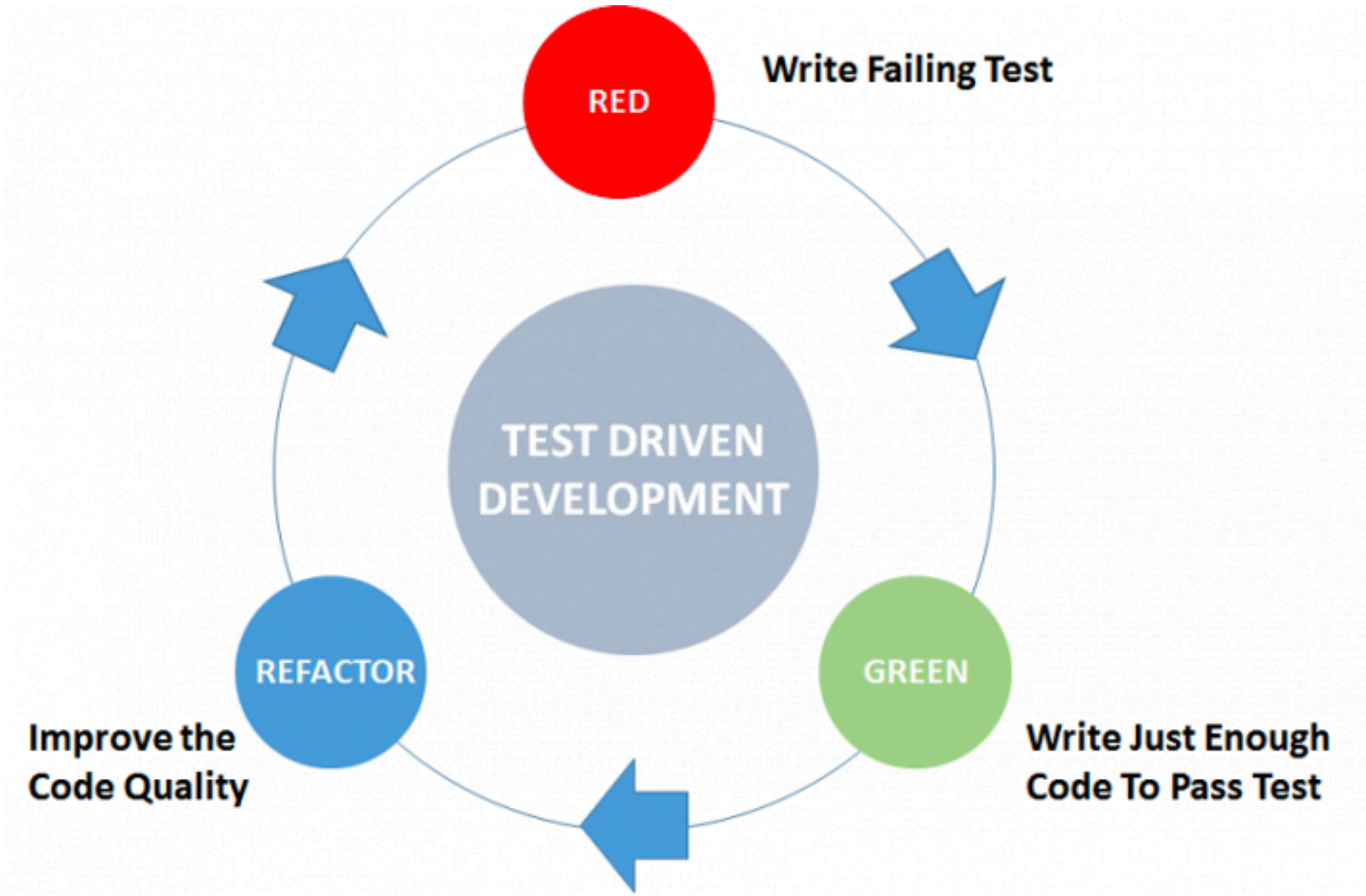
TDD in Iteration development



Small steps



TDD = ทำดีดี

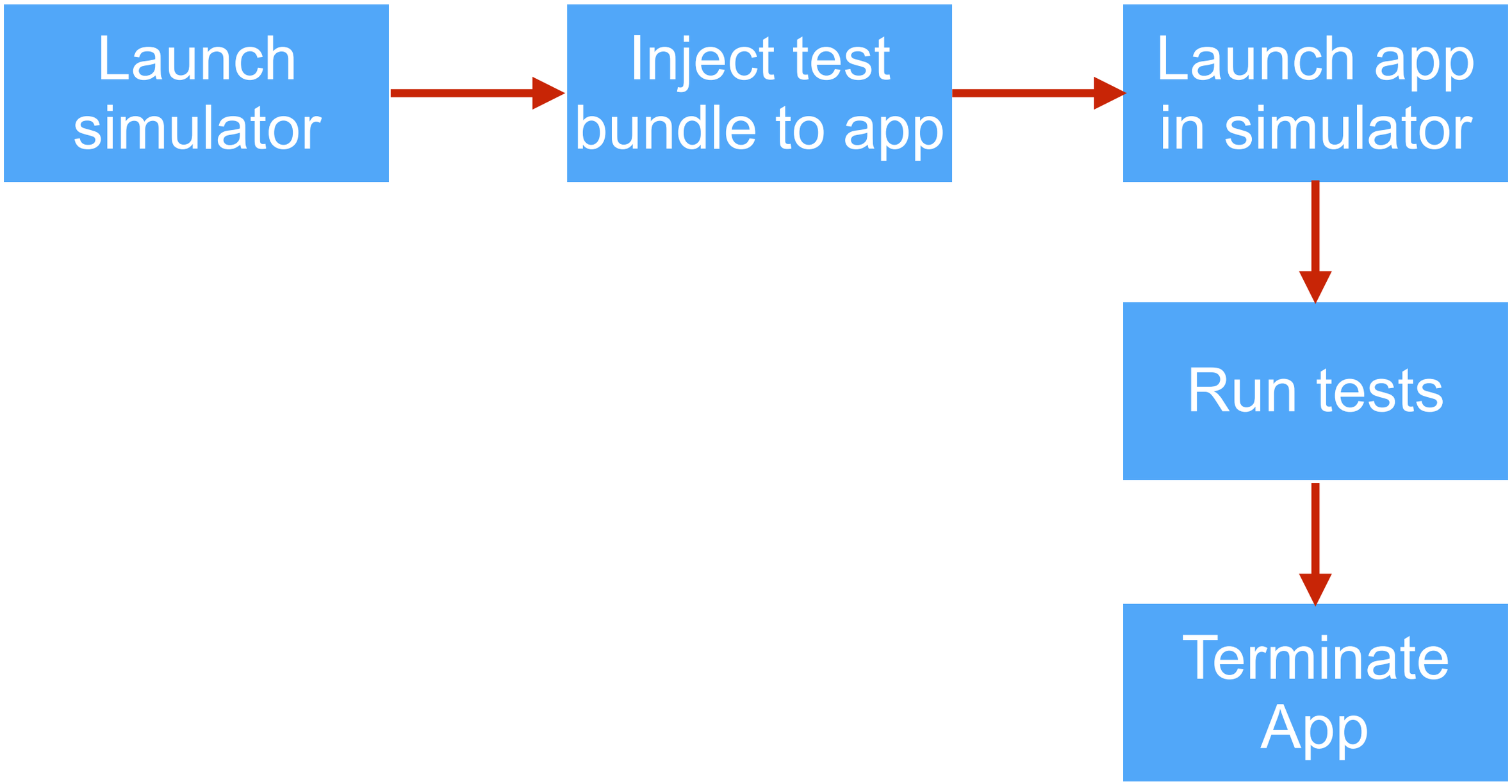


Practical testing

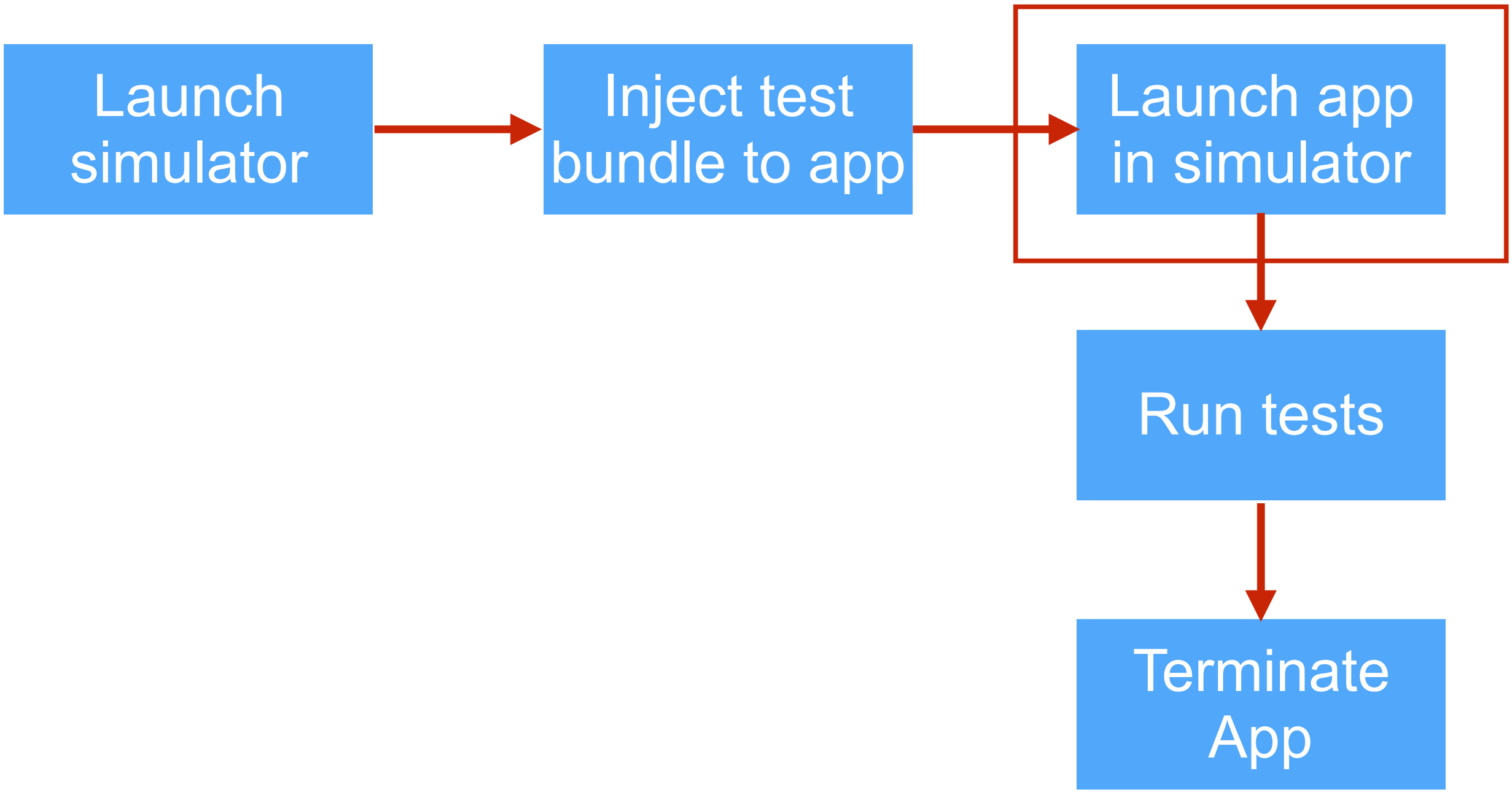
Basic of iOS app testing

Control application launch
Load ViewController
Manage dependencies

Control application launch



How to control in test ?

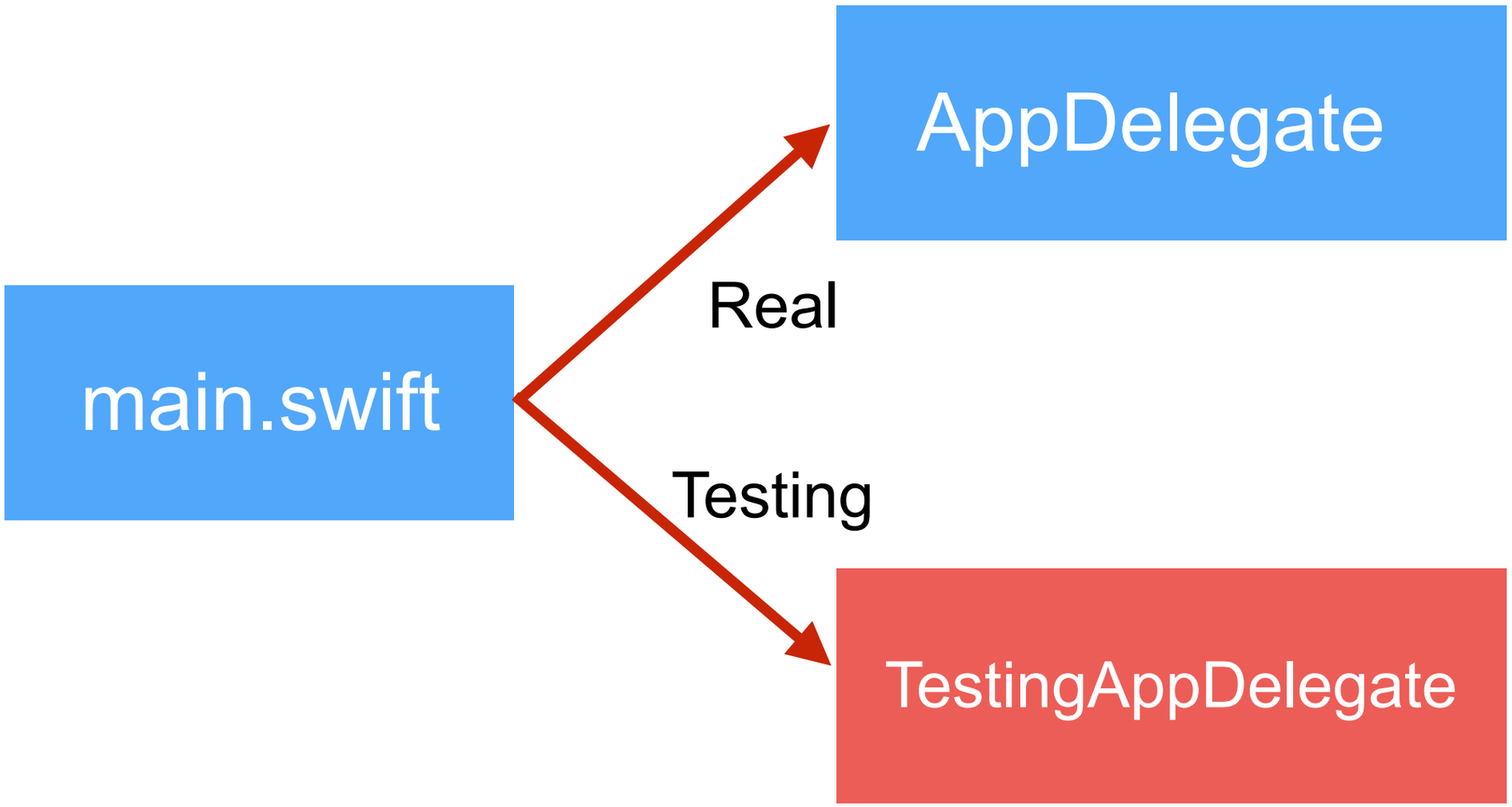


Working with UIKit …

Setup Core Data
Send specific-key to analytic service
Send request to fetch data

Need to control the App Delegate

By pass the AppDelegate



Step 1

Remove @UIApplicationMain from AppDelegate.swift

```
//@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchi
        launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> B

        print(">> Launched with real app delete")

        return true
    }
```

Step 2

Create file TestingAppDelegate.swift in Test folder
Should have only 1 function (application)

```
import UIKit

@objc(TestingAppDelegate)
class TestingAppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(
        _ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [UIApplication
            Any]?) -> Bool {

        print("<< Launched with testing app delegate")
        return true
    }
}
```

Step 3

Create file **main.swift** in Main module

To launch AppDelete in each context

```
import UIKit
let appDelegateClass: AnyClass =
    NSClassFromString("TestingAppDelegate") ?? AppDelegate.self

UIApplicationMain(
    CommandLine.argc,
    CommandLine.unsafeArgv, nil,
    NSStringFromClass(appDelegateClass))
```

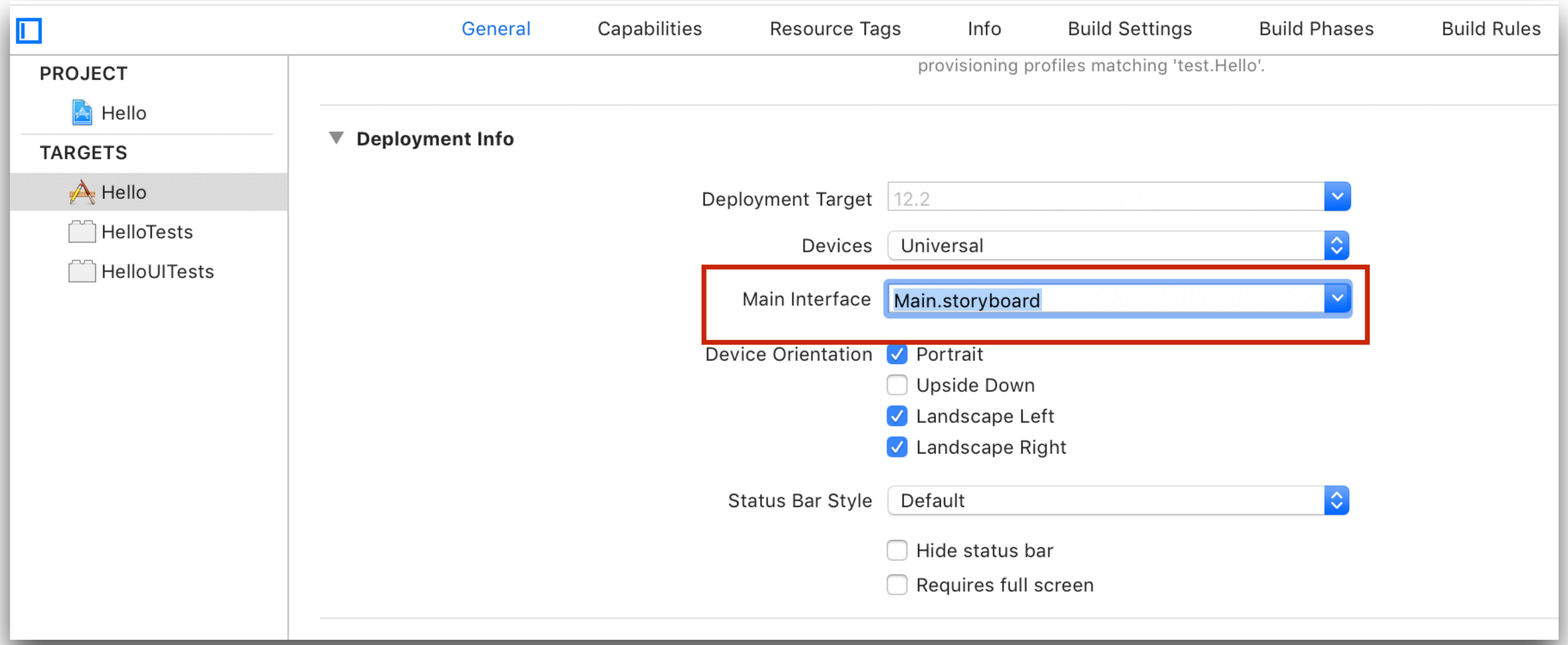
Try to tun test and app

Control app launch sequence during test

You can setup the environment for testing
Avoid doing any actual work

Try to custom main storyboard in code

Delete main interface in deployment info



Try to custom main storyboard in code

Modify in file AppDelegate.swift

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
    launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    print(">> Launched with real app delete")

    // Launch main storyboard
    window = UIWindow()
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    window?.rootViewController = storyboard.instantiateInitialViewController()
    window?.makeKeyAndVisible()

    return true
}
```

Load View Controller

Manage Dependencies

F.I.R.S.T unit test principles

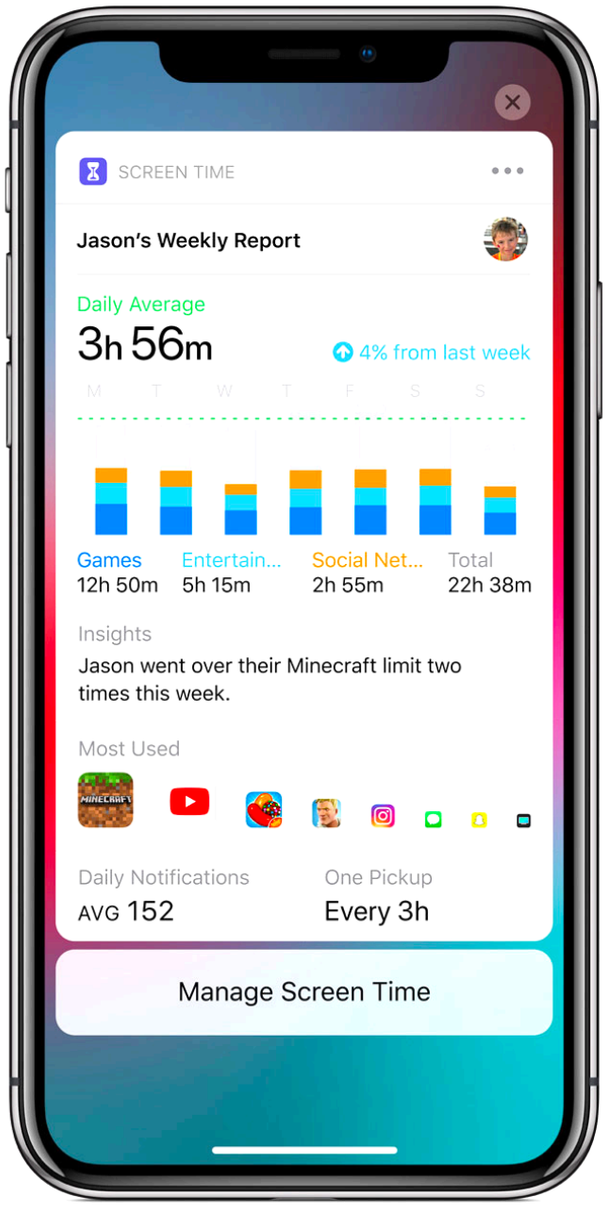
Fast
Isolate
Repeatable

Check your test …

Workshop

Login application

Login Page



Result

Success

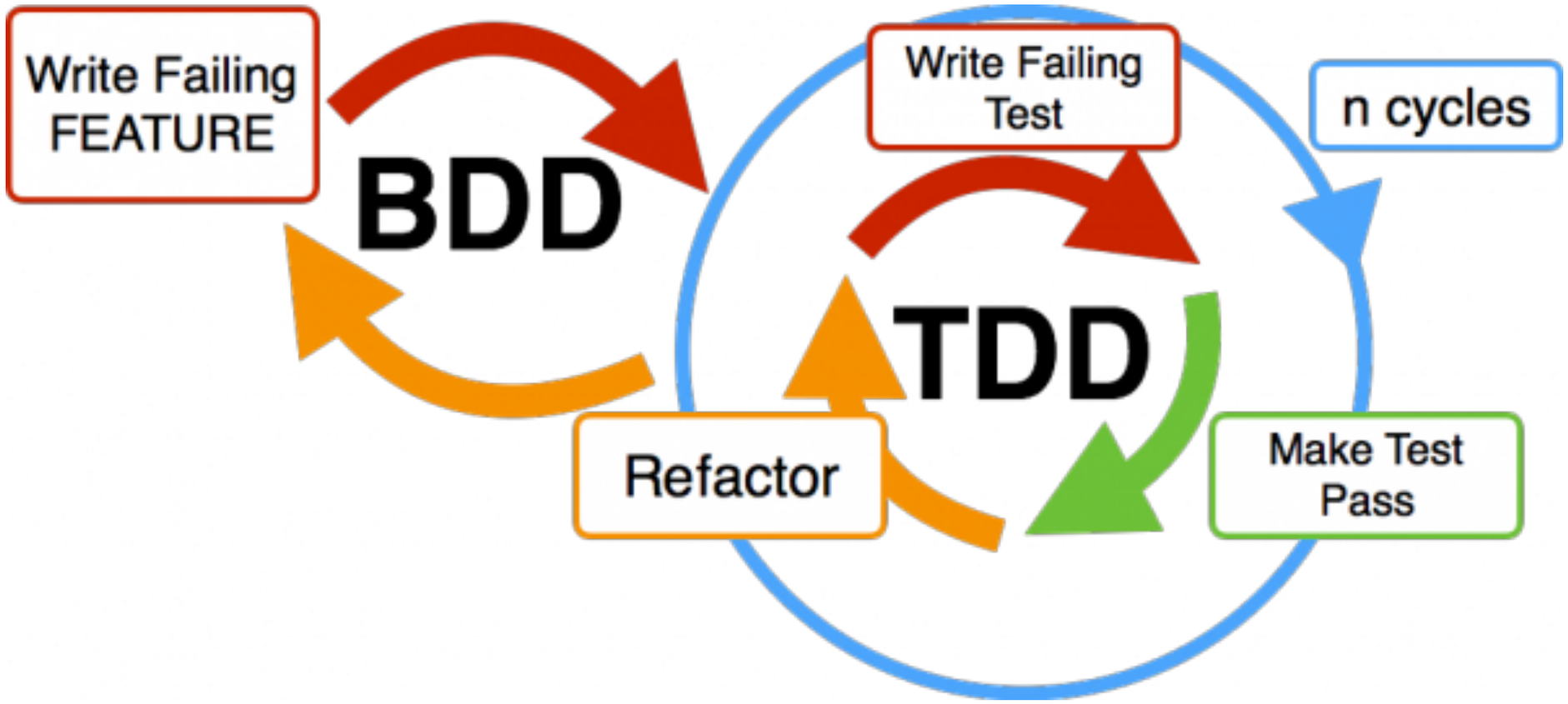
Failure

Write test case ?

Test cases

Test Case Name	Email	Password	Expected Result
Login success	somkiat@xxx.com		Show **success** message
Login fail			Show **fail** message

Start to write UI testing



Let’ start to write UI tests with UIXCTest

XCUITest

For UI testing
Xcode 9+ only
Combination XCTest and accessibility framework
Xcode Test recorder support

Accessibility framework

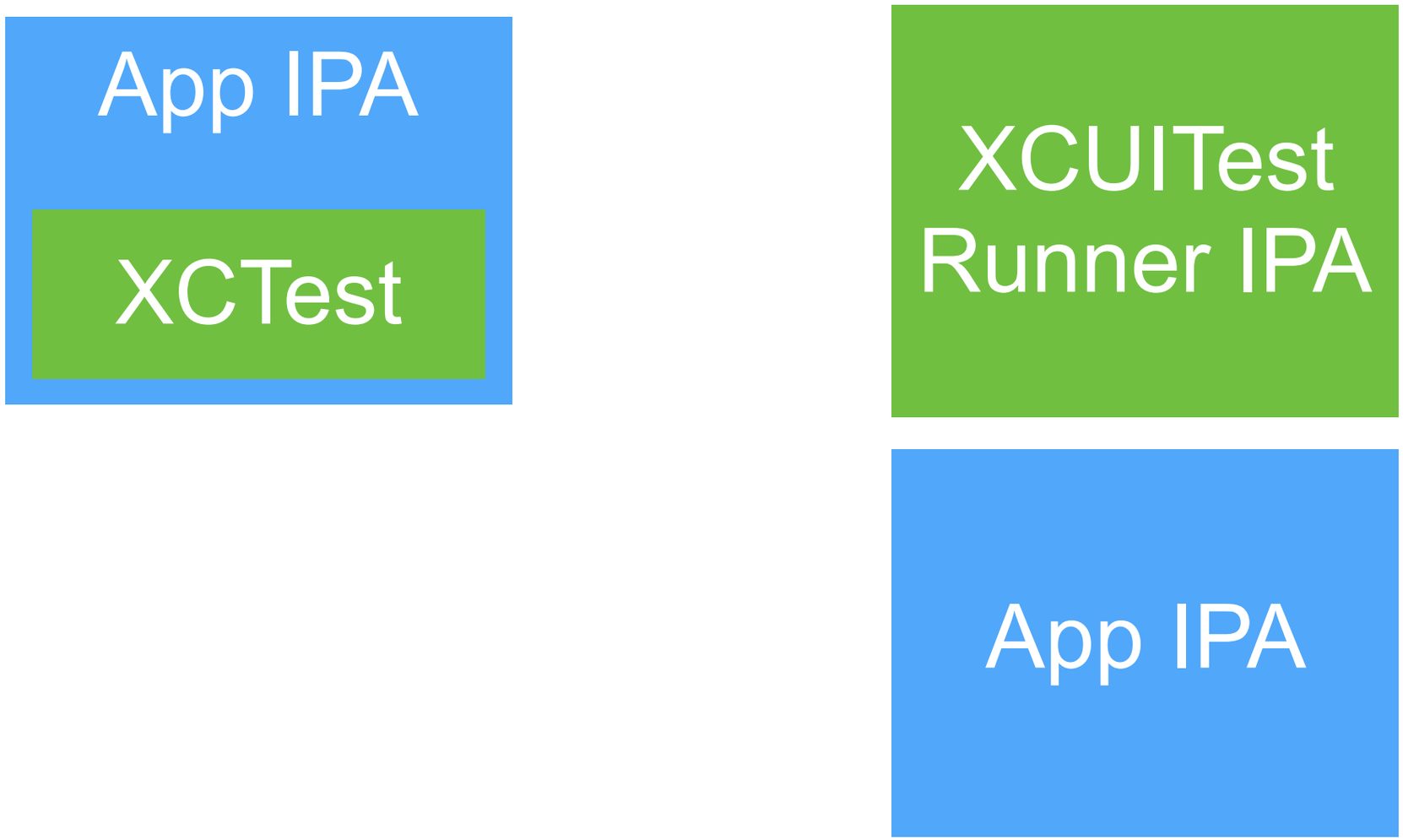
Provides the API to perform UI actions
Tap, Swipe, Key press and etc.

XCUITest allows UI testing

Find elements on UI interface
Perform UI actions on elements
Validate UI properties

XCTest vs XCUITest

XCTest bundle in App IPA file
XCUITest bundle in UI Test Runner



Let’ start to write tests

Create XCUITest

