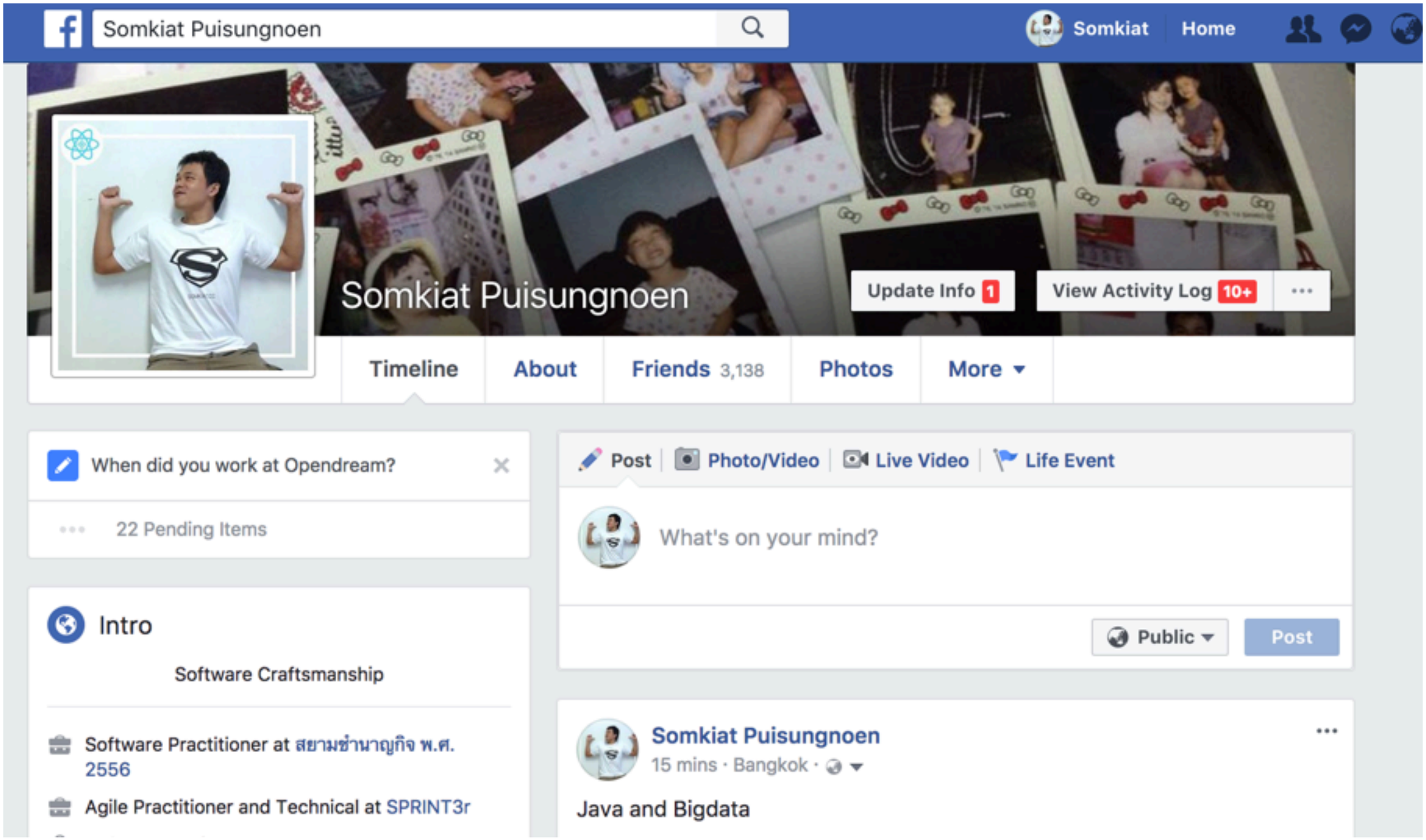
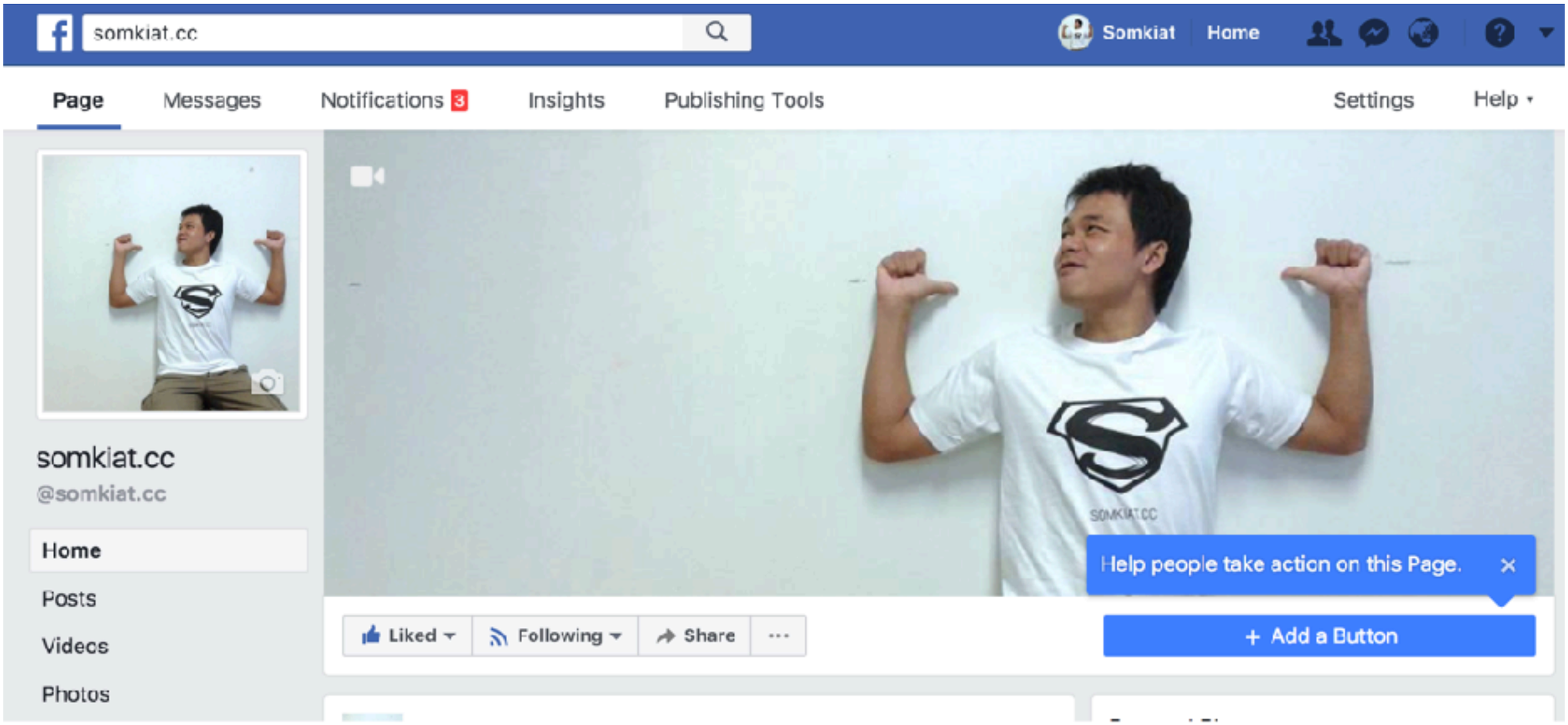
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Develop iOS app with Swift





Agenda

- Software requirement
- Programming language
- Create iOS project
- Command line tools
- Apple Human Interface Guideline (HIG)
- Workshop (step-by-step)

https://developer.apple.com/design/human-interface-guidelines/guidelines/overview/

Development needs

- Coding skills
- UI components
- App architecture
- Networking
- Debugging
- Testing
- Threads and concurrency
- Workshop (step-by-step)

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

Tools

Build

Test

Deploy

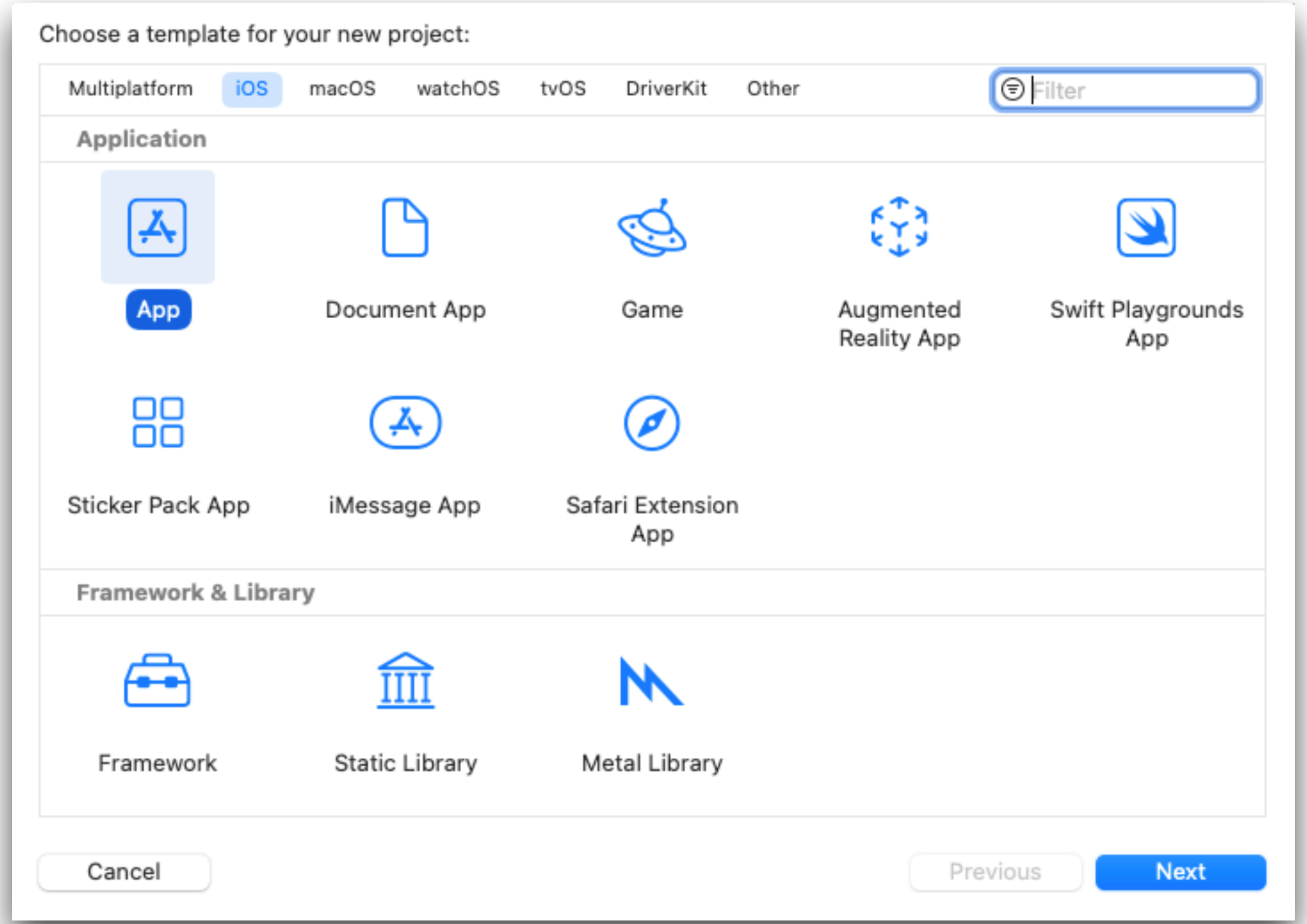
Software requirements

Software requirements

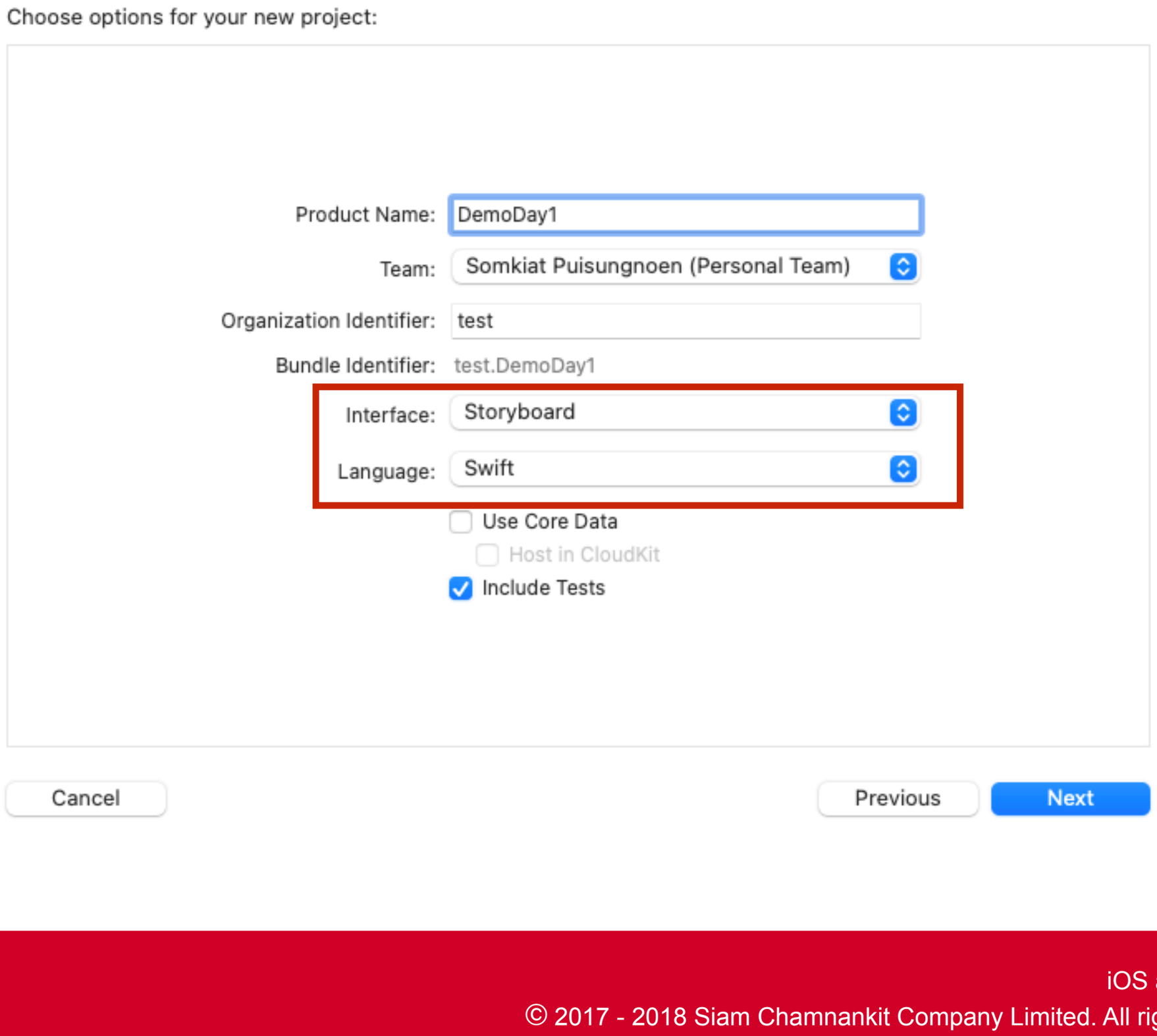
Xcode 14
CocoaPods
Fastlane

fastlane

Create a new project



Interface and Language



UI Interfaces

Story board

Swift UI (iOS 13+)

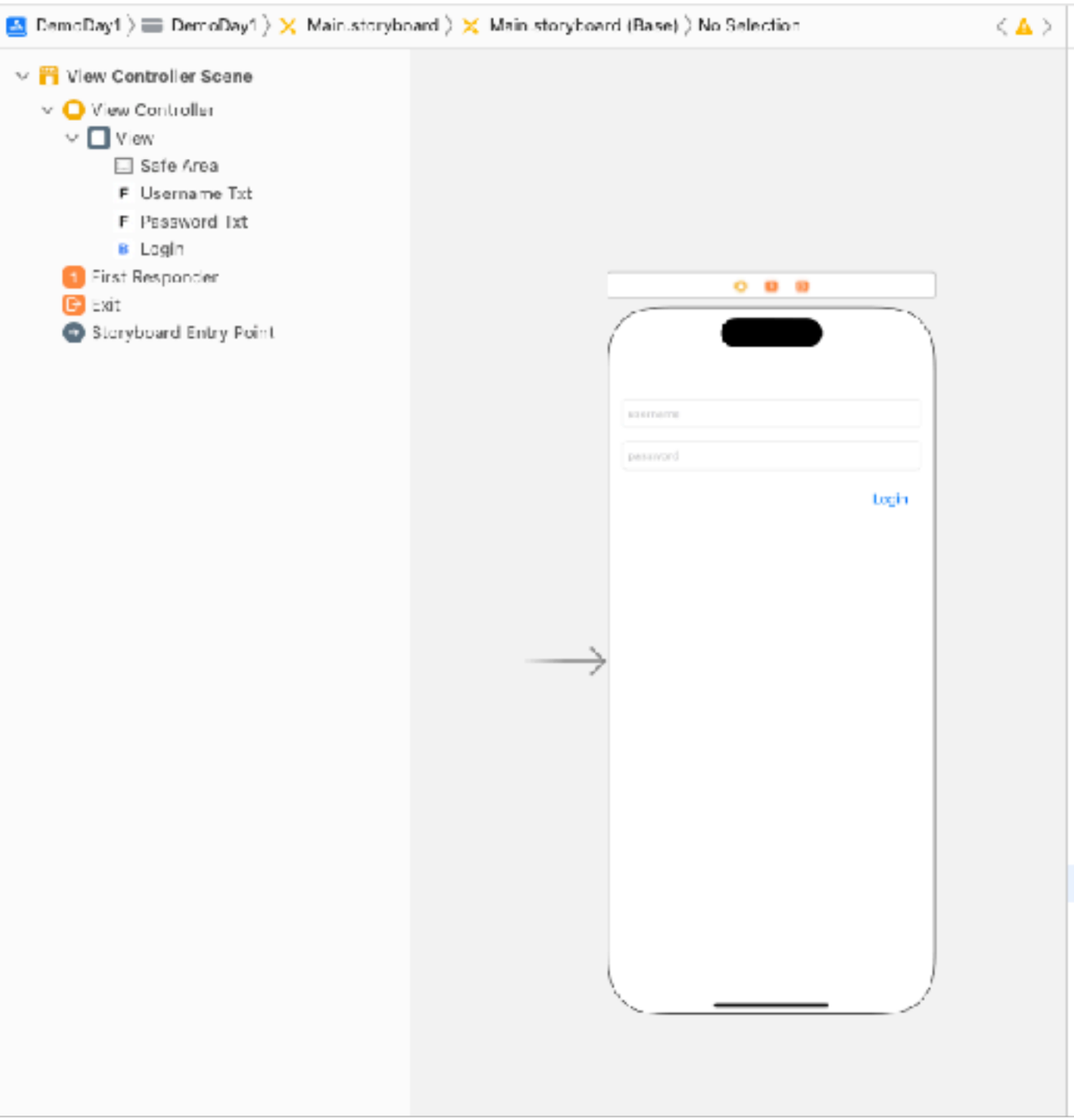
https://developer.apple.com/xcode/swiftui/