```
class HelloUITests: XCTestCase {

    override func setUp() {
        // In UI tests it is usually best to stop immediately when
            a failure occurs.
        continueAfterFailure = false

        // UI tests must launch the application that they test.
            Doing this in setup will make sure it happens for each
            test method.
        XCUIApplication().launch()
    }

    override func tearDown() {
        // Put teardown code here. This method is called after the
            invocation of each test method in the class.
    }

    func testExample() {
        // Use recording to get started writing UI tests.
        // Use XCTAssert and related functions to verify your tests
            produce the correct results.
    }

}
```

1 Initial state and start UI test

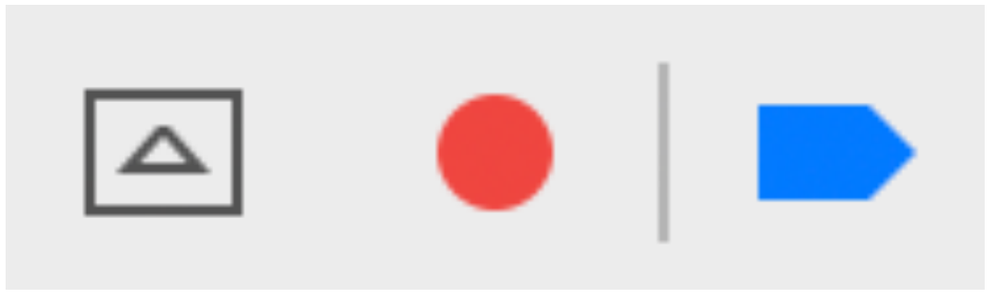
Create XCUITest

```
class HelloUITests: XCTestCase {

    override func setUp() {
        // In UI tests it is usually best to stop immediately when
            a failure occurs.
        continueAfterFailure = false

        // UI tests must launch the application that they test.
            Doing this in setup will make sure it happens for each
            test method.
        XCUIApplication().launch()
    }

    override func tearDown() {
        // Put teardown code here. This method is called after the
            invocation of each test method in the class.
    }

    func testExample() {
        // Use recording to get started writing UI tests.
        // Use XCTAssert and related functions to verify your tests
            produce the correct results.
    }