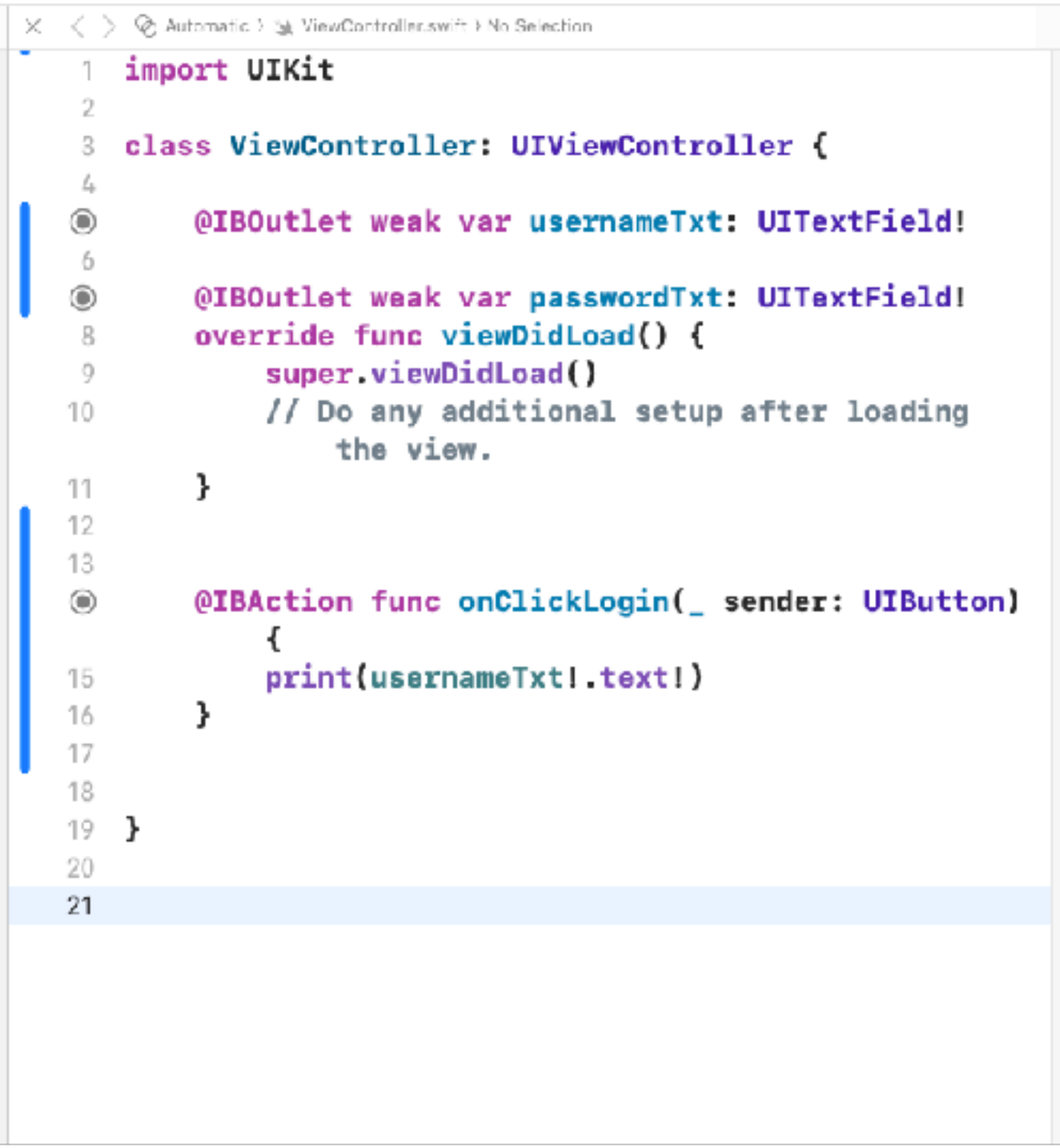
Programming Languages

Swift

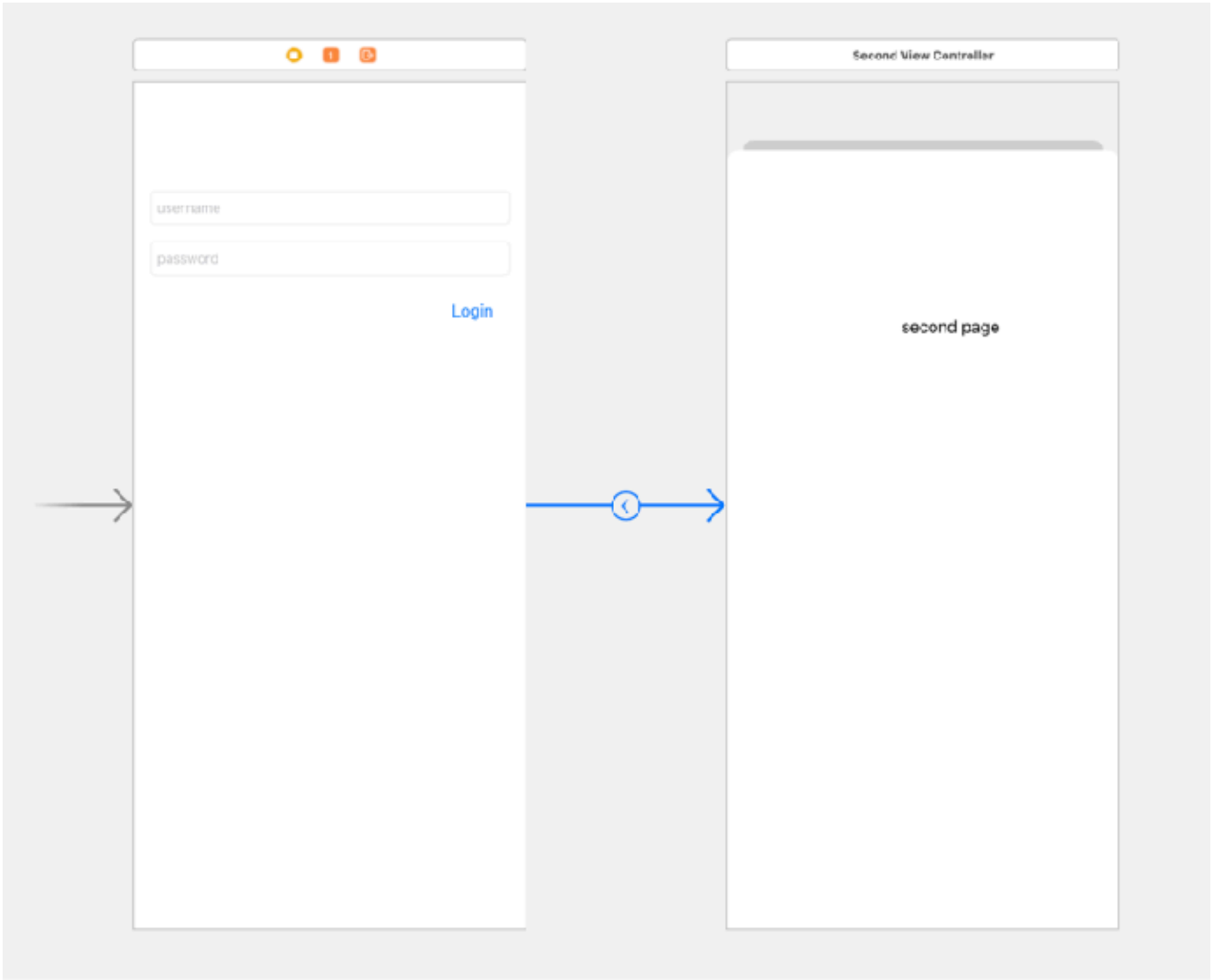
Objective-C

Working with Storyboard





Send data between ViewController



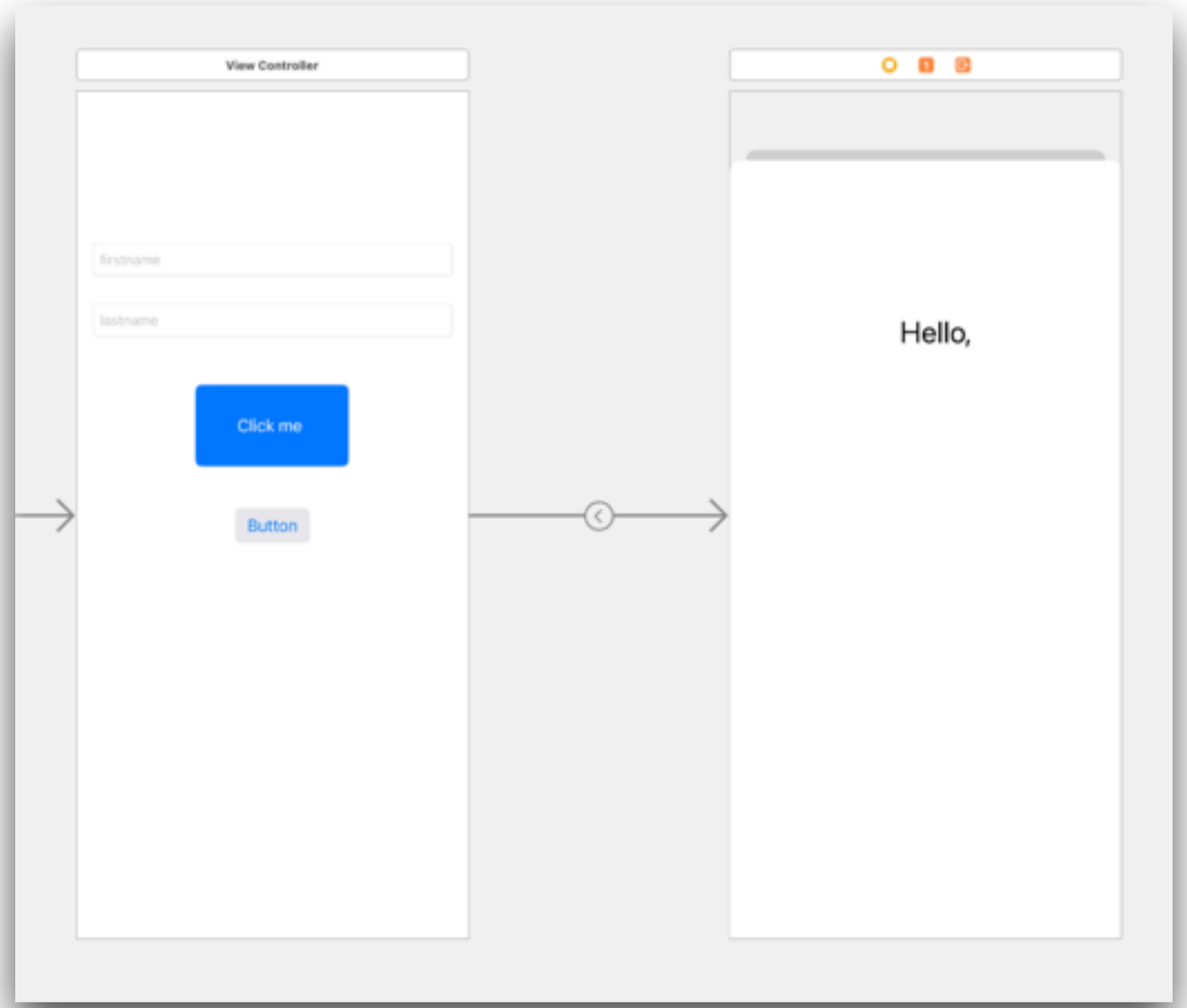
Send data between ViewController

Segue
Delegation pattern (Callback)
Closure
Notification Center
Singleton

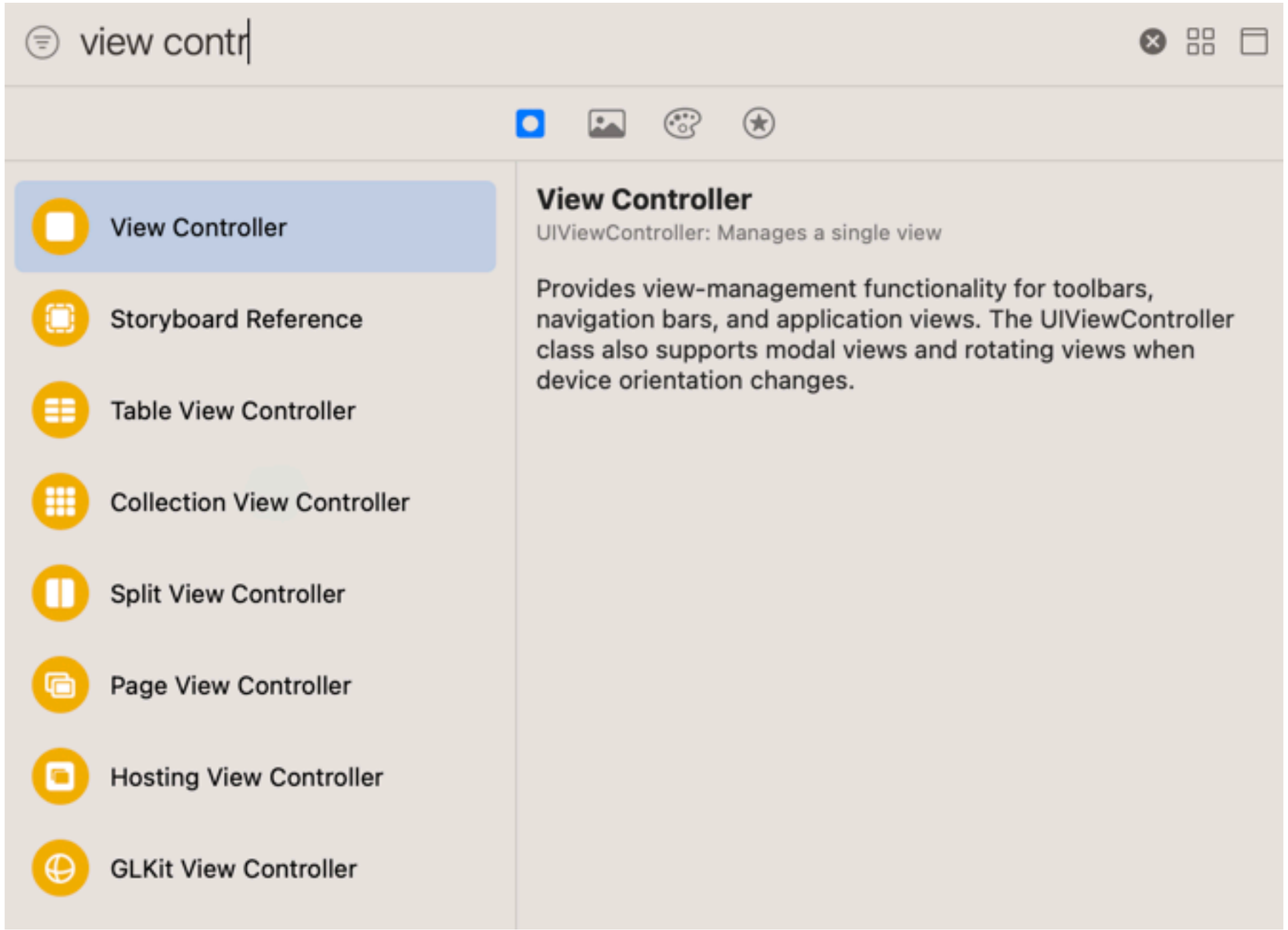
Segue

Segue

Pass data in storyboard mode

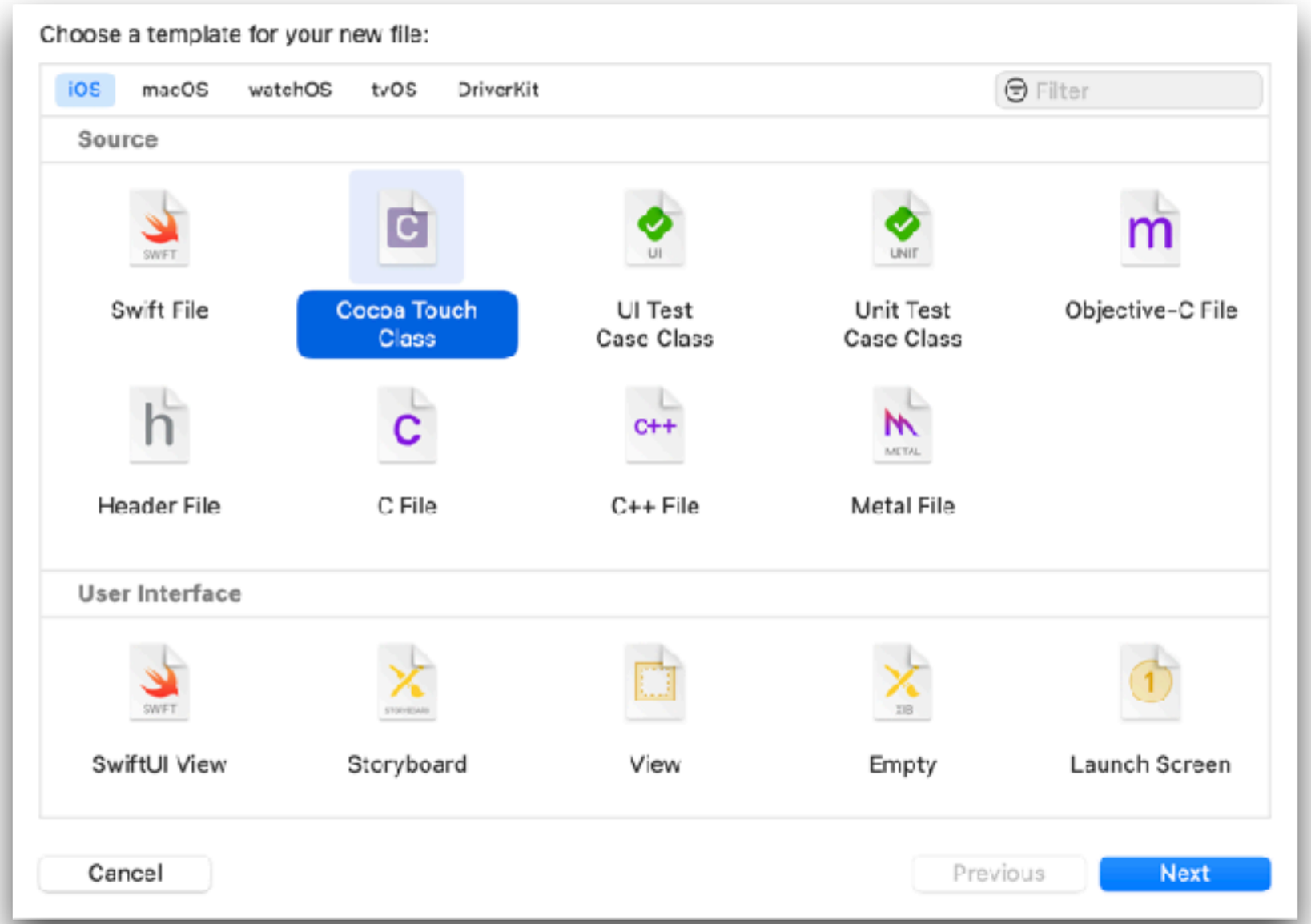


Create new View Controller



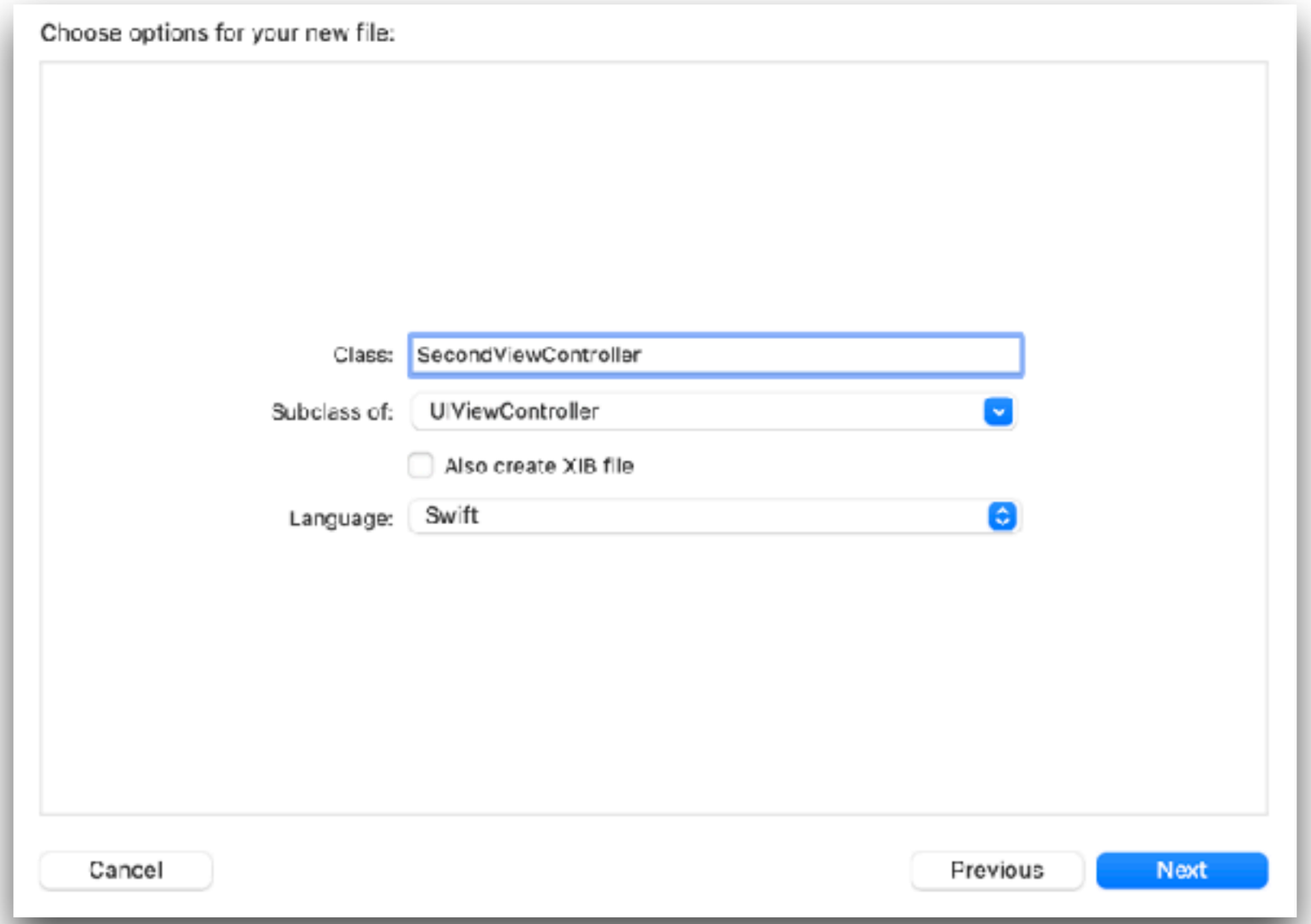
Create View Controller class

Choose Cocoa Touch Class

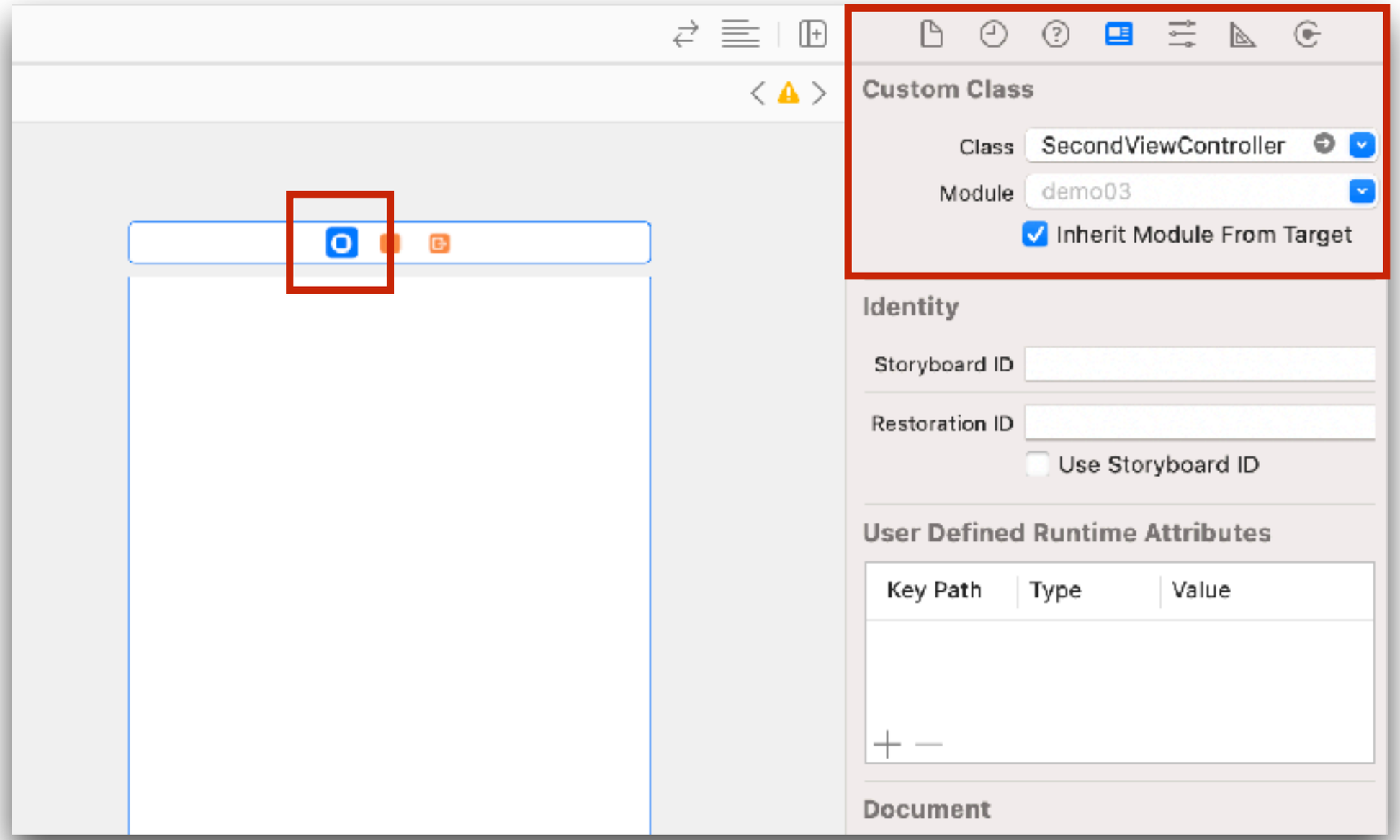


Create View Controller class

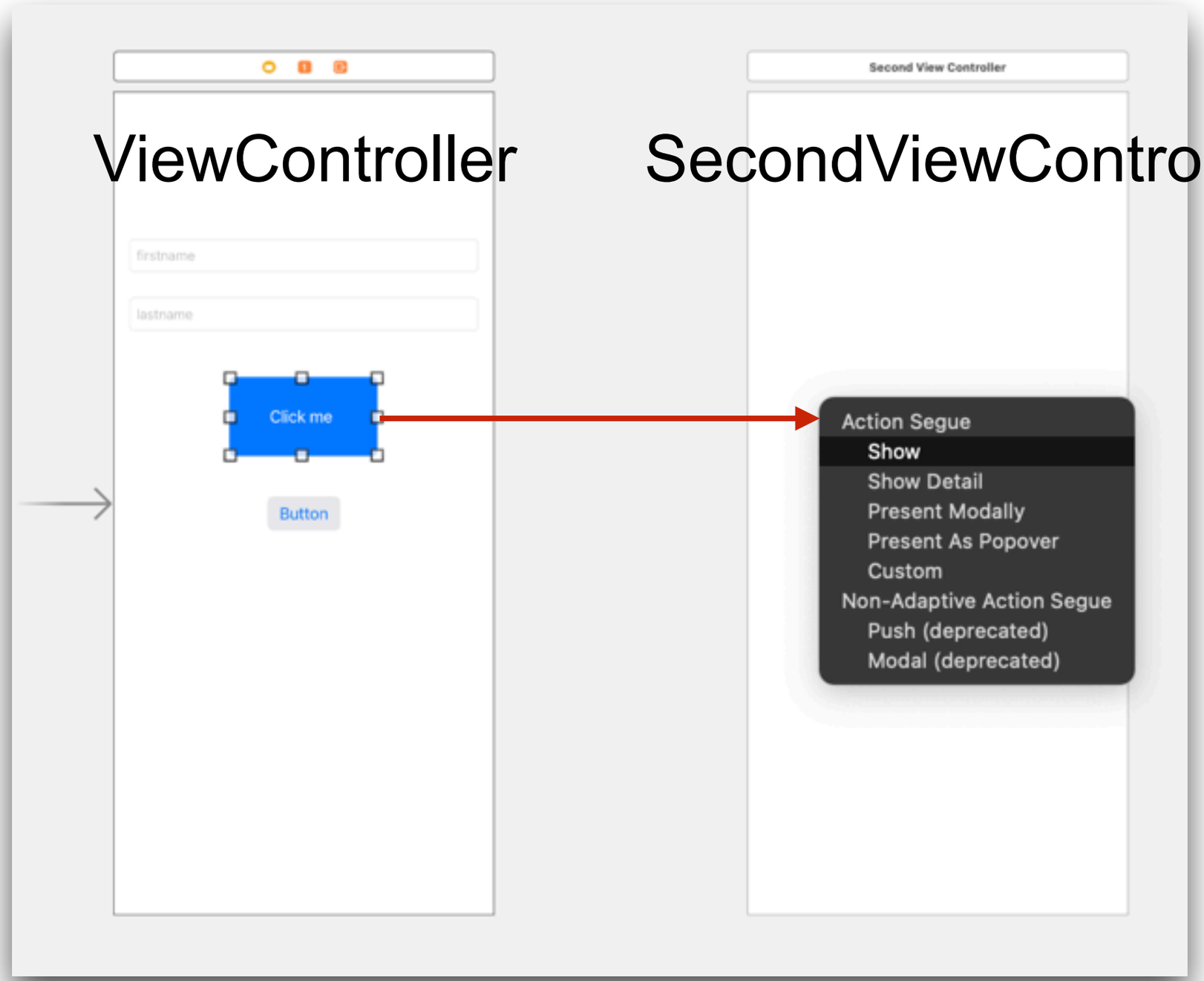
Class = SecondViewController



Mapping VC class in storyboard



Create segue in storyboard



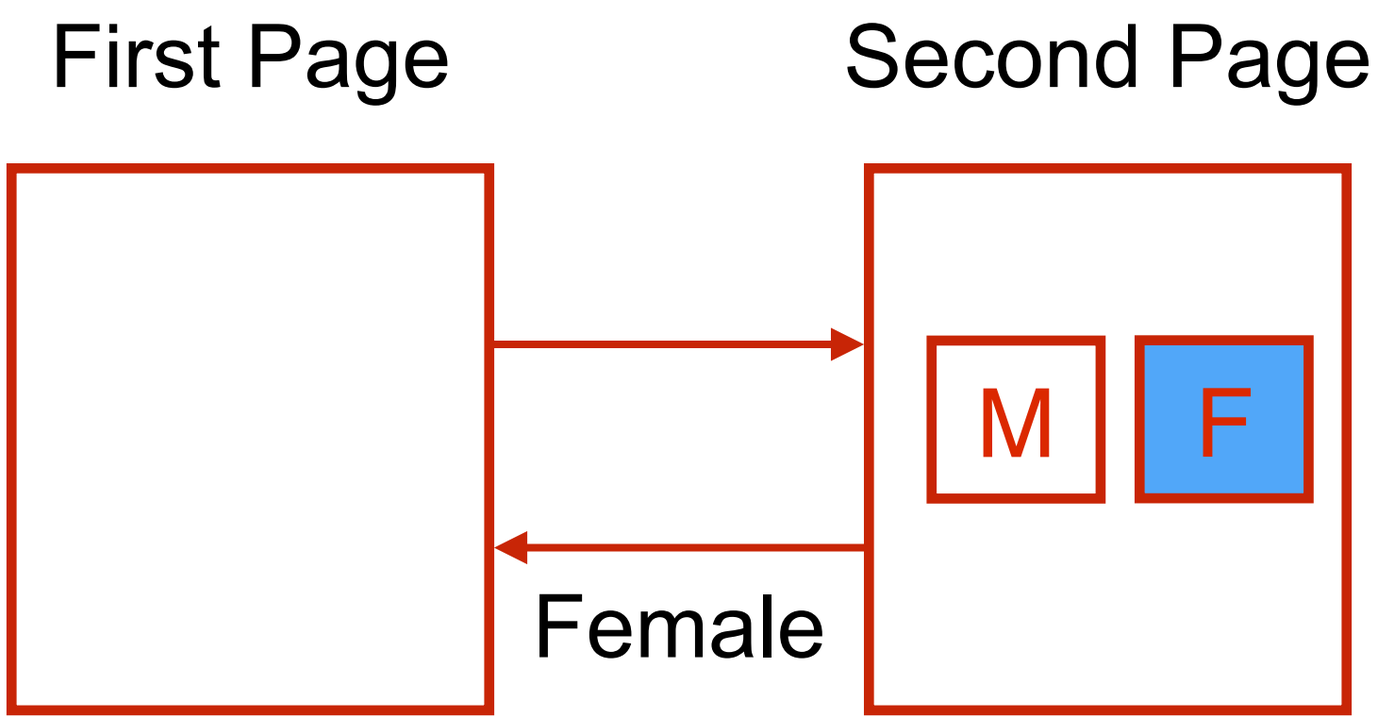
```swift
class ViewController: UIViewController {

    @IBOutlet weak var firstnameTF: UITextField!

    @IBOutlet weak var lastnameTF: UITextField!

    override func prepare(for segue: UIStoryboardSegue, sender:
        Any?) {
        let username = firstnameTF.text ?? ""
        let destVC = segue.destination as! SecondViewController
        destVC.username = username
    }
```

```swift
class SecondViewController: UIViewController {

    @IBOutlet weak var messageLabel: UILabel!

    var username: String = ""

    override func viewDidLoad() {
        super.viewDidLoad()

        messageLabel.text = "Hello, " + username
    }
```

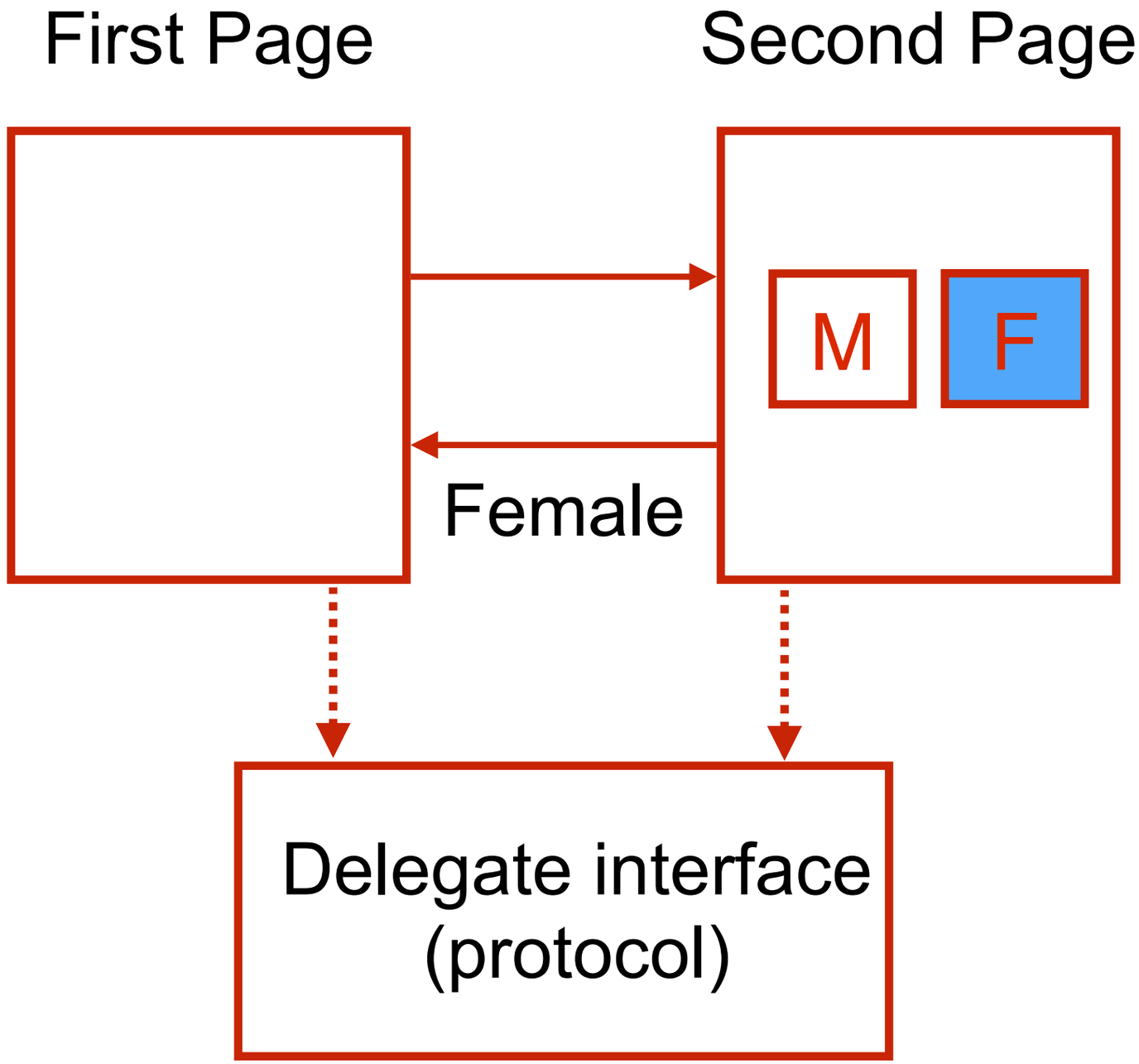
Delegate pattern

Delegate design pattern

Exchange data between objects
Allow object to communicate back with owner



Delegate design pattern

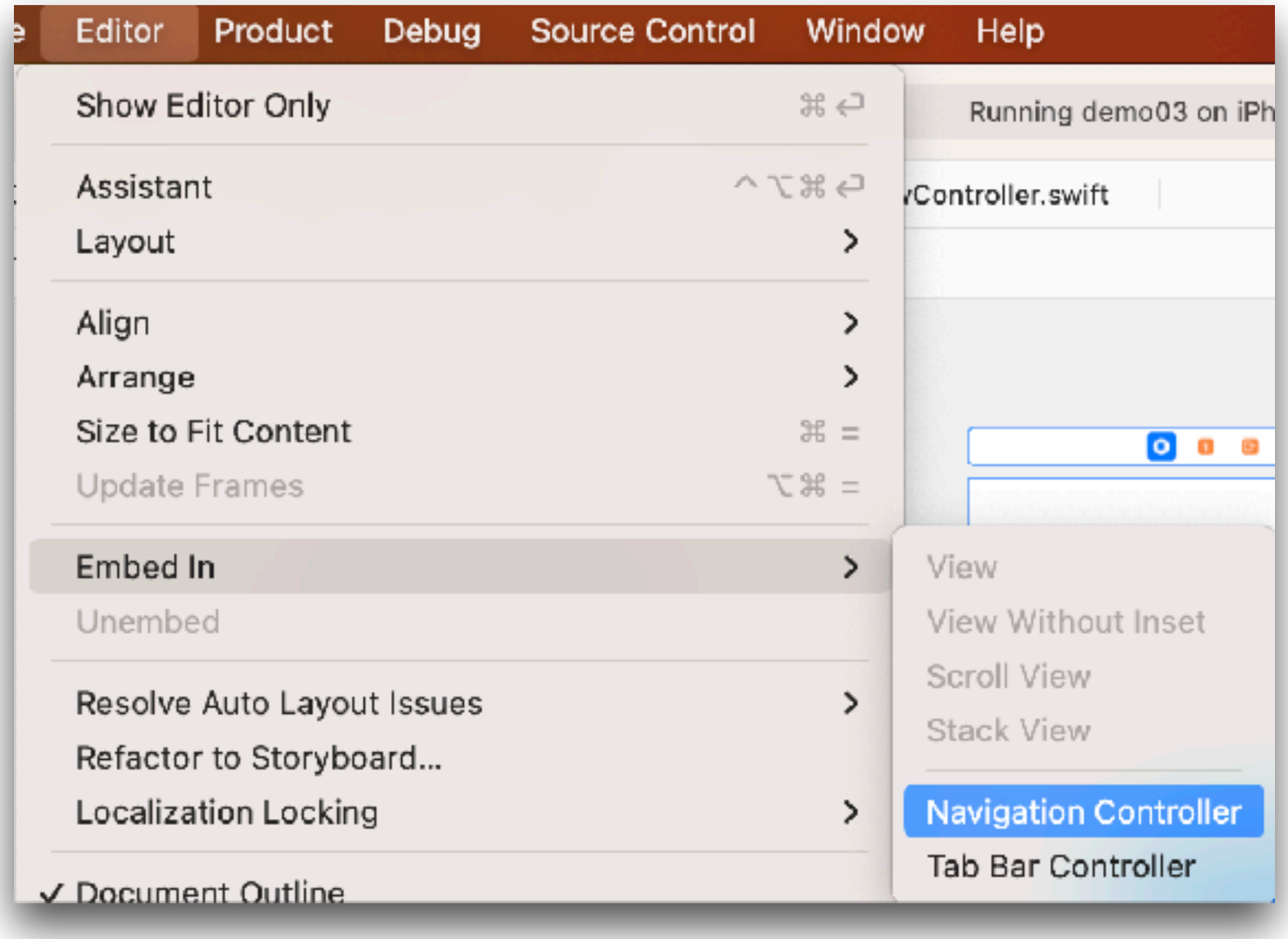


ChooseDelete with Protocol

```
protocol ChooseDelegate : AnyObject {

    func genderChoosen( gender: String)

}
```

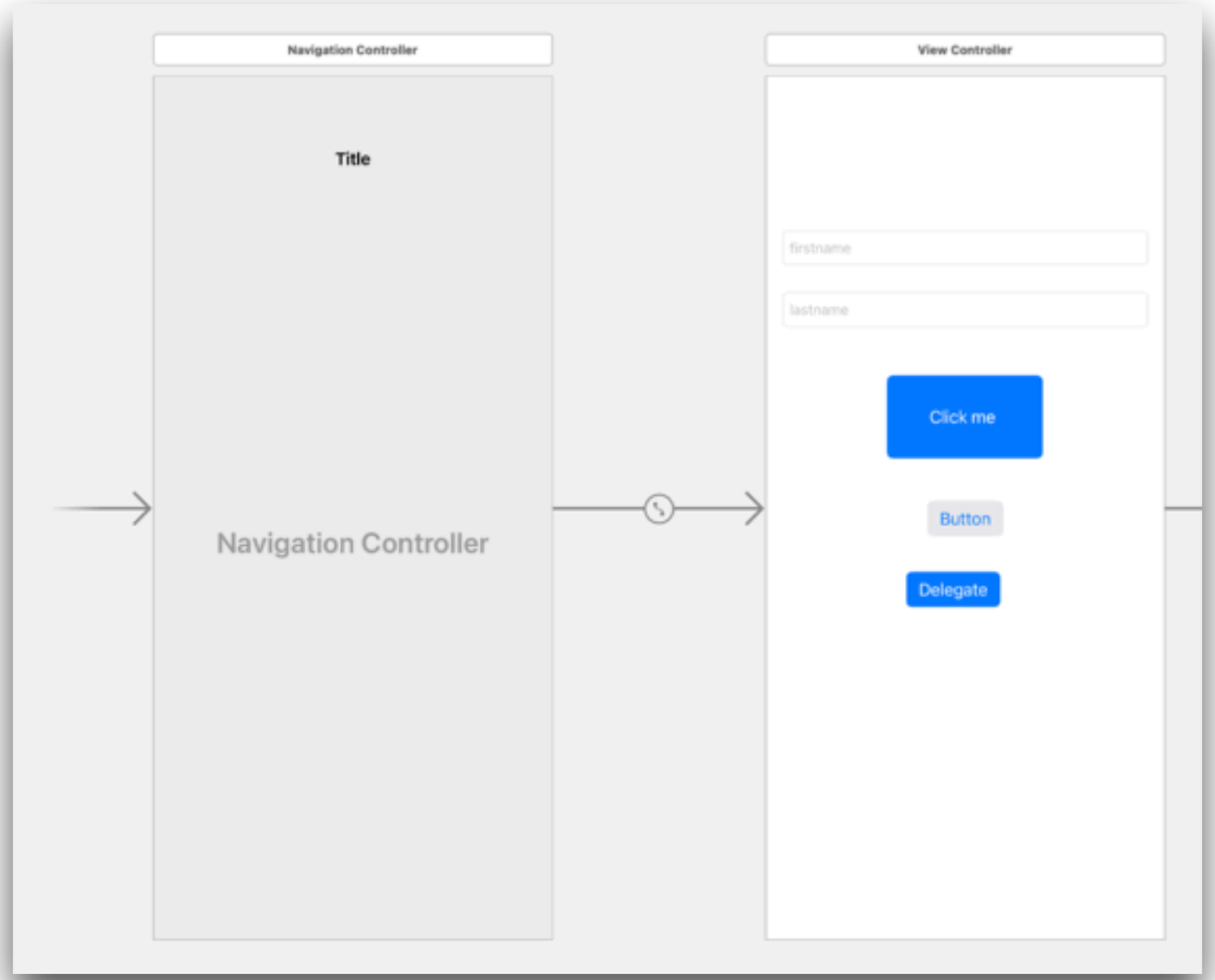
Create Navigation Controller

Editor -> Embed In



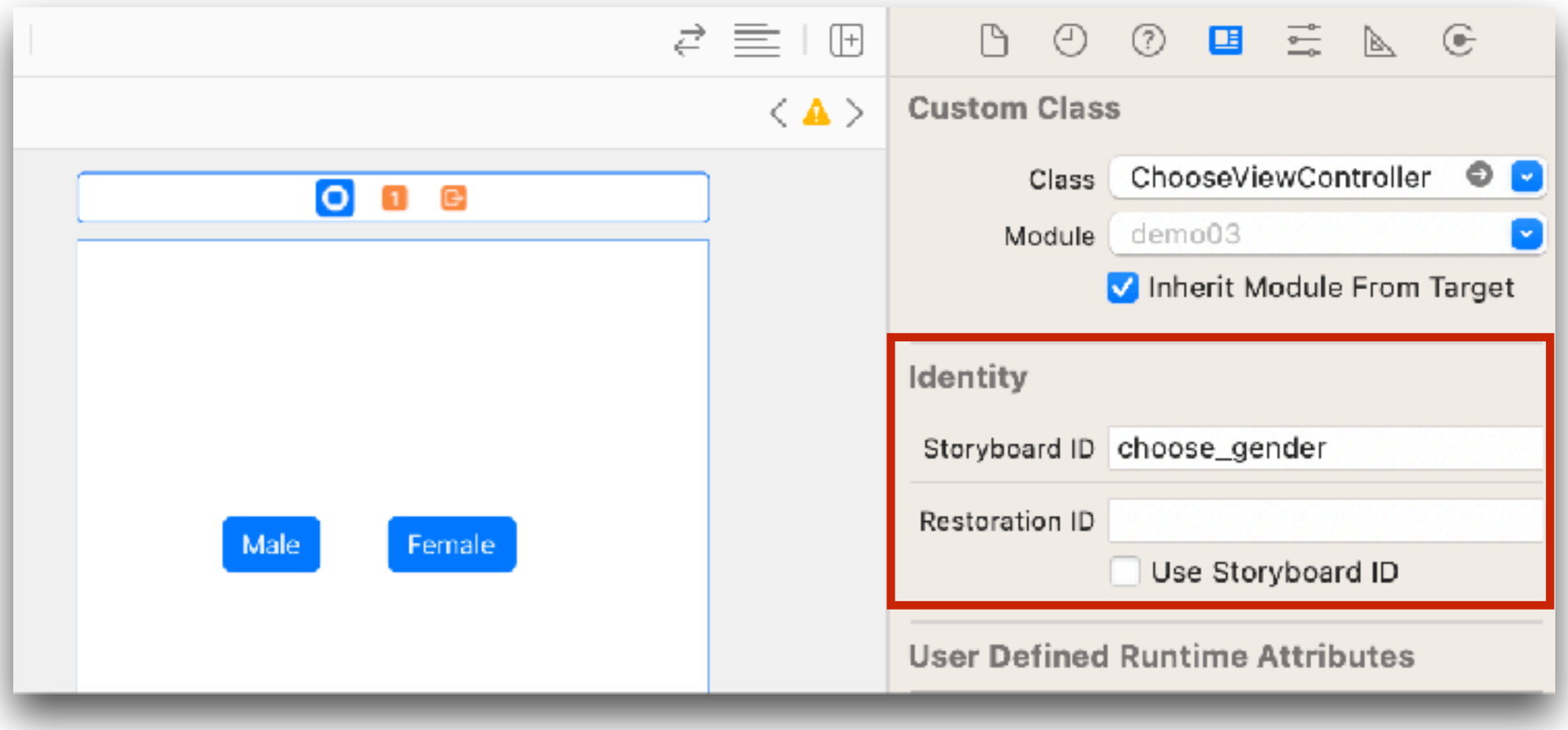
Create Navigation Controller

In ViewController



Create ChooseViewController

Identifier = choose_gender



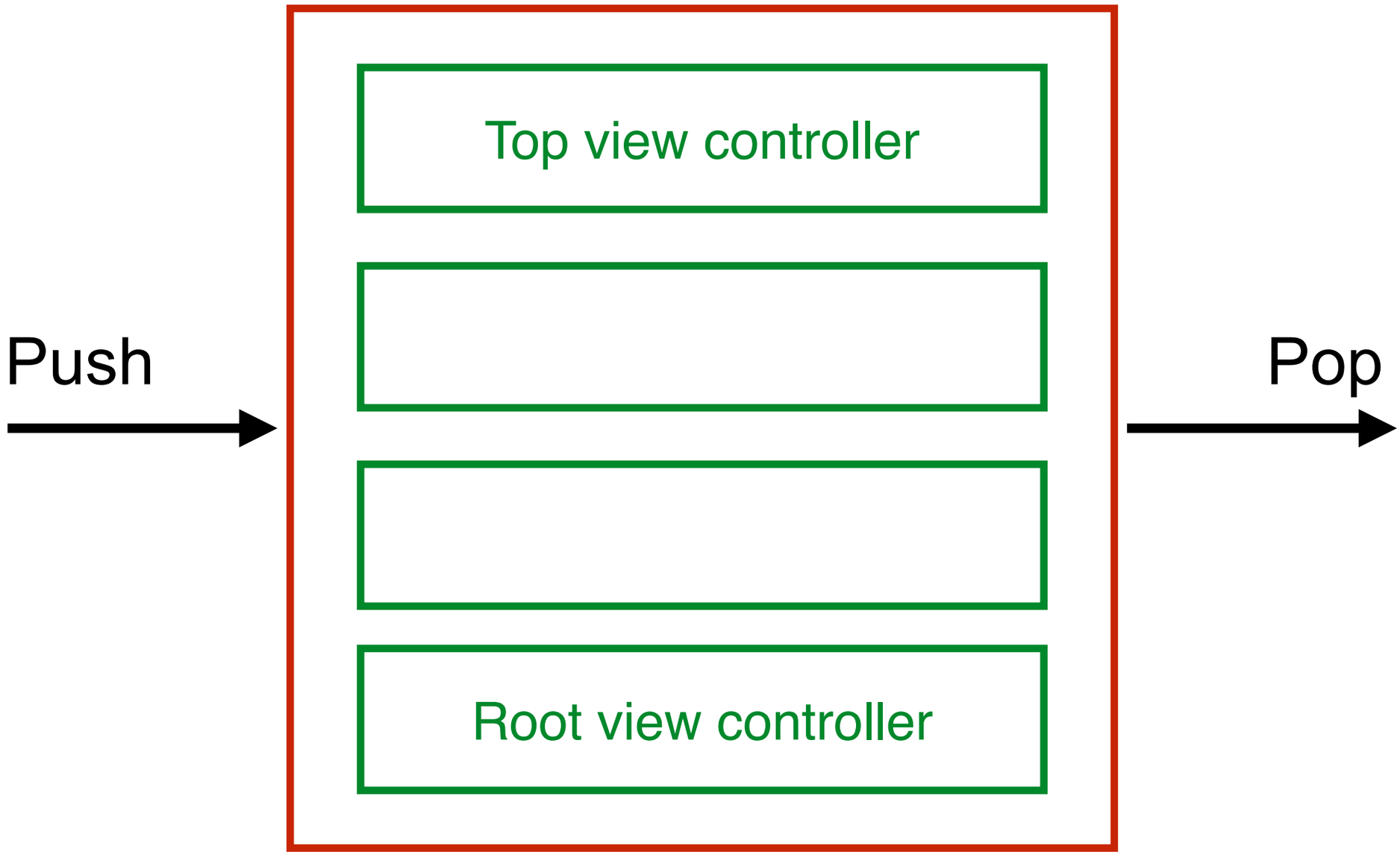
ChooseViewController

```
class ChooseViewController: UIViewController {

    weak var delegate: ChooseDelegate?

    @IBAction func pressGenderButton(_ sender: Any) {

        delegate?.genderChoosen(gender: "Female")

        // Return to previous VC
        navigationController?.popViewController(animated: true)
        dismiss(animated: true, completion: nil)
    }

}
```

ViewController

```
class ViewController: UIViewController, ChooseDelegate {

    @IBAction func onPressDelete(_ sender: Any) {

        if let vc = storyboard?.instantiateViewController(
                withIdentifier: "choose_gender") as? ChooseViewController {

                    vc.delegate = self
                    self.navigationController?.pushViewController(vc, animated: true)
                }
    }

    func genderChoosen(gender: String) {
        print("Choosen gender = " + gender)
    }

}
```

Navigation Stack



https://cocoacasts.com/uikit-fundamentals-1-how-to-get-the-root-view-controller-of-a-navigation-controller

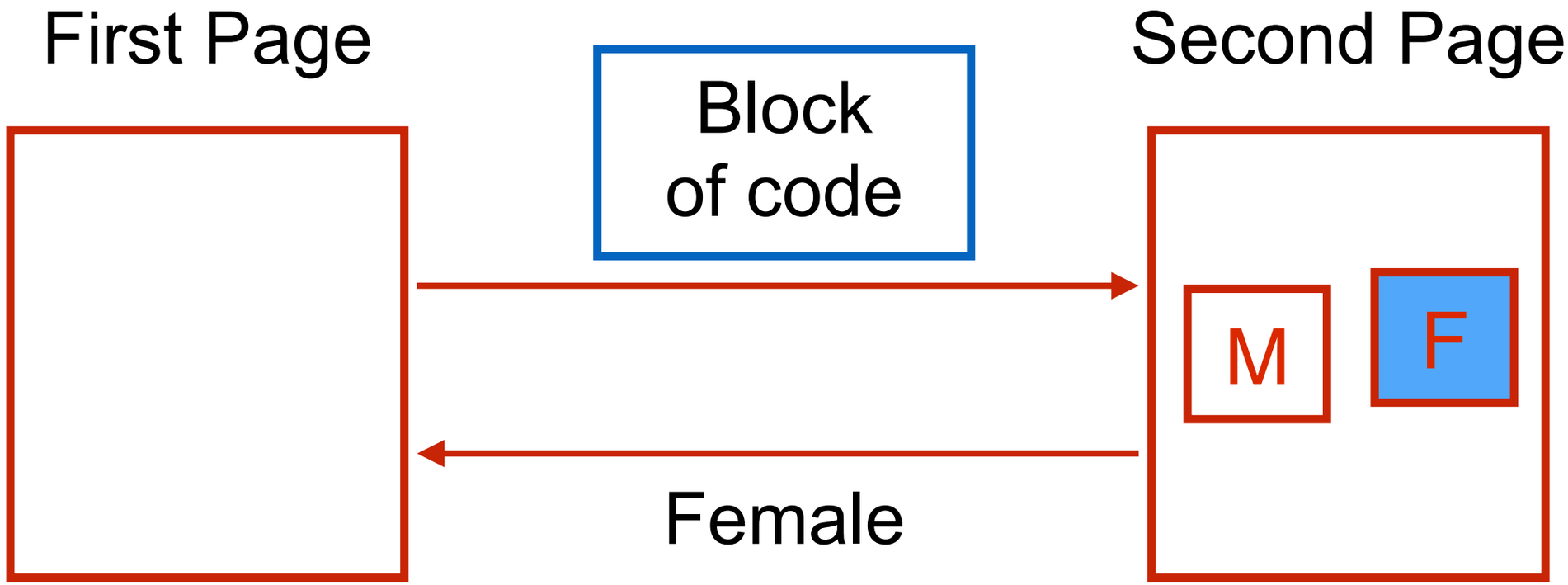
Closure

Closure

Block of code that passed in code to share data
Use closure in dismiss()

```
class ChooseViewController: UIViewController {

    @IBAction func pressGenderButton(_ sender: Any) {

        // Return to previous VC
        navigationController?.popViewController(animated: true)

        dismiss(animated: true, completion: nil)
    }

}
```

Closure



ChooseViewController

```
class ChooseViewController: UIViewController {

    var change : ((String) -> Void)?

    @IBAction func pressGenderButton(_ sender: Any) {
        navigationController?.popViewController(animated: true)

        dismiss(animated: true, completion: {
            self.change!("XXX")
        })

    }

}
```

Closure in swift

ViewController

```
class ViewController: UIViewController {

    @IBAction func onPressDelete(_ sender: Any) {
        vc.change = setGender
    }

func setGender(gender: String) {
    print("Set gender = " + gender)
}
```

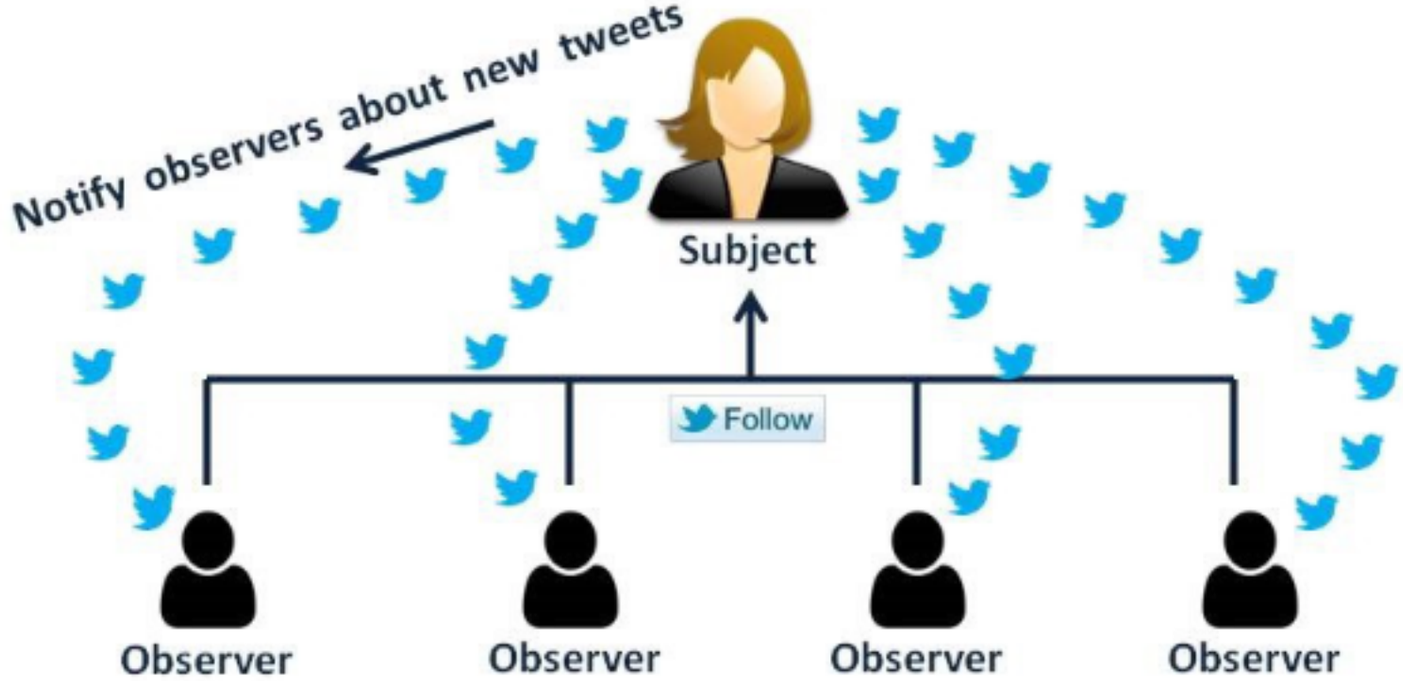
Notification Center

Notification Center

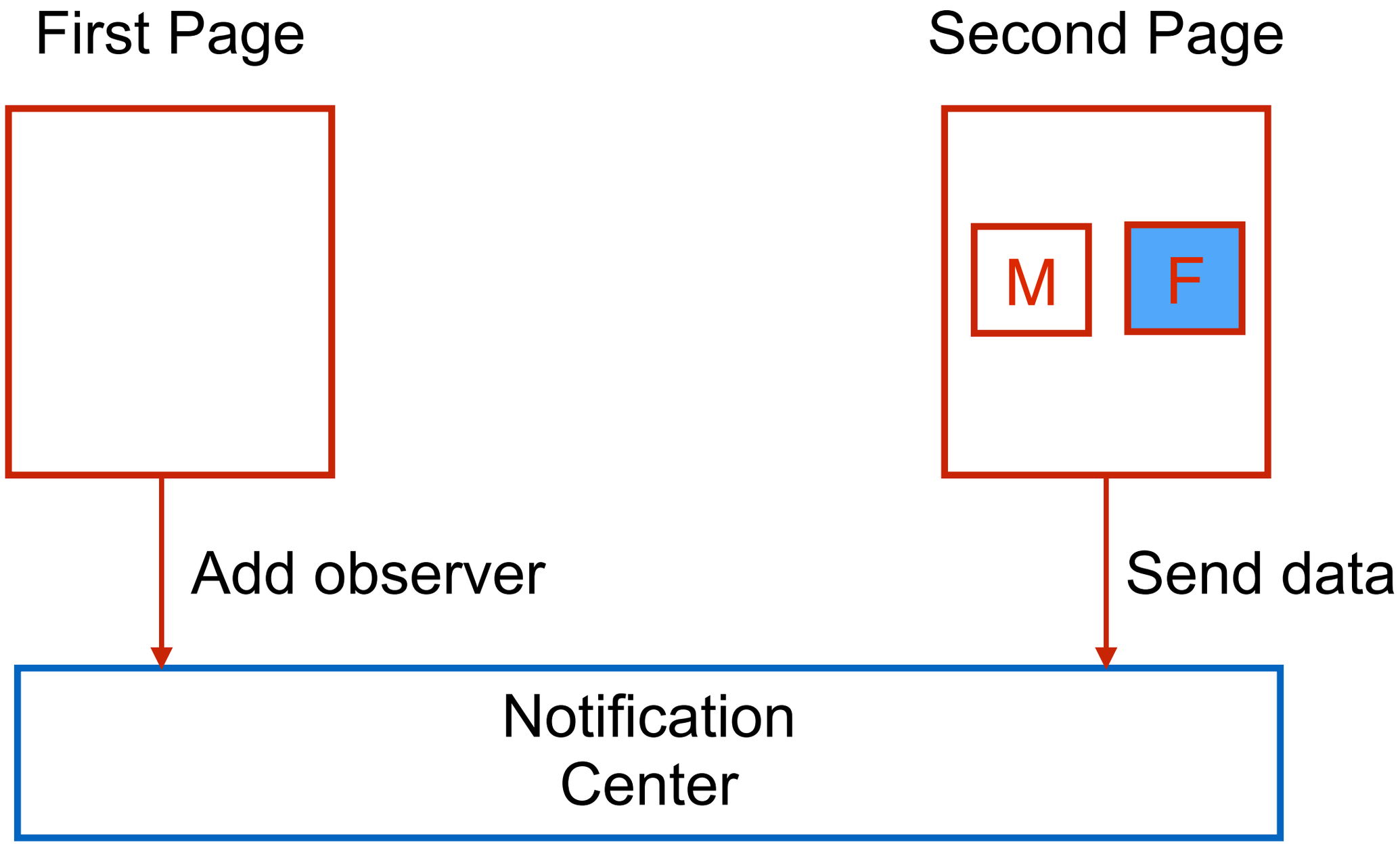
Observer design pattern

Decouple between sender and receiver

Global scope !!



Notification Center



Add Observer

```
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        NotificationCenter.default.addObserver(self,
          selector: #selector(didReceiveData(_:)),
          name: Notification.Name("DataNotification"), object: nil)
    }

    @objc func didReceiveData(_ notification: Notification) {
        if let data = notification.userInfo?["data"] as? String {
            setGender(gender: data)
        }
    }

    deinit {
        NotificationCenter.default.removeObserver(self)
    }

}
```

Remove Observer

```
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        NotificationCenter.default.addObserver(self,
          selector: #selector(didReceiveData(_:)),
          name: Notification.Name("DataNotification"), object: nil)
    }

    @objc func didReceiveData(_ notification: Notification) {
        if let data = notification.userInfo?["data"] as? String {
            setGender(gender: data)
        }
    }

    deinit {
        NotificationCenter.default.removeObserver(self)
    }

}
```

Send data to NotificationCenter

```
class ChooseViewController: UIViewController {

    @IBAction func pressGenderButton(_ sender: Any) {

        let data = "Some data"
        NotificationCenter.default.post(name: Notification.Name("DataNotification"),
             object: nil, userInfo: ["data": data])

    }

}
```

Singleton

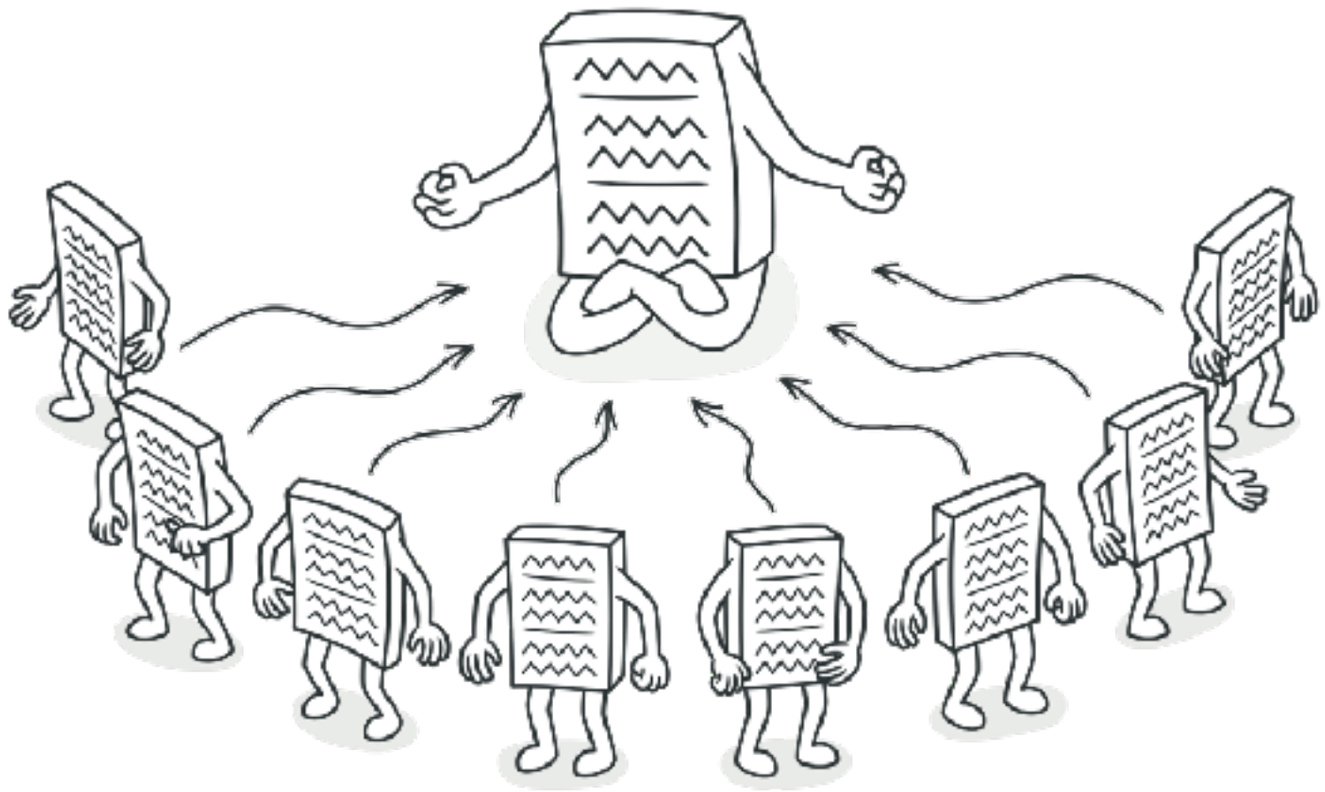
Singleton design pattern

Guarantee only one instance for class

Access anywhere in app

Global scope !!

Easy solution but be carefully !!



Singleton class

```
class MySharedData {

    static let instance = MySharedData()

    // Private constructor
    private init(){

    }

    // Global variables and function
    var isOpen = true

    func getDataByKey(key: String) -> String {
        return ""
    }
}
```

Manage Dependencies

Manage dependencies

Dependencies

Name	Description
Moya	Network
Alamorefire	Network
RxSwift	Reactive programming with Swift
Realm	Database for mobile
KeychainAccess	Wrapper for Keychain (keep secure user data)
IOSSEcuritySuite	Security Detect jailbreak

Working with CocoaPods

CocoaPods

Dependency Manager for Swift/Obj-C
Build with Ruby

https://cocoapods.org/

Install Ruby with RVM

Command line tool to manage Ruby

https://rvm.io/

Installation

$sudo gem install cocoapods
$pod --version

https://cocoapods.org/

Use dependency with CocoaPods

$pod init
Edit file **Podfile**

$pod install

Open file .xcworkspace in Xcode

https://cocoapods.org/

Use dependency with CocoaPods

$pod init
Edit file **Podfile**

$pod install

Open file .xcworkspace in Xcode

https://cocoapods.org/

Podfile

Specification file

Define dependencies of targets in Xcode project

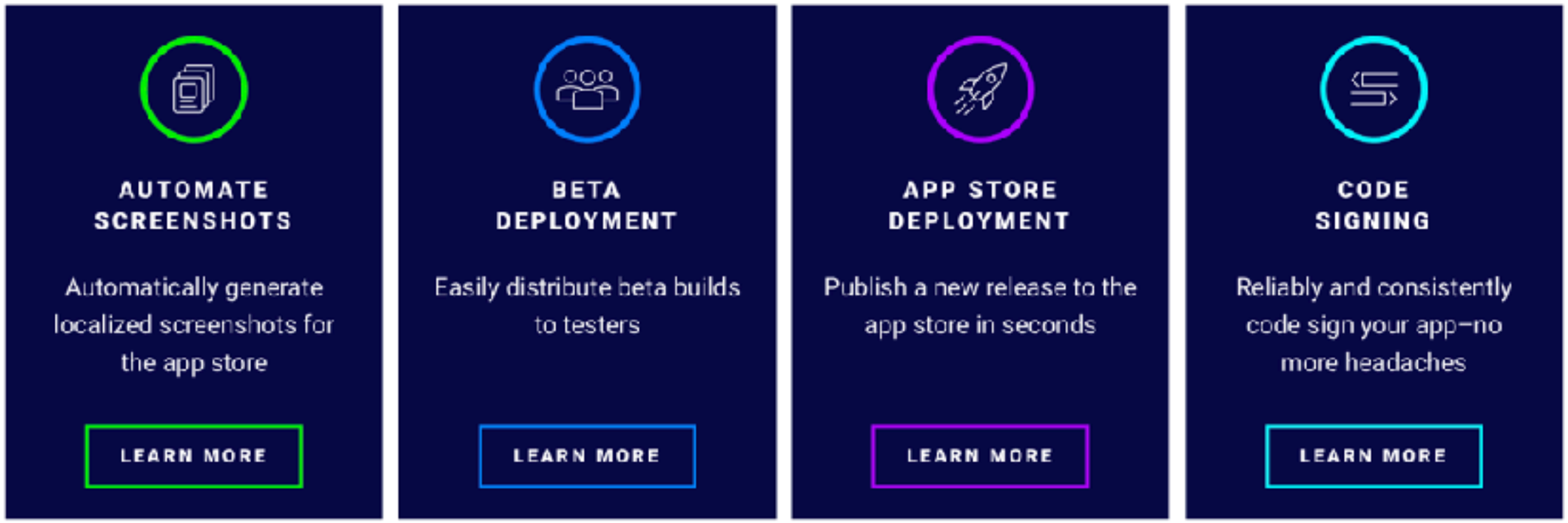
Podfile

```
target 'DemoDay1' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for DemoDay1

  target 'DemoDay1Tests' do
    inherit! :search_paths
    # Pods for testing
  end

  target 'DemoDay1UITests' do
    # Pods for testing
  end

end
```

Working with Fastlane

Fastlane

Automation tools for Mobile app
iOS and Android



https://fastlane.tools/

Installation

$brew install fastlane

$fastlane init

Podfile

```
target 'DemoDay1' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for DemoDay1

  pod 'Moya', '~> 15.0'

  target 'DemoDay1Tests' do
    inherit! :search_paths
    # Pods for testing
  end

  target 'DemoDay1UITests' do
    # Pods for testing
  end

end
```

Install dependencies

$pod install

```
Analyzing dependencies
Downloading dependencies
Installing Alamofire (5.6.4)
Installing Moya (15.0.0)
Generating Pods project
Integrating client project
```

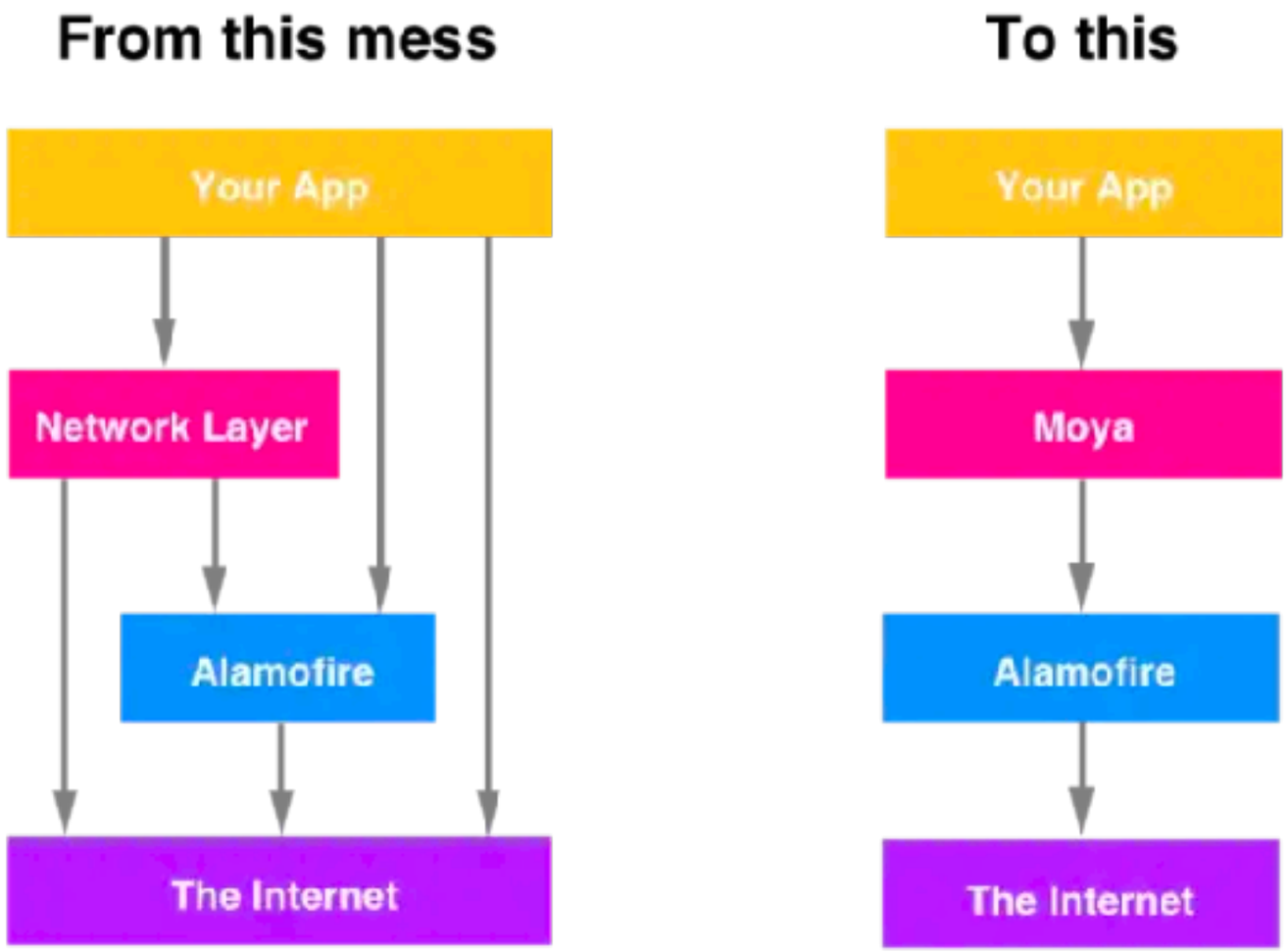
```
DemoDay1                DemoDay1Tests           Podfile.lock
DemoDay1.xcodeproj      DemoDay1UITests         Pods
DemoDay1.xcworkspace    Podfile
```

Networking

Working with Network

Moya

Alamofire



https://github.com/Moya/Moya

Podfile

```
pod 'Moya', '~> 15.0'
pod 'SwiftyJSON', '~> 4.0'
pod 'Moya/RxSwift', '~> 15.0'
pod 'Moya/ReactiveSwift', '~> 15.0'
```

Working with Alamofire

```
class MyAPI {
    func fetchUsers() {
        print("Fetch")

        AF.request("https://jsonplaceholder.typicode.com/photos?_limit=5")
            .validate()
            .responseJSON { response in
                switch response.result {
                case .success(let value):
                    print("JSON: \(value)")
                case .failure(let error):
                    print("Error: \(error)")
                }
            }
    }
}
```

Working with Moya

Split implementation

```
class MyAPI {

    func fetchUsersWithMoya(){
        let provider = MoyaProvider<MyService>()
        provider.request(.photos(limit: 5)) { result in
            switch result {
            case let .success(response):
                do {
                    let json = try JSON(response.data)
                    print("JSON: \(json)")
                } catch {
                    print("Error parsing JSON: \(error)")
                }
            case let .failure(error):
                print("Error: \(error)")
            }
        }
    }
```