}
```

2 Write your test

Create XCUITest

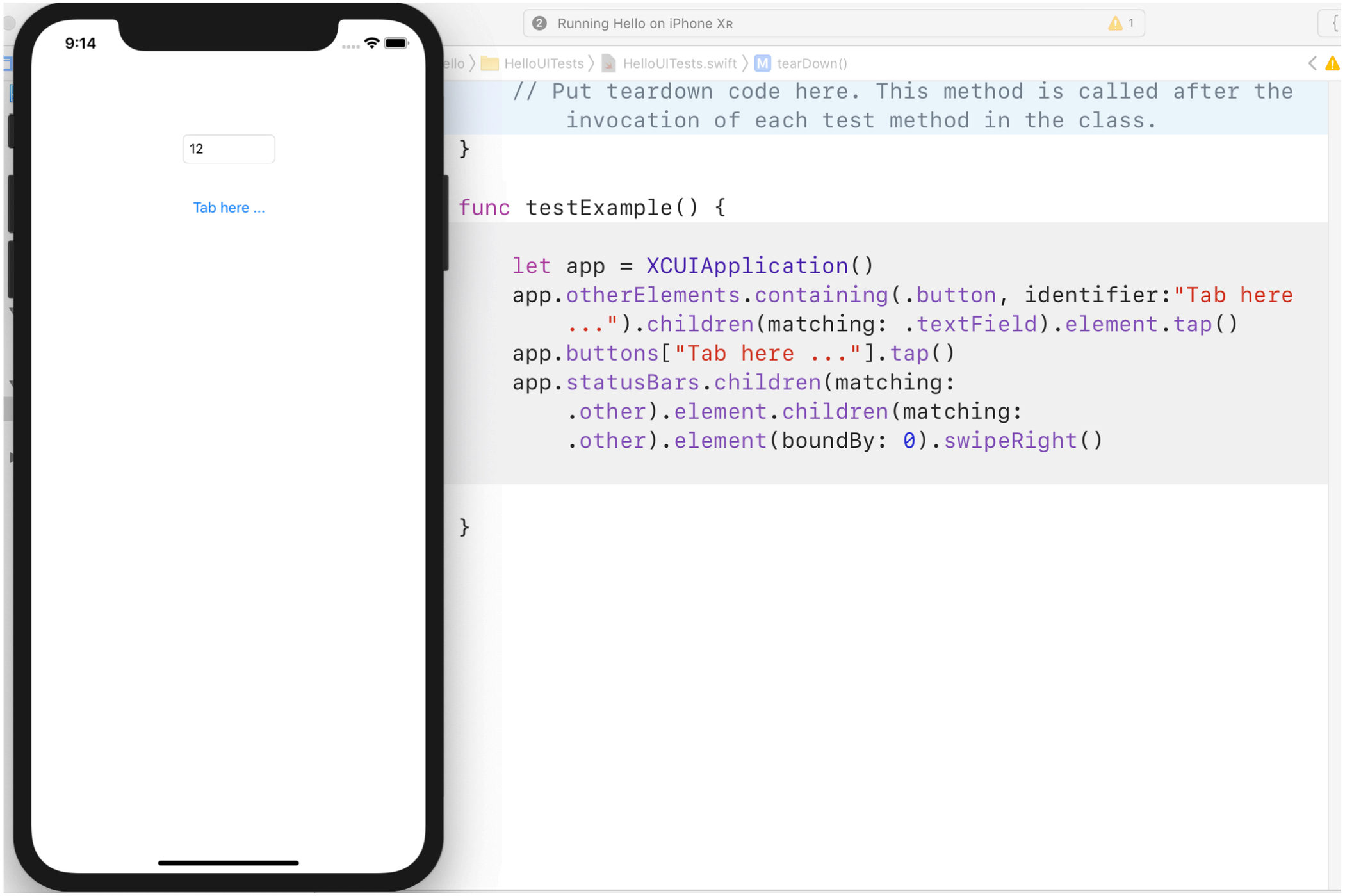
Write by yourself or use UI Test Recorder



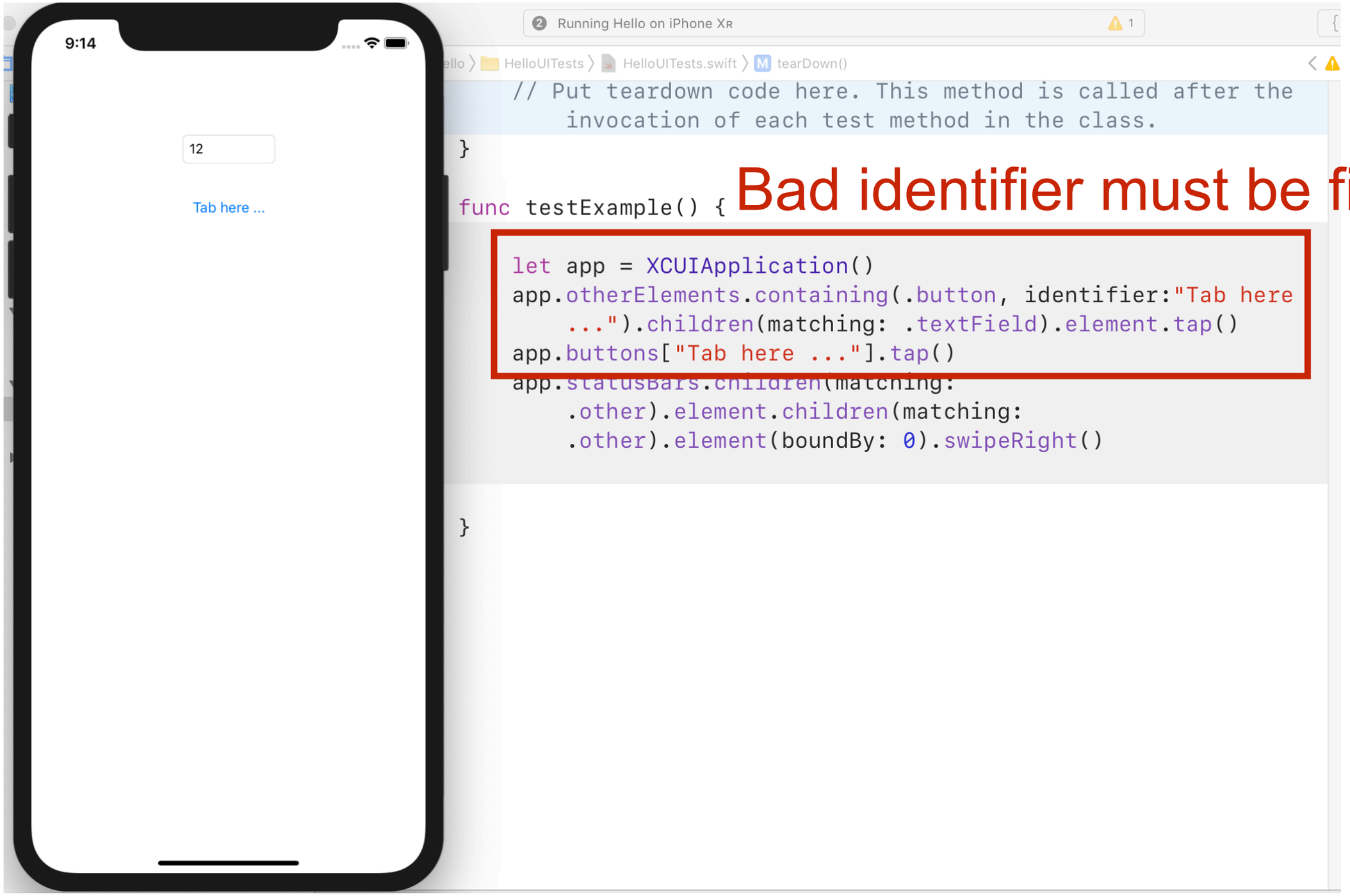
Record UI Test

Create test function with **test** prefix
Tab record button and exercise UI
Tap again to stop
Can record additional actions later
Run your test

Record UI Test



Record UI Test



Accessing UI elements

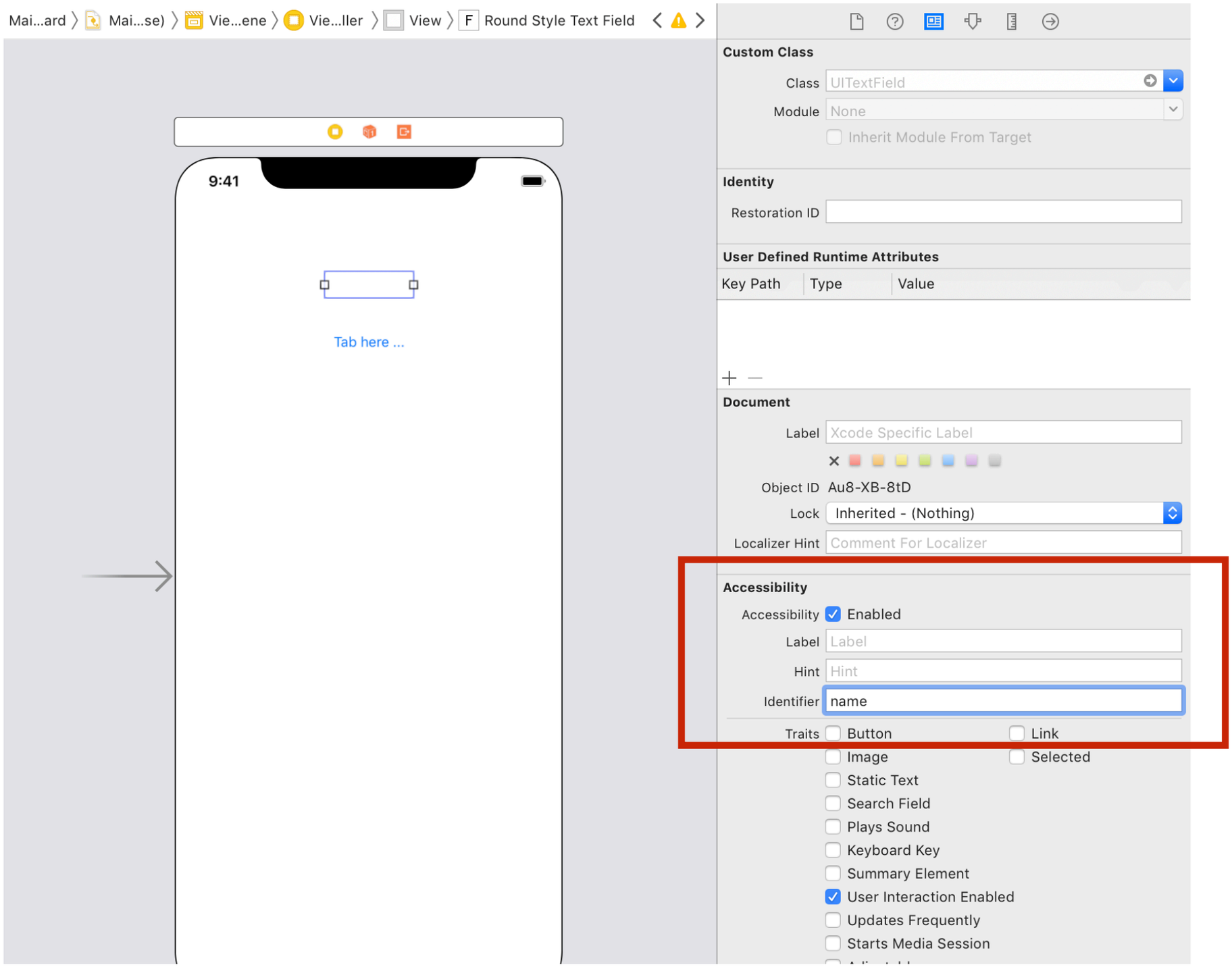
XCUIApplication to launch tests

XCUIApplication use to access all elements

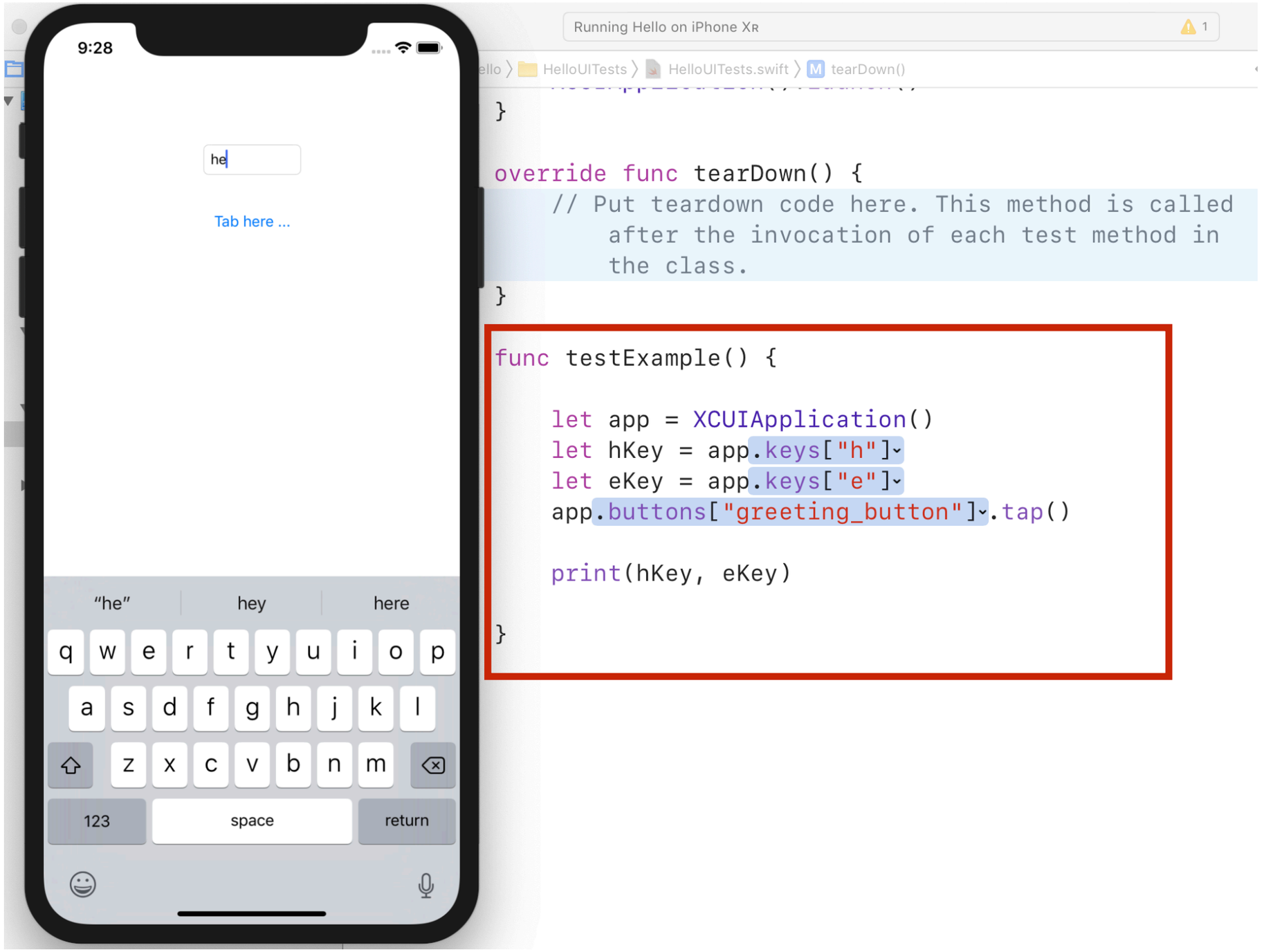
Access with text and indices that can be change !!

Test fail !!

Improve identifier of elements



Record again !!



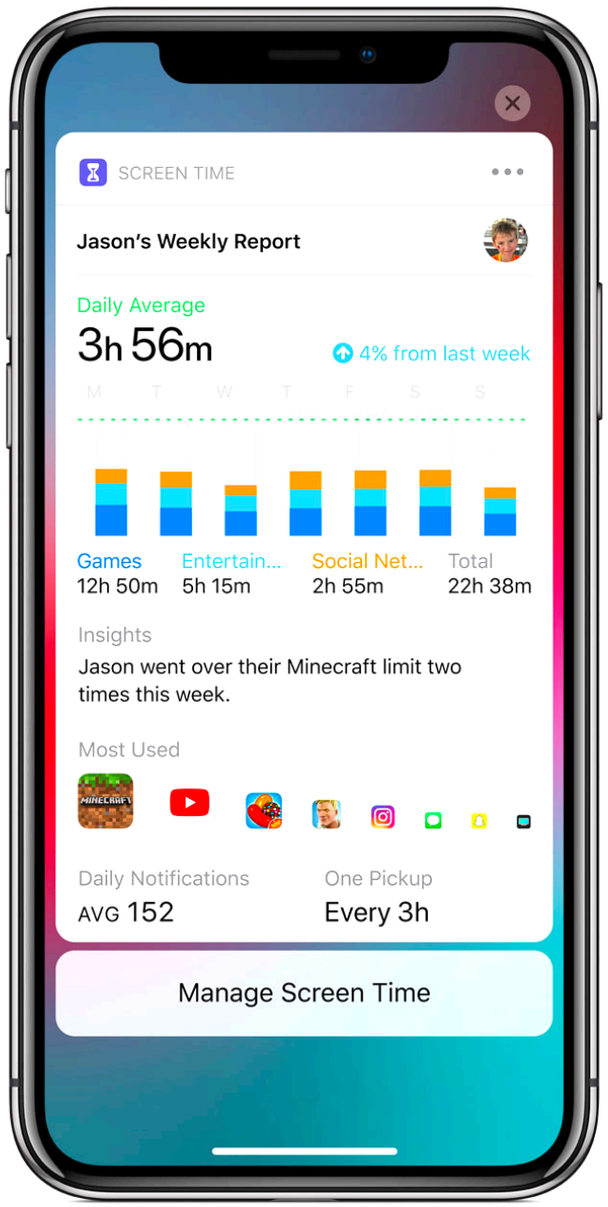
Don’t forget to verify !!

Using XCTAssertion
Working with **Asynchronous process**

Workshop

Flow 1 :: Login Success

Login Page

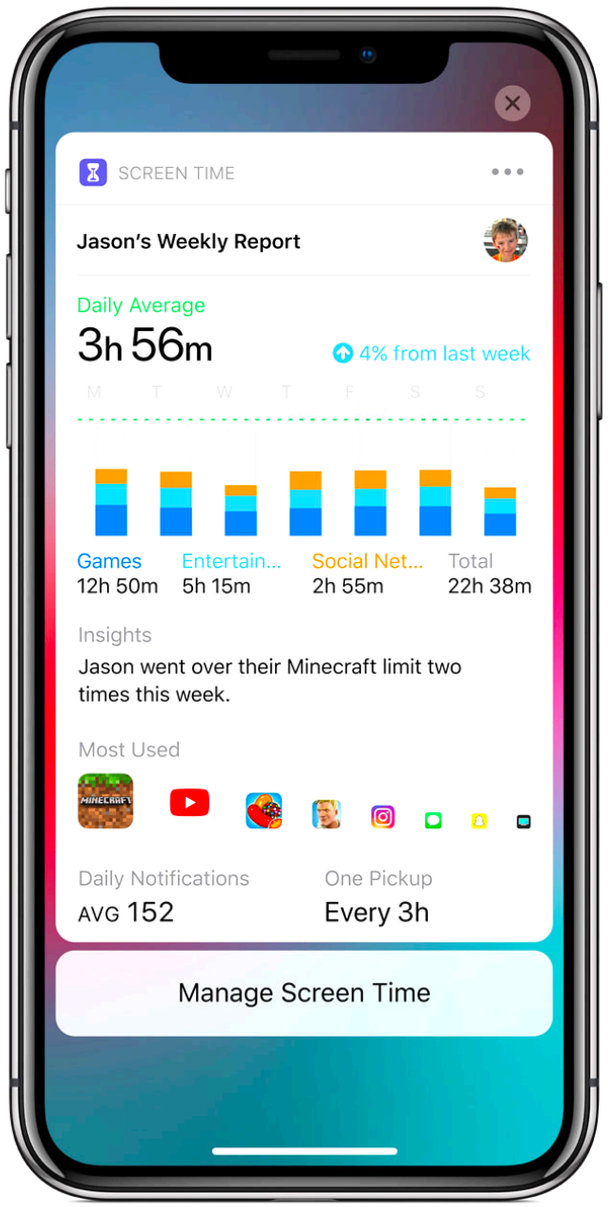


Result Page

Success

Define ID in each element

Login Page

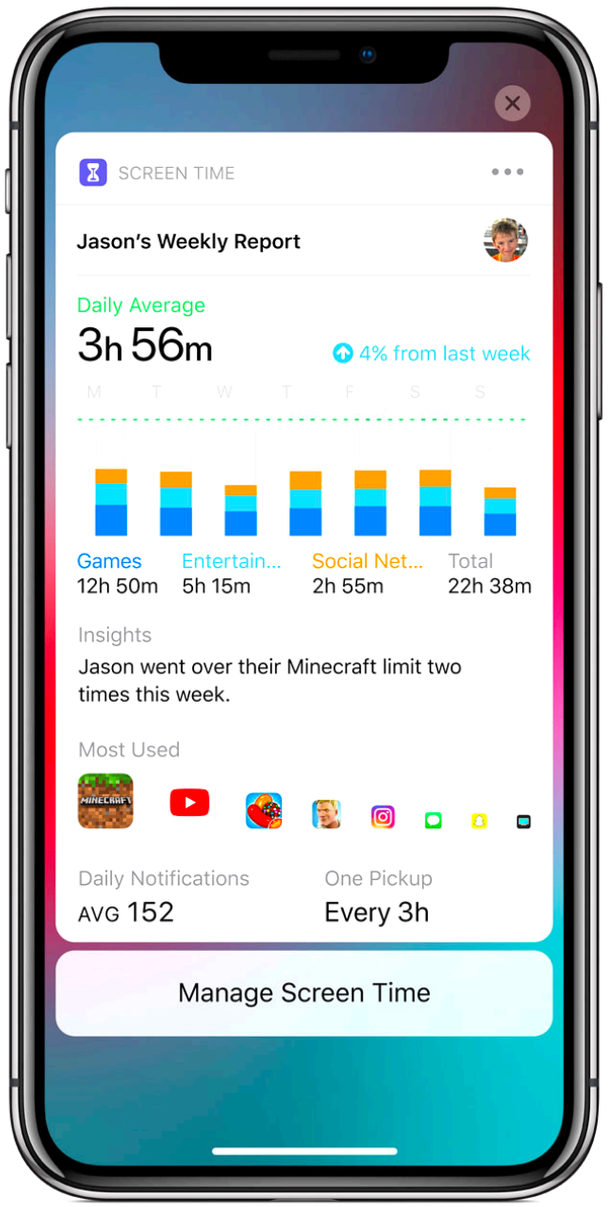


Result Page

Success

Flow 2 :: Login Fail

Login Page

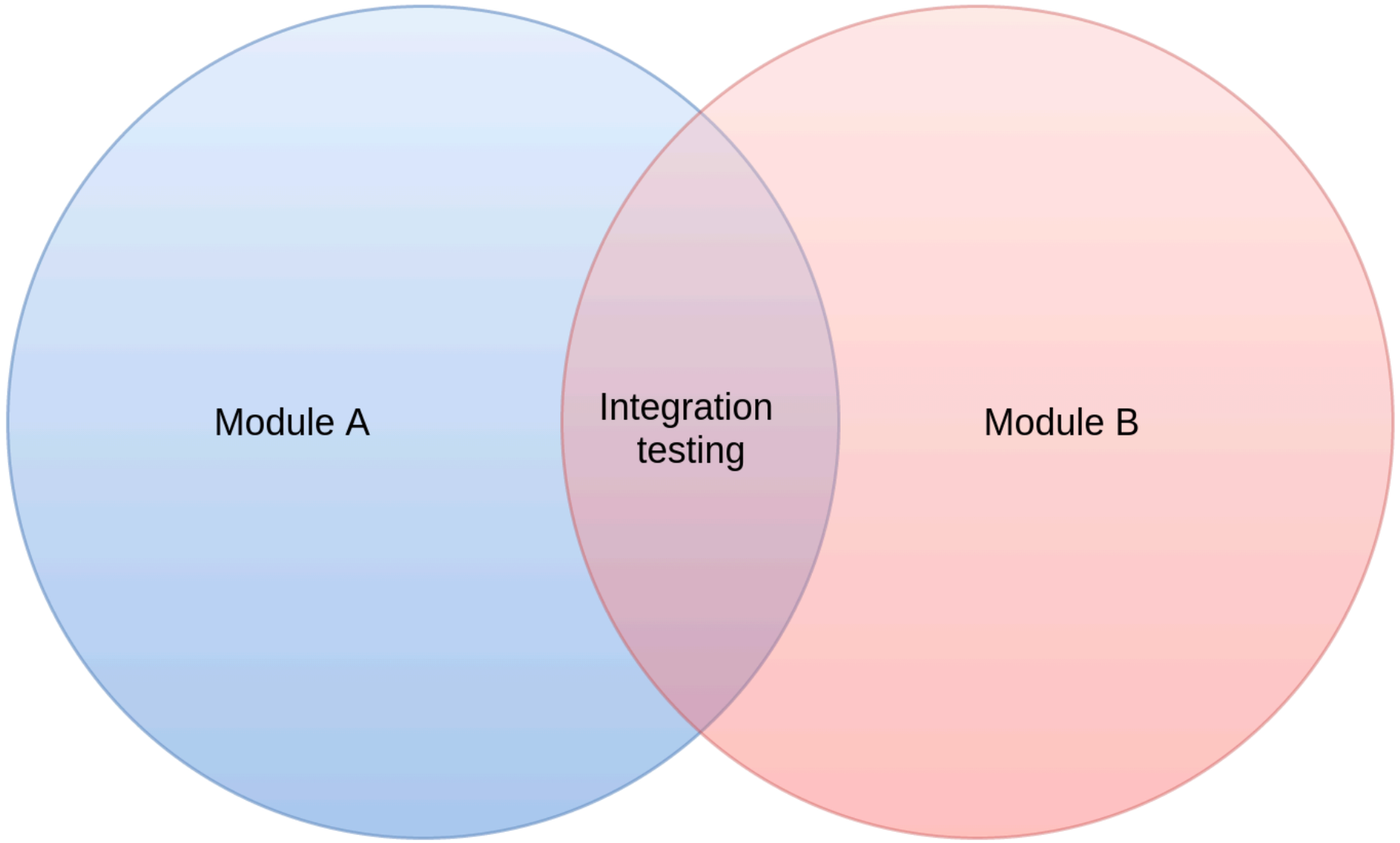


Result Page

Failure

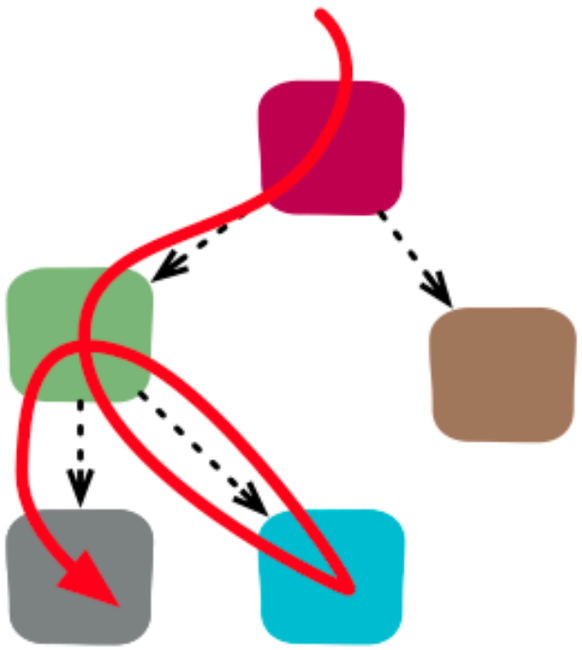
Integration testing !!

Integration testing

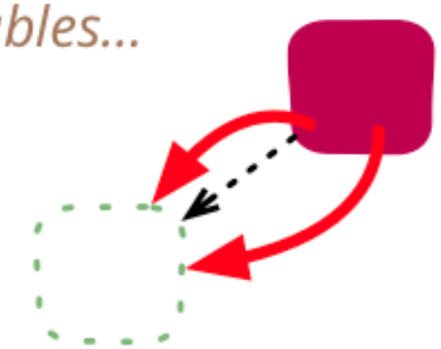


Integration testing !!

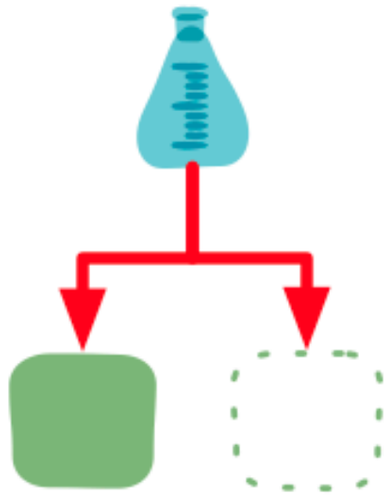
Integration Testing commonly refers to broad tests done with many modules active...



...but it can be done with narrow tests of interactions with individual Test Doubles...

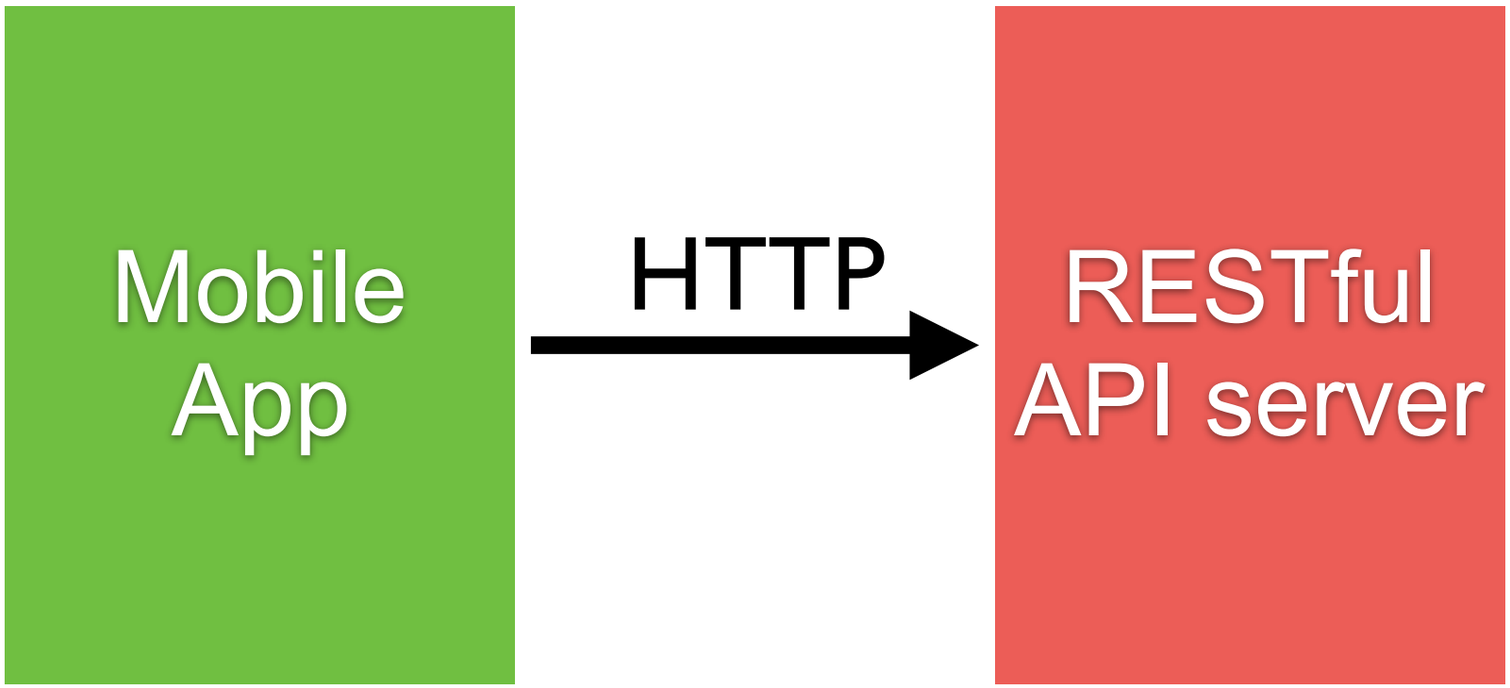


...supported by Contract Tests to ensure the faithfulness of the double



https://martinfowler.com/bliki/IntegrationTest.html

How to testing ?

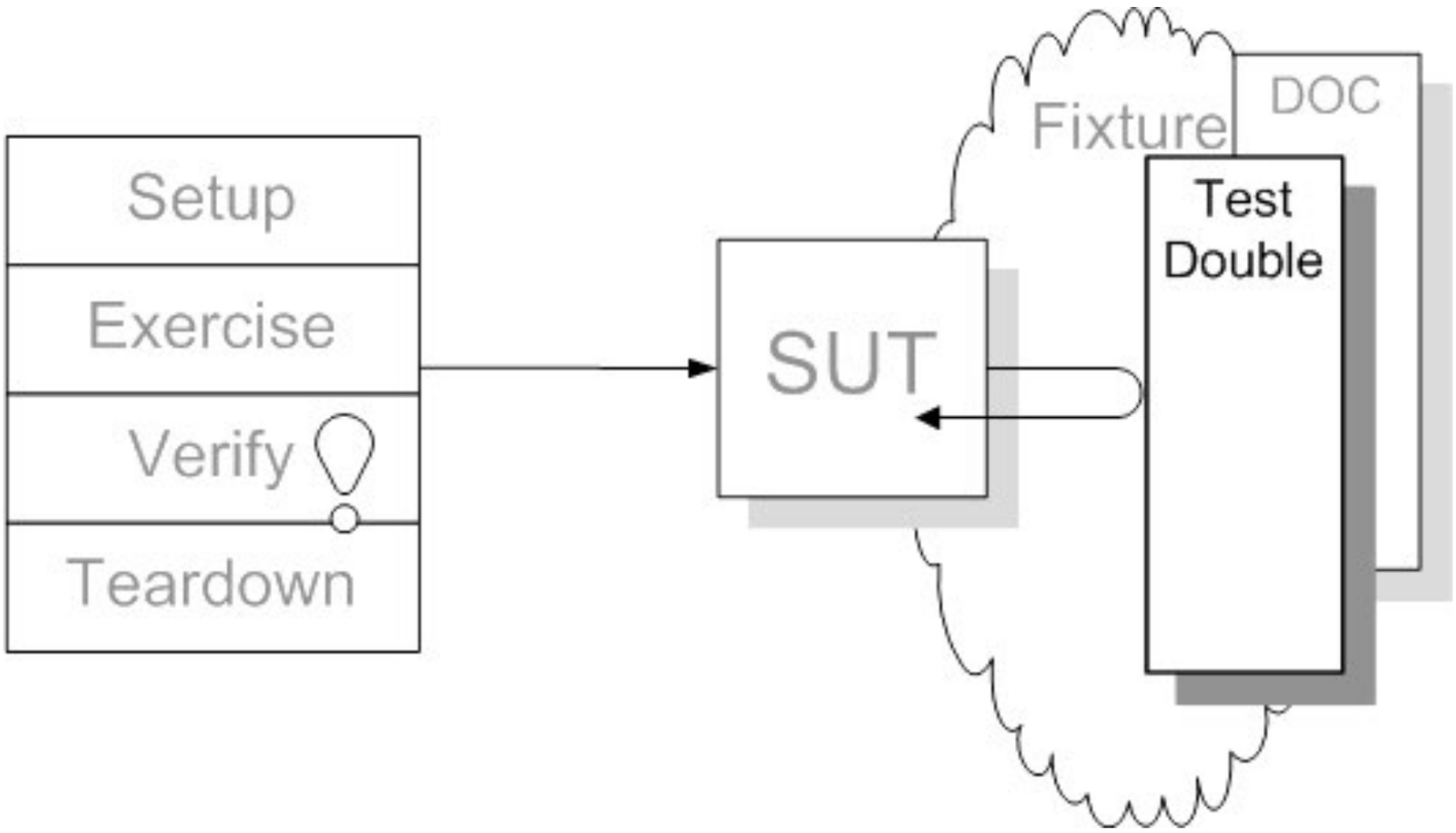


How to testing with network ?

Mock API Server (Internal or External)
Configurable endpoint of API
Using Test double
No need to change code

Unit test VS. UI test

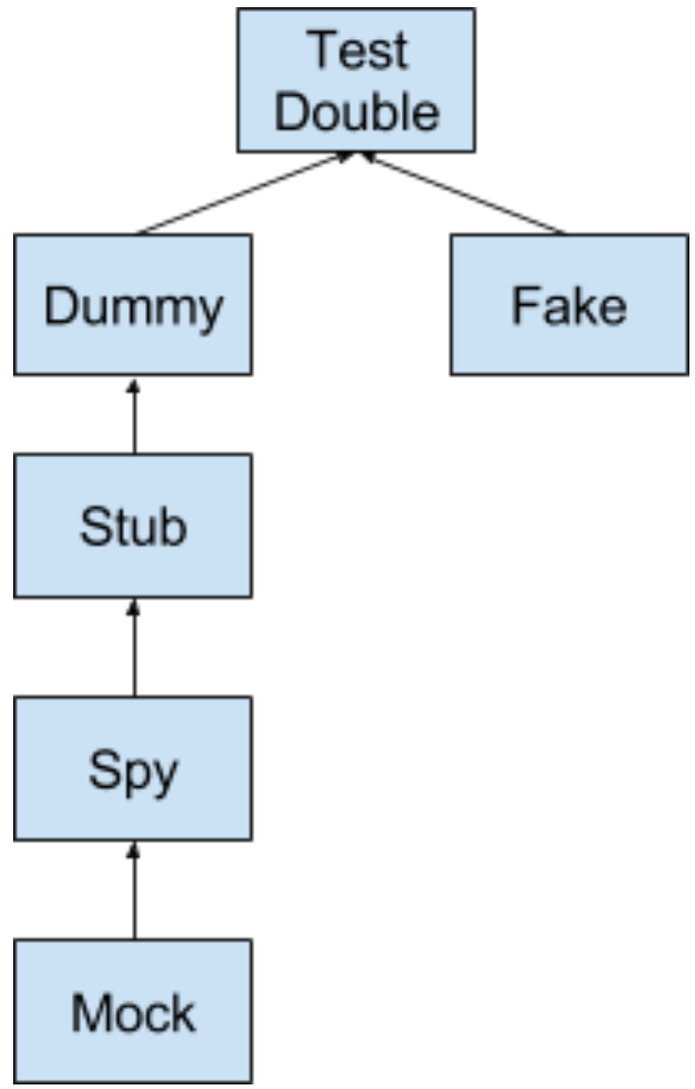
Test double



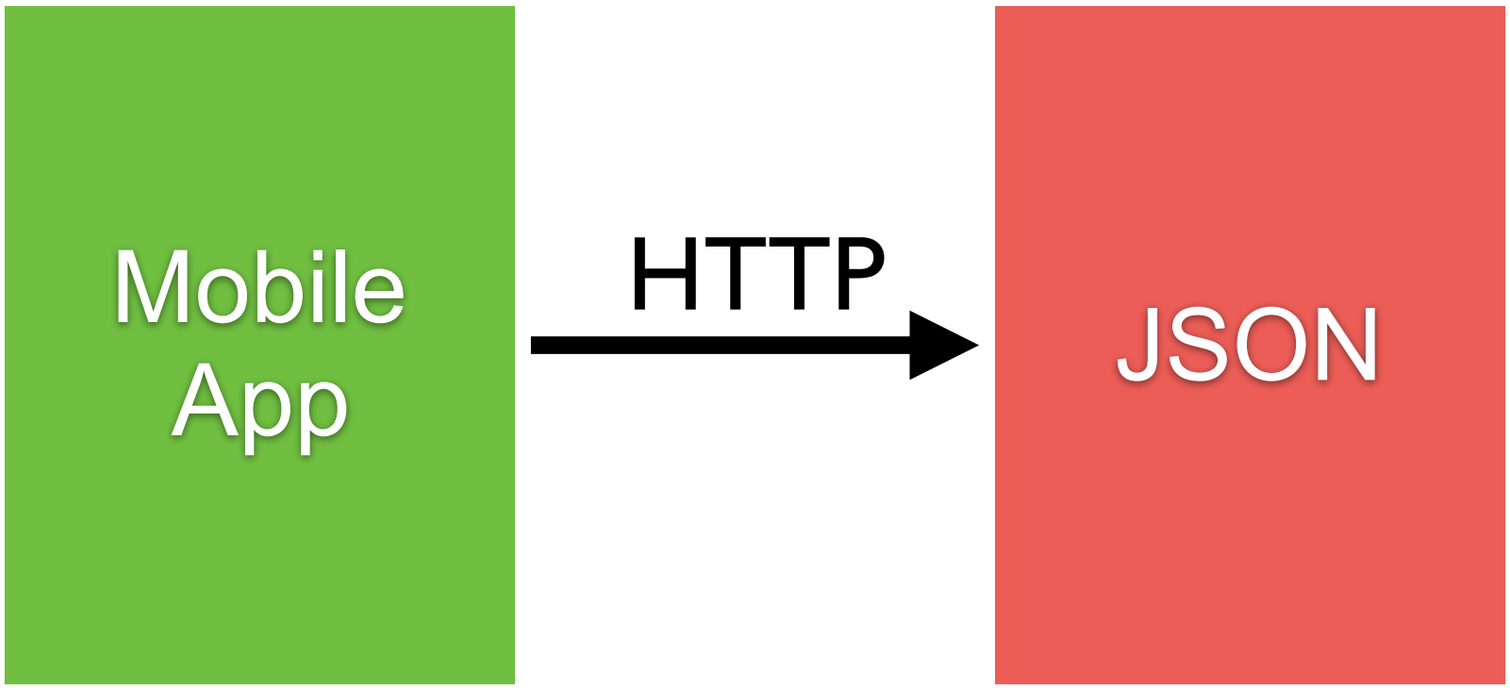
http://xunitpatterns.com/Test%20Double.html

Test double

Dummy
Stub
Spy
Mock
Fake



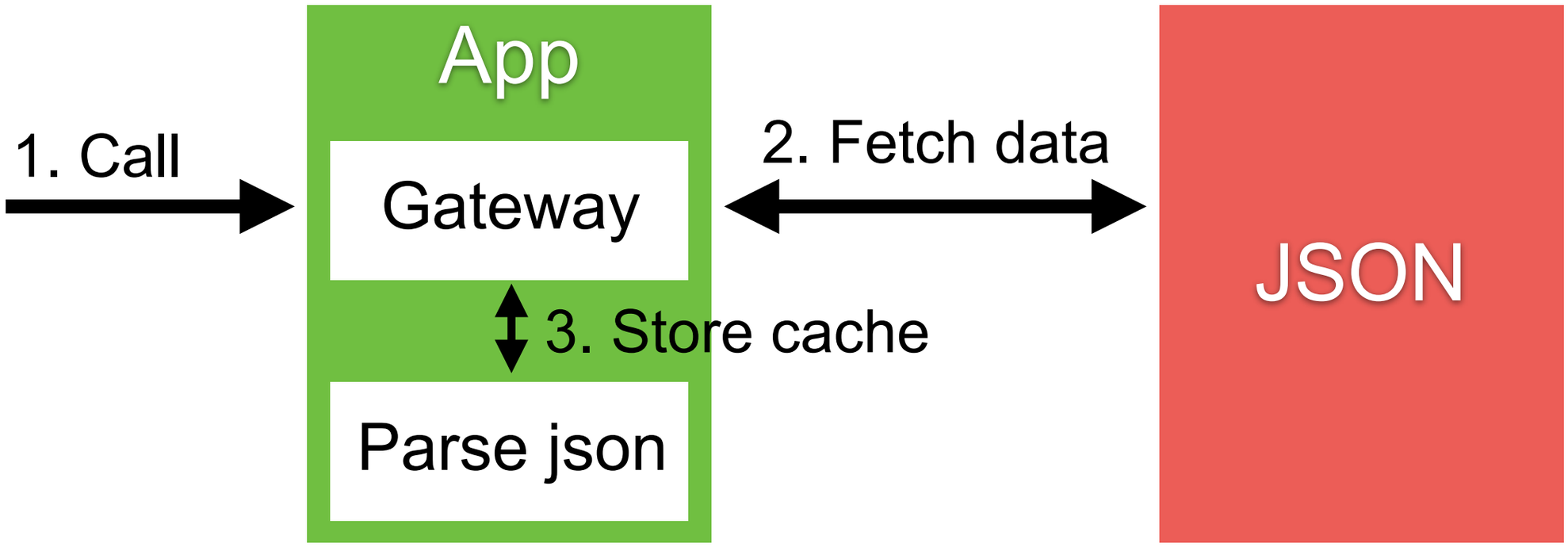
Example



Process to fetch data

1. Refresh calls fetch
2. Fetches and call parse
3. Parse stores cache
4. Refresh stores date

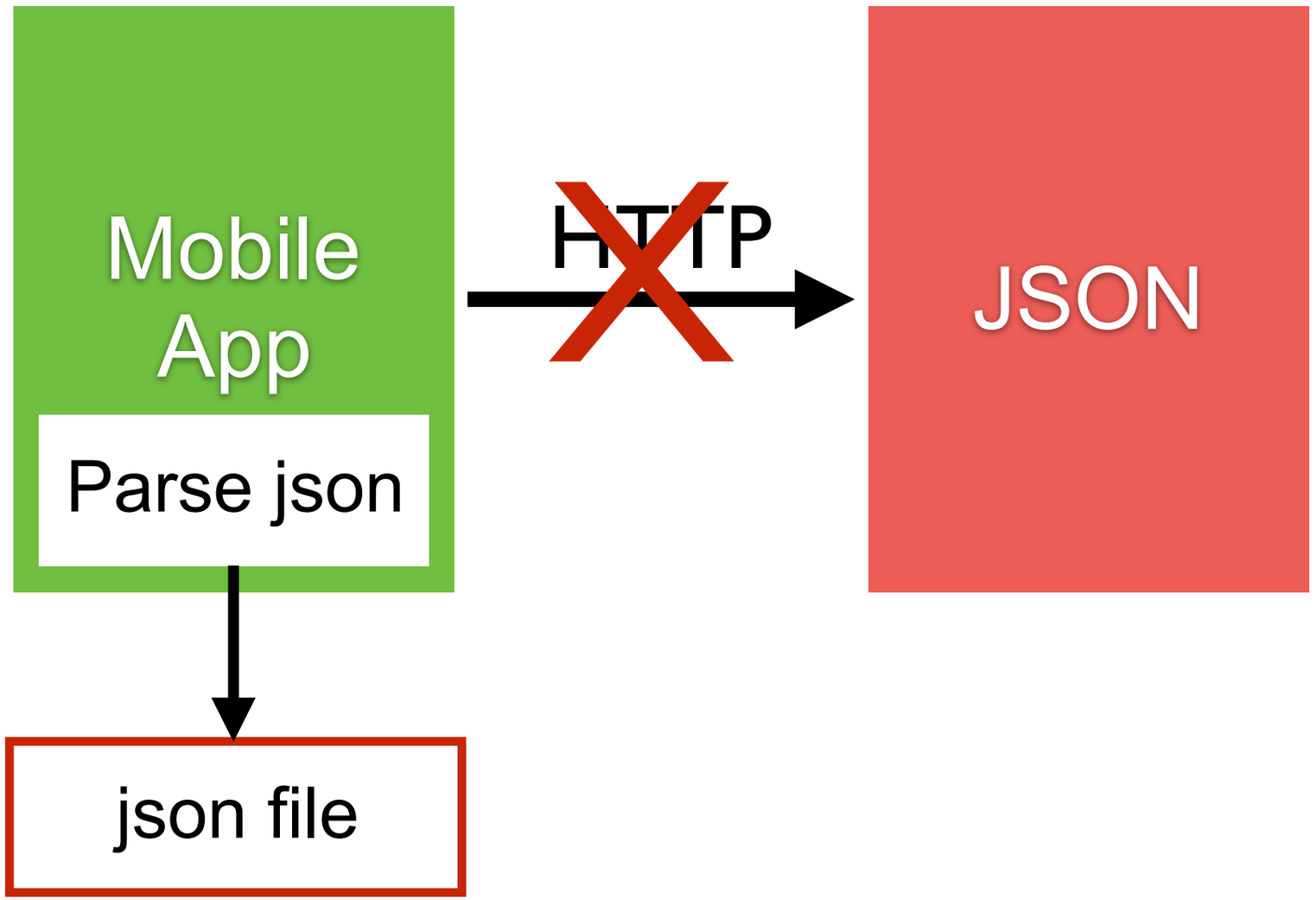
Example



How to testing ?

Test JSON data (success and failure)
Test process to call API with mock class
Test process to call API with real class

1. Mock JSON data



Mock JSON data

Try to parse JSON data only

```
func testParseJSON() {
    let music = Music()
    let bundle = Bundle(for: type(of: self))
    if let path = bundle.path(forResource: "JSON", ofType: "txt") {
        if let data = try? Data.init(contentsOf: URL.init(fileURLWithPath: path)) {
            let result = music.parseJSON(data: data)
            XCTAssertNotNil(result!, "should not be nil")
            XCTAssertGreaterThan(result!.count, 0, "should have values")
        }
        else {
            XCTFail()
        }
    }
    else {
        XCTFail()
    }
}
```

2. Mock class

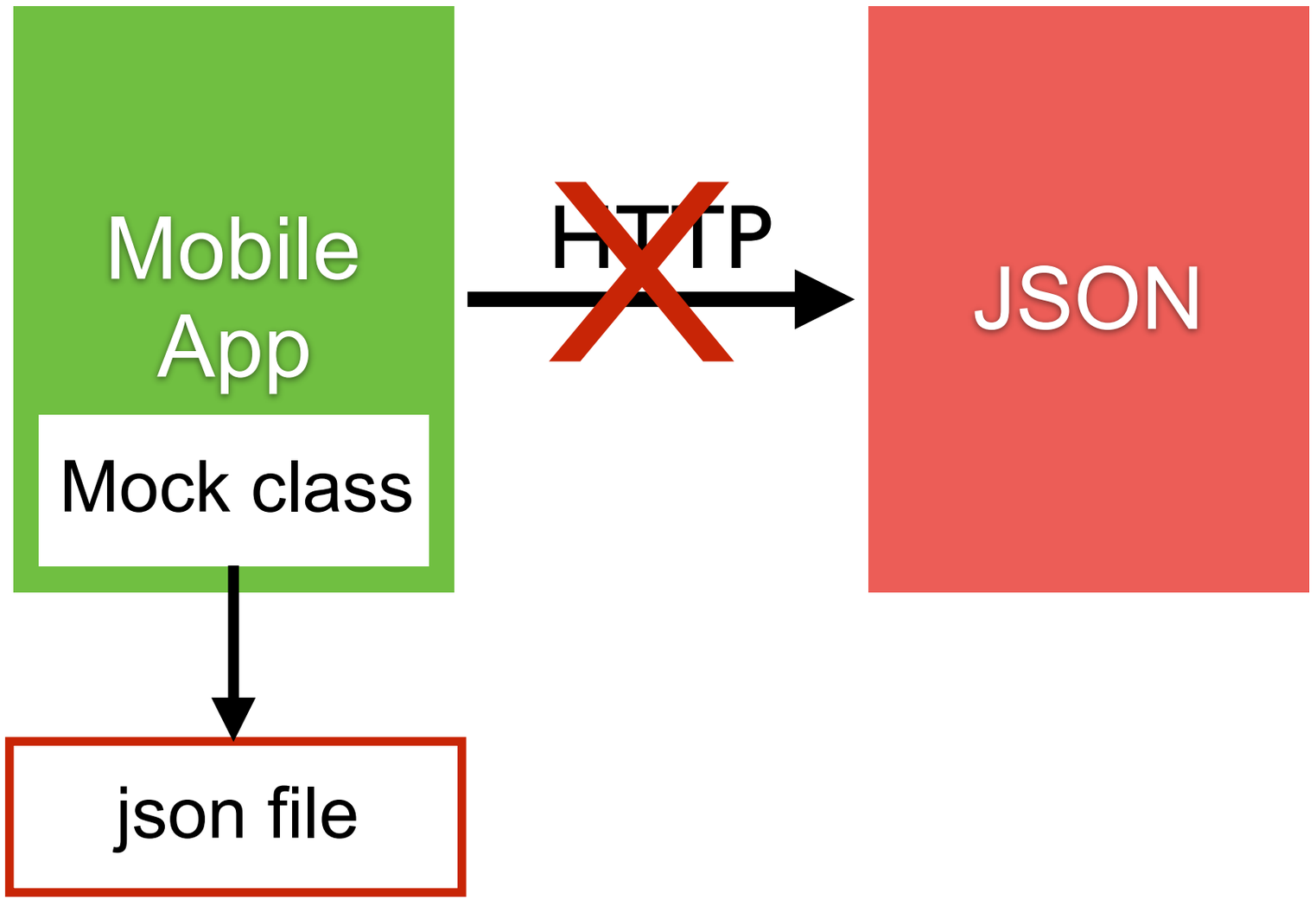
Design to mock what should be mocked
Use another functions
Avoid unwanted functionality
Benefit from base class implementation

Process to fetch data

1. Refresh calls fetch
2. Fetches and call parse
3. Parse stores cache
4. Refresh stores date

Avoid to fetch data from server/api !!

Mock class



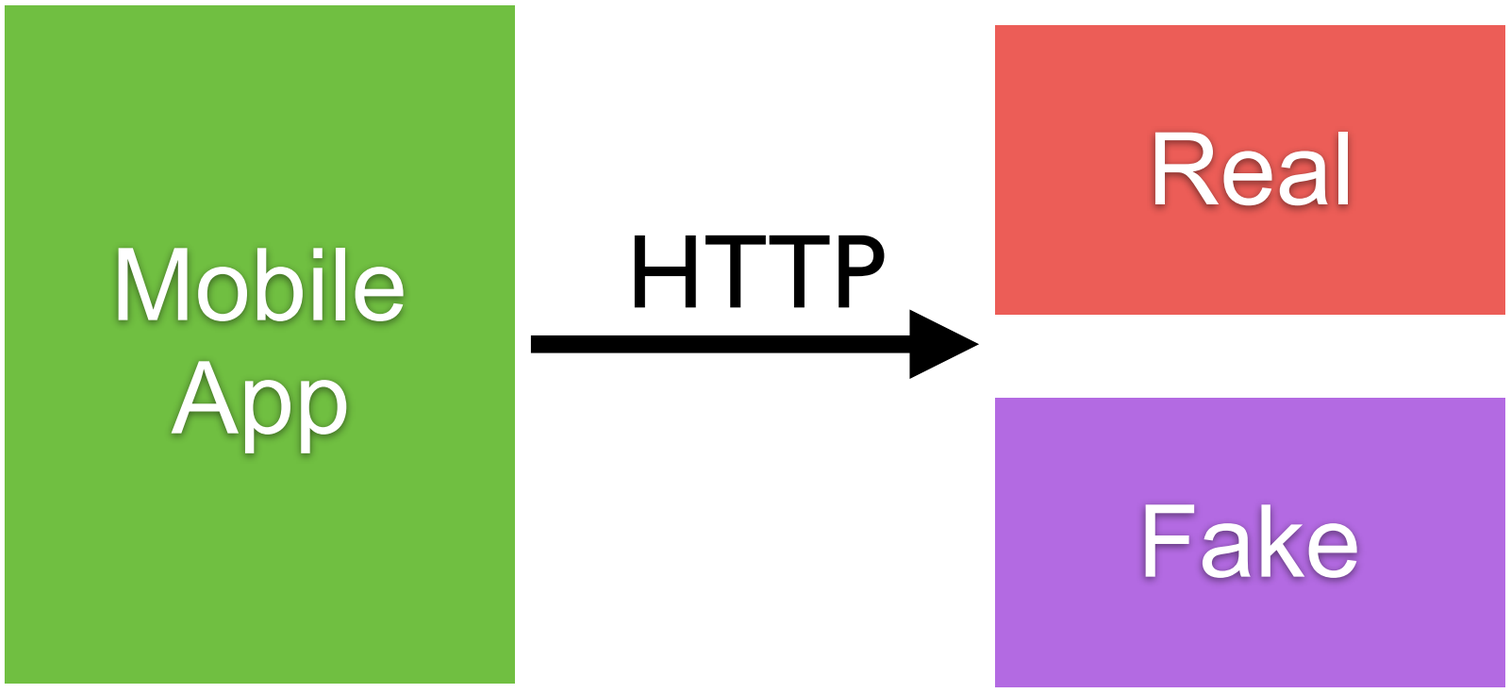
Create Mock class

Load data from file and call parse

```
class MockMusic : Music {
    override func fetchMusic(completion: @escaping ([[String : Any]]?,
        let bundle = Bundle(for: type(of: self))
        if let path = bundle.path(forResource: "JSON", ofType: "txt") {
            if let data = try? Data.init(contentsOf: URL.init(fileURLWi
                let parsedData = self.parseJSON(data: data)
                completion(parsedData, nil)
            }
        }
    }
}
```

3. Real class

Call to real API server
Call to Fake API server



Mock API server

External (Stubby4j, Wiremock)
Internal (OHHTTPStubs)

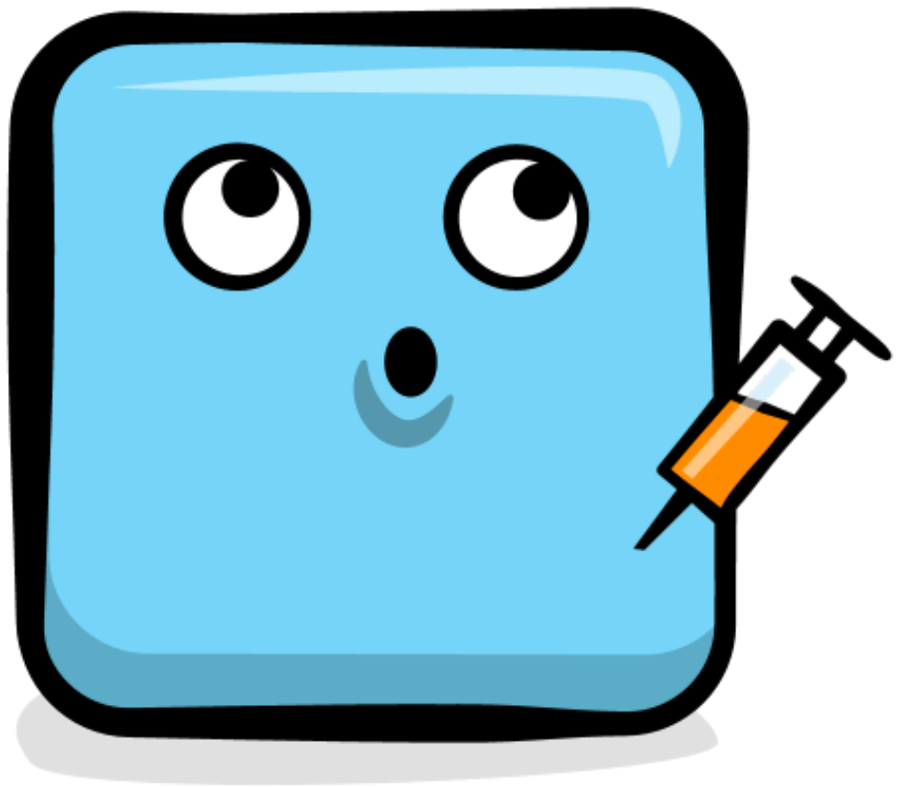
How to change URL of API ?

Constant variable

Build config

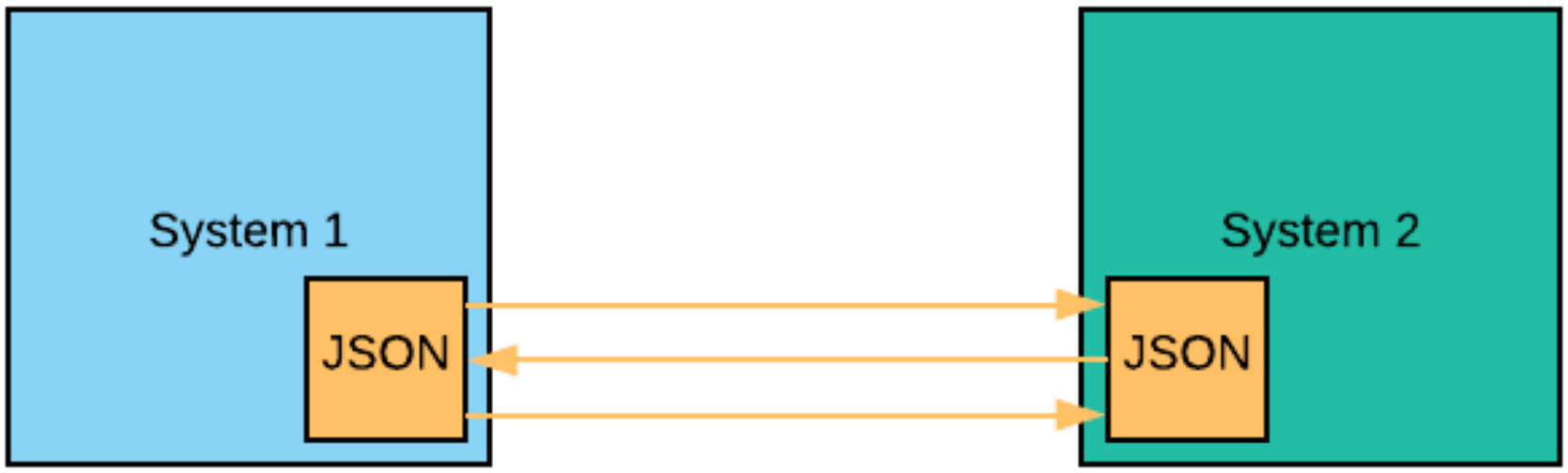
Dependency Injection (DI)

Dependency Injection (DI)



Tight coupling !!

Loose coupling



Dependency Injection (DI)

Technique whereby one object supplies the dependencies of another object

Techniques to inject

Constructor injection
Setter/property injection
Method injection
Interface injection

Benefits of Dependency Injection

Reusability
Maintainability
Scalability
Testability

DI frameworks

Create by your own
Using framework

DI frameworks

Swinject

Cleanse

Demo and Workshop

Asynchronous testing

When the real callback is part of it
Control can be returned too soon
Set expectation (create, fulfil and waiting)

Fetch data from network

Set expectation

```
func testFetch() {
    let exp = expectation(description: "server fetch")

    let music = Music()

    music.fetchMusic { (items, error) in
        XCTAssertNotNil(items, "items should not be nil")
        XCTAssertTrue(items!.count > 0, "items should not be
            empty")
        exp.fulfill()
    }