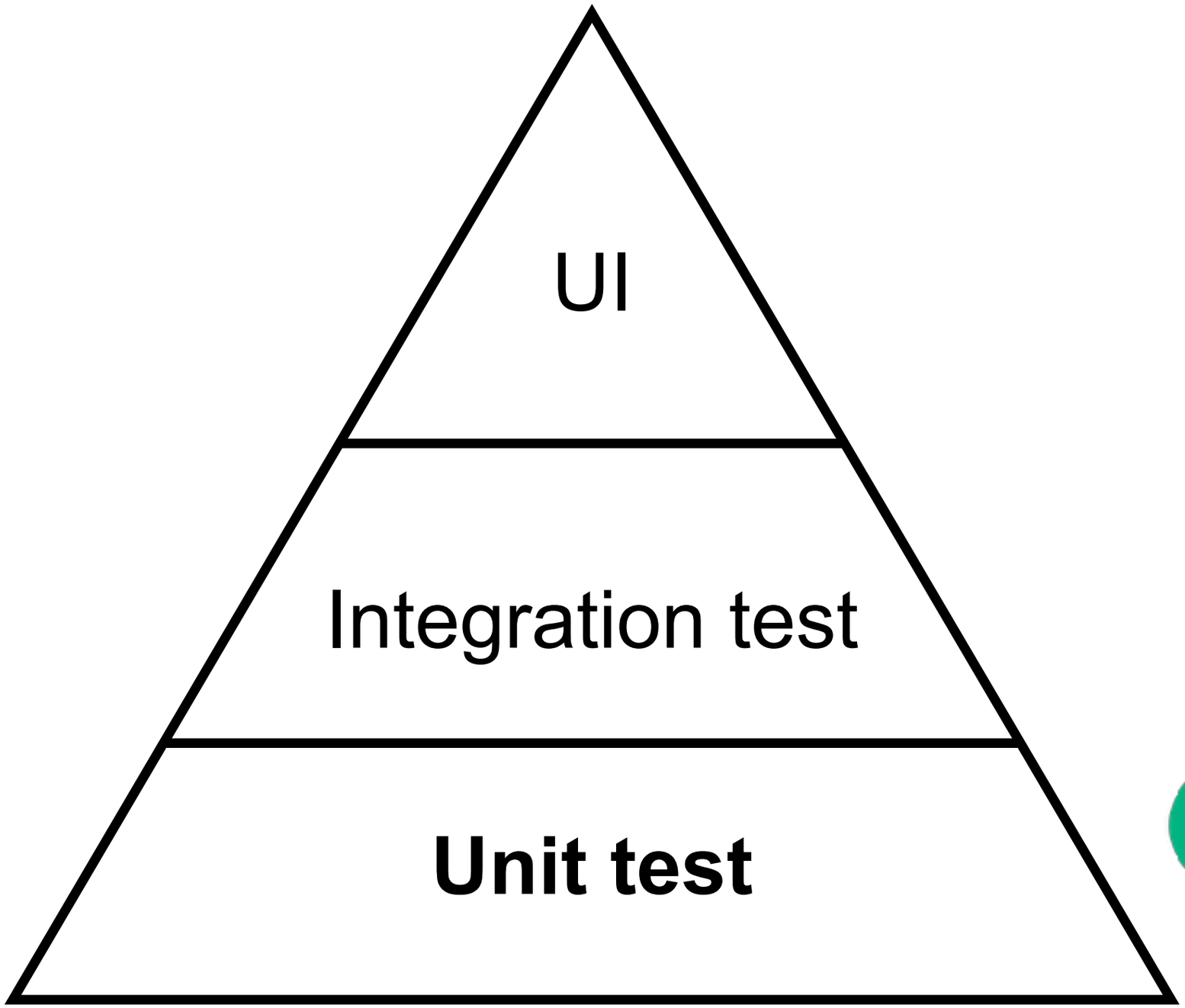
MyService ?

```
enum MyService {
    case photos(limit: Int)
}
```

```
extension MyService: TargetType {
    var baseURL: URL {
        return URL(string: “url of api")!
    }
    var path: String {
        switch self {
        case .photos:
            return "/photos"
        }
    }
    var task: Task {
        switch self {
        case let .photos(limit):
            return .requestParameters(
                parameters: ["_limit": limit],
                encoding: URLEncoding.queryString)
        }
    }
}
```

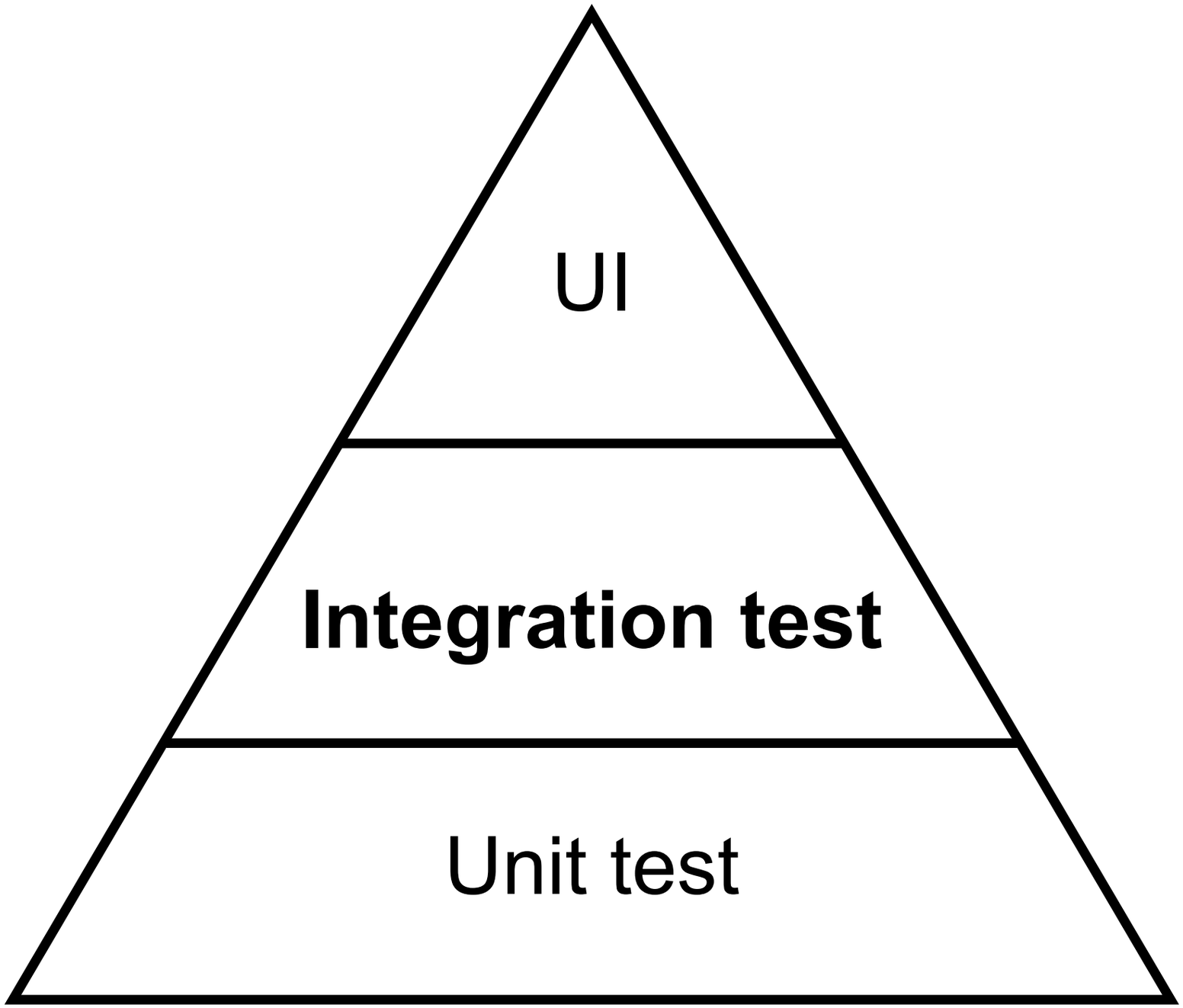
All about Testing

Testing in iOS app ?

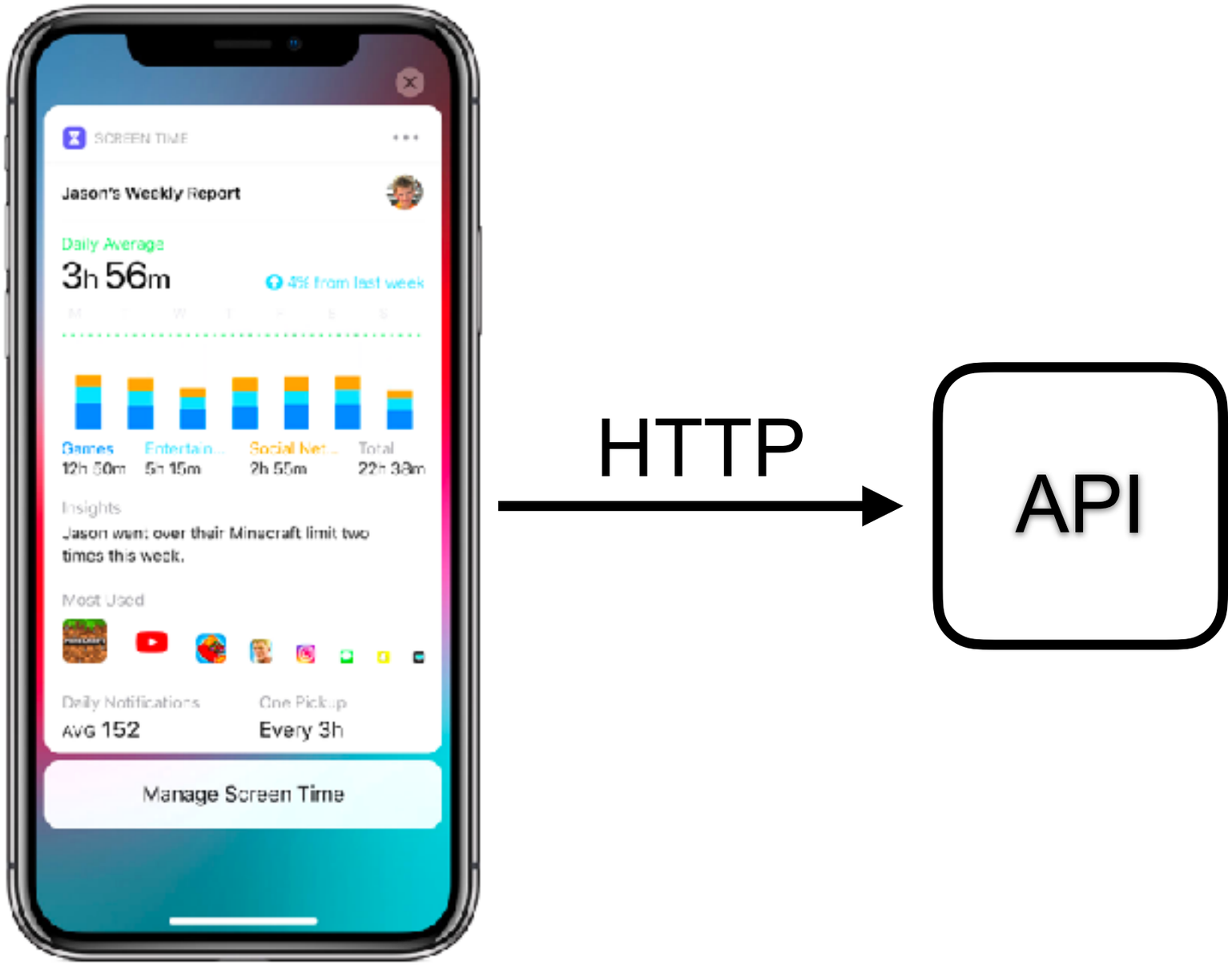


Nimble

Unit XCTest



Integration test ?

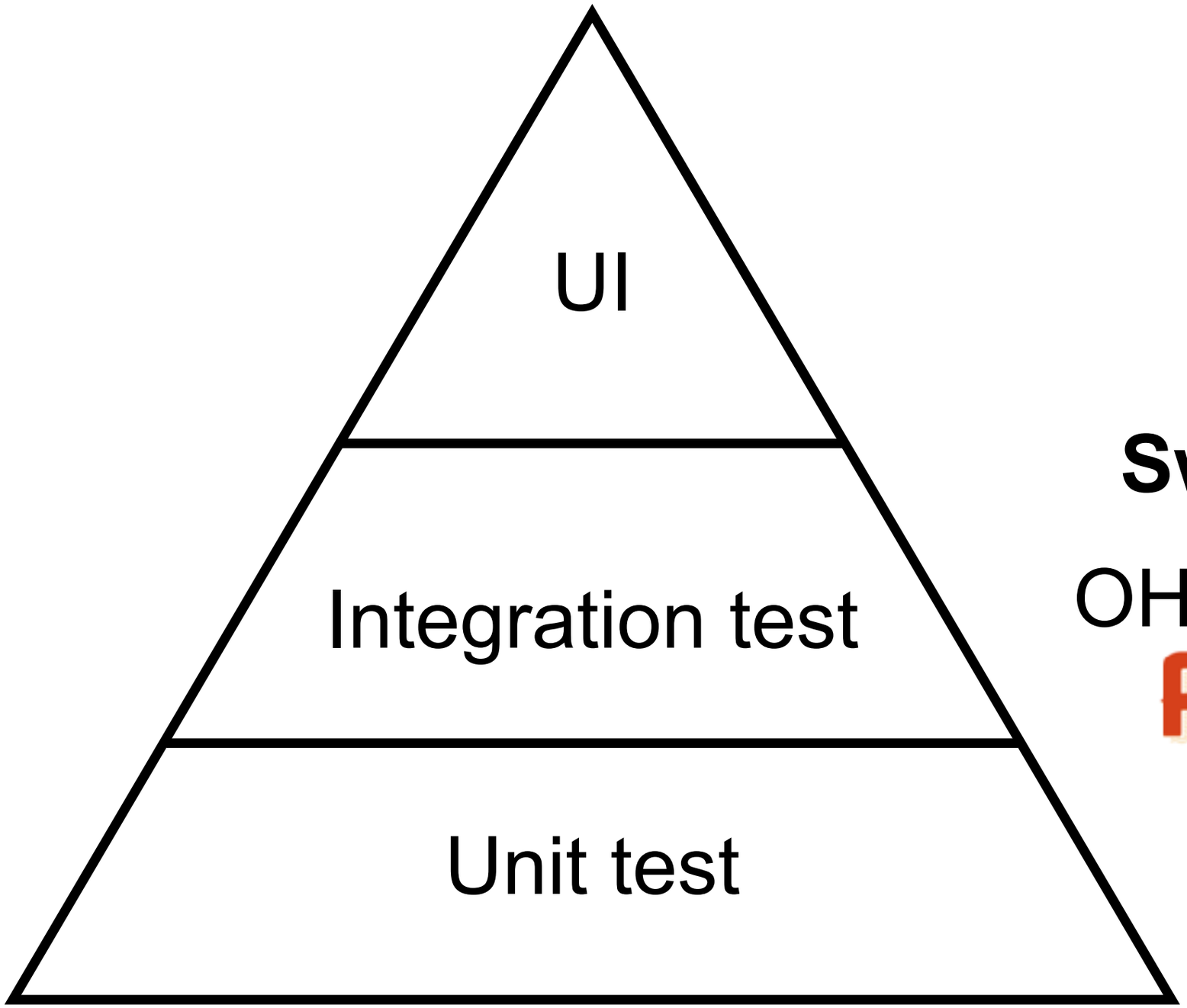


Integration test ?

Network request
UserDefaults
Core Data
Singleton objects

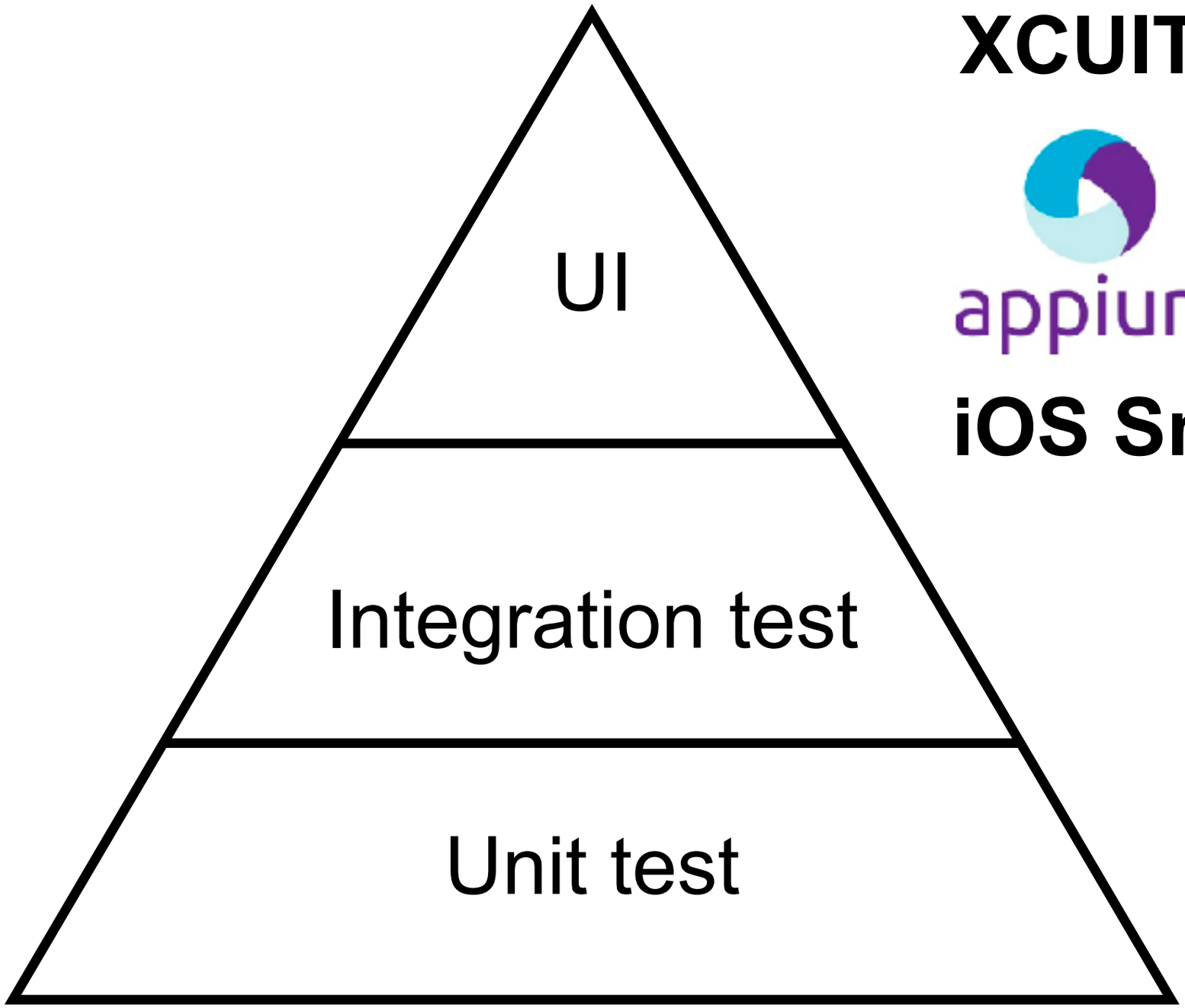
Singleton object ?

NSFileManager
NSApplication
UIApplication
etc …



SwiftLocalhost

OHHTTPStubs with ALAMOFIRE

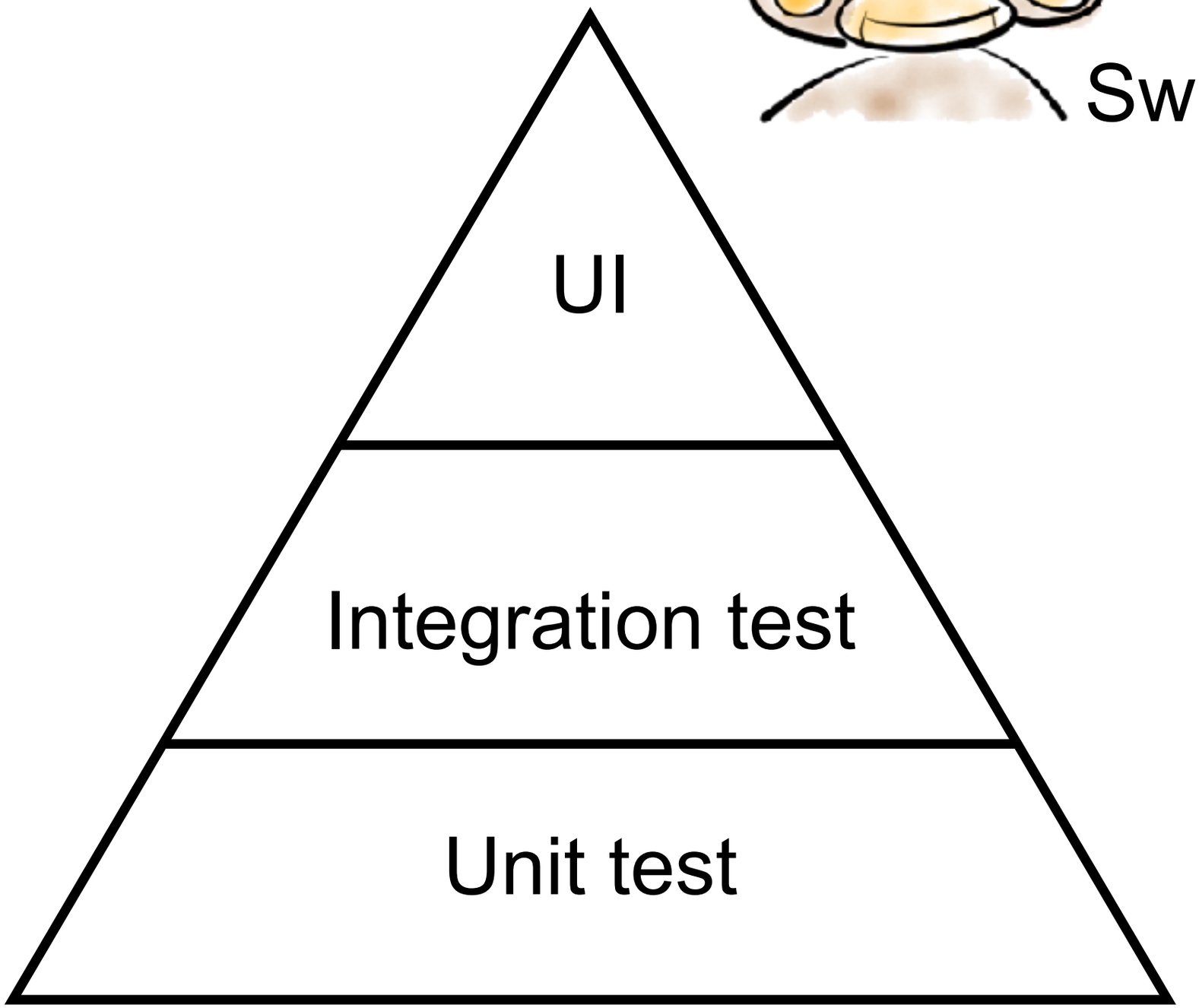


XCUITest

KIF

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

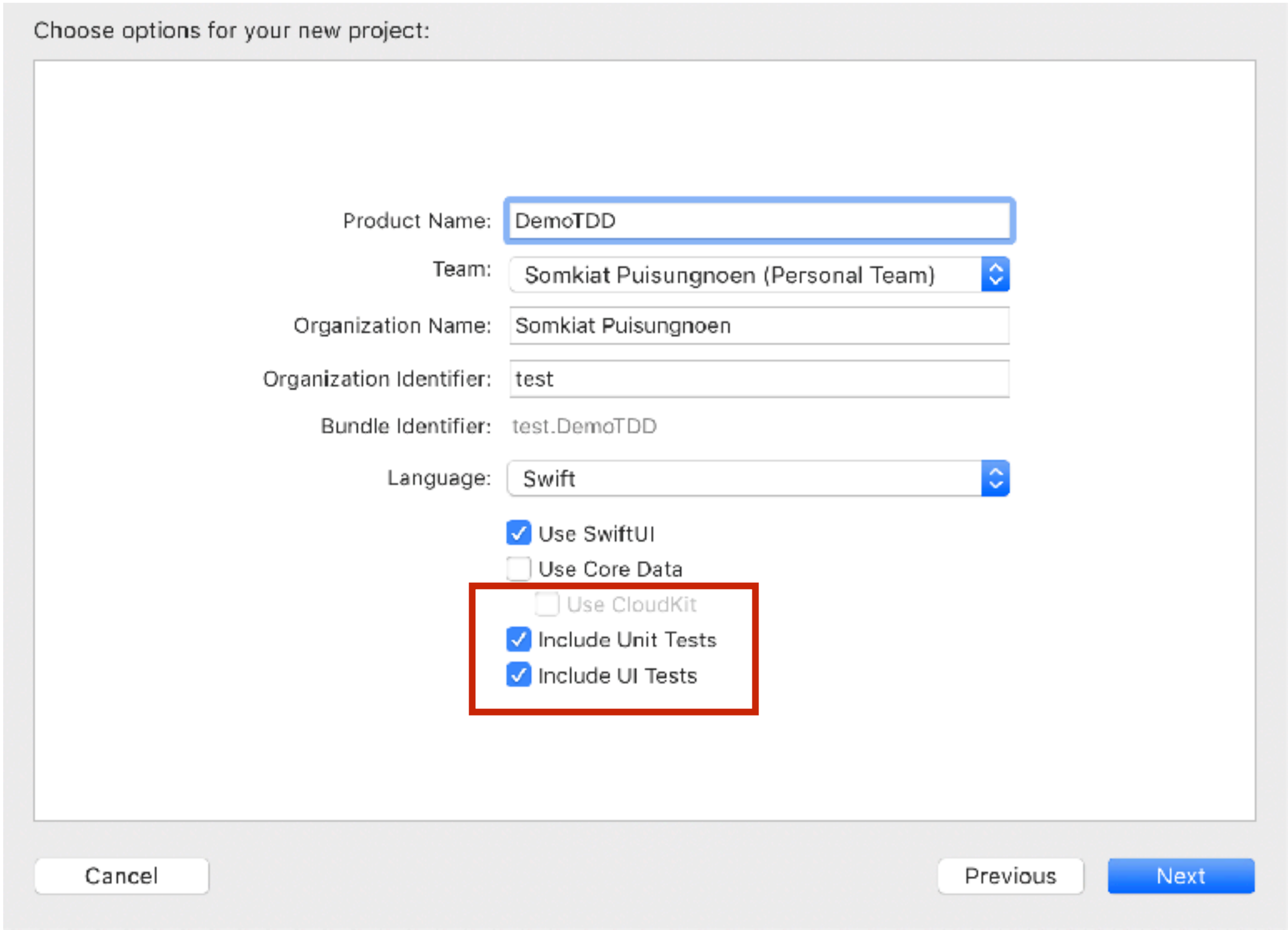
Self-validate

Thorough/Timely

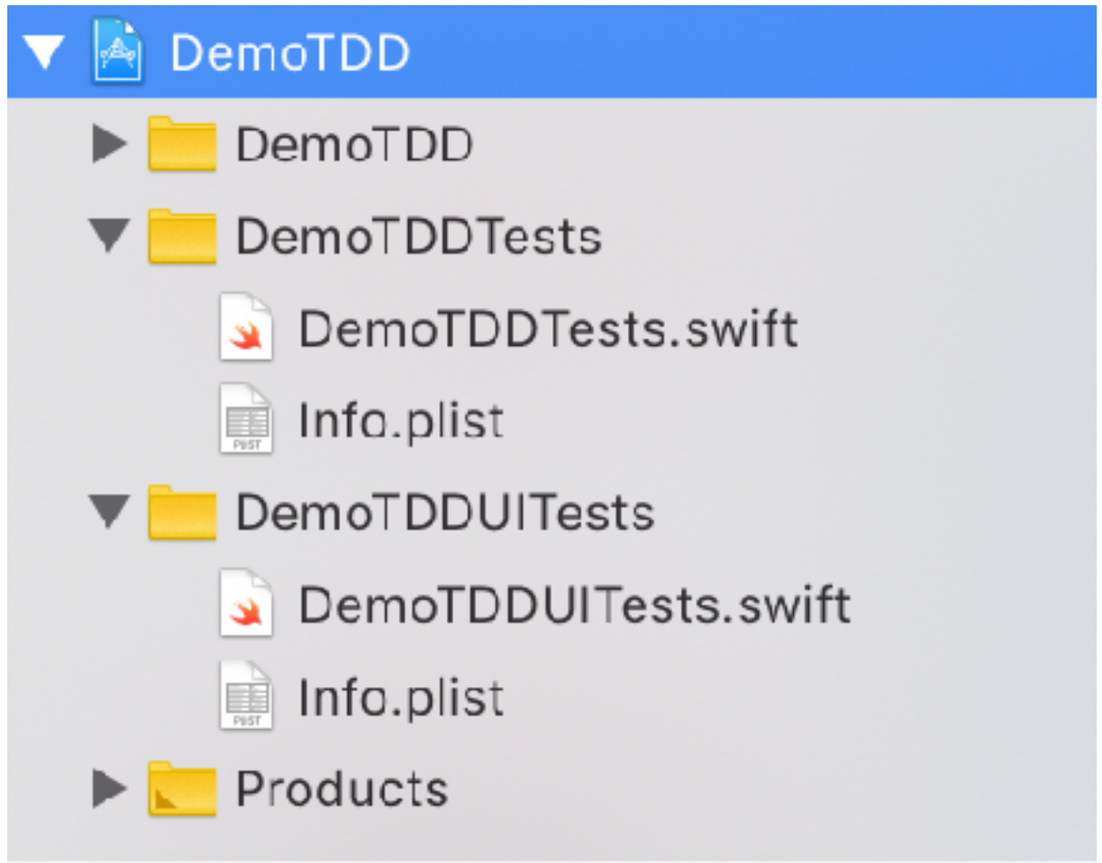
Not Unit test if …

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

```
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.
        }
    }

}
```