    waitForExpectations(timeout: 0.000001) { (error) in
        print (error?.localizedDescription)
    }
}
```

Waiting for expectation

Workshop

Testing more …

UserDefaults
Core Data
Singletons

Testable code structure

We need good structure ?

Cost of maintain

Cost of change

Time to market

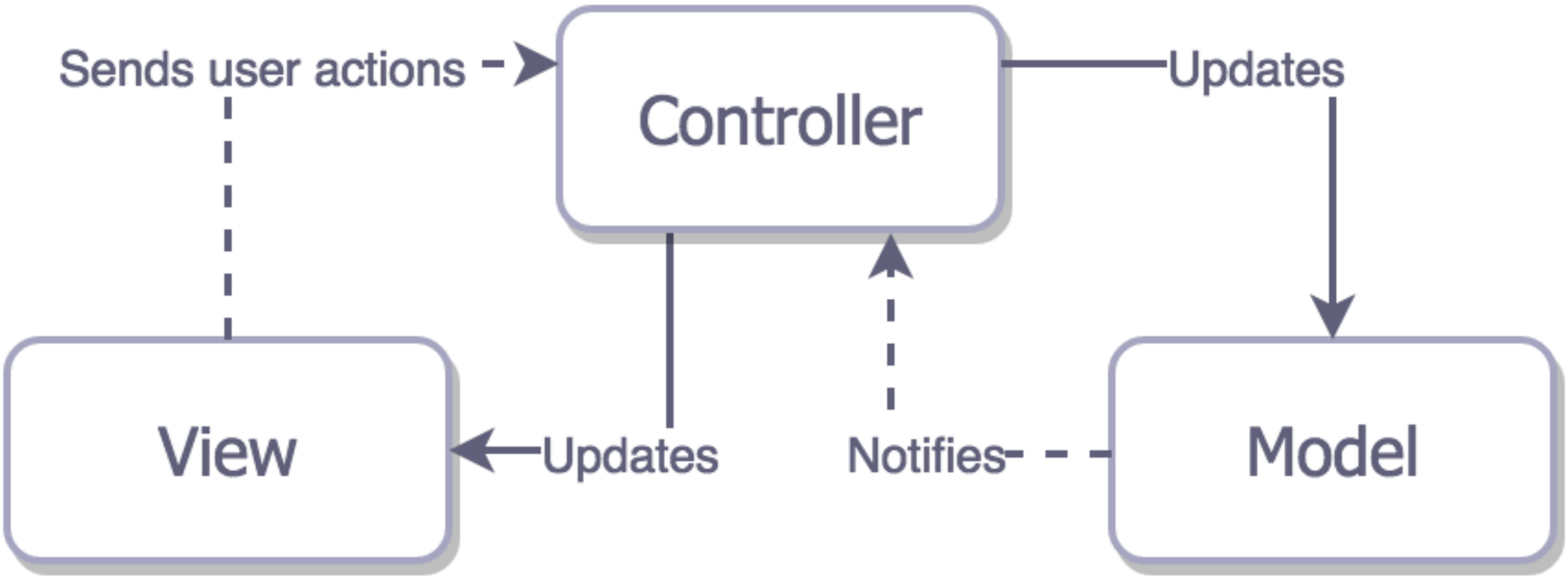
Features of good structure

Balance distribution of responsibilities
Testability
Ease of use and low maintain cost

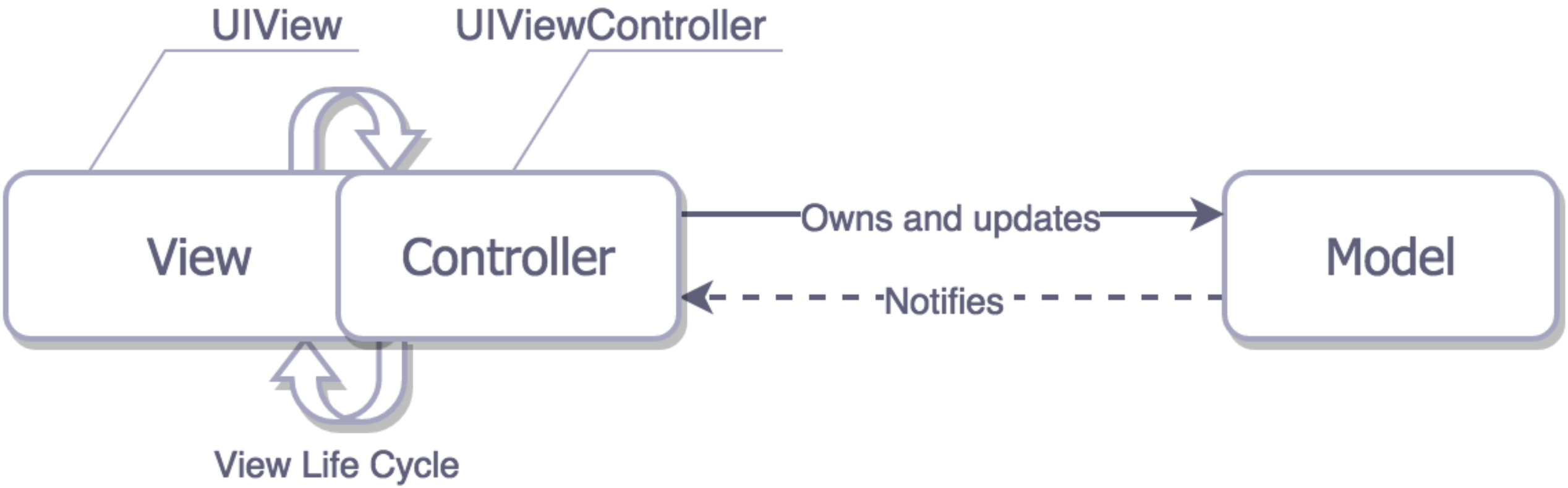
Design patterns

Apple MVC
MVC
MVP
MVVM
VIPER
POP
Clean Architecture
Redux-Like

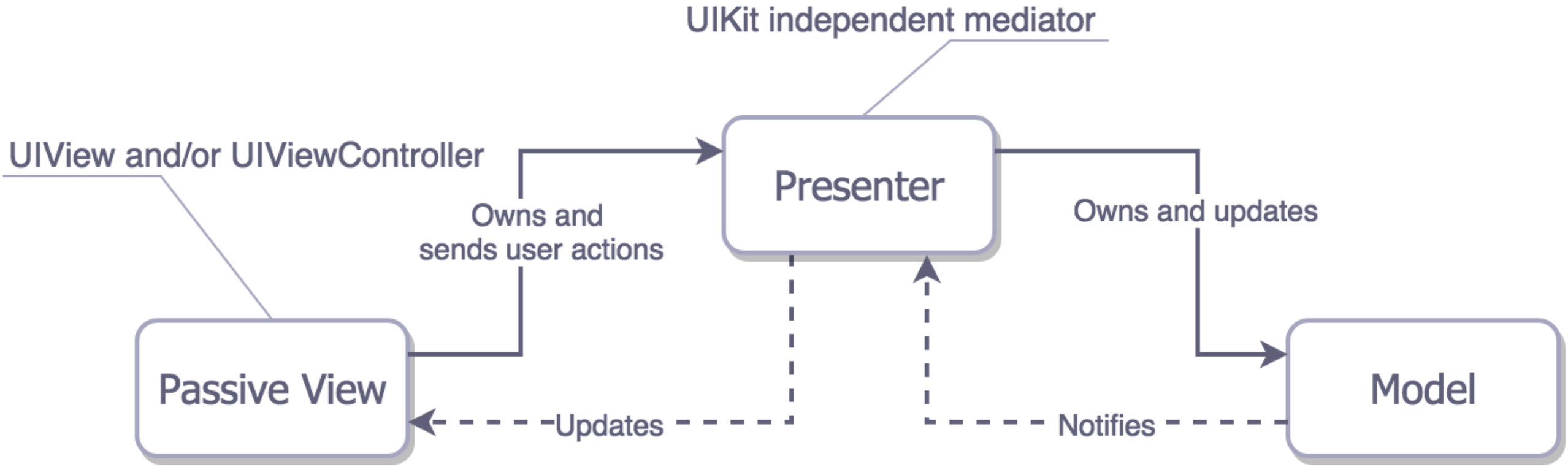
Apple MVC



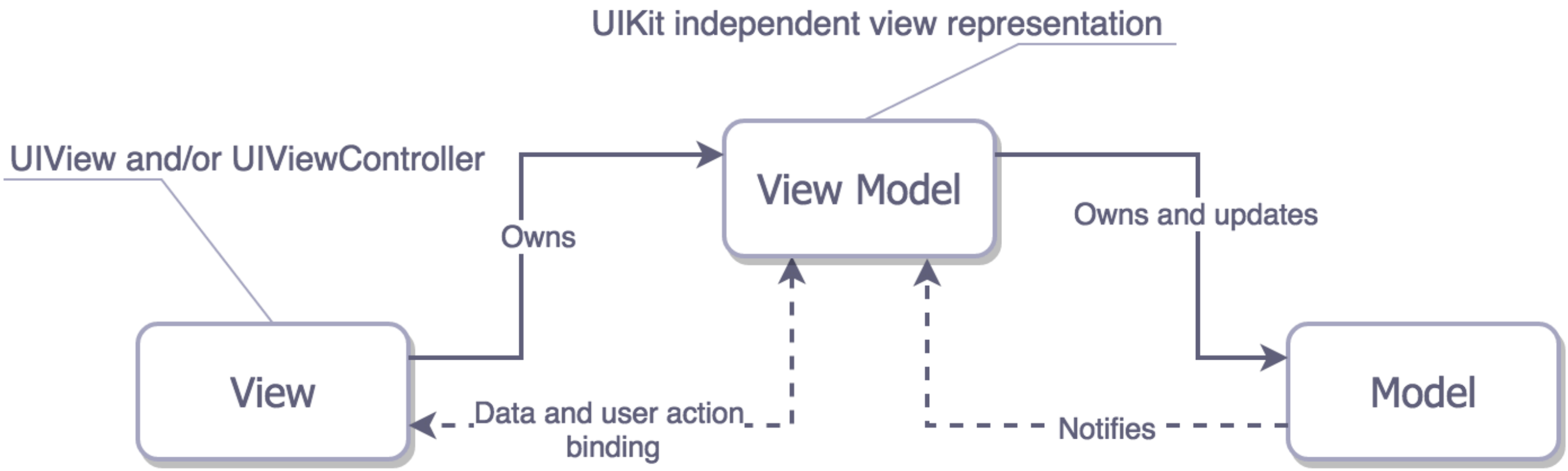
Apple MVC



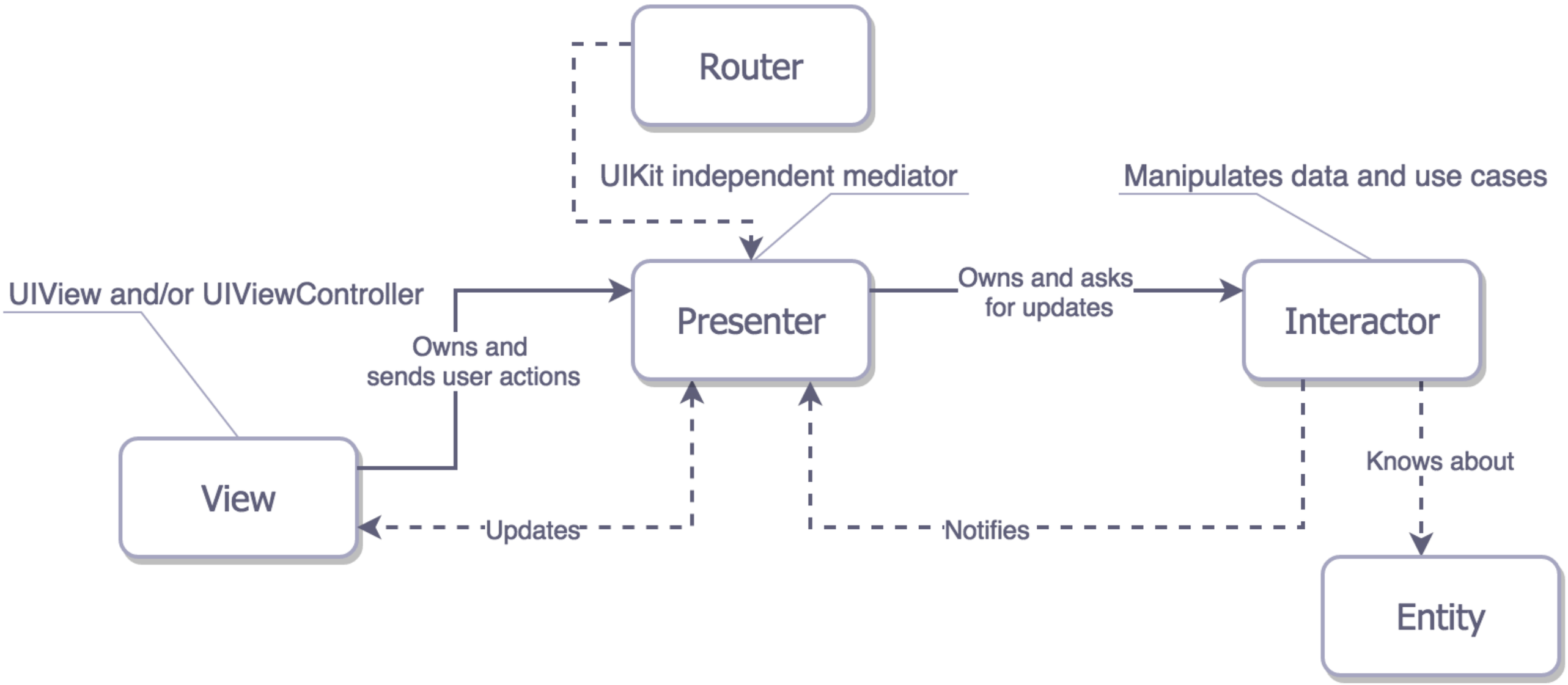
MVP



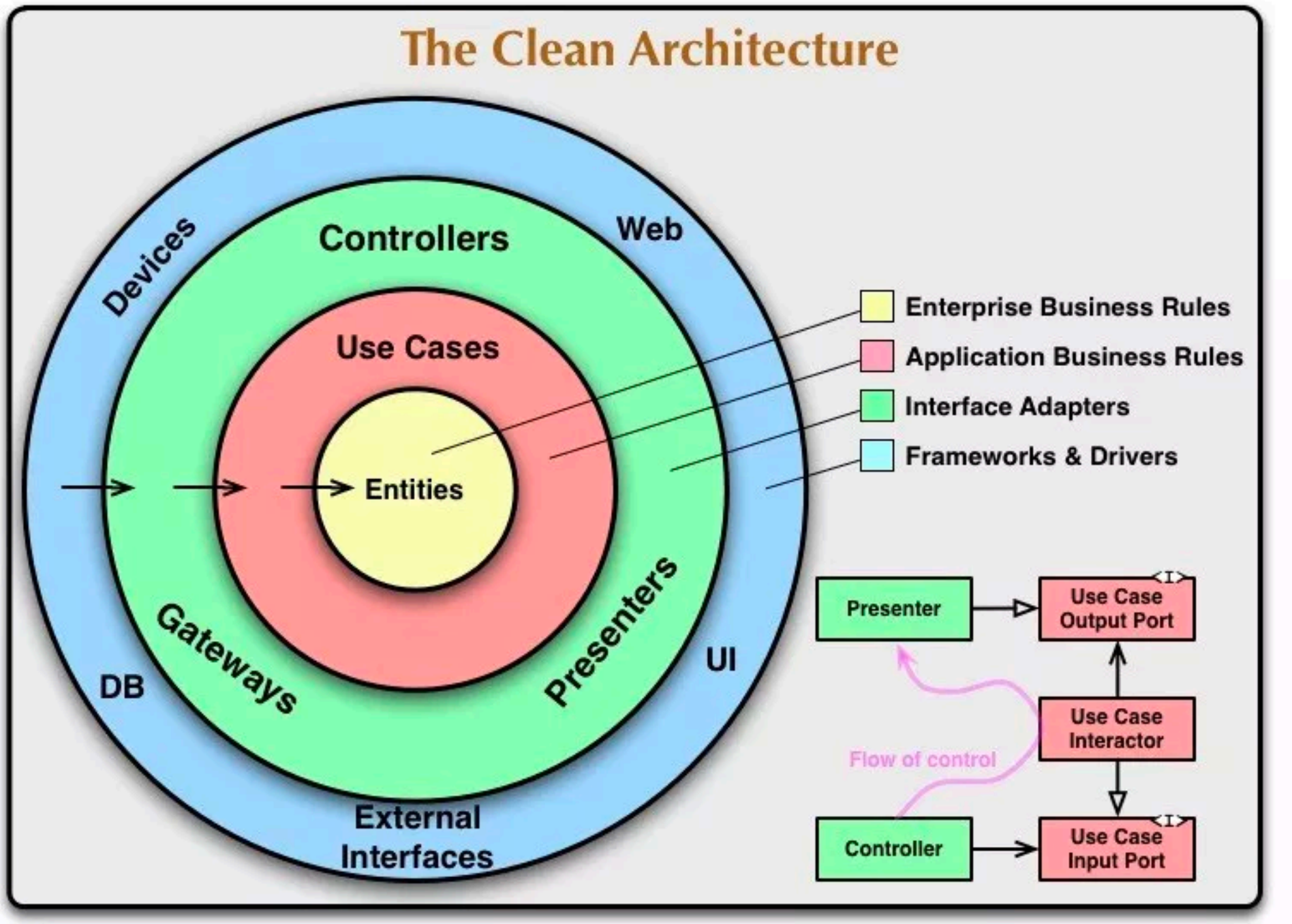
MVVM



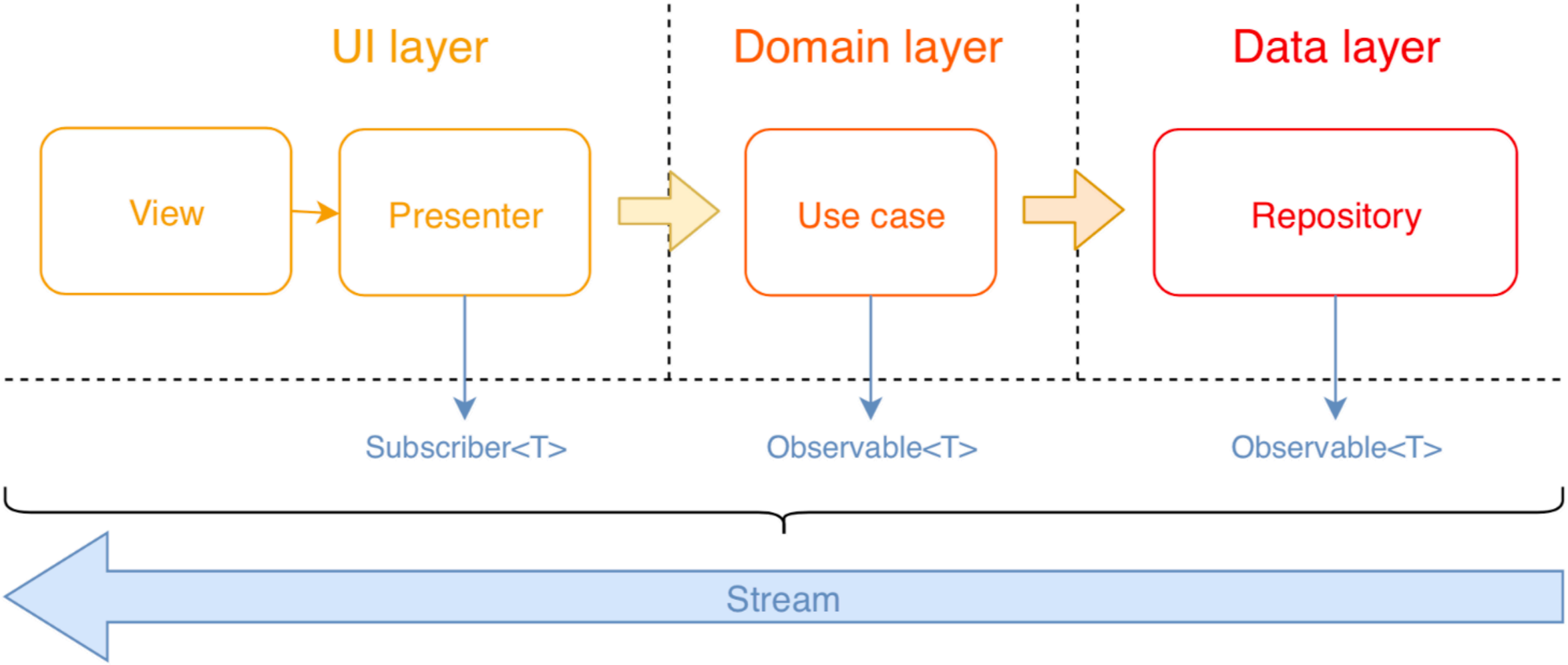
VIPER



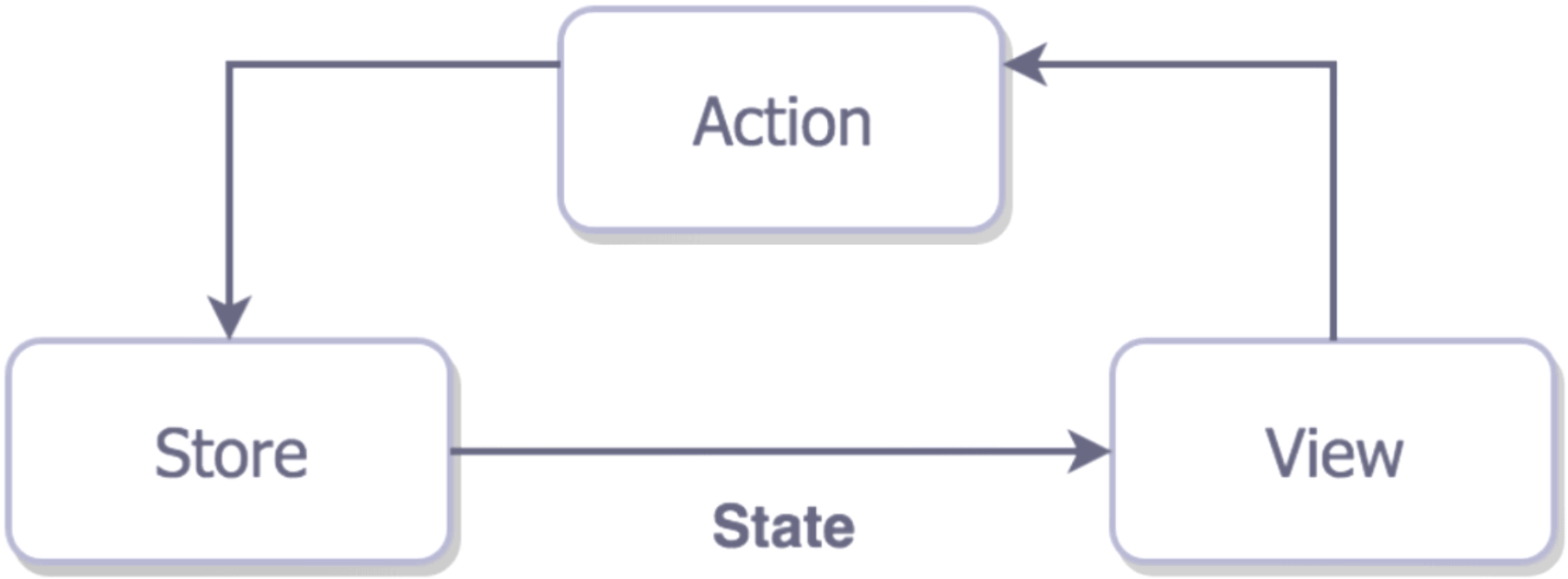
Clean Architecture



Clean Architecture



Redux-Like



No silver bullet

Workshop

More testing types

More testing type and tools

Snapshot testing
Mokey testing
Performance testing
Quick and Nimble

Performance testing

Function name start with **test**

Using **self.messure** closure for your code

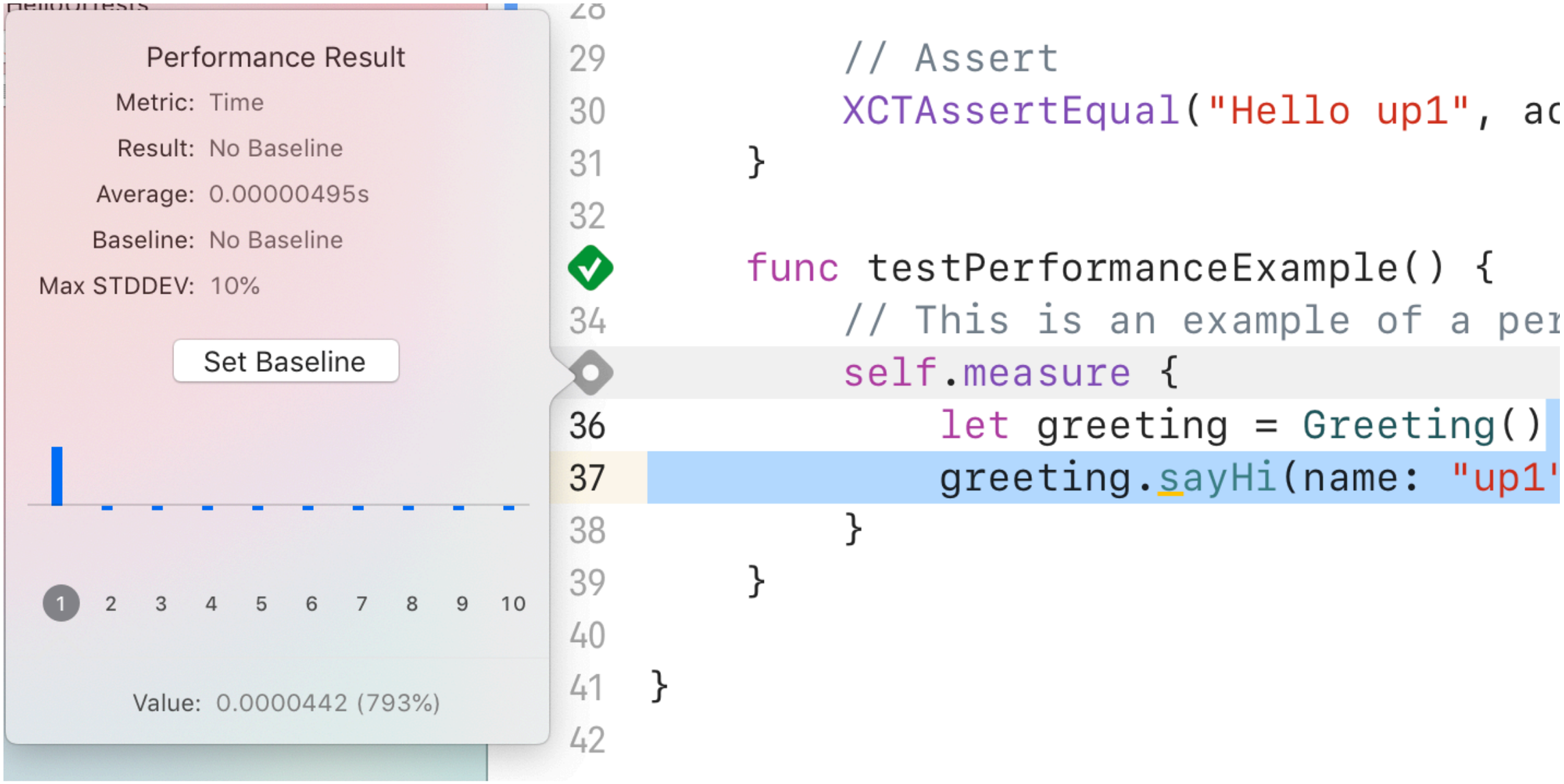
No assertions

Run 10 times

Averaged for baseline

Collecting coverage data may effect performance !!

Run and set Baseline



Example

```
func testMusic() {
    // This is an example of a performance test case.
    self.measure {
        let exp = expectation(description: "server fetch")
        let music = Music()
        music.fetchMusic(completion: { (items, error) in
            exp.fulfill()
        })
        waitForExpectations(timeout: 10.0, handler: { (error) in
            print (error?.localizedDescription)
        })
    }
}
```

Workshop

Working with

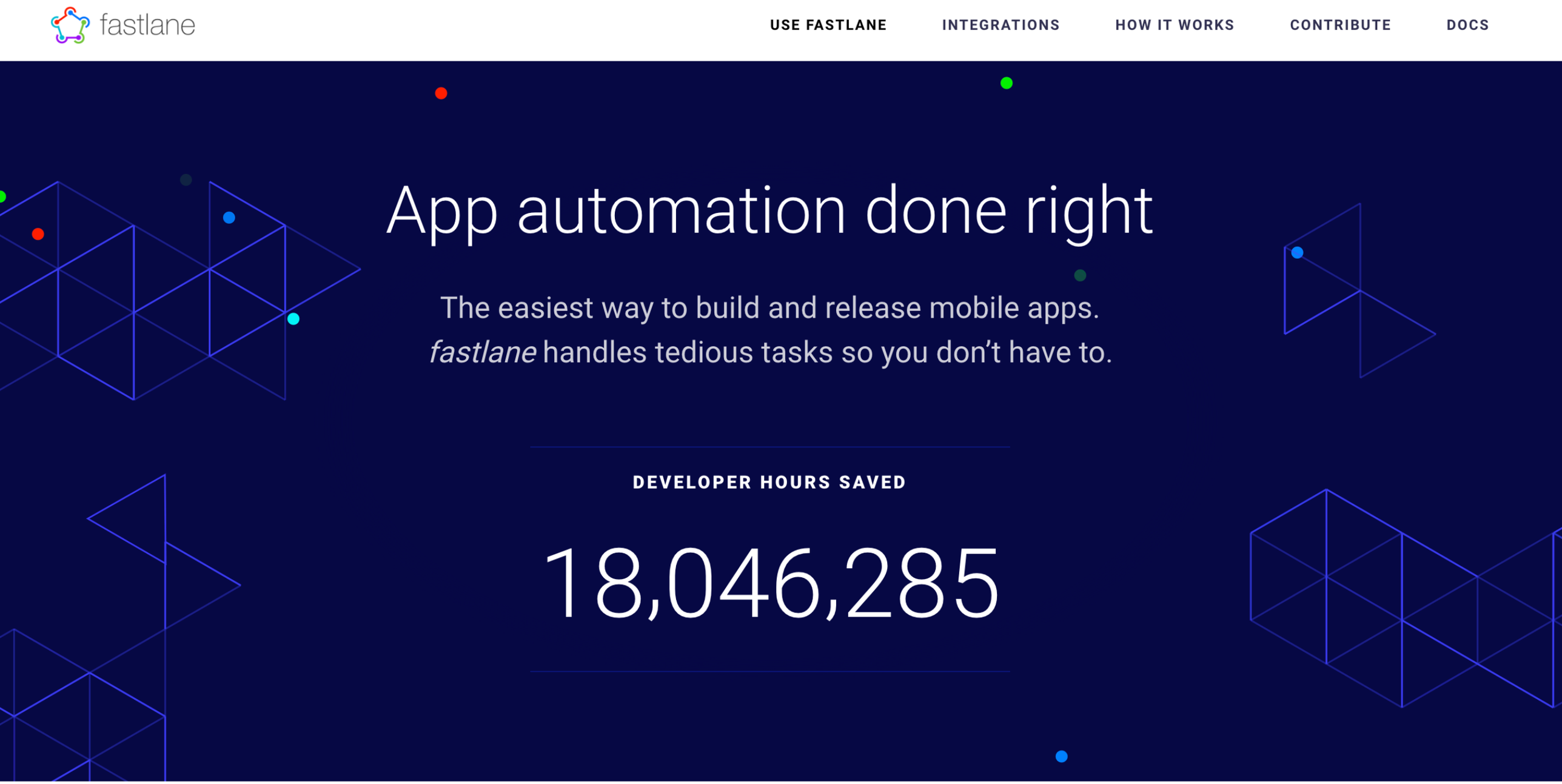
Why fastlane ?

CI/CD for iOS app

To help deployment and testing easier

Series of tools to test, build, sign and deploy iOS app

Fastlane



https://fastlane.tools/

Fastlane

Automate your development and release process

fastlane is an open source platform aimed at simplifying Android and iOS deployment.
fastlane lets you automate every aspect of your development and release workflow.

AUTOMATE SCREENSHOTS

Automatically generate localized screenshots for the app store

LEARN MORE

BETA DEPLOYMENT

Easily distribute beta builds to testers

LEARN MORE

APP STORE DEPLOYMENT

Publish a new release to the app store in seconds

LEARN MORE

CODE SIGNING

Reliably and consistently code sign your app–no more headaches

LEARN MORE

https://fastlane.tools/

Fastlane for iOS app

https://fastlane.tools/

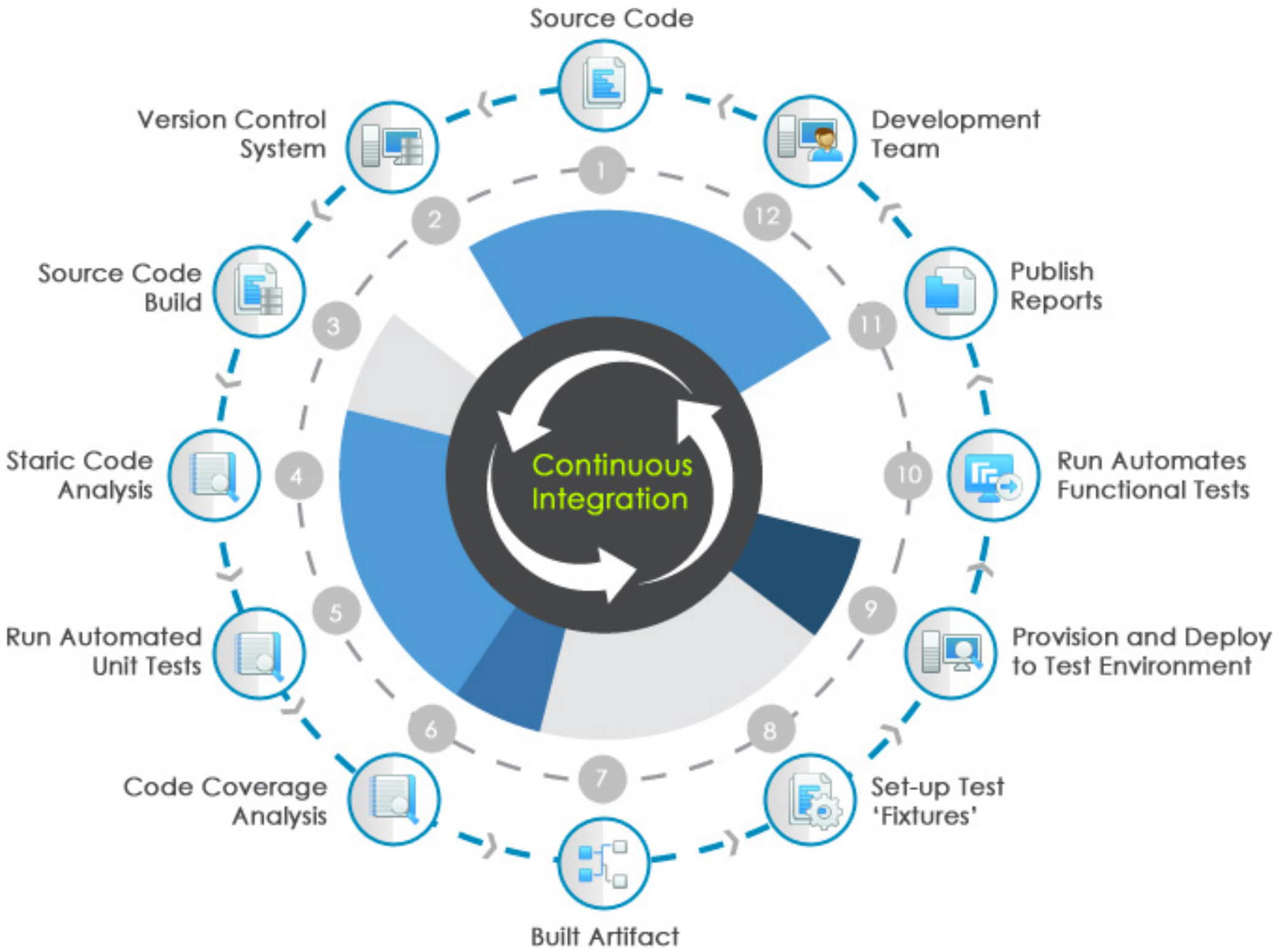
Installation

Required Ruby
$xcode-select --install
$sudo gem install fastlane -NV

https://docs.fastlane.tools/getting-started/ios/setup/

Demo and Workshop

Continuous Integration



Jenkins

Bamboo

TeamCity

go

Visual Studio
Team Foundation Server

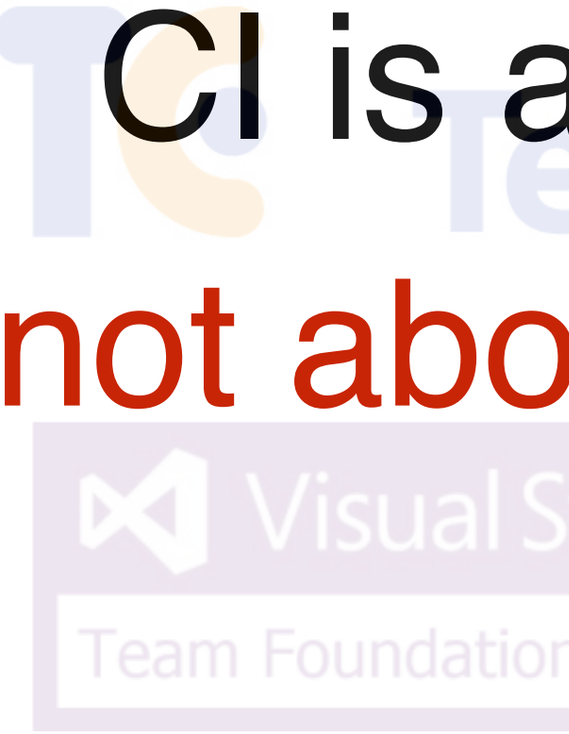
Hudson

Travis

wercker

circleci

CI is about what people do not about what tools they use



Continuous Integration

Discipline to **integrate frequently**

Continuous Integration

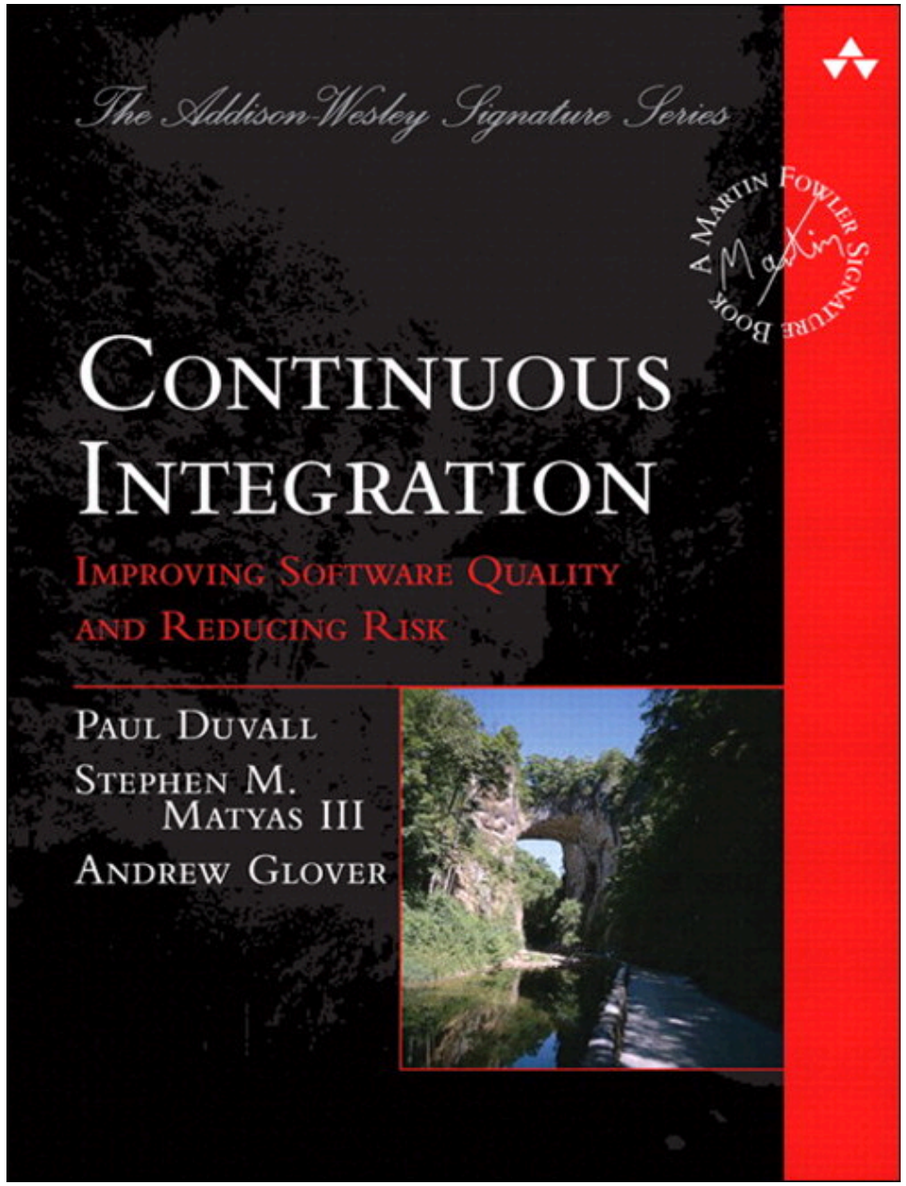
Strive to make **small change**

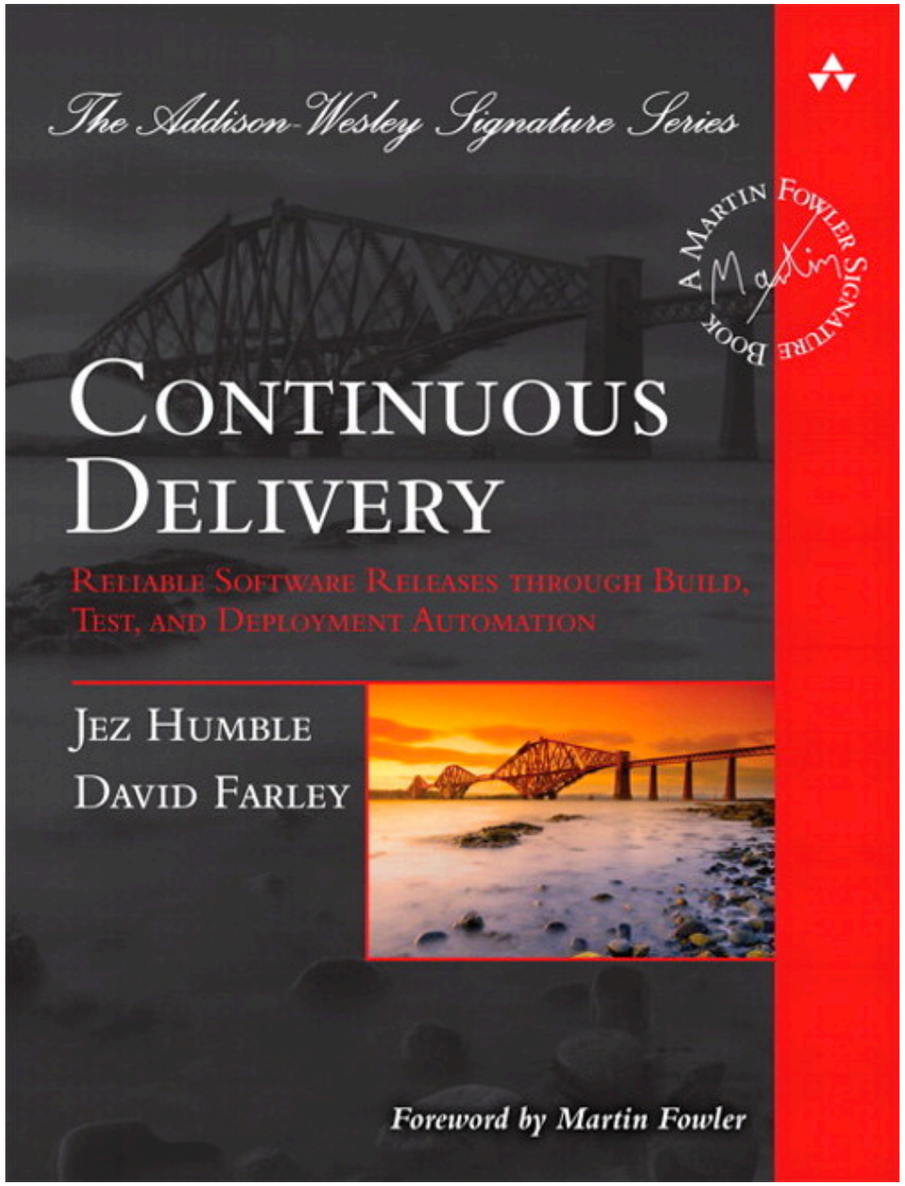
Continuous Integration

Strive for **fast feedback**

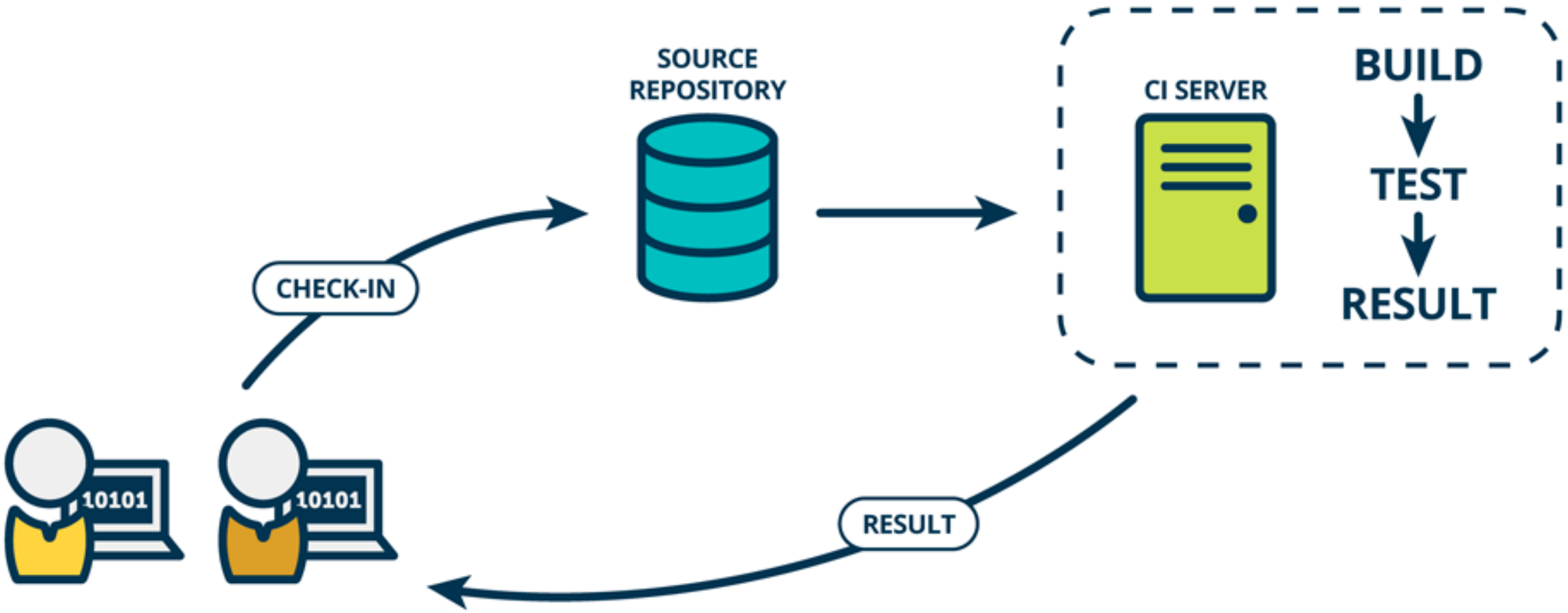
Practices of Continuous Integration

Improve quality and reduce risk

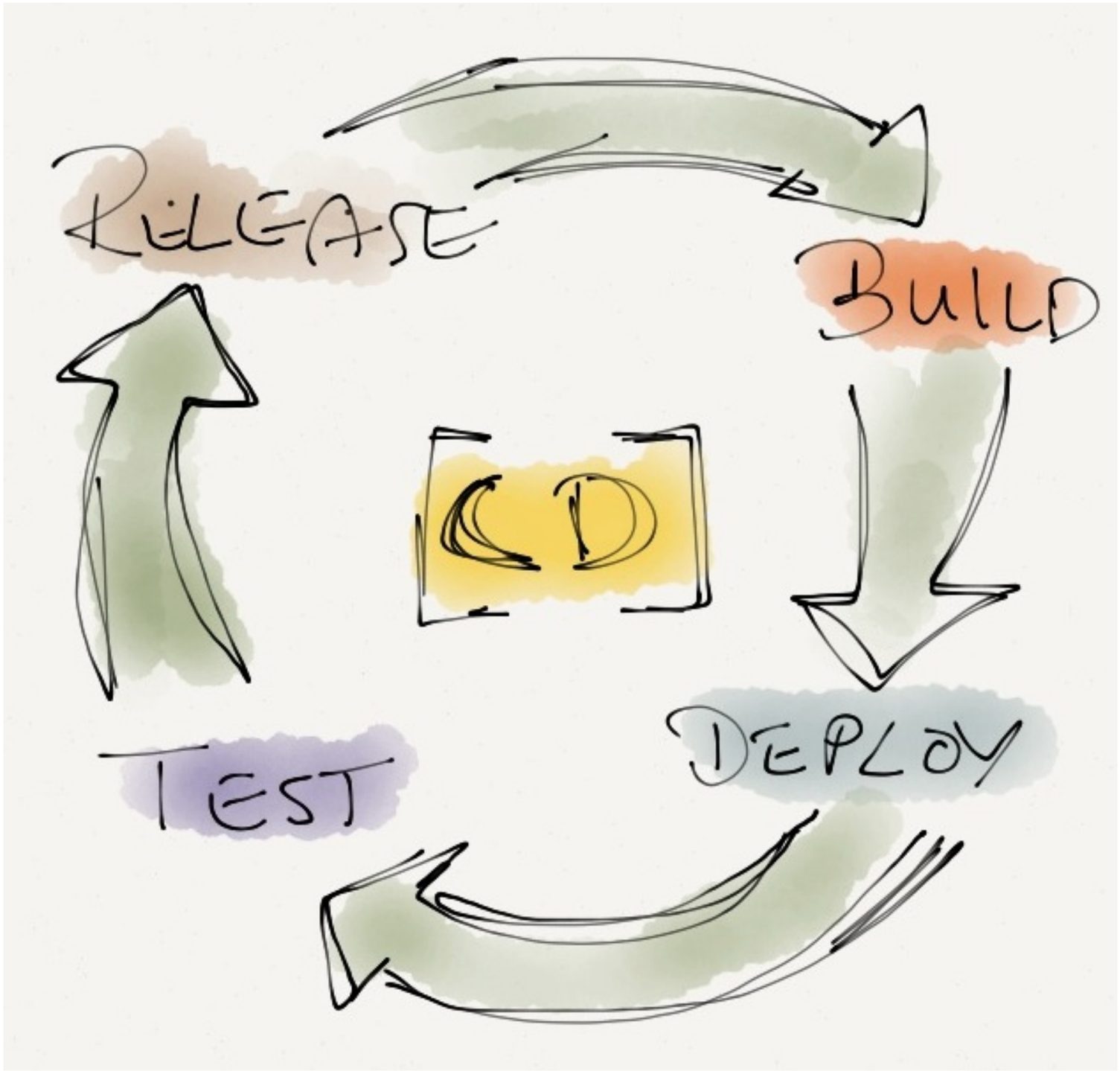