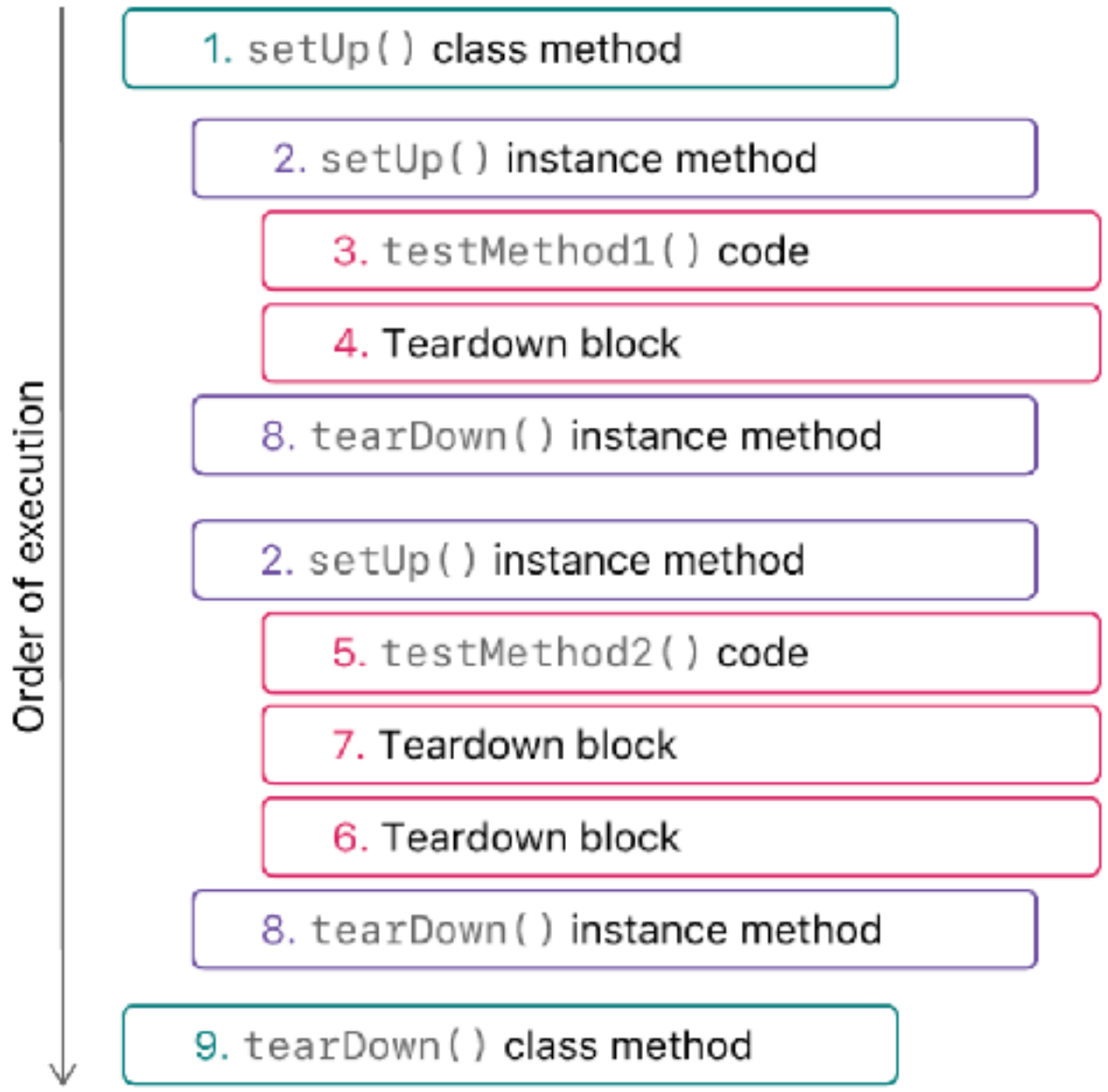
1 Import XCTest

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)
}
```

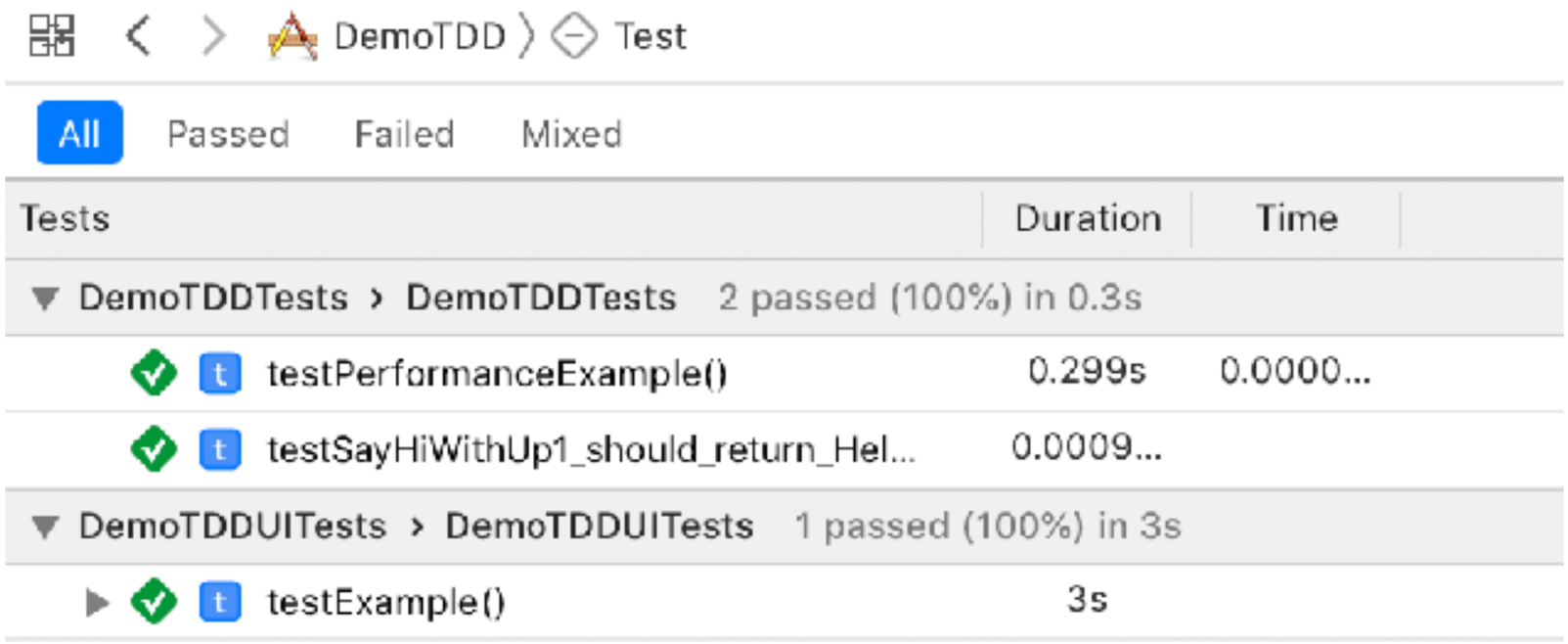
Assertion in XCTest

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

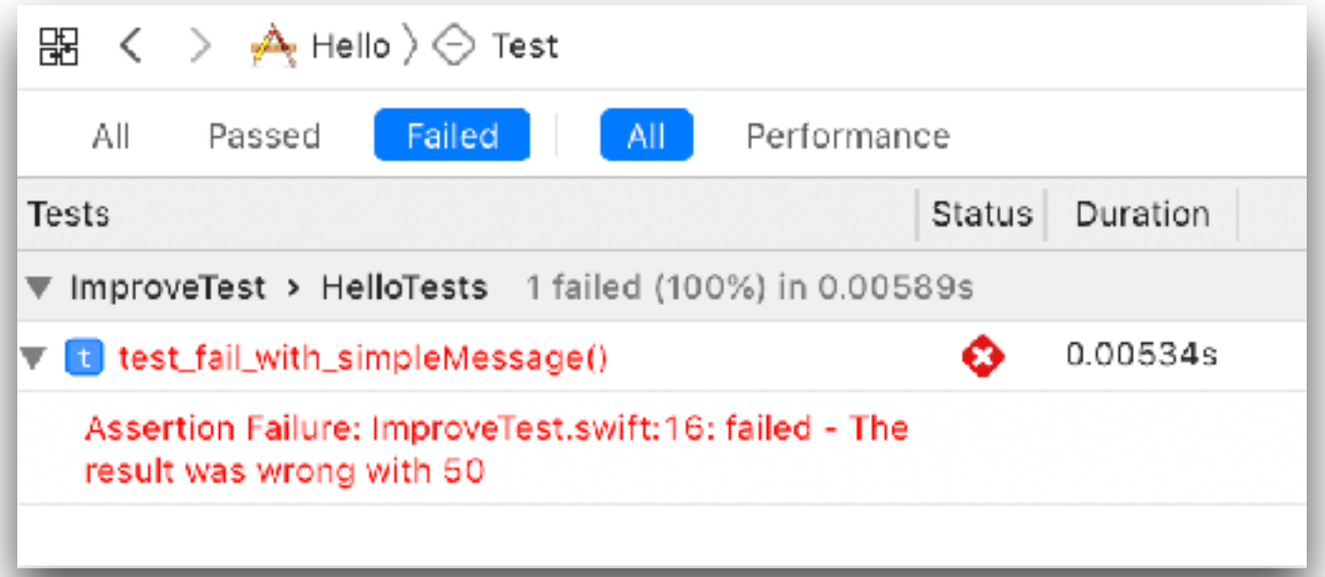


Write a second test case

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).
```

Working with Error

```
class MyRange {
    var input: String
    init(input: String) throws {
        self.input = input
        if !startWithInclude() {
            throw MyRange.InputError.emptyStartText
        }
    }
}

extension MyRange {
    enum InputError: Error{
        case emptyStartText
    }
}
```

https://docs.swift.org/swift-book/LanguageGuide/ErrorHandling.html

Testing with Error

```
func testErrorWithOutStart() {

        var thrownError: Error?

        XCTAssertThrowsError(try MyRange(input: "1,5]")) {
            thrownError = $0
        }

        XCTAssertTrue(
            thrownError is MyRange.InputError,
            "Unexpected error type: \(type(of: thrownError))"
        )

        XCTAssertEqual(thrownError as? MyRange.InputError, .emptyStartText)

}
```

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

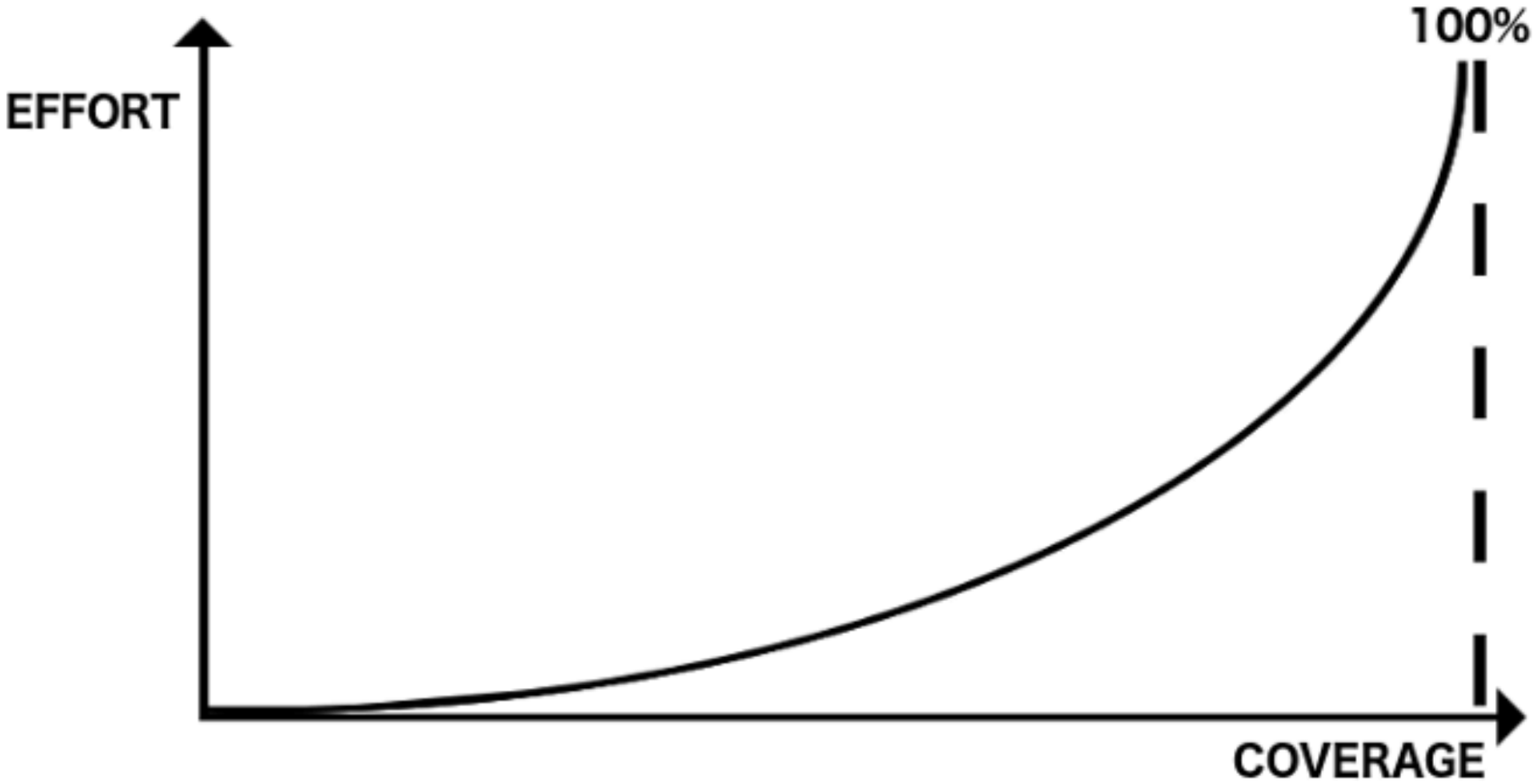
But 100% of code coverage **does not mean** that your code is 100% correct

Code coverage

Powerful tool to improve the quality of your code

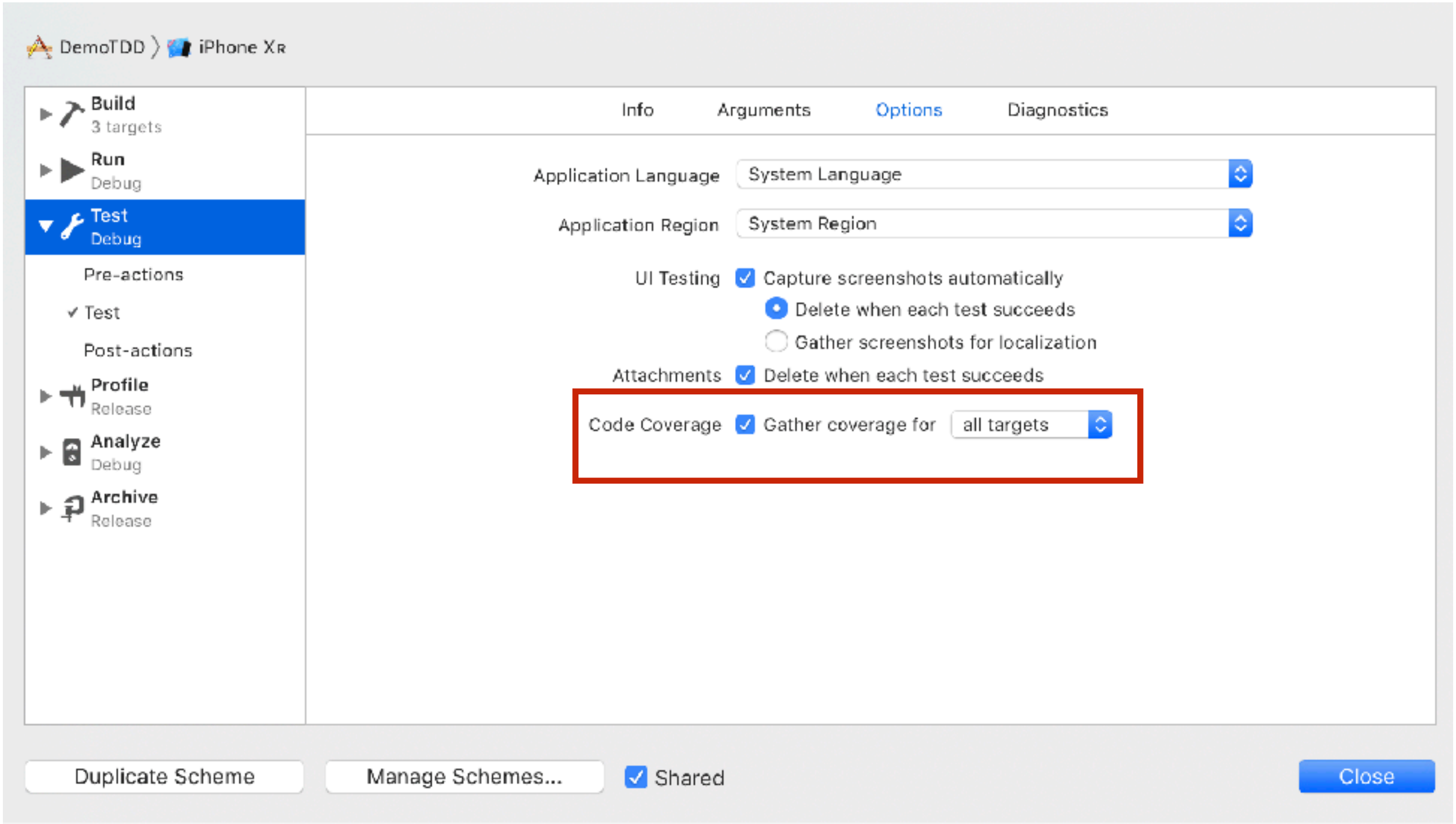
Code coverage != quality of tests

Code coverage 100% ?

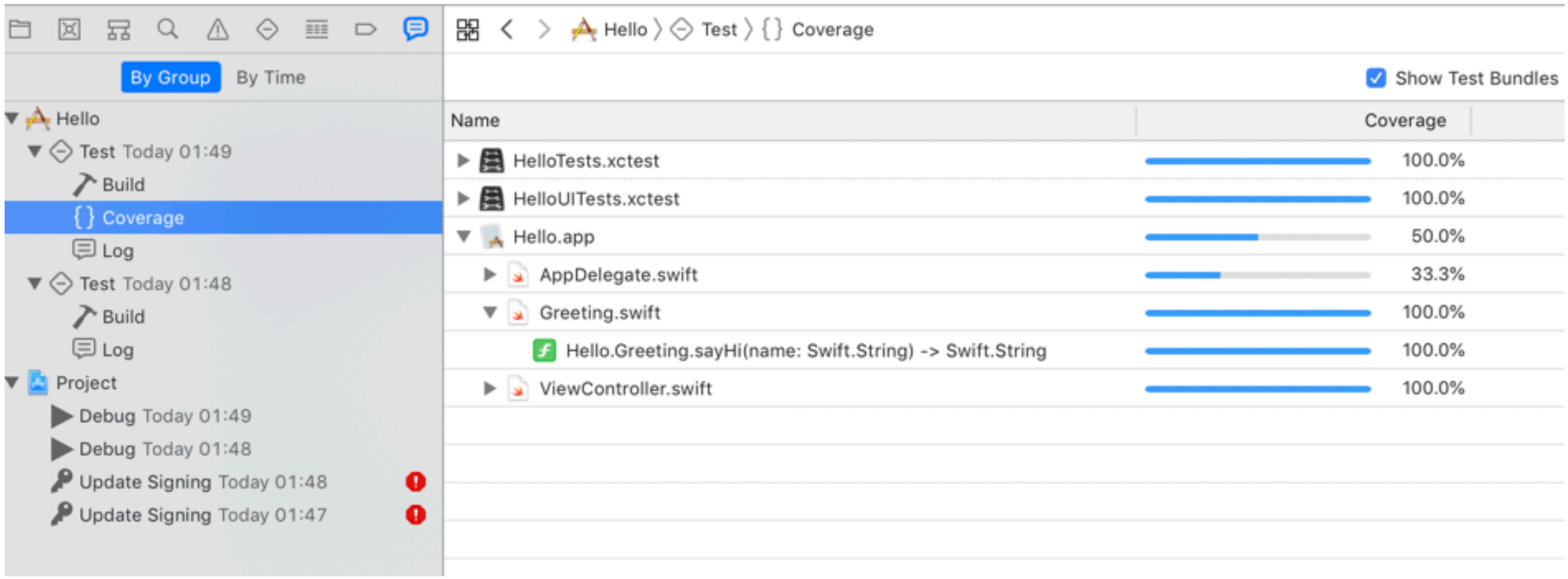


Code coverage in Xcode

Enable code coverage in Scheme



Code coverage report

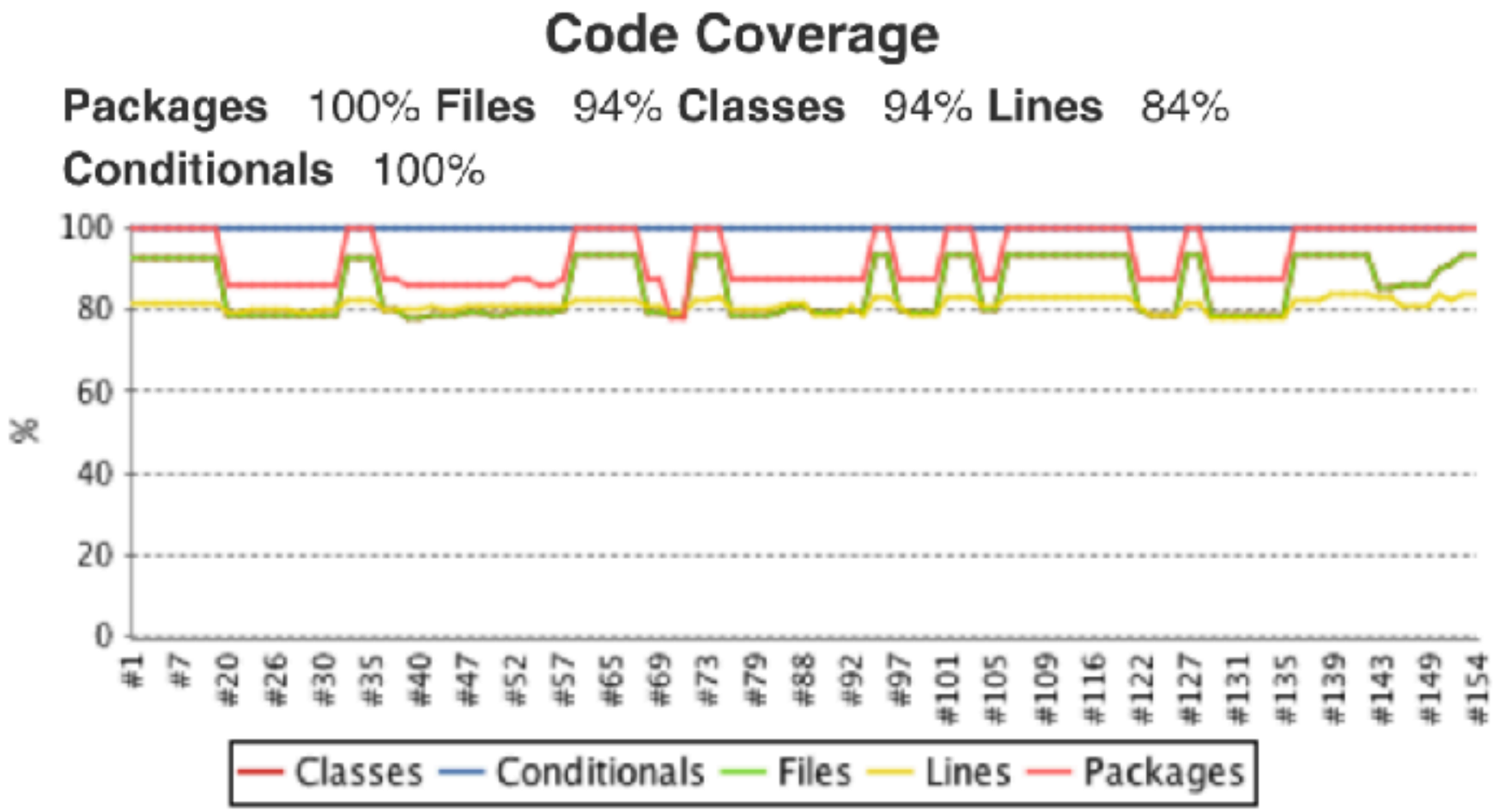


Code coverage in Editor

```
class MyClass {

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

    deinit {
        print("Called deinit")
    }

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

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

}
```