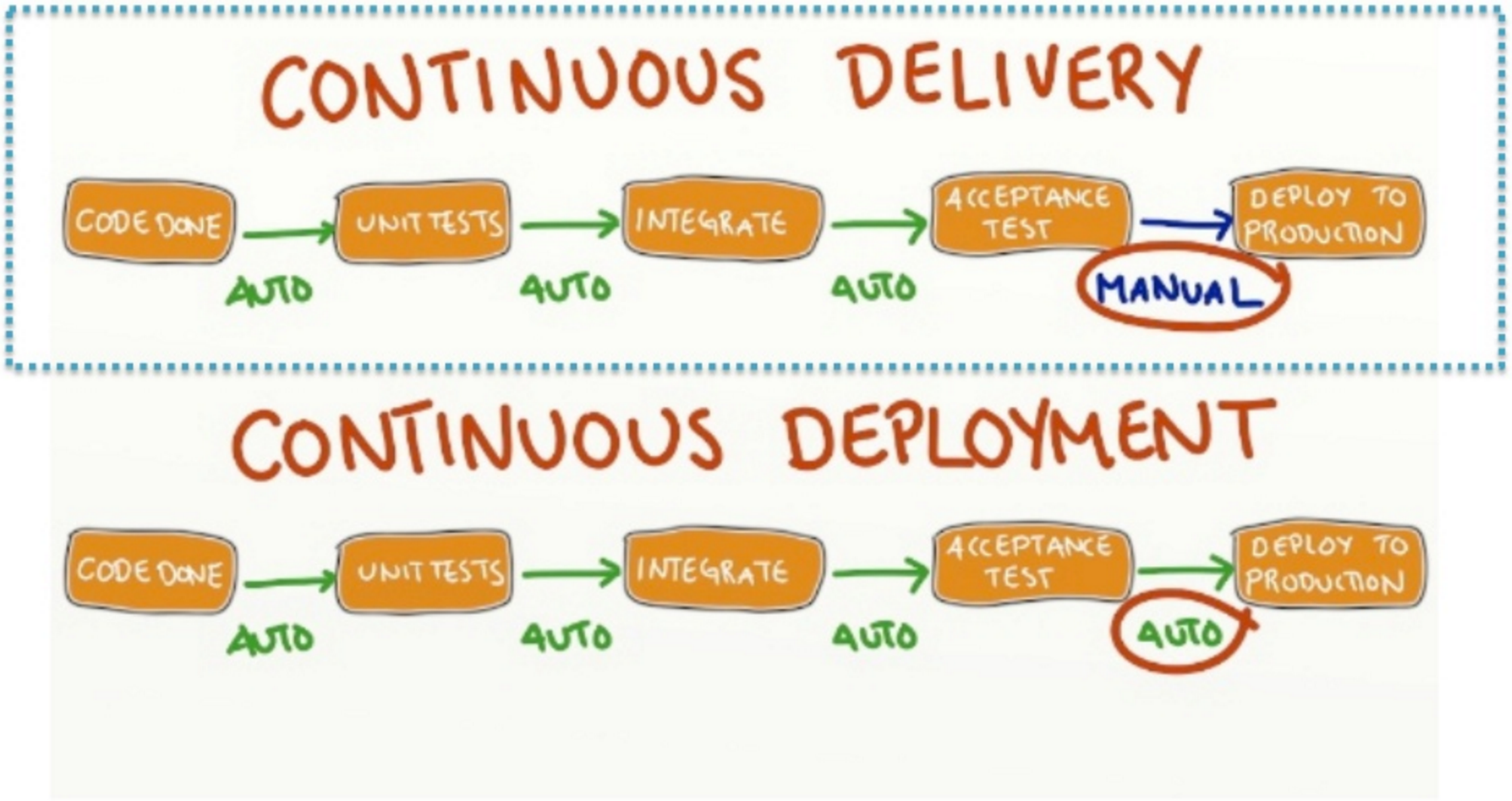
Continuous Integration



CD ?

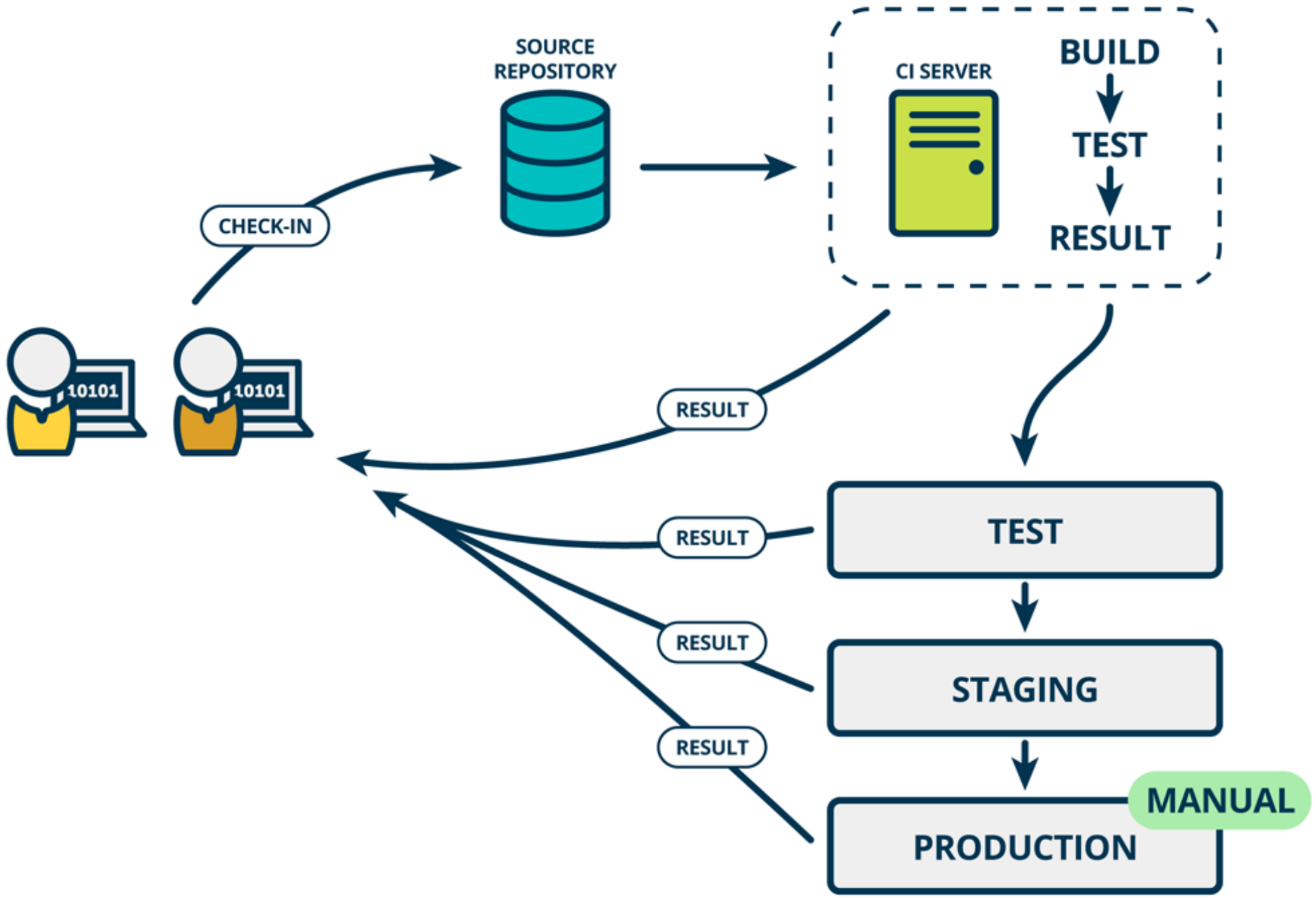


CD ?

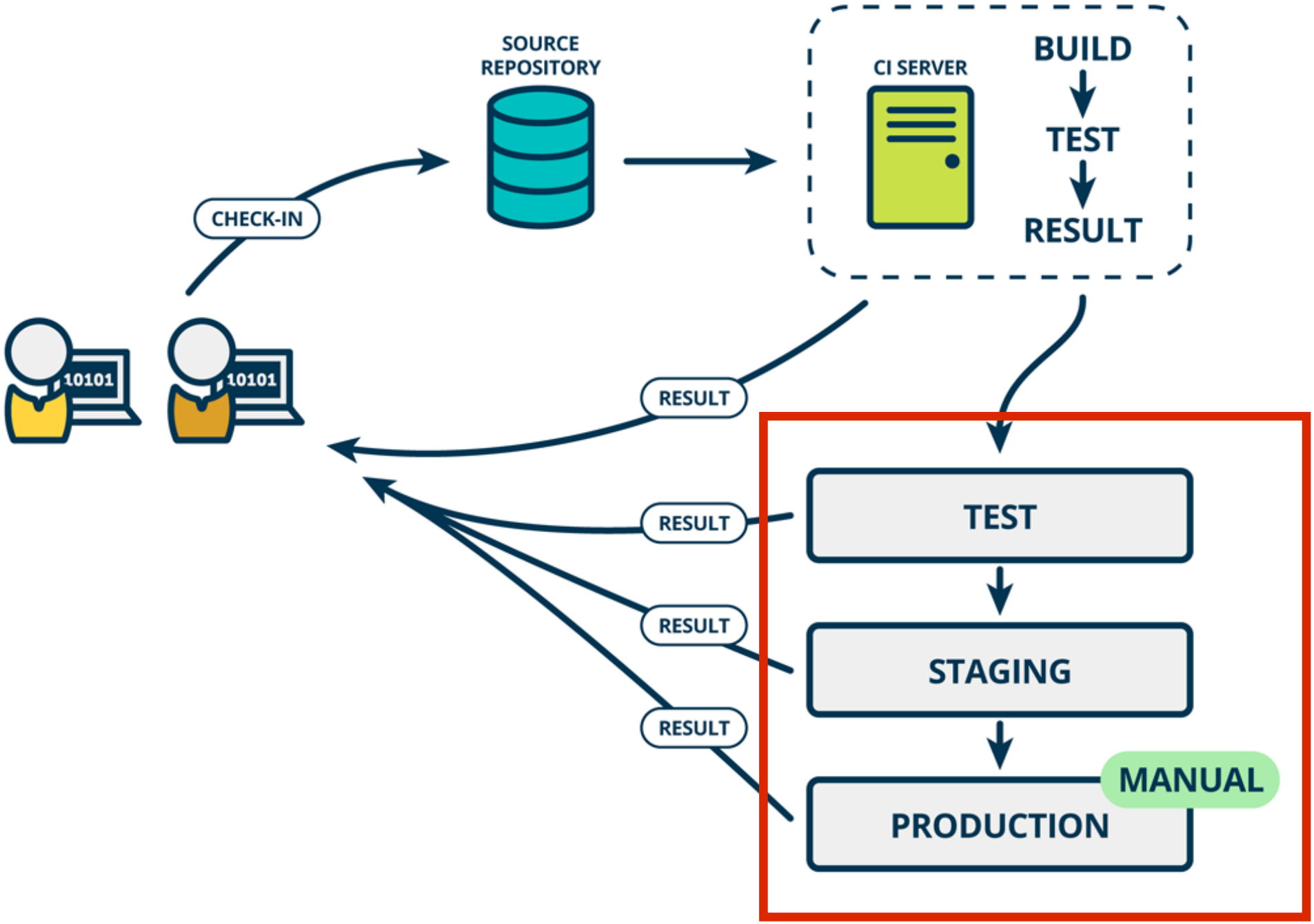


http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment

Continuous Delivery



Rise of DevOps



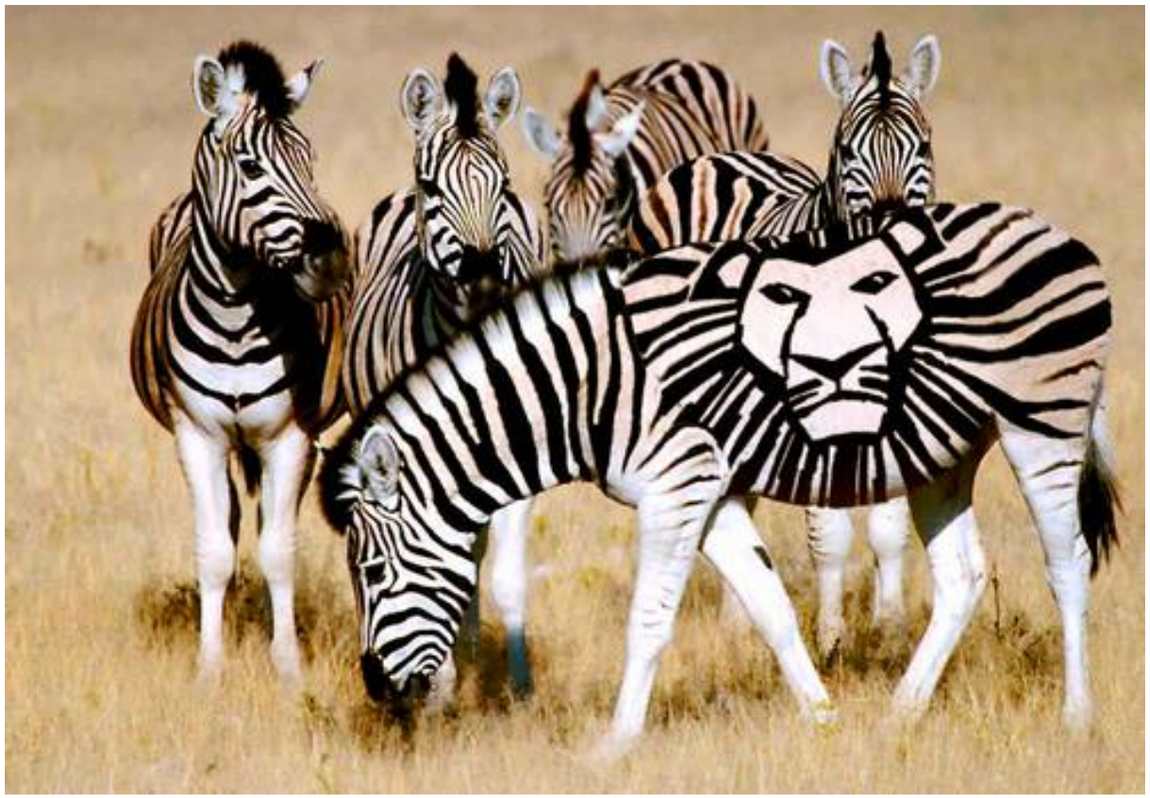
Continuous Integration

is a Software development **practices**

Practice 1

Maintain a single source repository

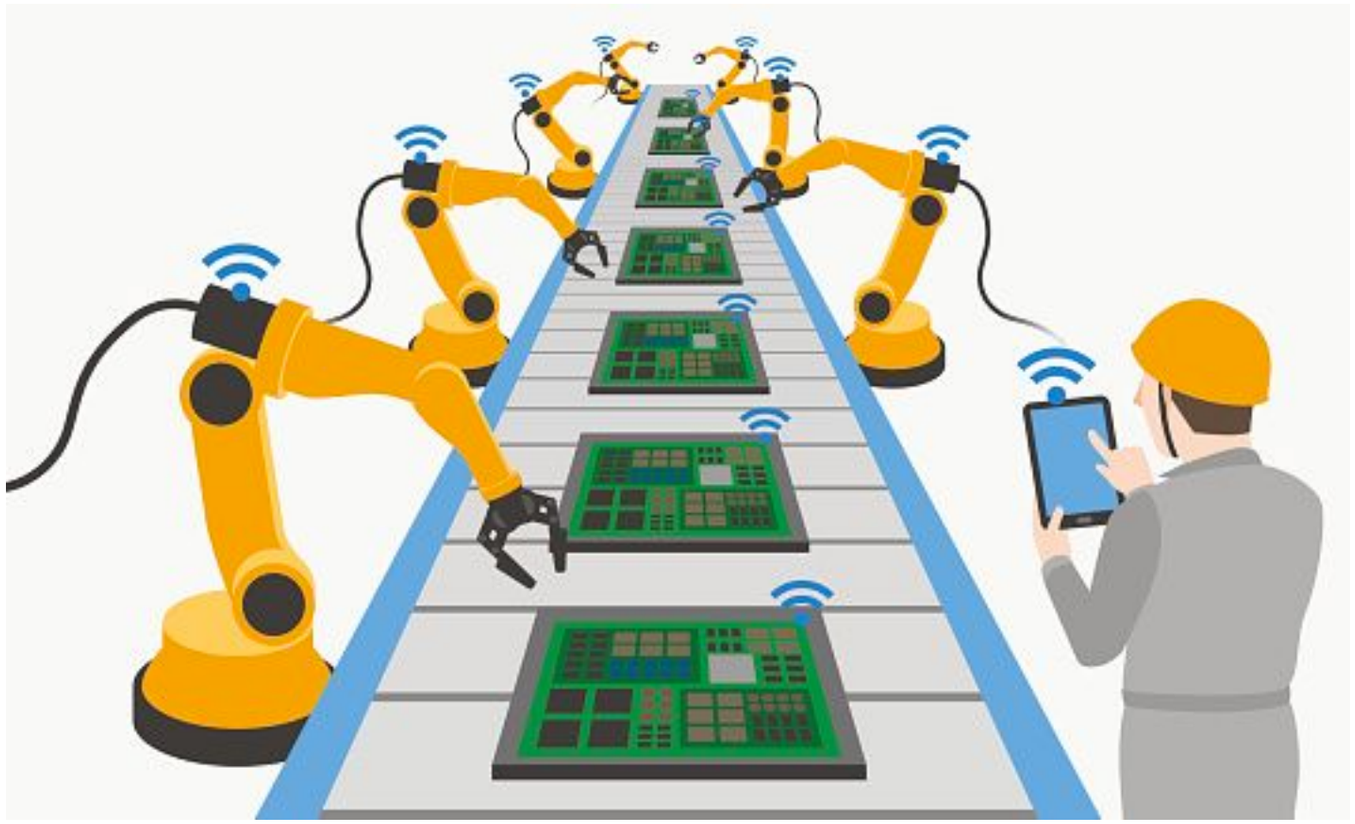
In general, you should store in source control everything you need to build anything



Practice 2

Automated the build

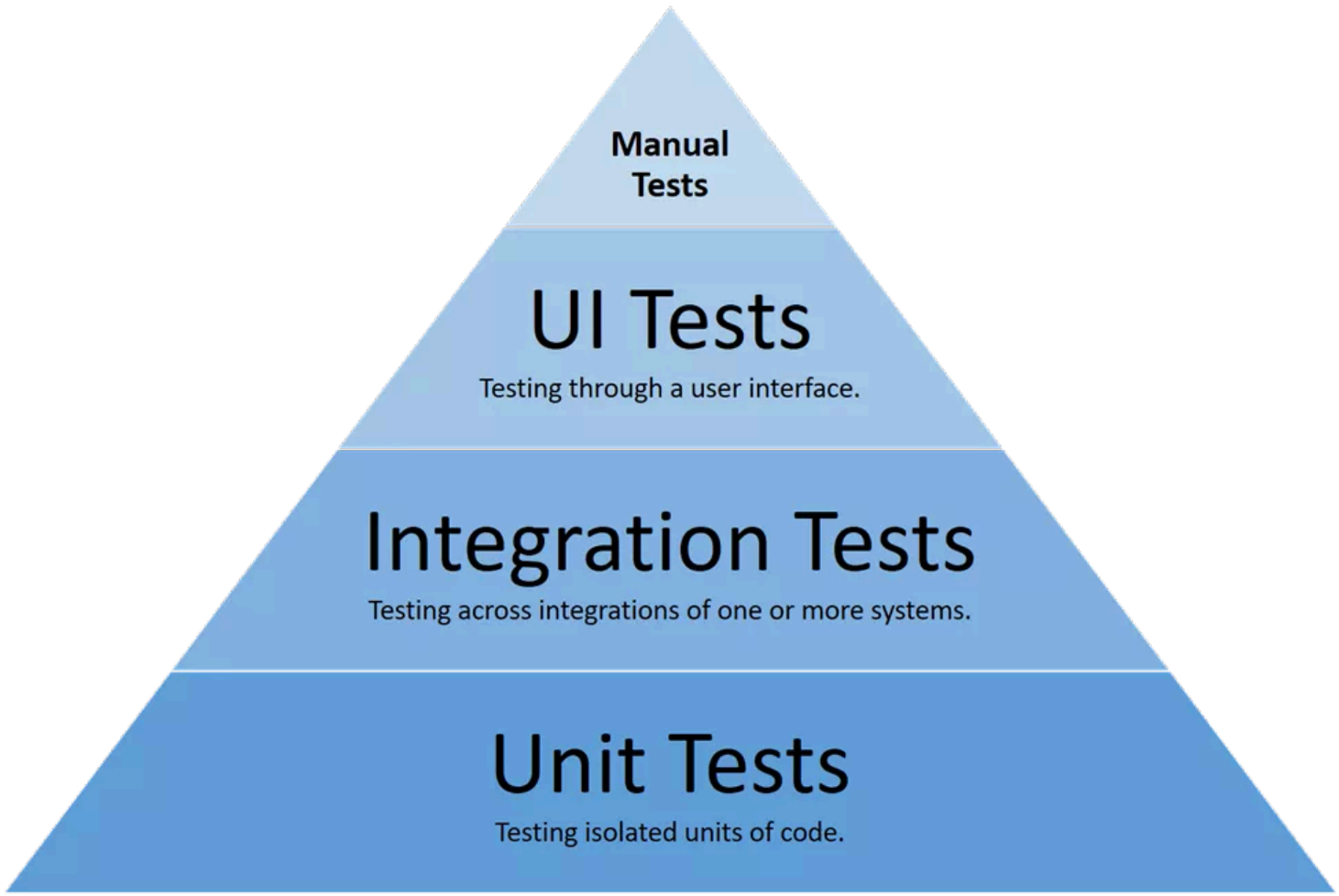
Automated environment for builds



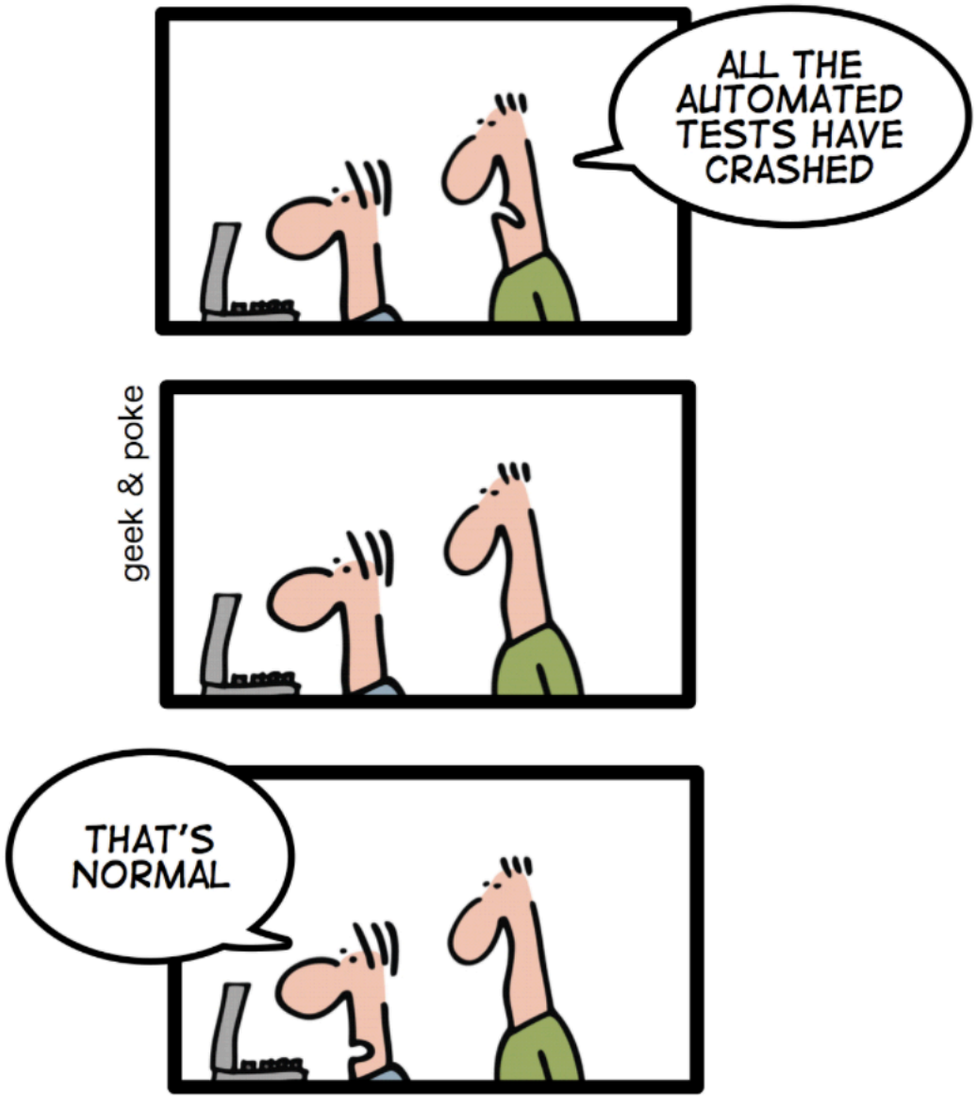
Practice 3

Make your build **self-testing**

Build process => compile, linking and **testing**



TODAY: CONTINUOUS INTEGRATION GIVES YOU THE COMFORTING FEELING TO KNOW THAT EVERYTHING IS NORMAL



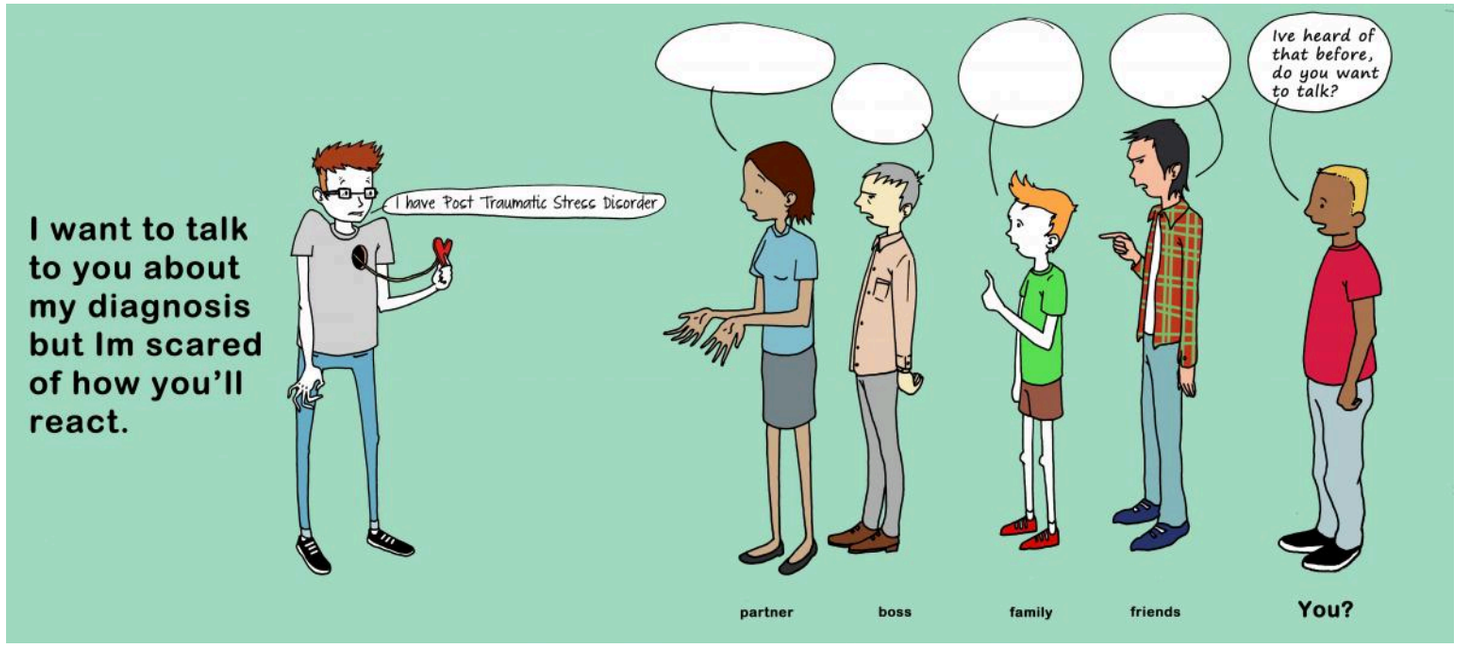
http://geekandpoke.typepad.com/

Practice 4

Everyone **commits** to the mainline everyday

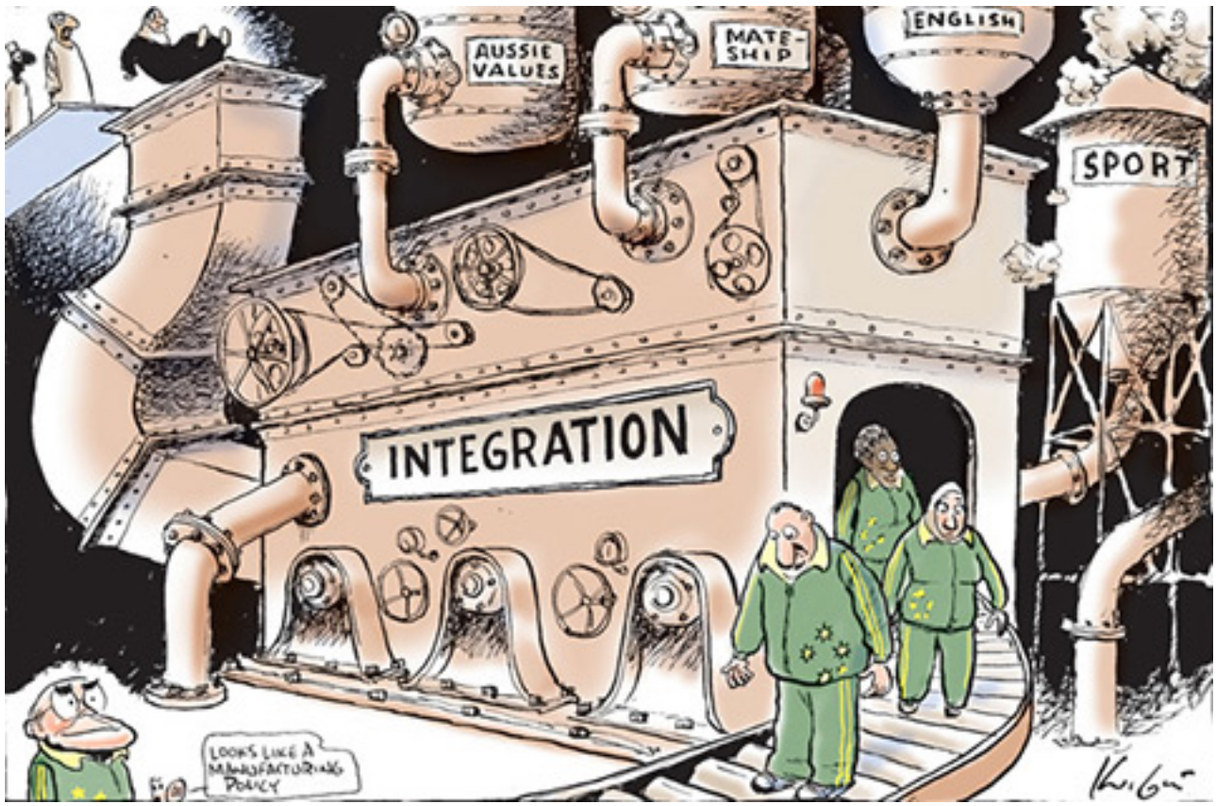
Integration is about **communication**

Integration allows developers to **tell** other developers



Practice 5

Every commits should build the mainline on an **Integration machine**



Nightly build is not enough for Continuous Integration

Practice 6

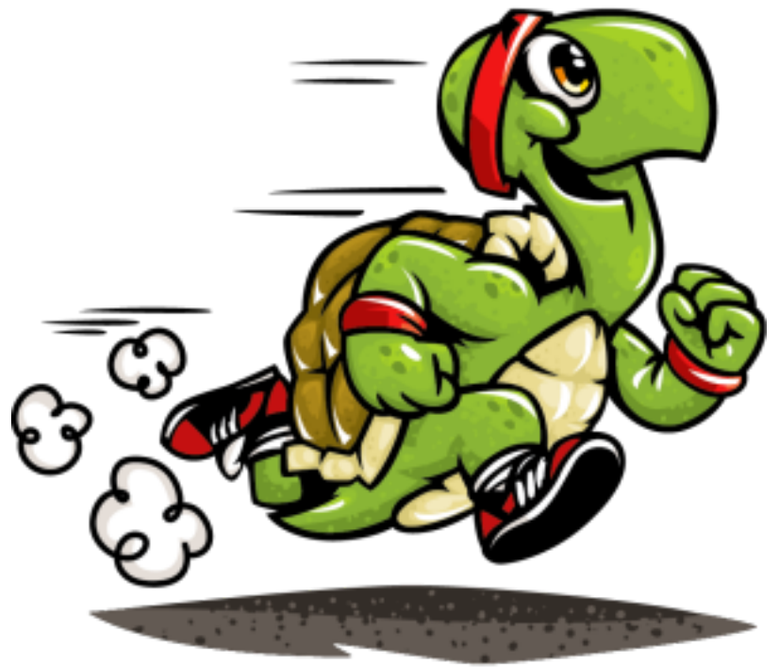
Fix broken builds **immediately**

“Nobody has a higher priority task than fixing the build”

Practice 7

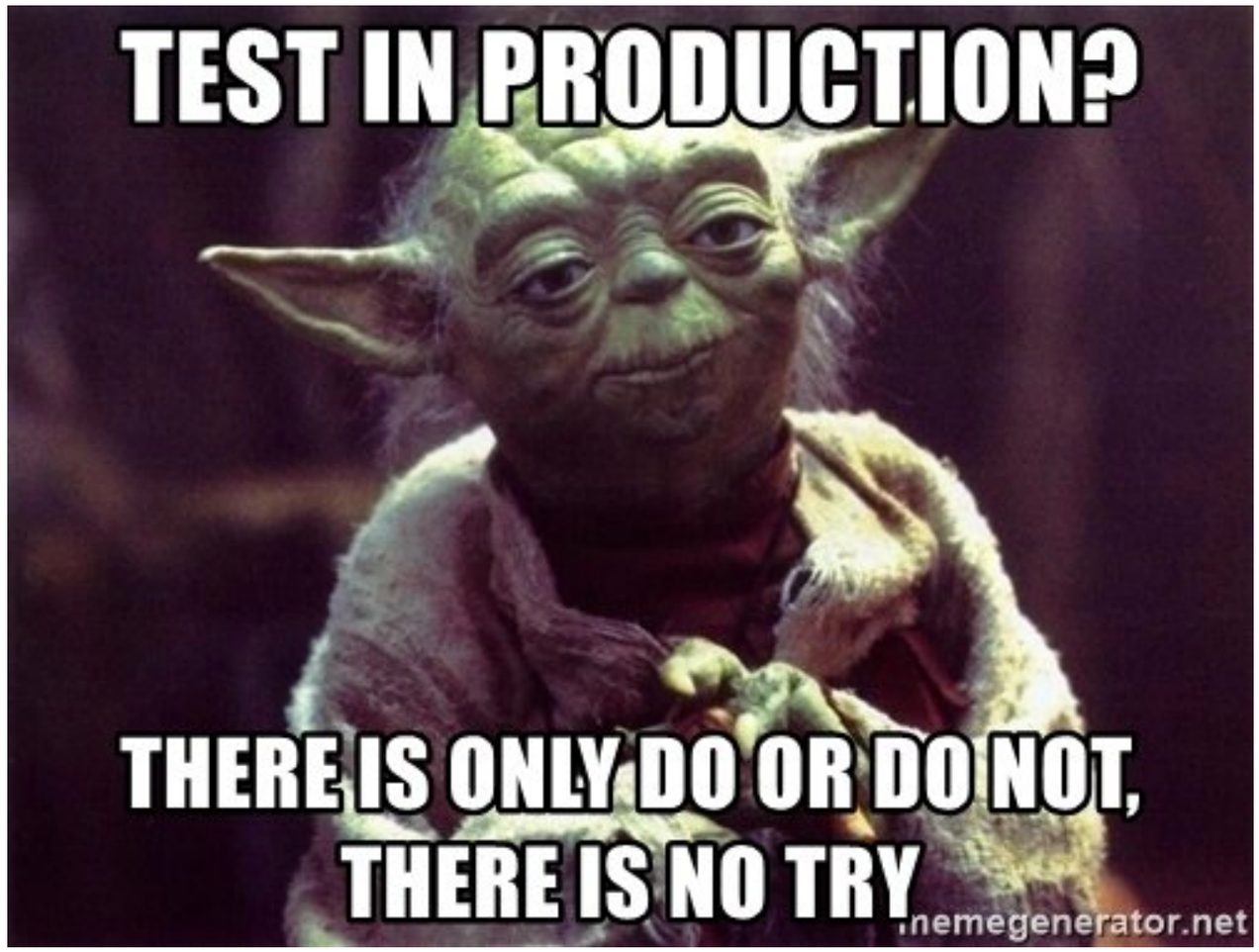
Keep the build **fast**

Continuous Integration is to provide rapid feedback



Practice 8

Test in clone of the **Production** environment



Practice 9

Make it easy for anyone to get
the latest executable

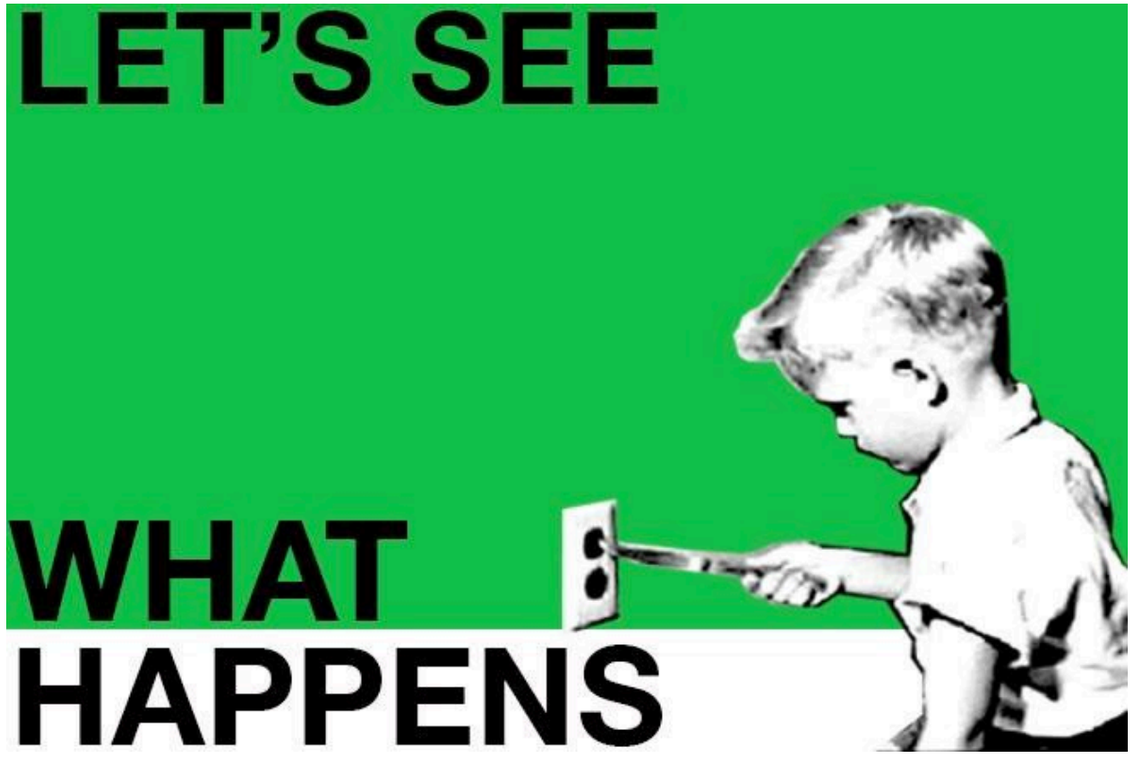
Make sure well known place where people can find

Practice 10

Everyone can see what's happening

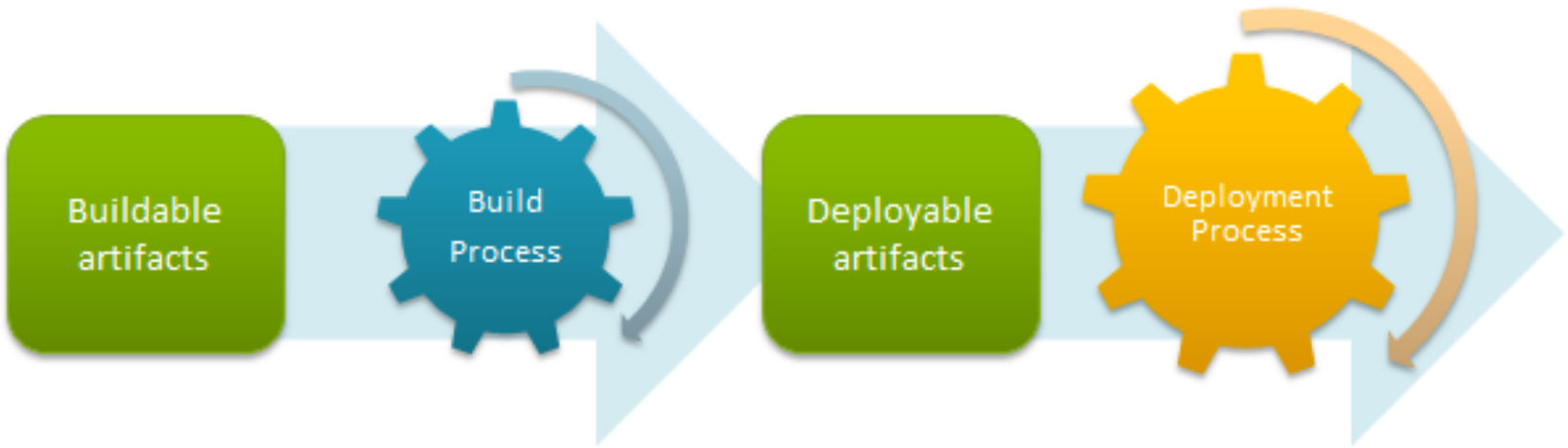
Easier to see the state of the system and changes

Show the good information



Practice 11

Automated deployment



How to achieve the CI ?

1. Use good version control

Local

Centralize

Distributed