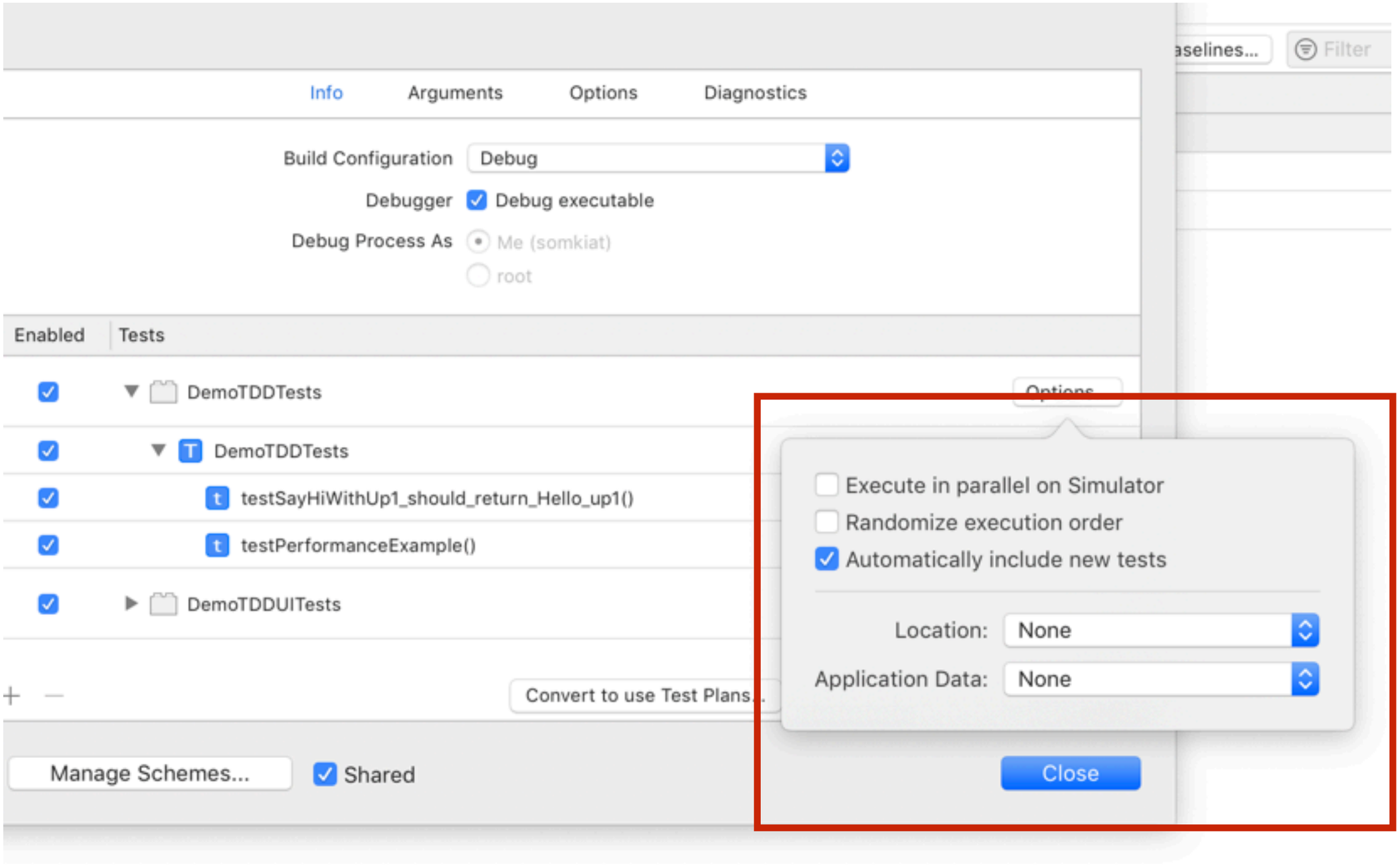
Code coverage report

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

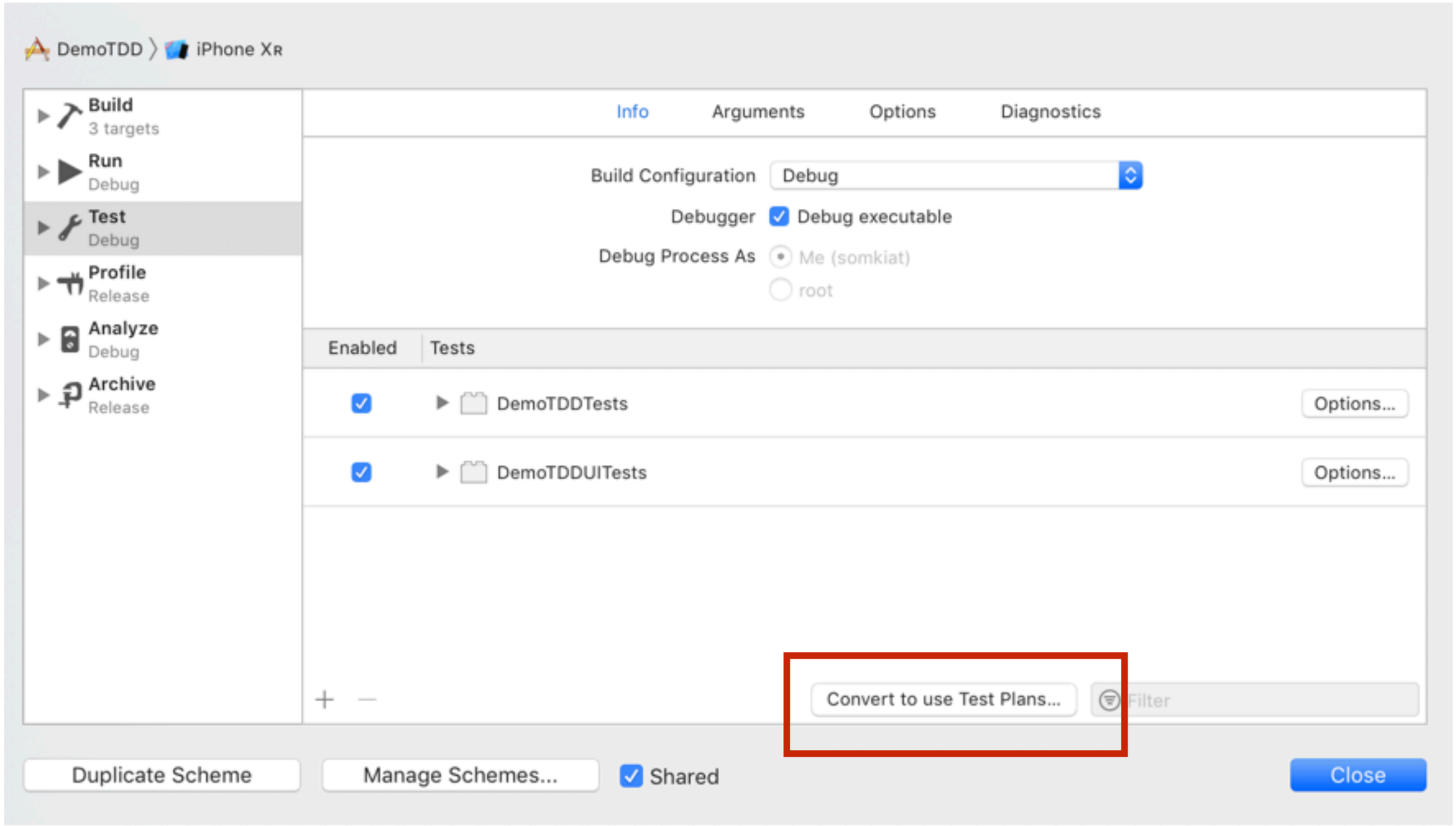


More test features in Xcode

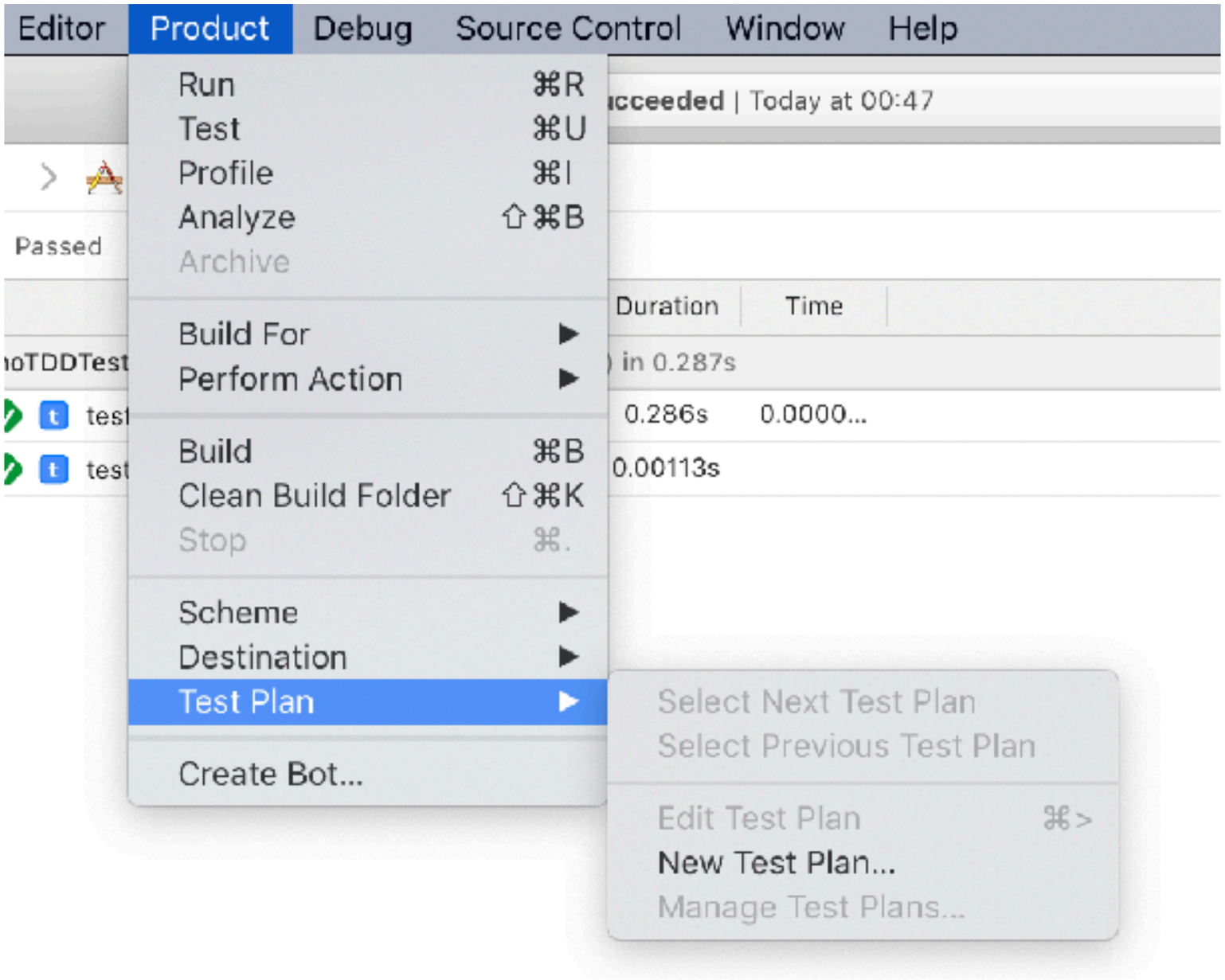
Run random and parallel



Test plan in Xcode 11



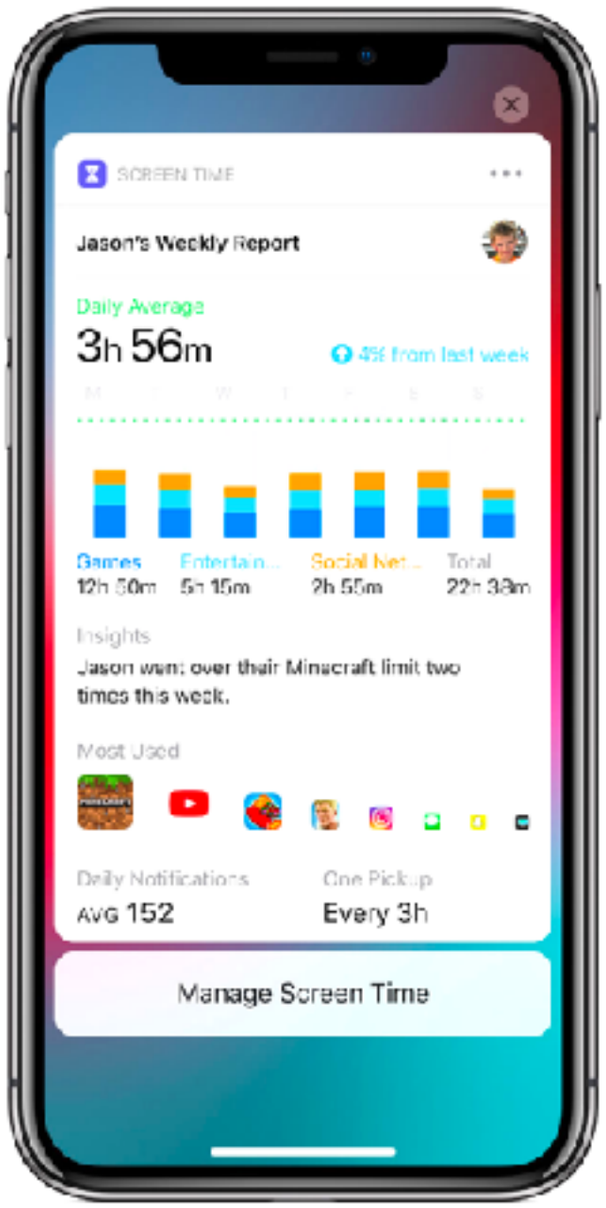
Test plan in Xcode 11



Workshop

Login application

Login Page



Result

Success

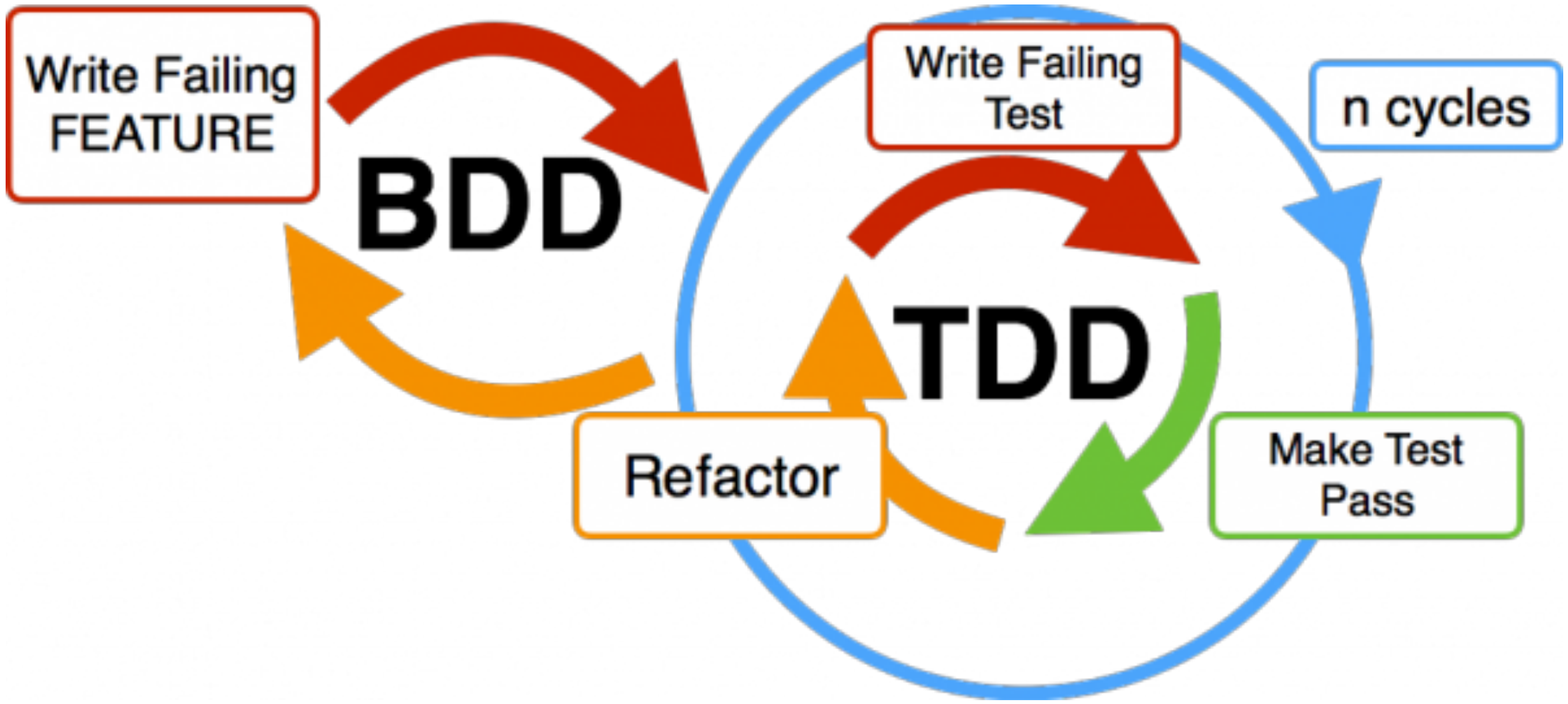
Failure

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

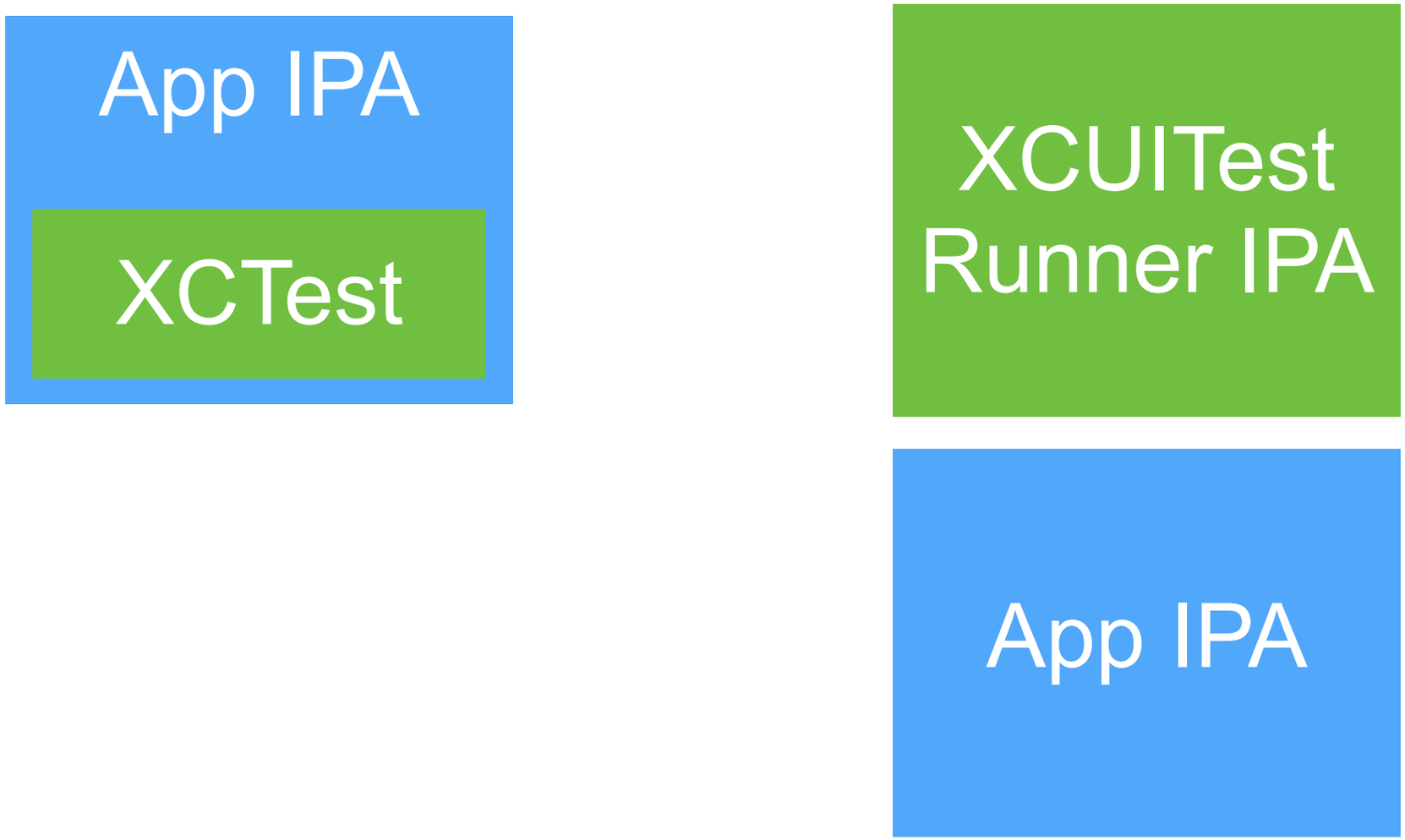
Provides the API to perform UI actions
Tap, Swipe, Key press and etc.

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

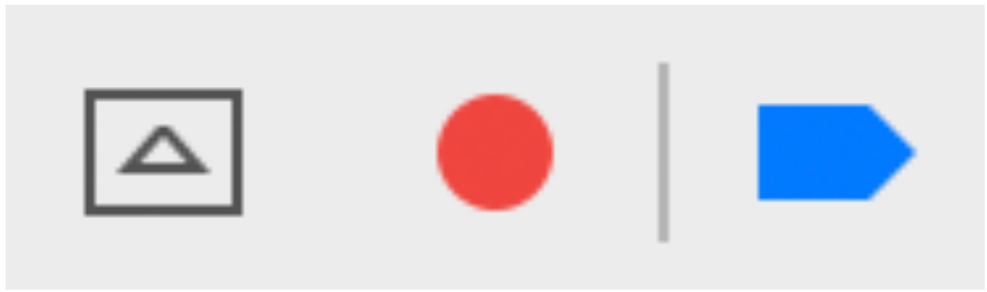
Create XCUITest

```
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

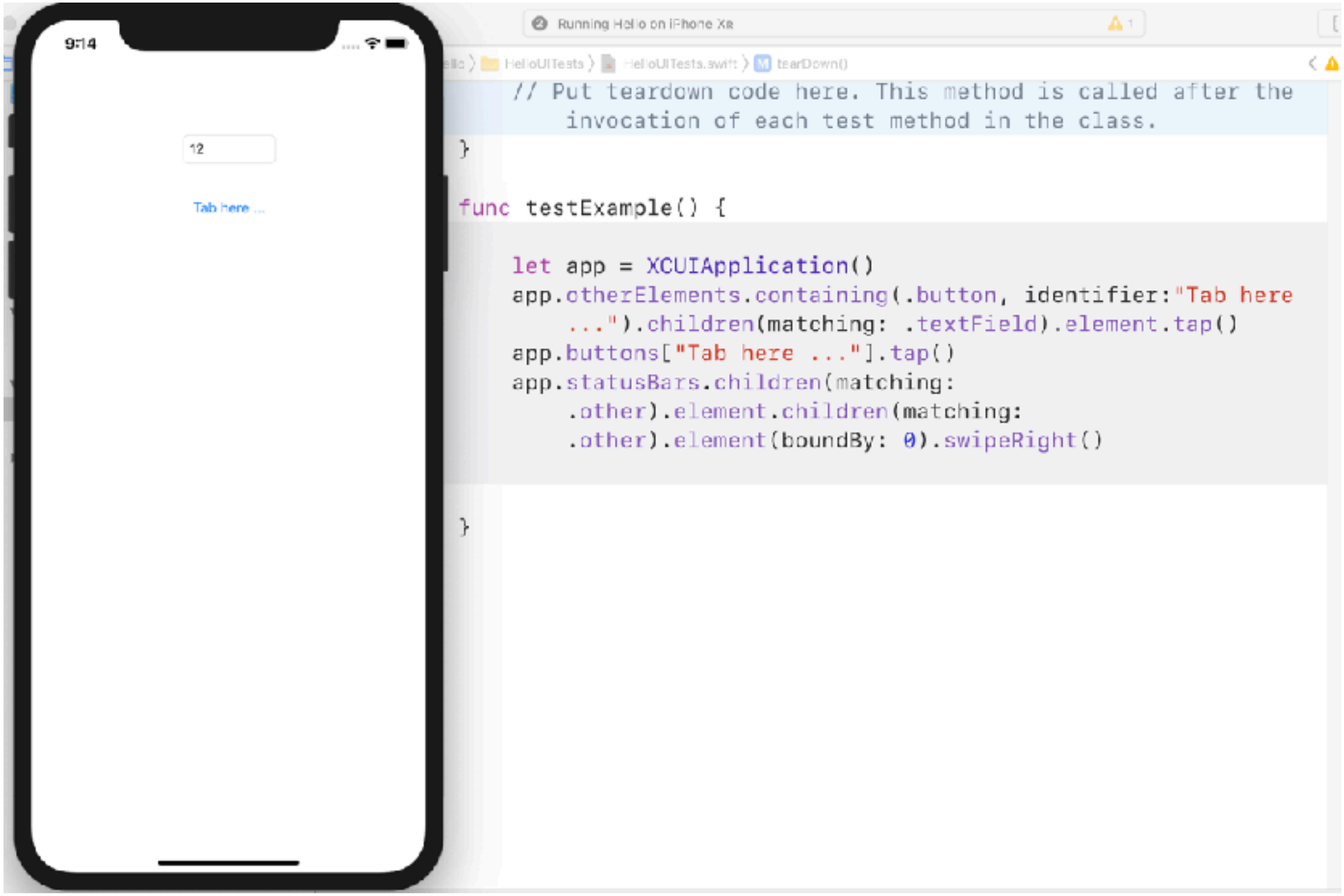
Write by yourself or use UI Test Recorder



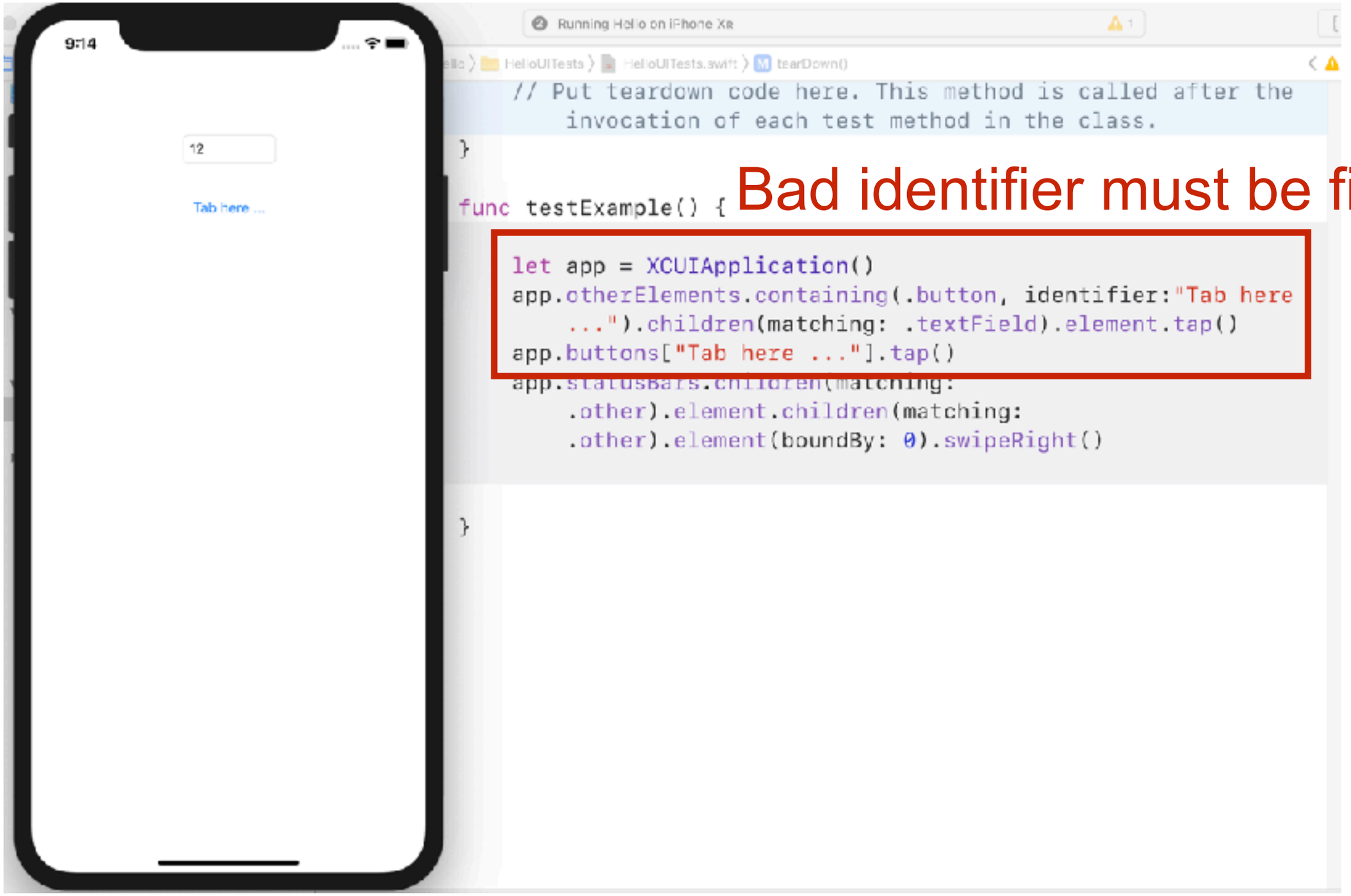
Record UI Test

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

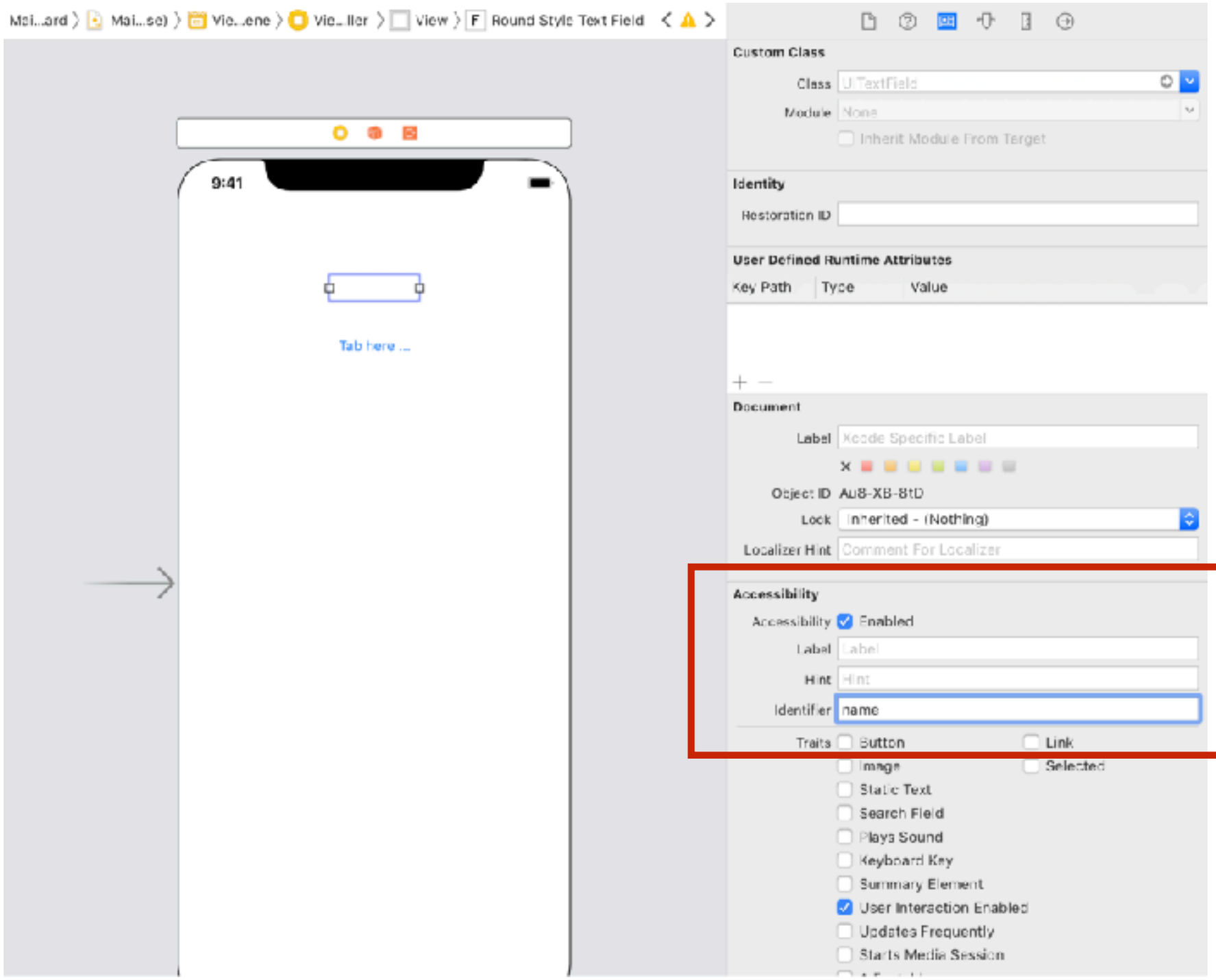
XCUIApplication to launch tests

XCUIApplication use to access all elements

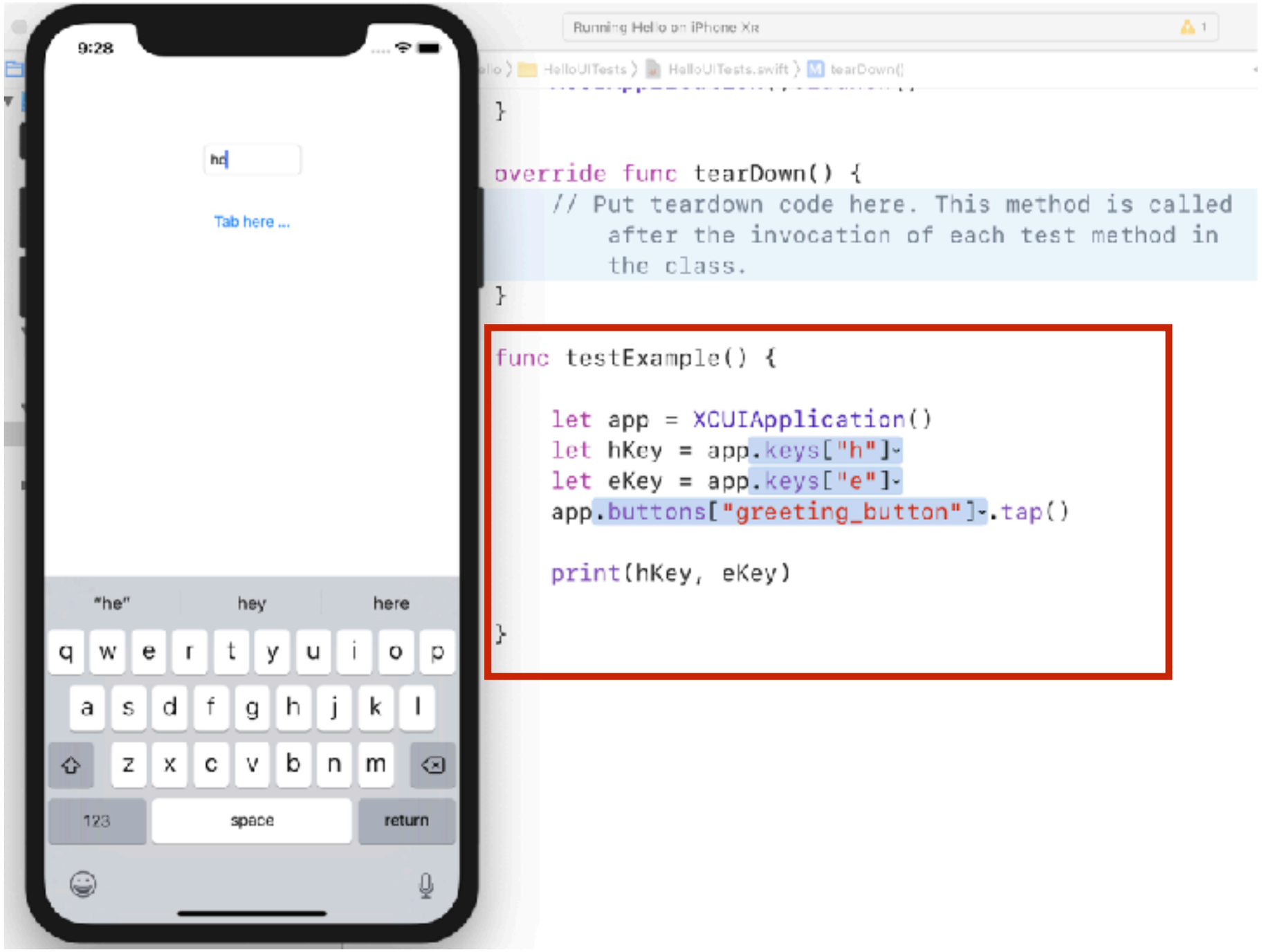
Access with text and indices that can be change !!

Test fail !!

Improve identifier of elements



Record again !!

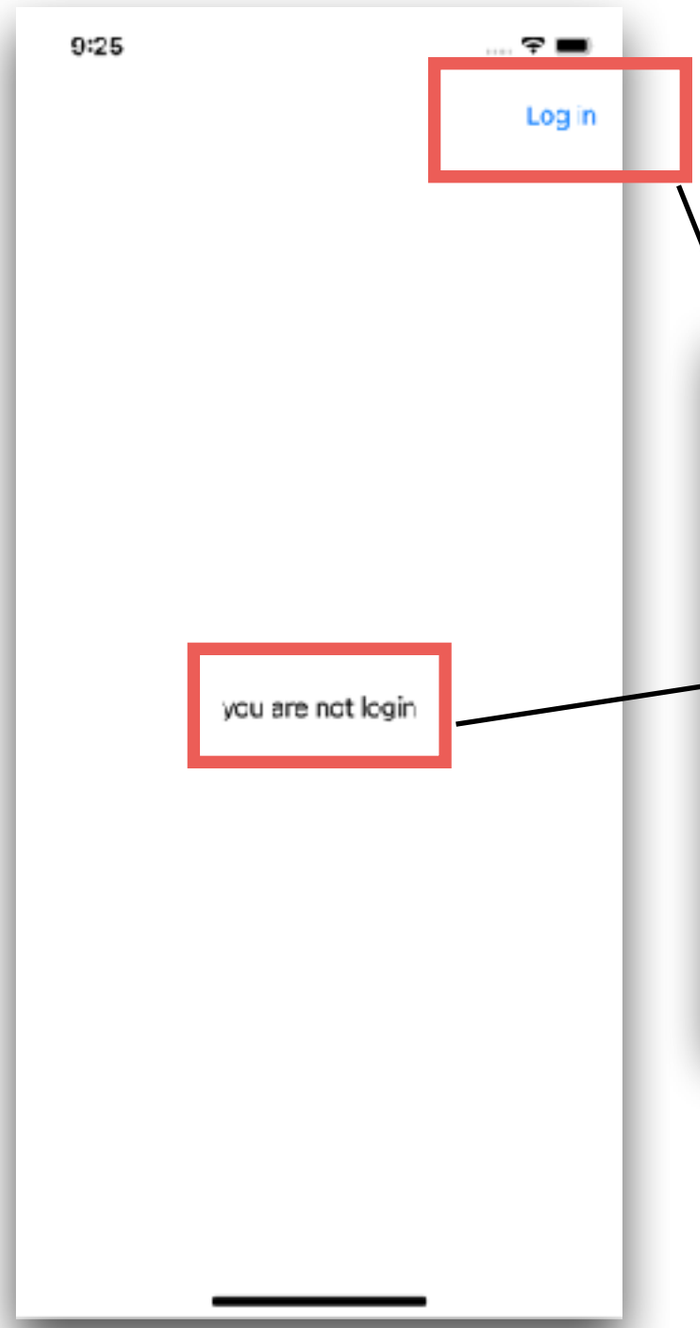


Don’t forget to verify !!

Using XCTAssertion
Working with **Asynchronous process**

Working with SwiftUI

Add accessibility to elements



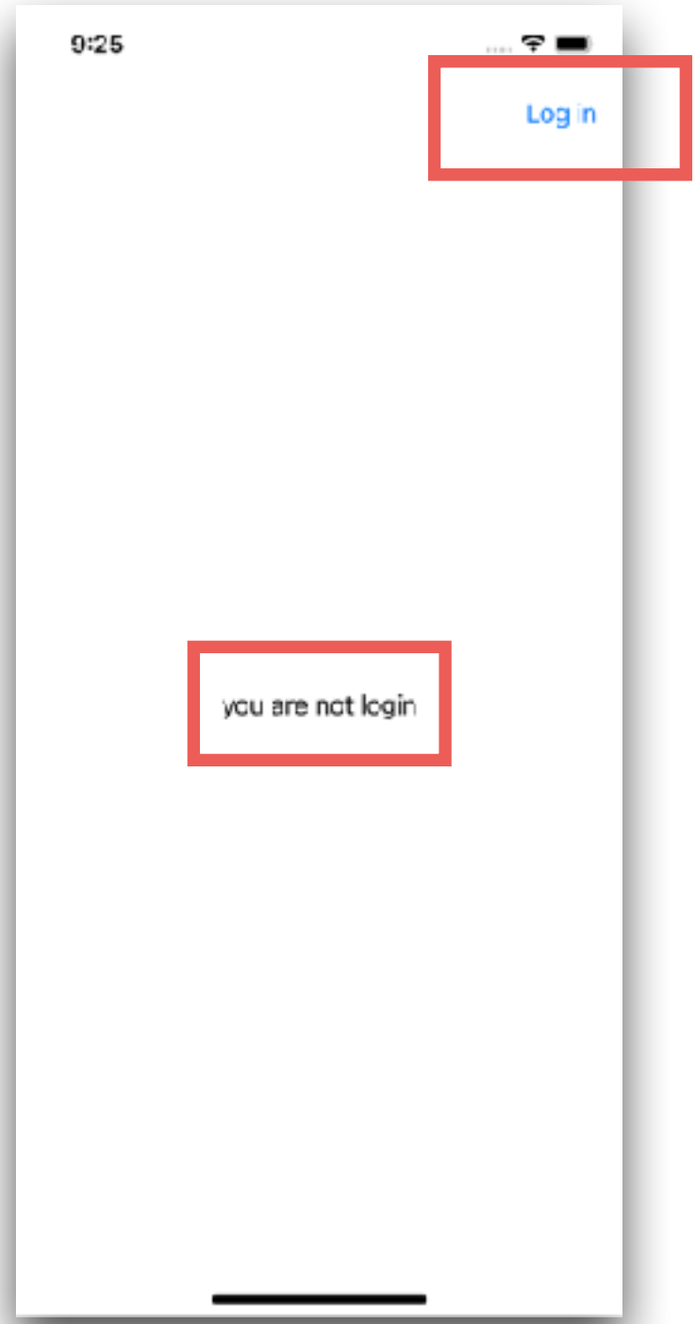

```
struct NonLoggedInView: View {

    @EnvironmentObject var appState: AppState
    @ObservedObject var viewModel: NonLoggedInViewModel = NonLoggedInViewModel()

    var body: some View {
        Text("you are not login").accessibility(identifier: "txtResult")
        .toolbar {
            Button("Log in"){
                viewModel.startLogin()
            }.onReceive(viewModel.onLoggedInSuccess) { value in
                appState.isLogin = true
            }.accessibility(identifier: "btnLogin")
        }
    }
}
```

Write UI Test or Recording

```
 8  import XCTest
 9
11  class LoginFlowTests: XCTestCase {
11
12      override func setUpWithError() throws {
13          // Put setup code here. This method is ca
14
15          // In UI tests it is usually best to stop
16          continueAfterFailure = false
17
```

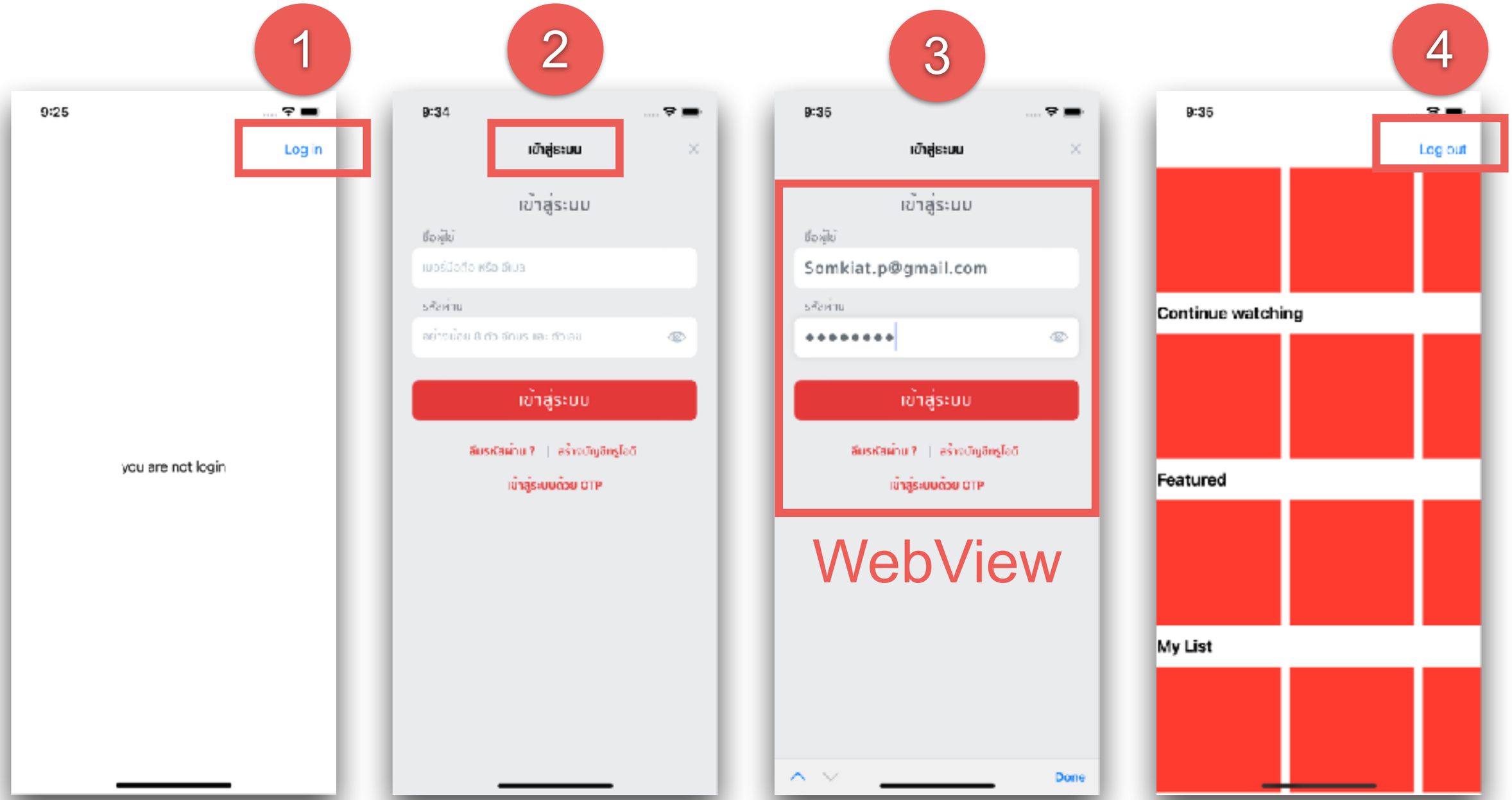
Recording UI Test

Basic flow

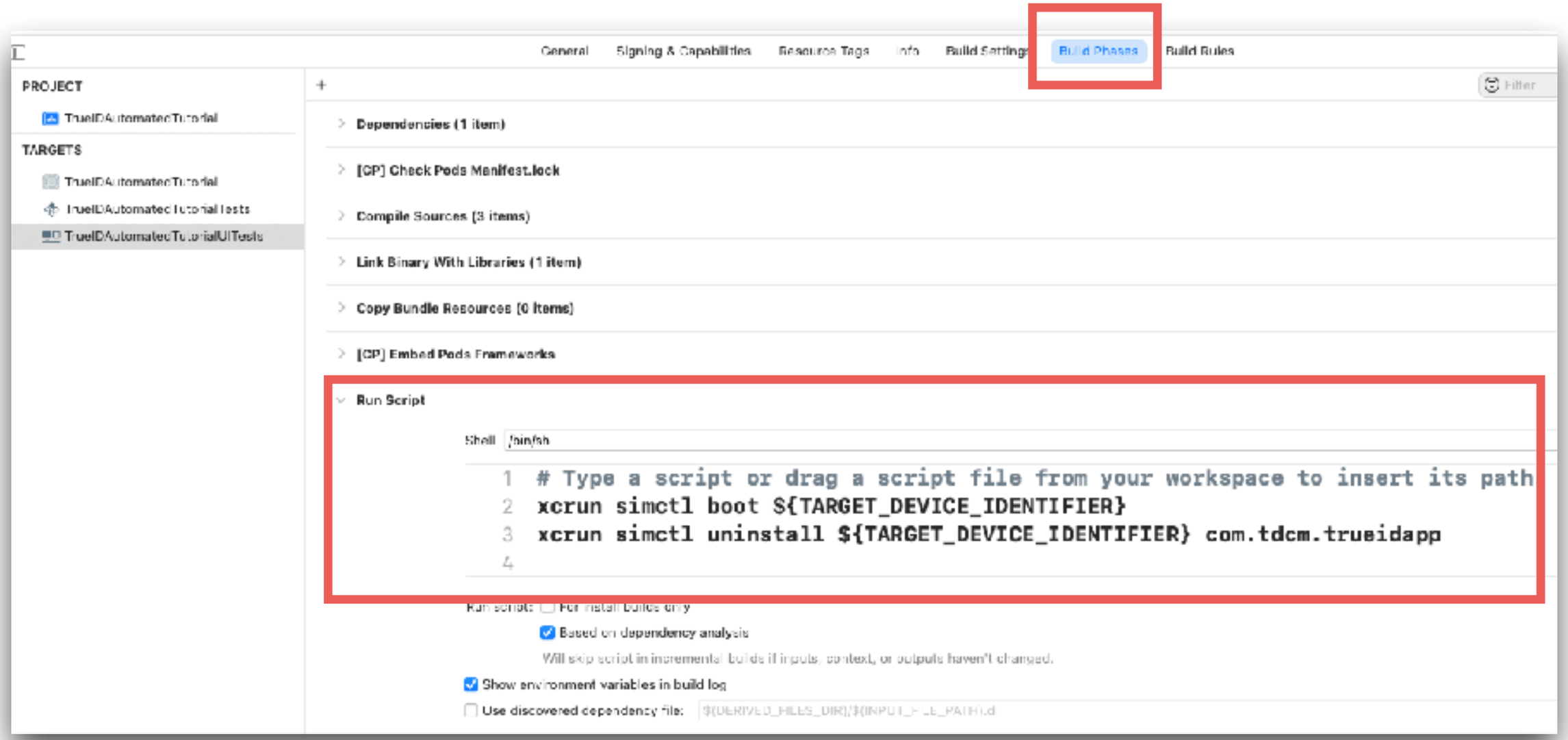



```
func testCheckNotLoginPage() throws {
    let app = XCUIApplication()
    let result = app.staticTexts["txtResult"].label
    XCTAssertEqual("you are not login", result)
    XCTAssert(app.buttons["btnLogin"].exists)
}
```

User Flow



Uninstall app before test

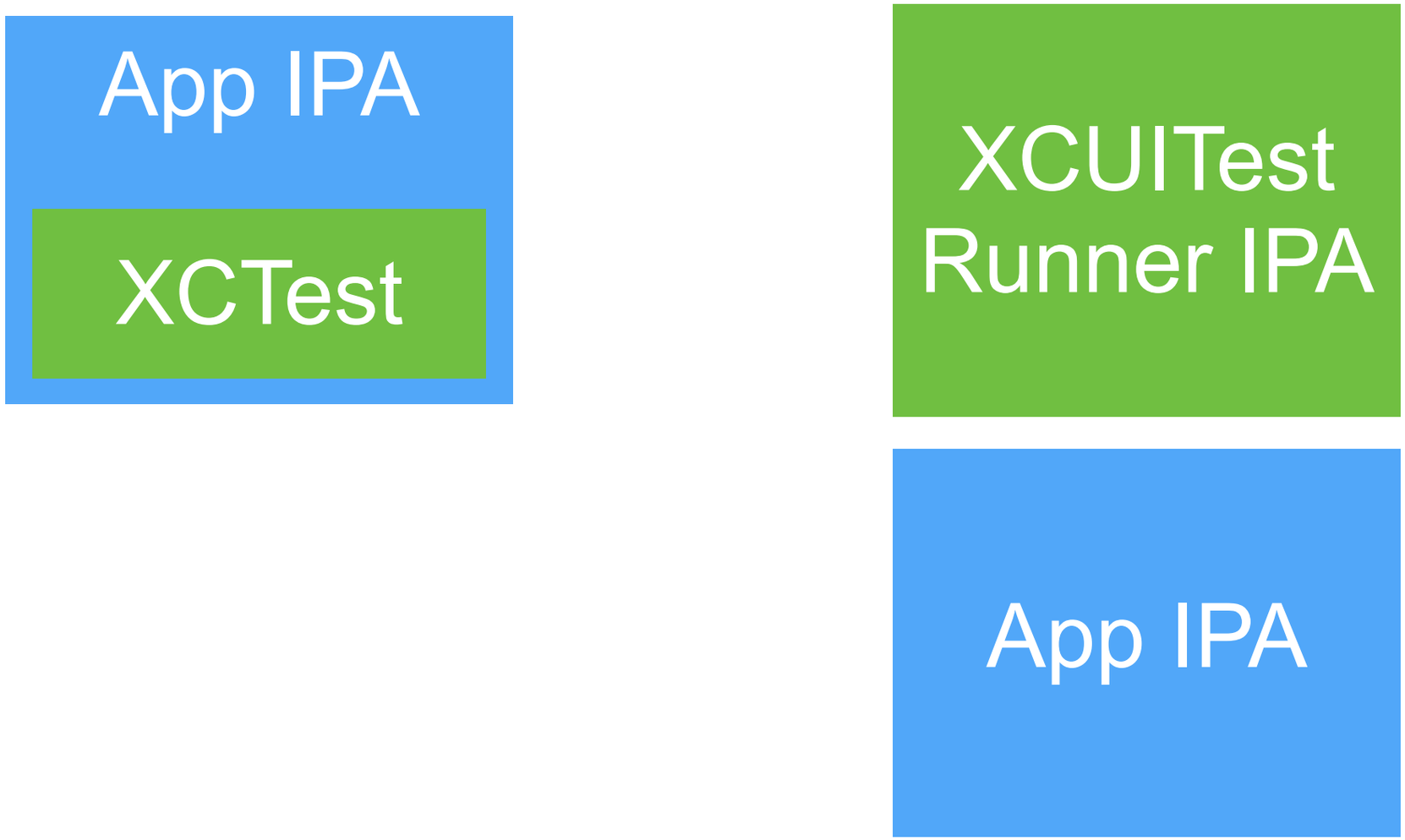


Workshop

Send data
from UI test to app

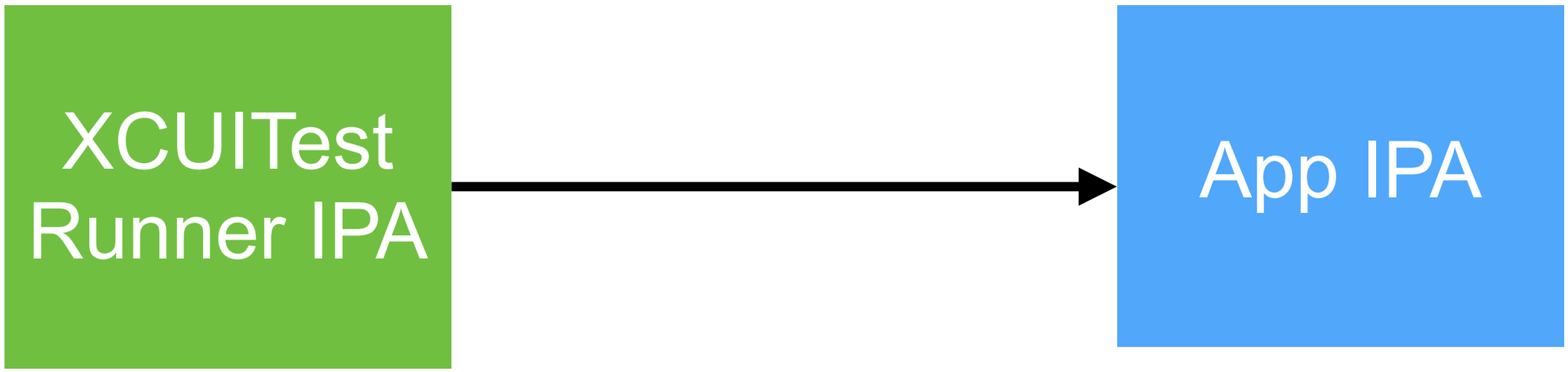
XCTest vs XCUITest

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



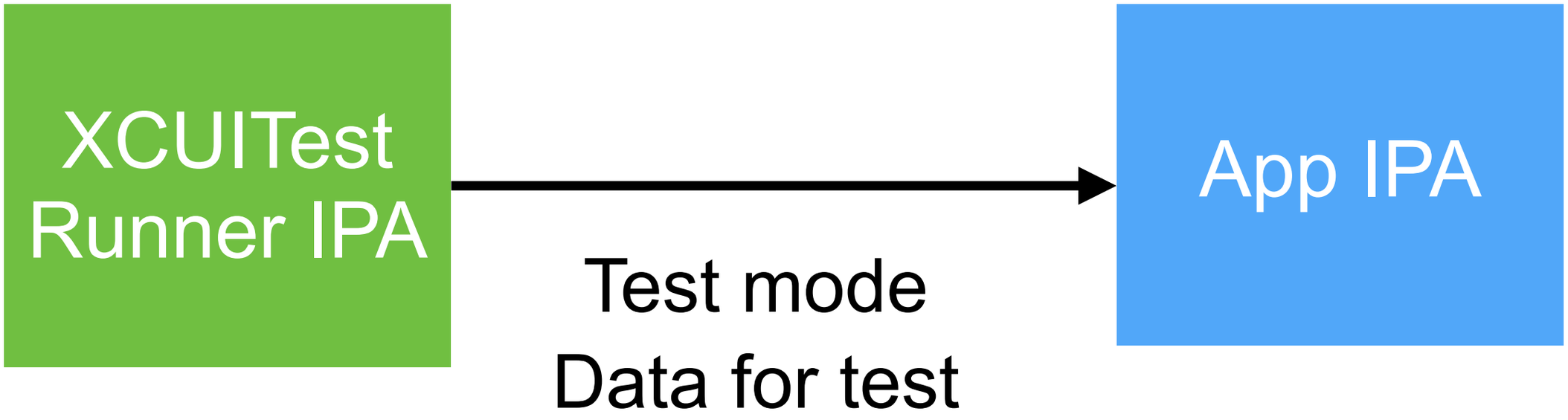
XCUITest

XCUITest bundle in UI Test Runner




```
open var launchArguments: [String]
open var launchEnvironment: [String : String]
```

Demo



Demo

```
class SendDataToUITests: XCTestCase {

    let app = XCUIApplication()

    override func setUpWithError() throws {
        continueAfterFailure = false
        app.launchArguments = LaunchArguments.standardForTest
    }

    override func tearDownWithError() throws {
        app.terminate()
    }

    func testCheckNotLoginPageWithTestData() throws {
        app.launchEnvironment["login"] = "false"
        app.launchEnvironment["result-text"] = "HELLO FROM TEST"
        app.launch()

        let result = app.staticTexts["txtResult"].label
        XCTAssertEqual("HELLO FROM TEST", result)
        XCTAssertTrue(app.buttons["btnLogin"].exists)
    }
```

Demo

```swift
class SendDataToUITests: XCTestCase {

    let app = XCUIApplication()

    override func setUpWithError() throws {
        continueAfterFailure = false
        app.launchArguments = LaunchArguments.standardForTest
    }

    override func tearDownWithError() throws {
        app.terminate()
    }

    func testCheckNotLoginPageWithTestData() throws {
        app.launchEnvironment["login"] = "false"
        app.launchEnvironment["result-text"] = "HELLO FROM TEST"
        app.launch()

        let result = app.staticTexts["txtResult"].label
        XCTAssertEqual("HELLO FROM TEST", result)
        XCTAssertTrue(app.buttons["btnLogin"].exists)
    }
```

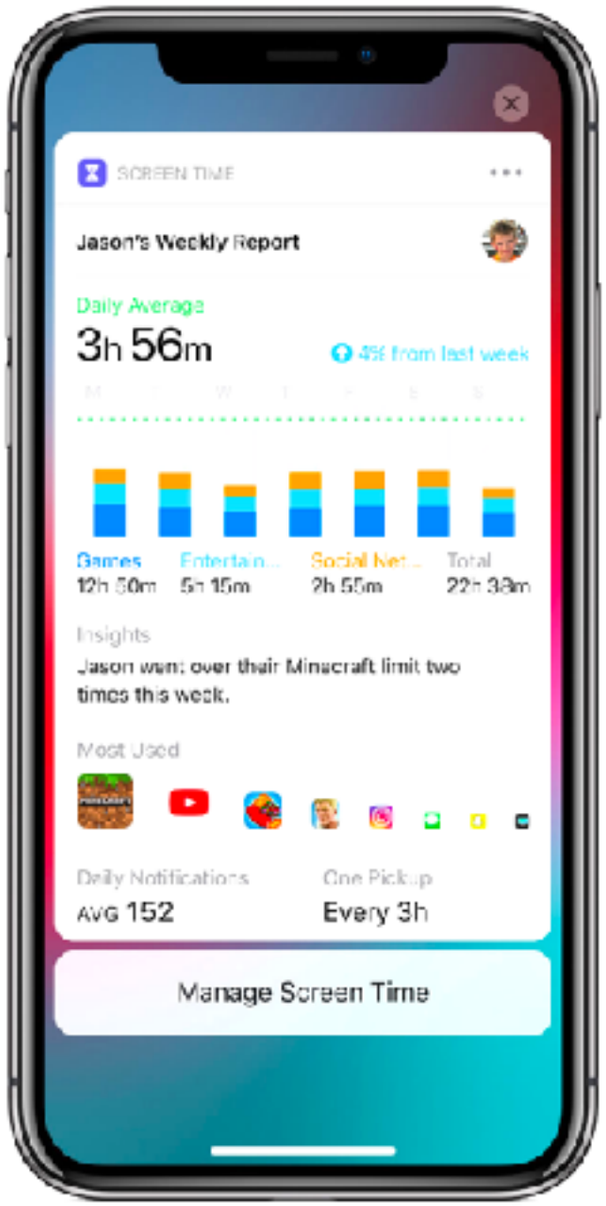
Change Production Code

Manage state in App from UI Test

```
struct MainView: View {

    @StateObject var appState: AppState

    private func isLoginStateFromUITest() -> Bool {
        guard TestHelper.isRunningUITests() else { return appState.isLogin }
        return Bool(TestHelper.getData(key: "login") ) ?? true
    }