2. Choose Branch strategy

Main only

Development isolation

Feature isolation

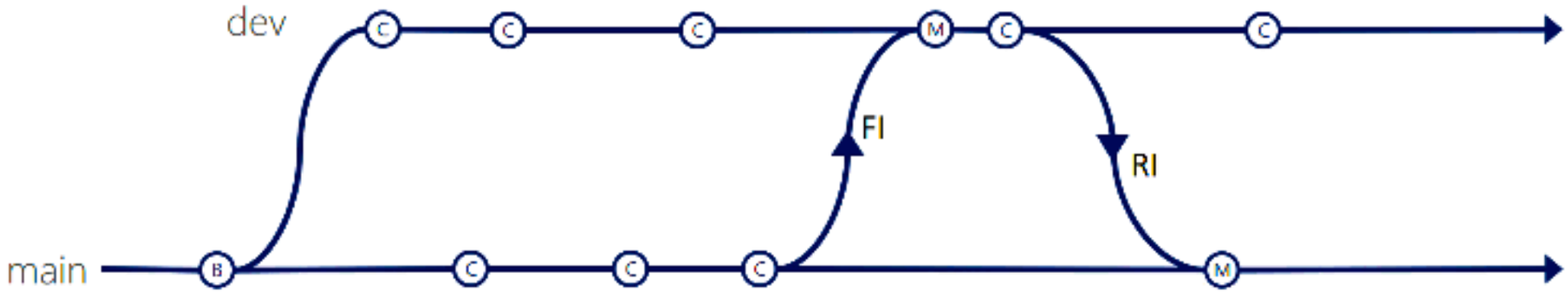
Release isolation

Service and Release isolation

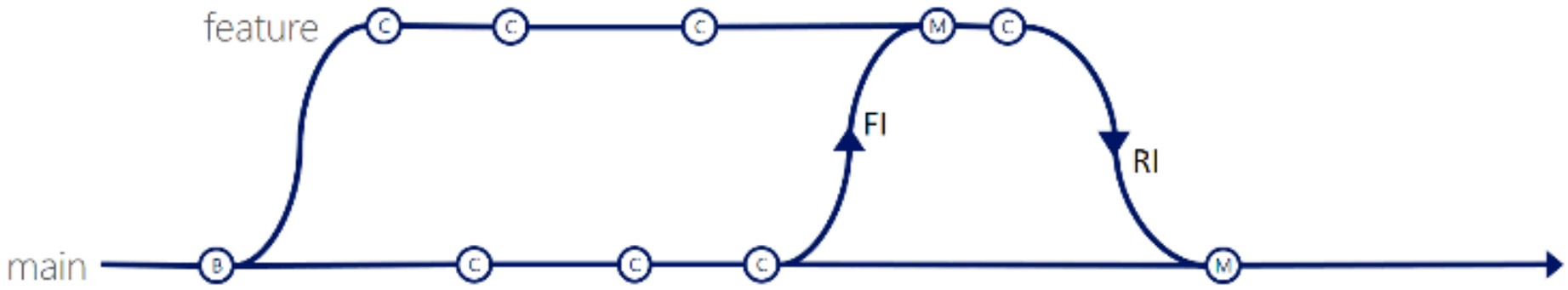
Service, Hotfix and Release isolation

Main only

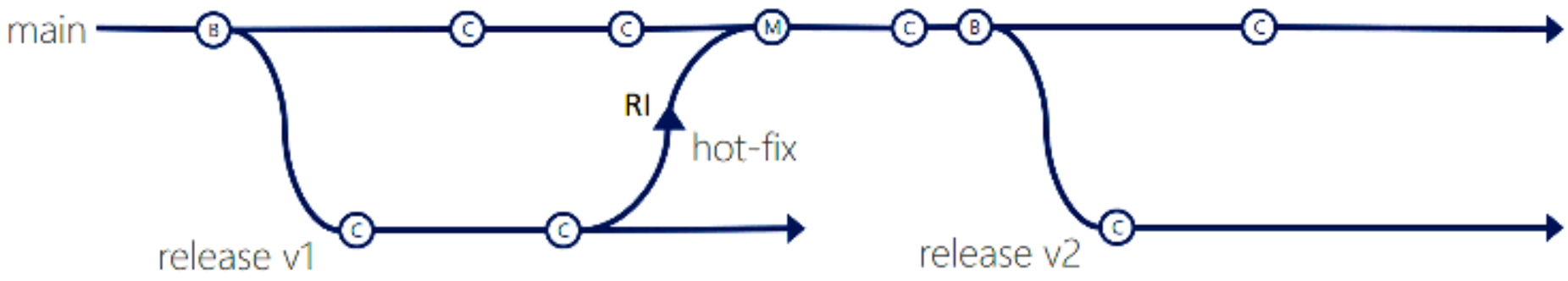
Development isolation



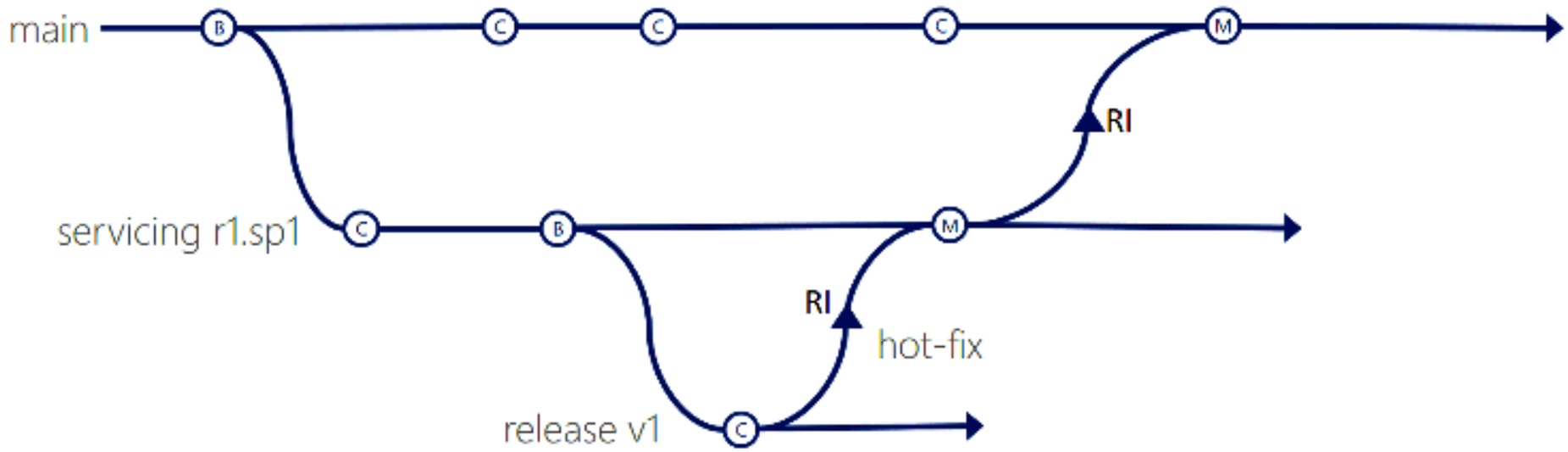
Feature isolation



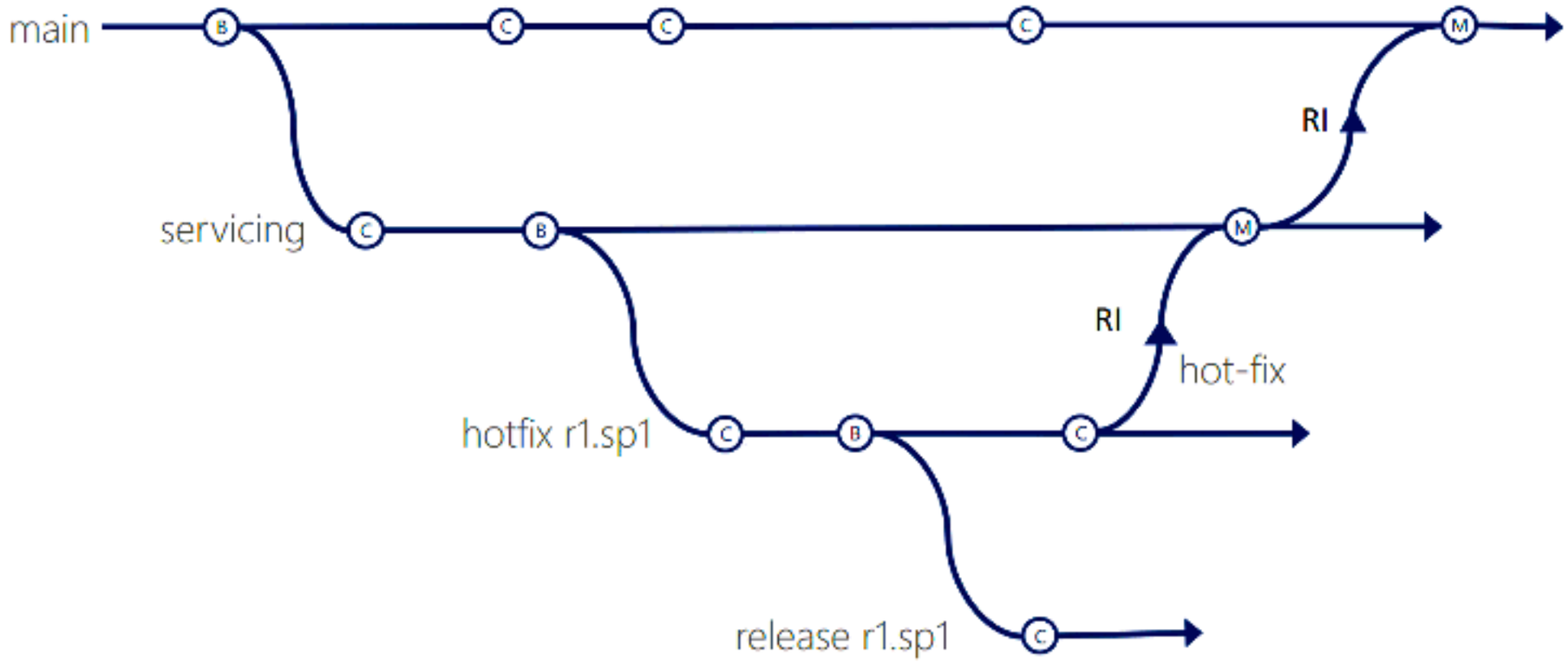
Release isolation



Service and Release isolation



Service, Hotfix, Release isolation

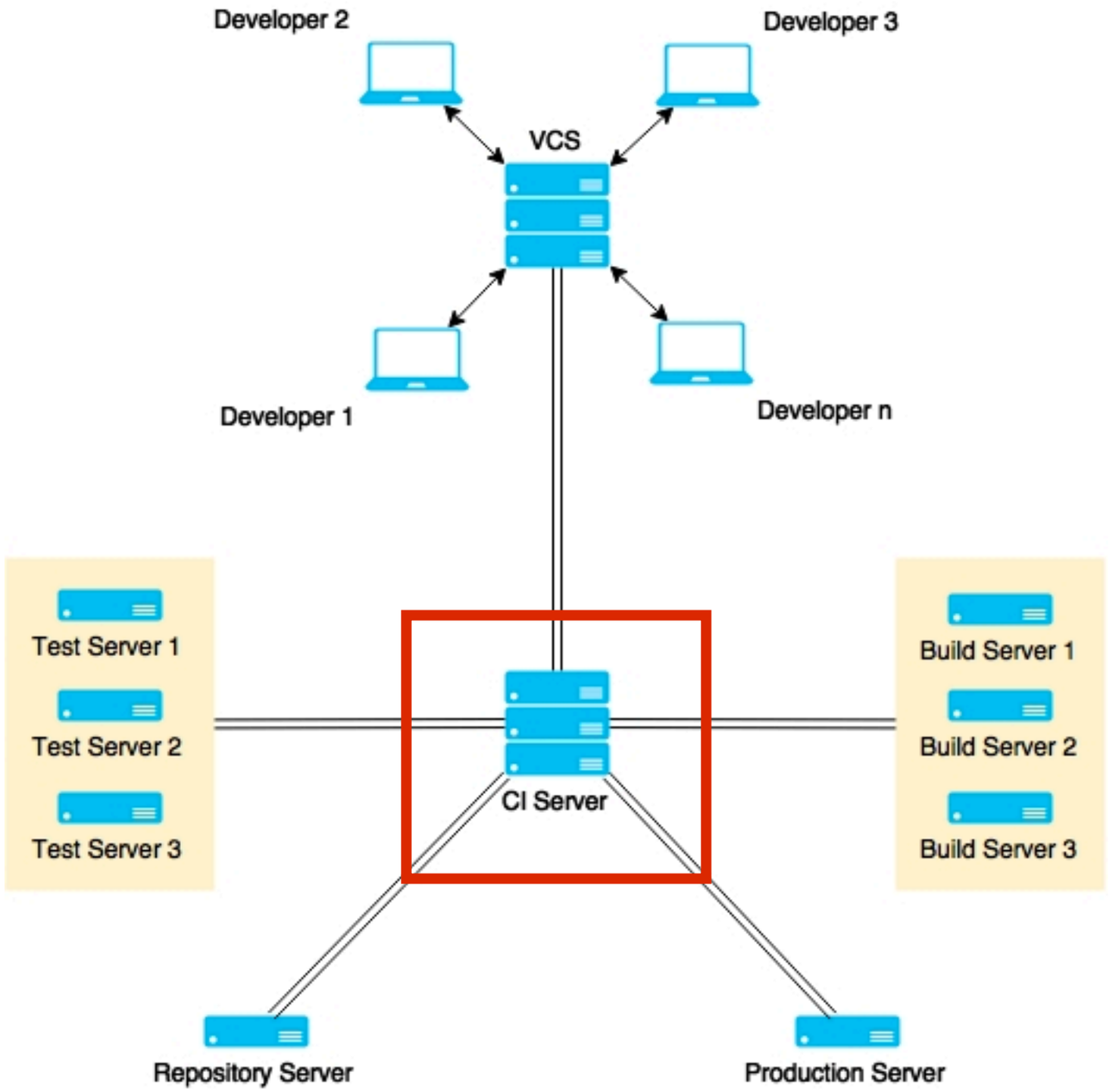


Validate, Validate and Validate

Suggestion

Keeping branches short-lived, merge conflicts are keep to as few as possible

3. Use good CI tool



Jenkins

Bamboo

TeamCity

go

Visual Studio Team Foundation Server

Hudson

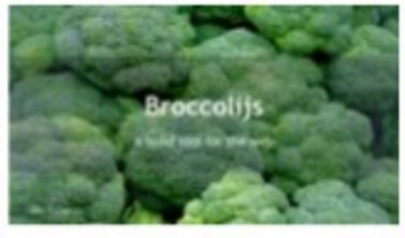
Travis

wercker

circleci

4. Use good build tool

- Javascript
  - Gulp, Grunt, Brocolli



- C#/.NET
  - Nant, MSBuild

- Java/JVM
  - Ant, Maven, Gradle, SBT, Leiningen

More …

Use static code analysis

Automated testing

Automated deployment

People discipline/habit

“Behind every successful agile project, there is a

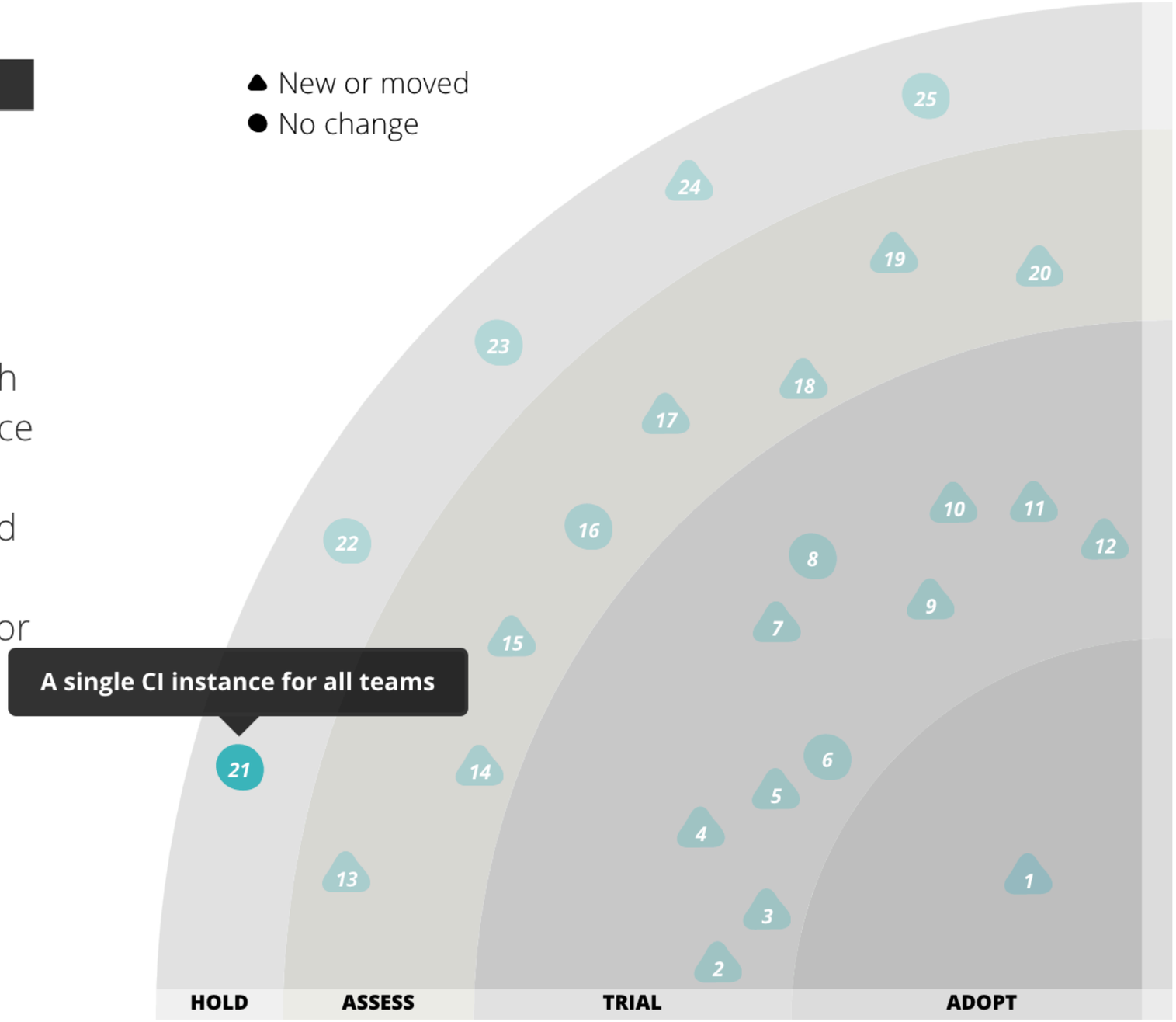
Continuous Integration Server”

Anti-pattern for CI Server

HOLD ?

21. A single CI instance for all teams

We're compelled to caution, again, against creating **a single CI instance for all teams**. While it's a nice idea in theory to consolidate and centralize Continuous Integration (CI) infrastructure, in reality we do not see enough maturity in the tools and products in this space to achieve the desired outcome. Software delivery teams which must use the centralized CI offering regularly have long delays depending on a central team to perform minor configuration tasks, or to troubleshoot problems in the shared infrastructure and tooling. At this stage, we continue to recommend that organizations limit their centralized investment to establishing patterns, guidelines and support for delivery teams to operate their own CI infrastructure.



https://www.thoughtworks.com/radar/techniques

Demo CI/CD with Jenkins