    var body: some View {
        if isLoginStateFromUITest() {
            LoggedInView().environmentObject(appState)
        } else {
            NonLoggedInView().environmentObject(appState)
        }
    }
}
```

Flow 1 :: Login Success

Login Page

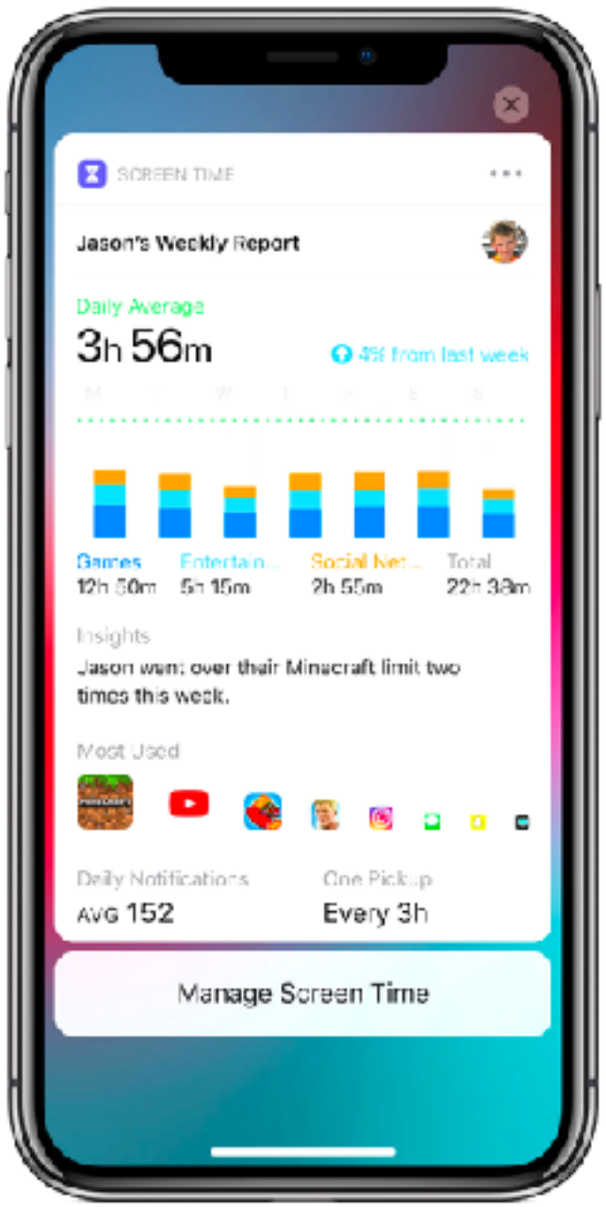


Result Page

Success

Define ID in each element

Login Page

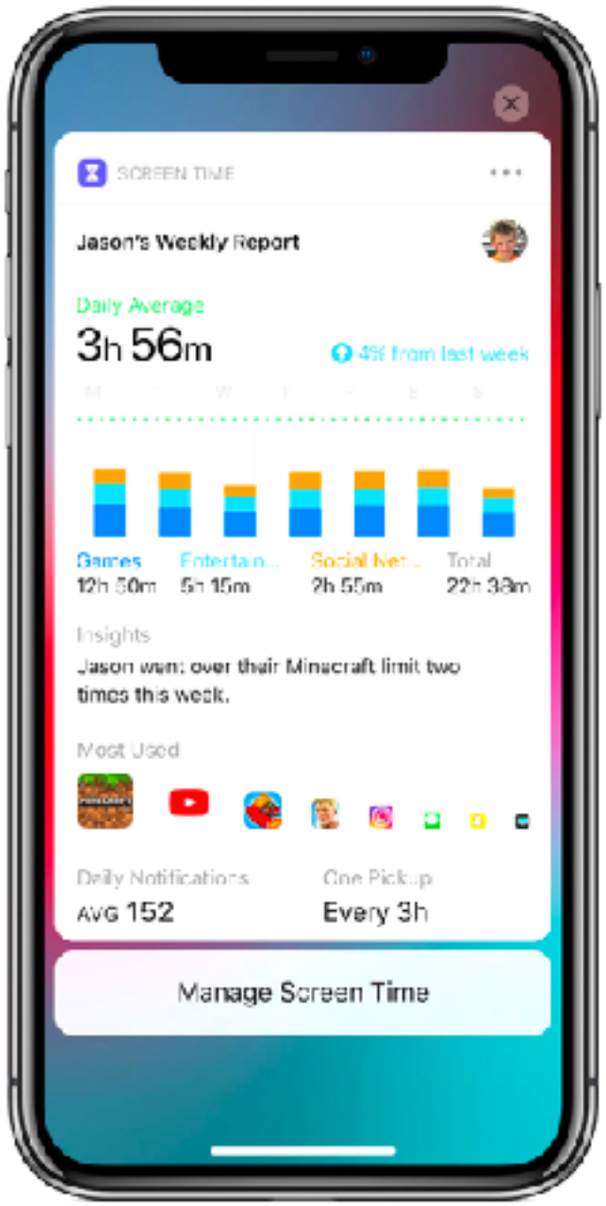


Result Page

Success

Flow 2 :: Login Fail

Login Page

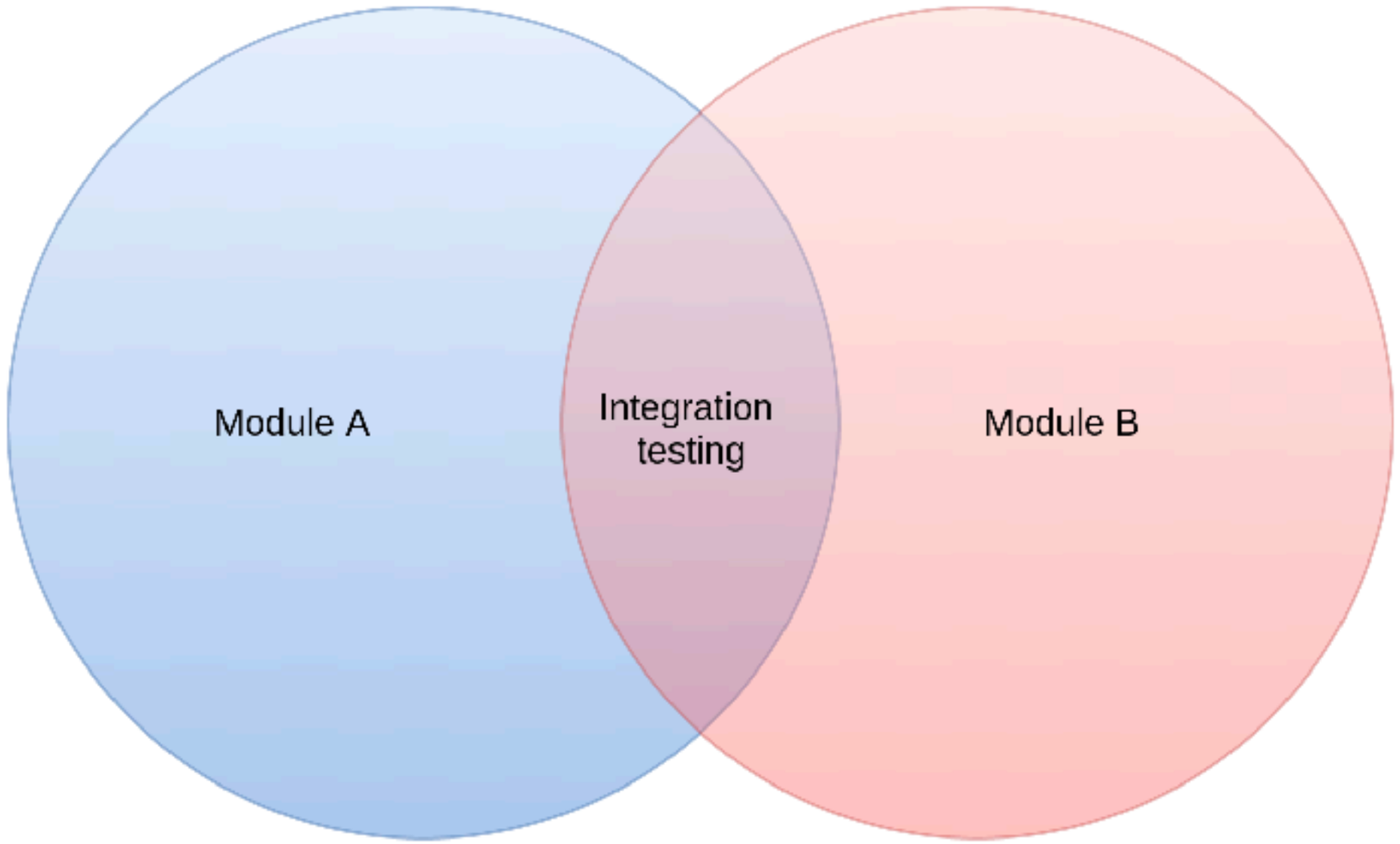


Result Page

Failure

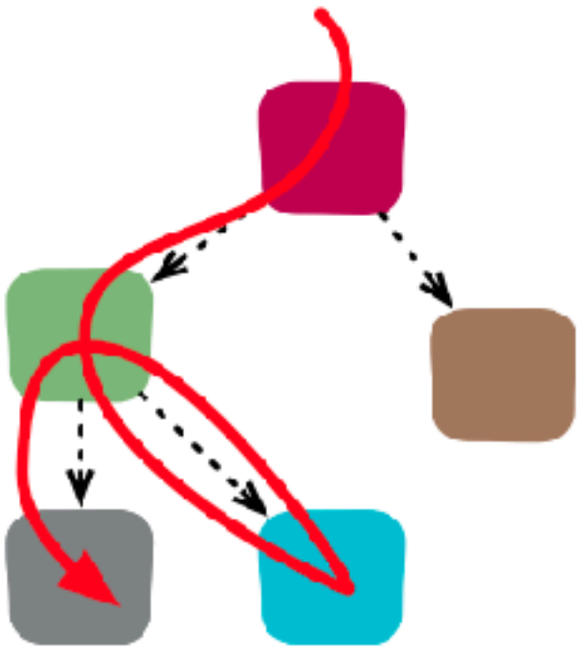
Integration testing !!

Integration testing

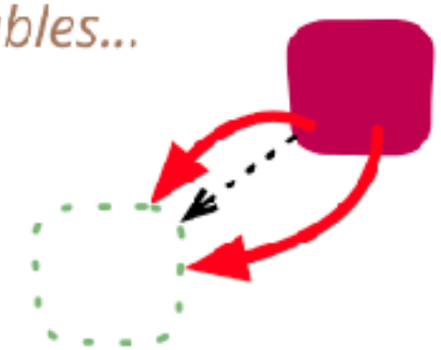


Integration testing !!

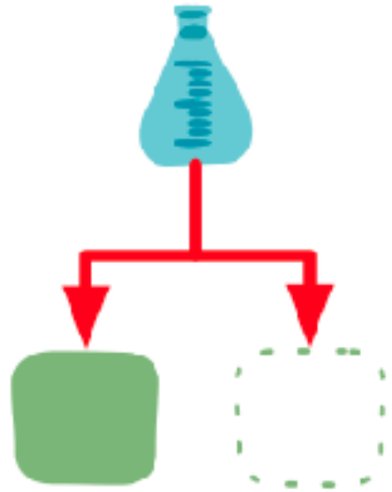
Integration Testing commonly refers to broad tests done with many modules active...



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

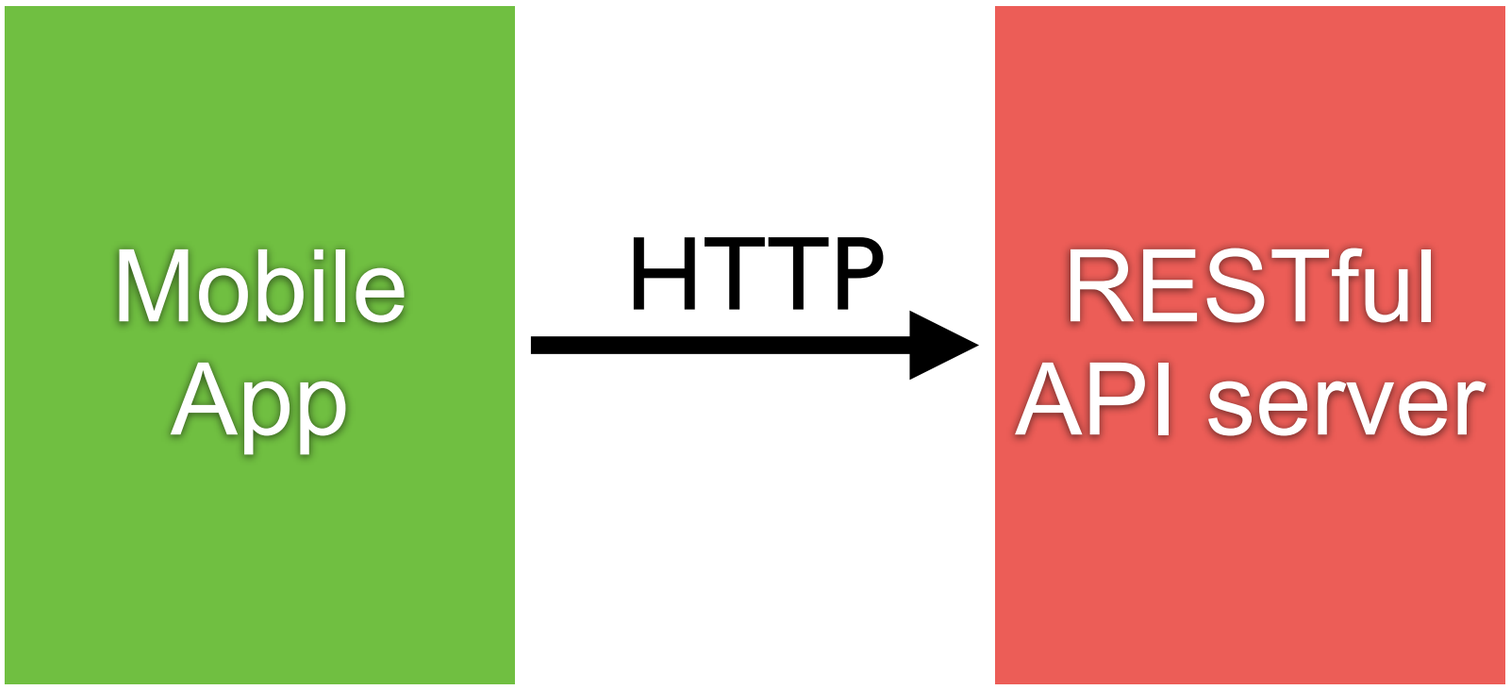


...supported by Contract Tests to ensure the faithfulness of the double



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

How to testing ?

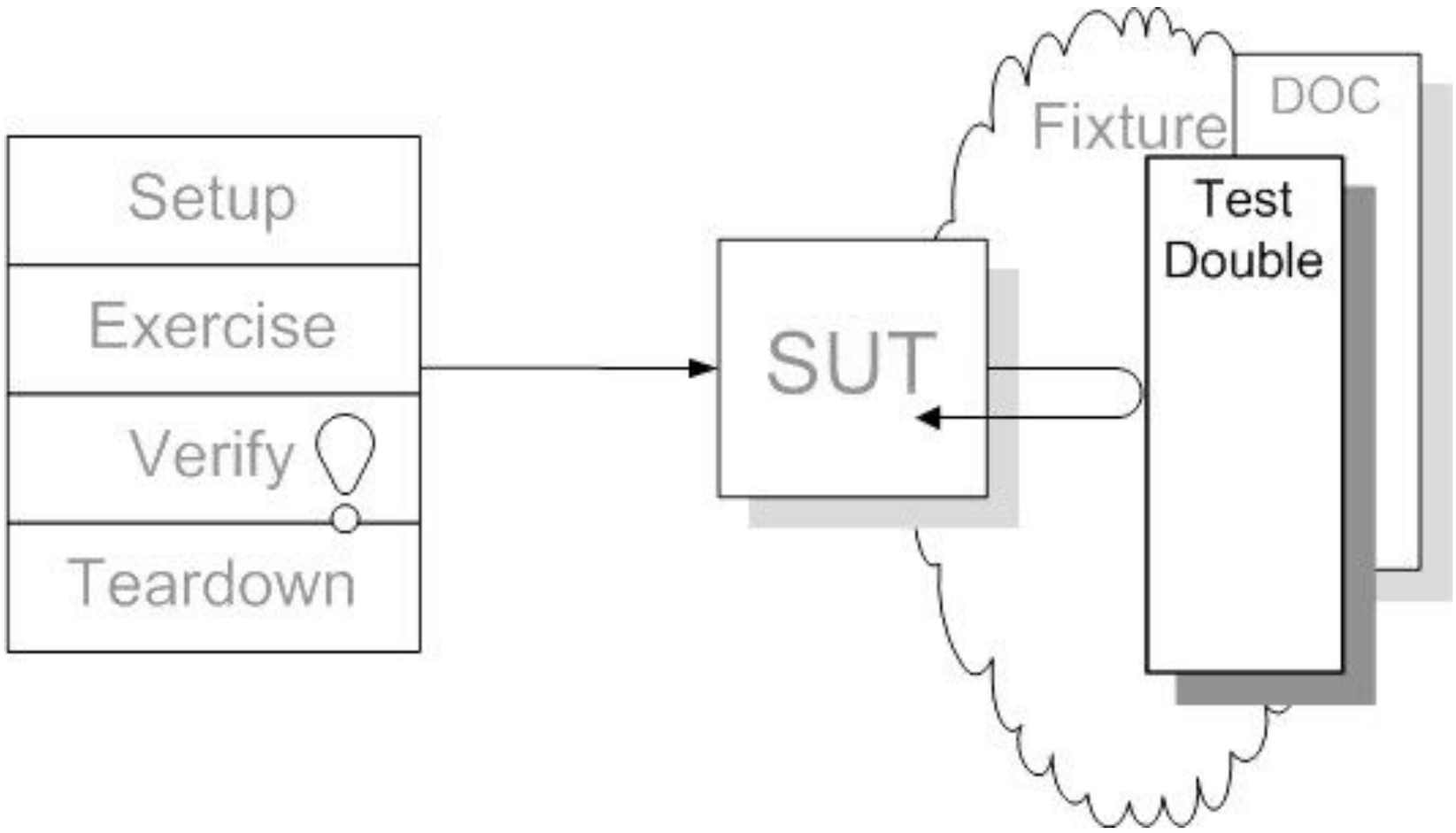


How to testing with network ?

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

Unit test VS. UI test

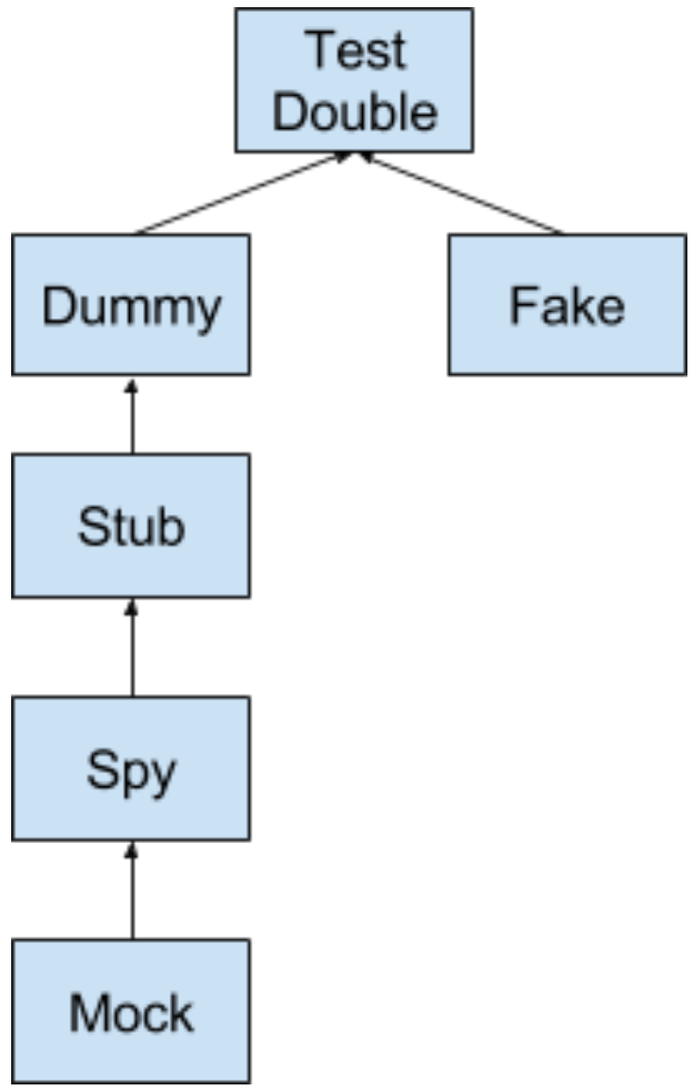
Test double



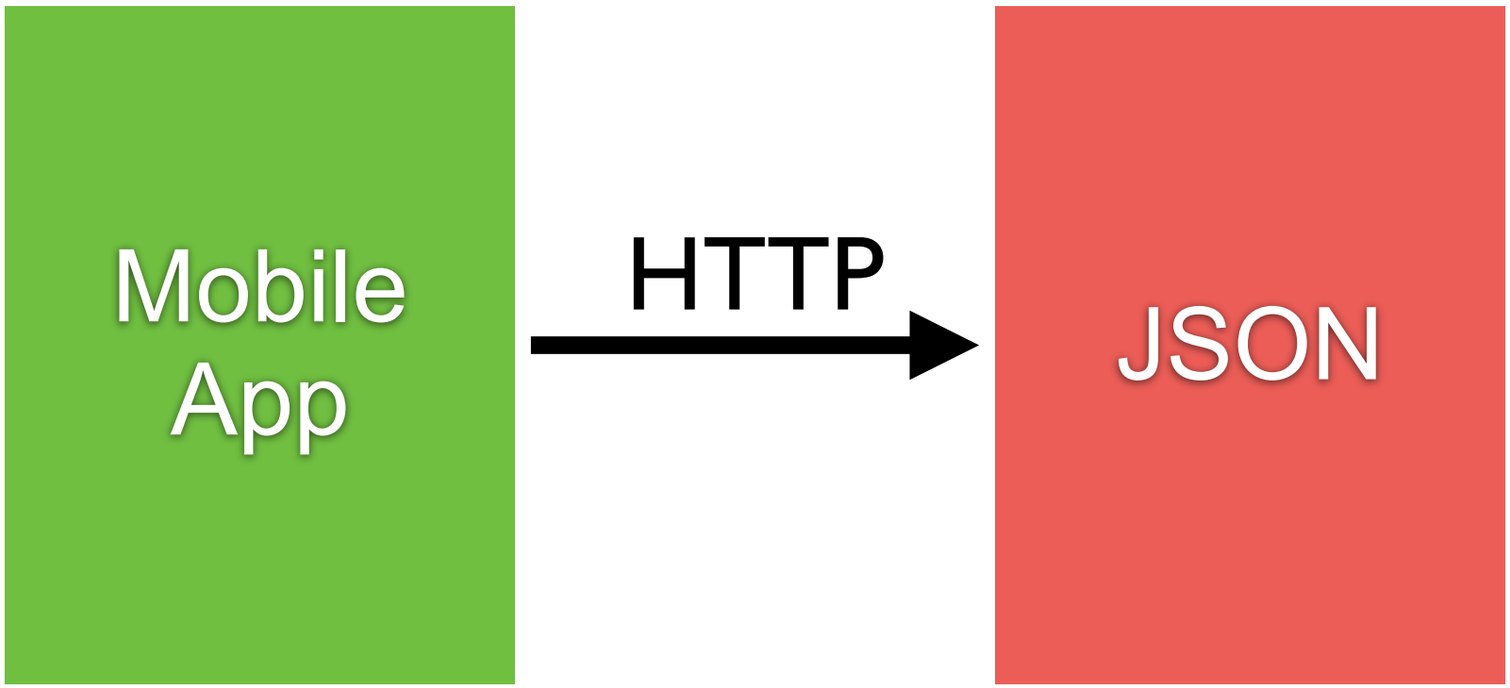
http://xunitpatterns.com/Test%20Double.html

Test double

Dummy
Stub
Spy
Mock
Fake



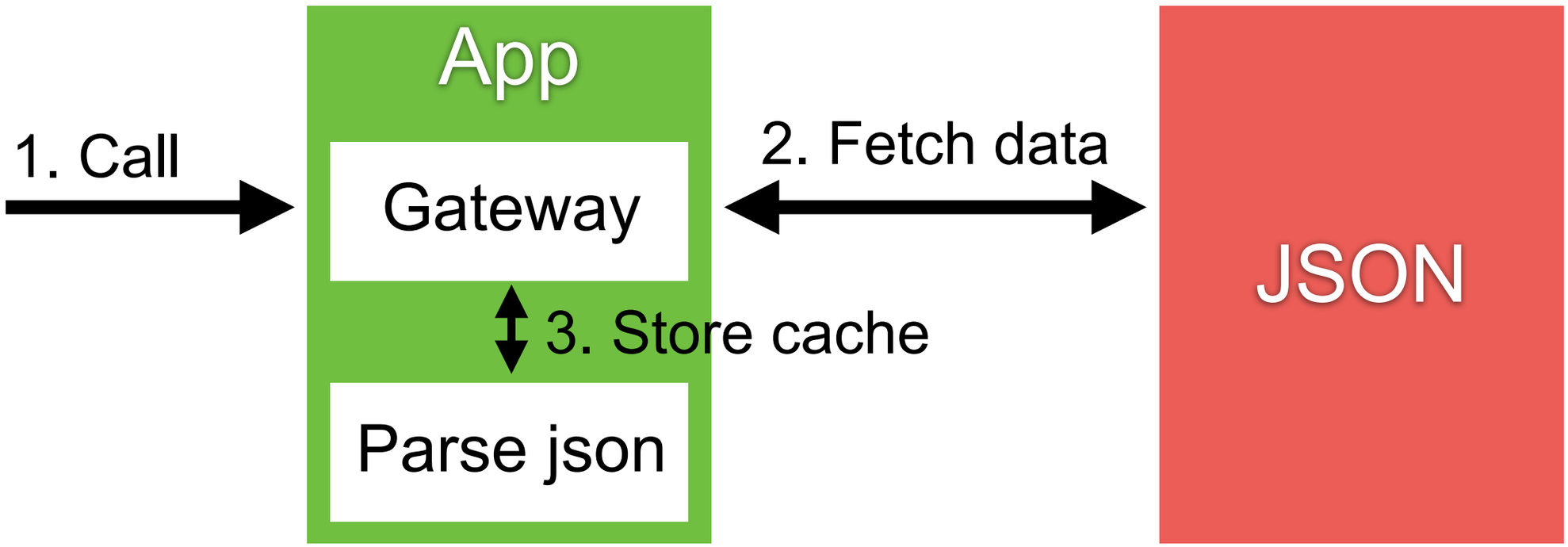
Example



Process to fetch data

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

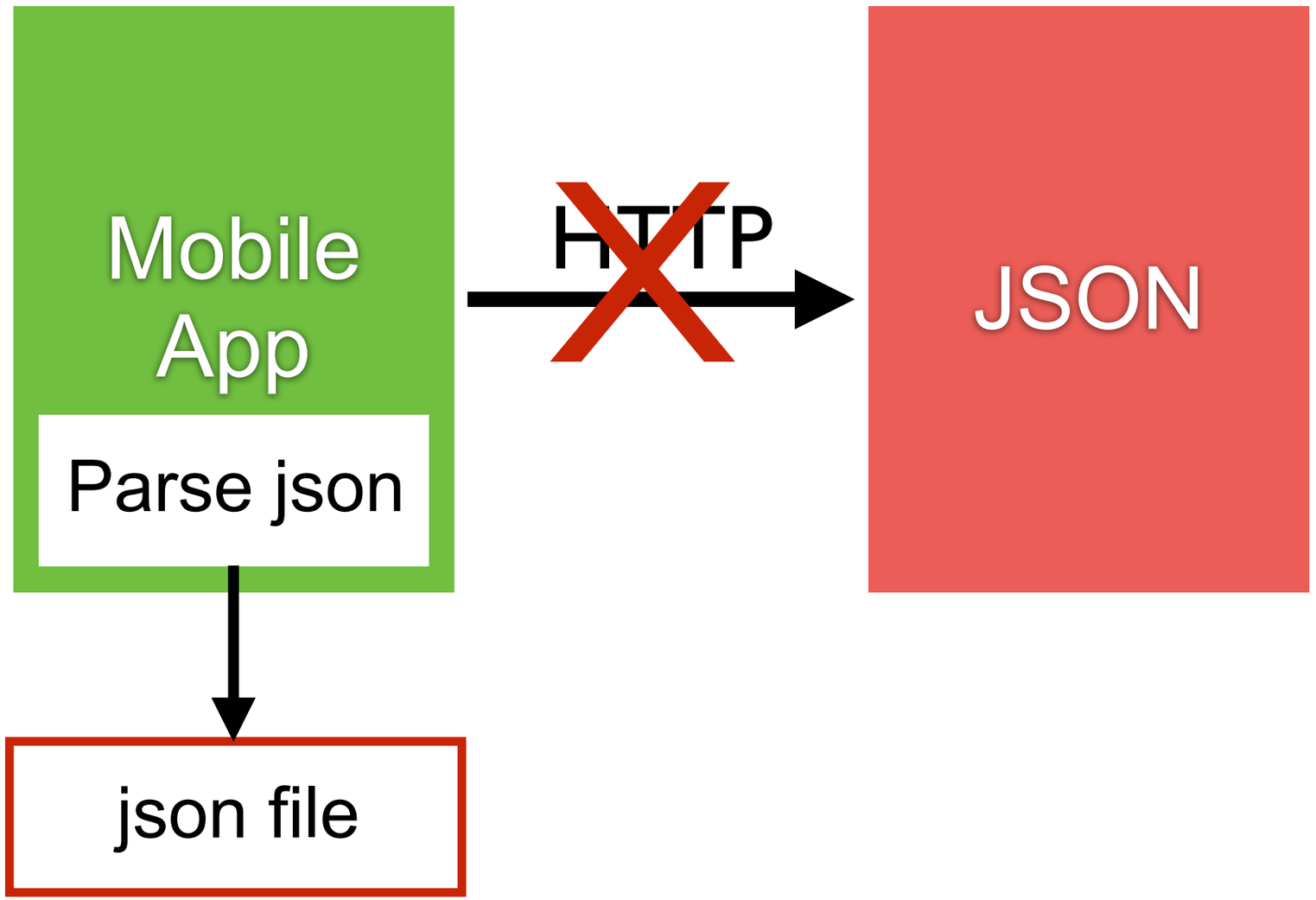
Example



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

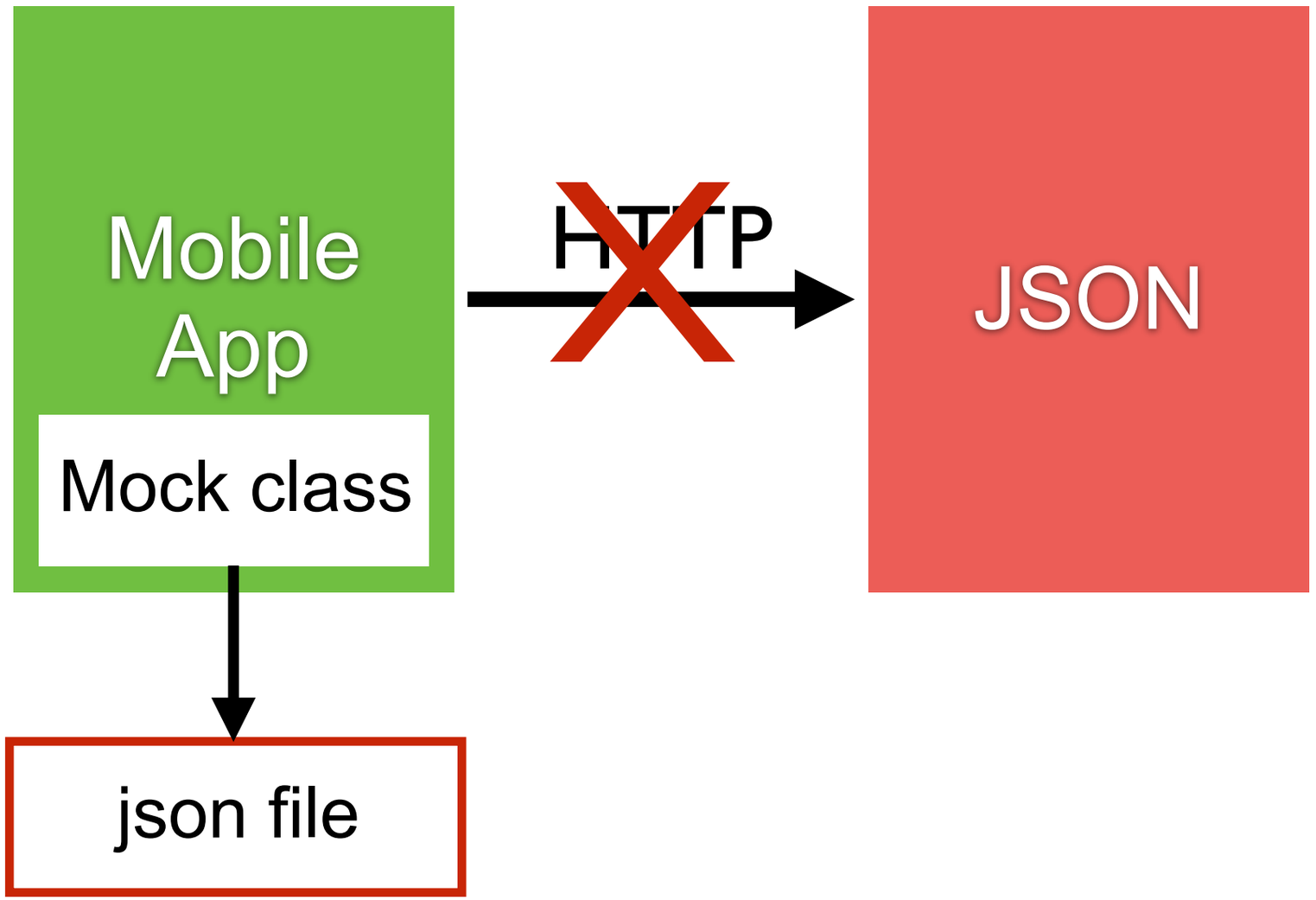
Design to mock what should be mocked
Use another functions
Avoid unwanted functionality
Benefit from base class implementation

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



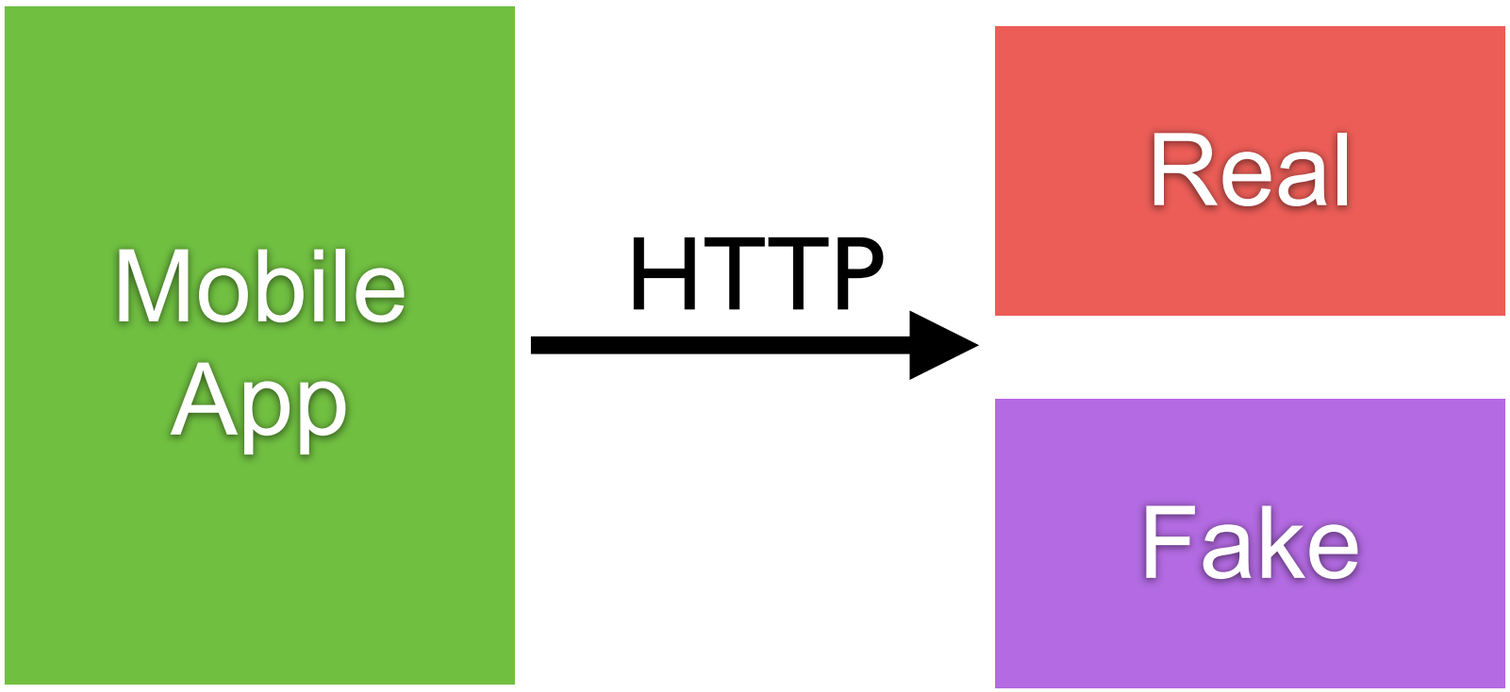
Create Mock class

Load data from file and call parse

```
class MockMusic : Music {
    override func fetchMusic(completion: @escaping ([[String : Any]]?, I
        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)

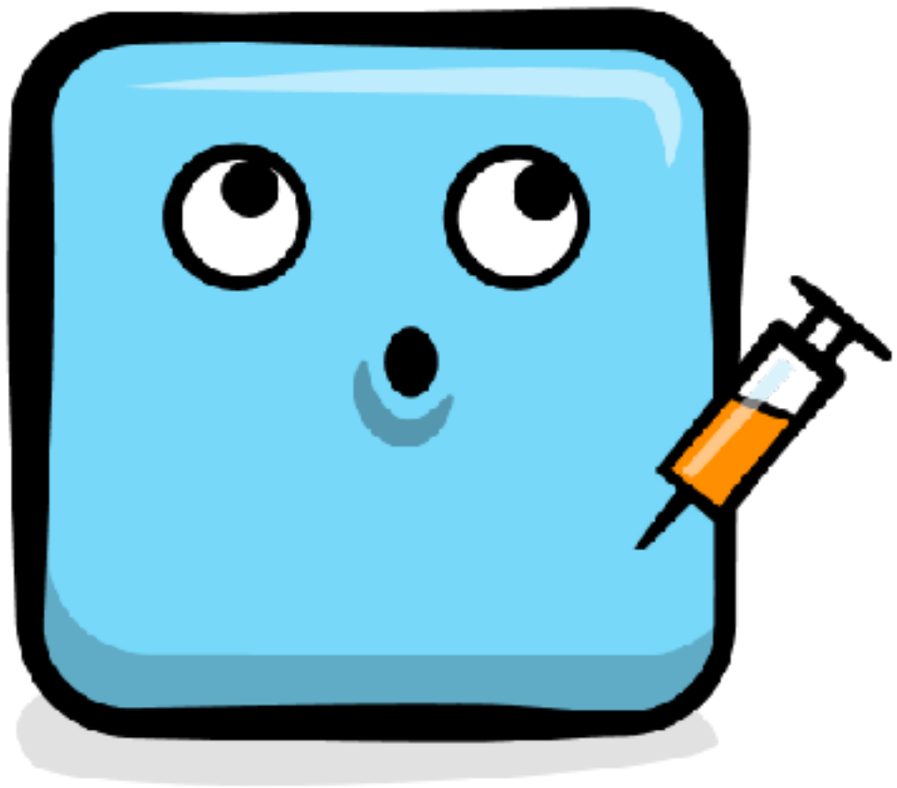
How to change URL of API ?

Constant variable

Build config

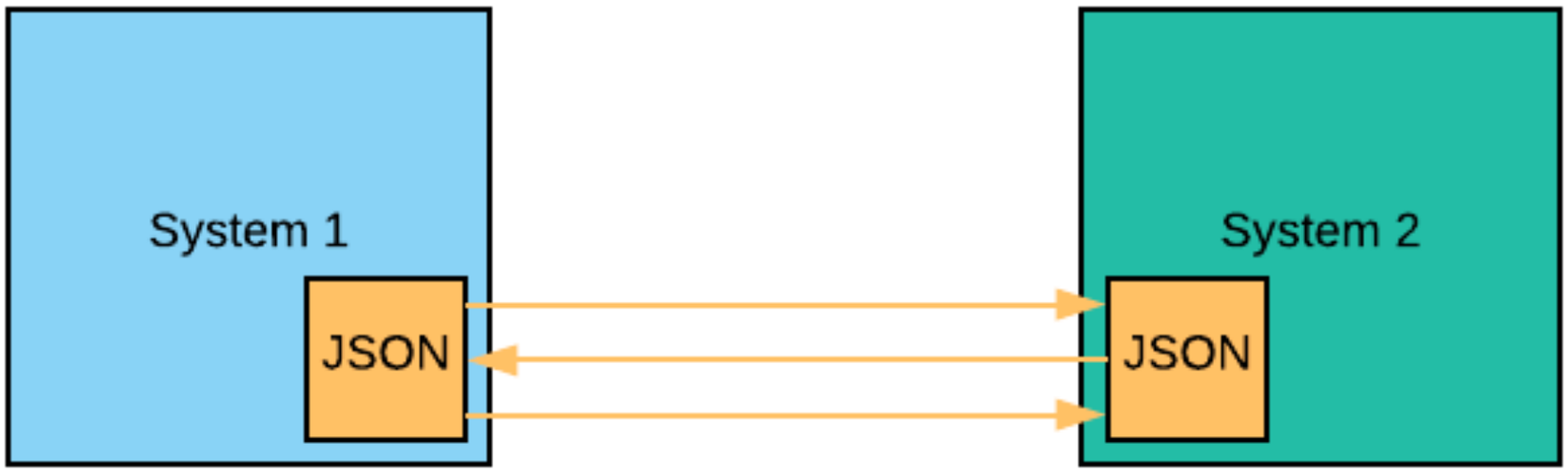
Dependency Injection (DI)

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

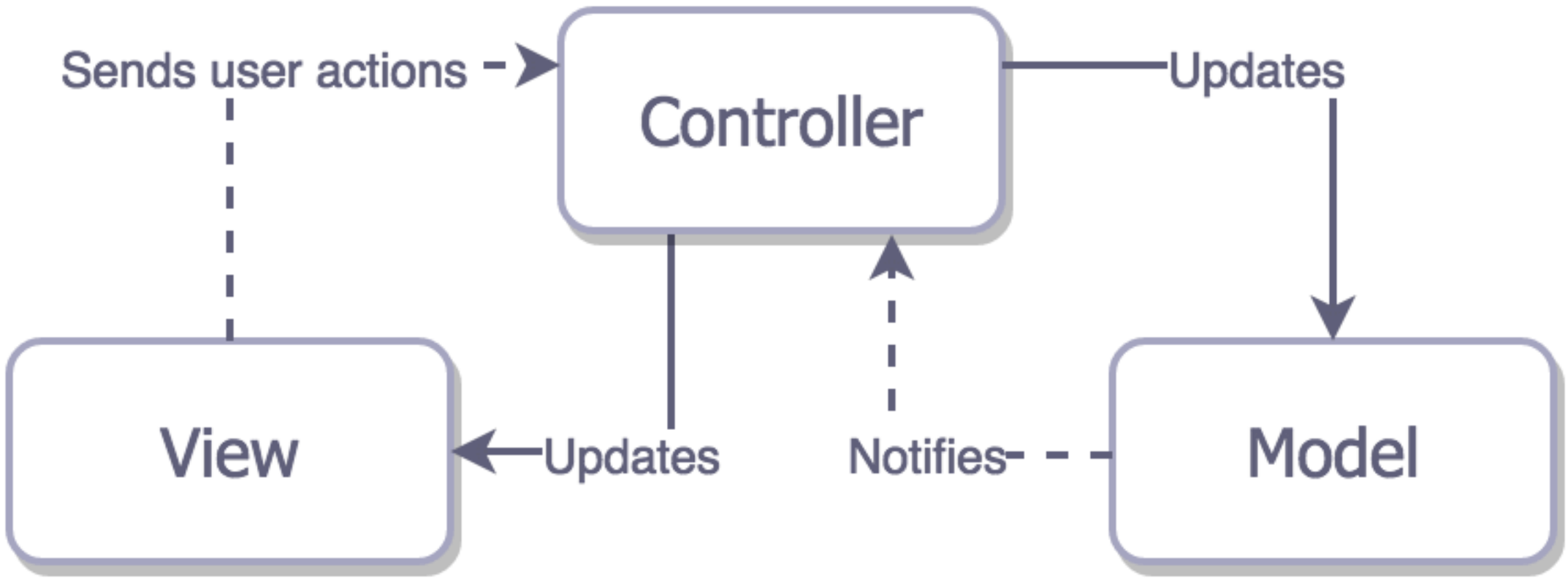
Features of good structure

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

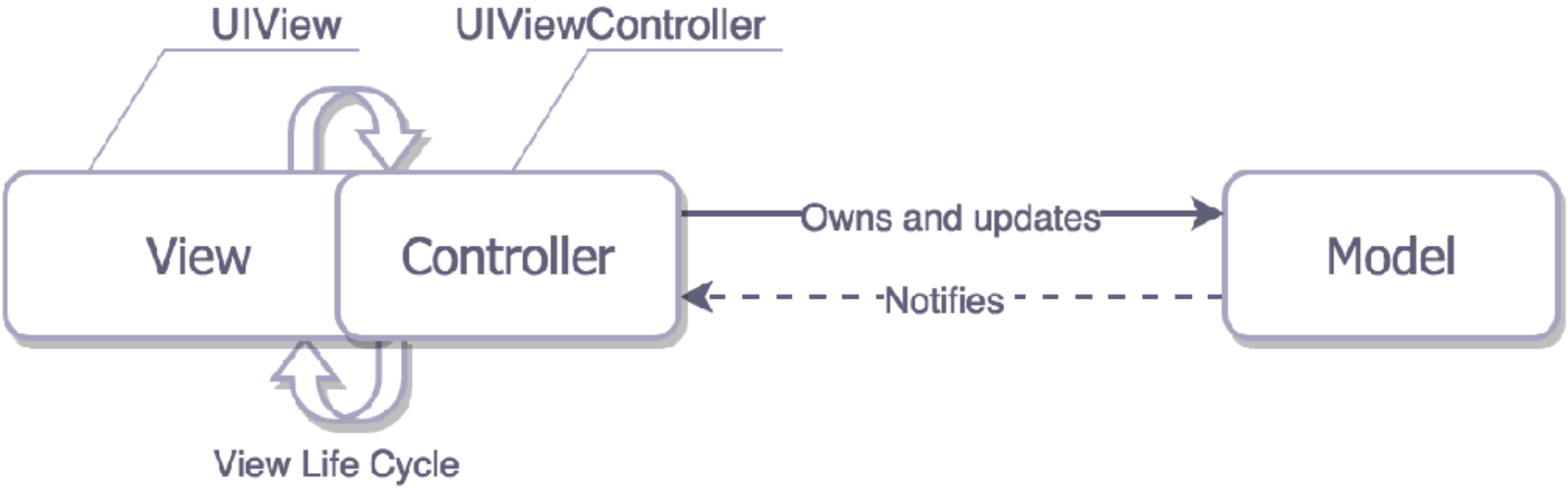
Design patterns

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

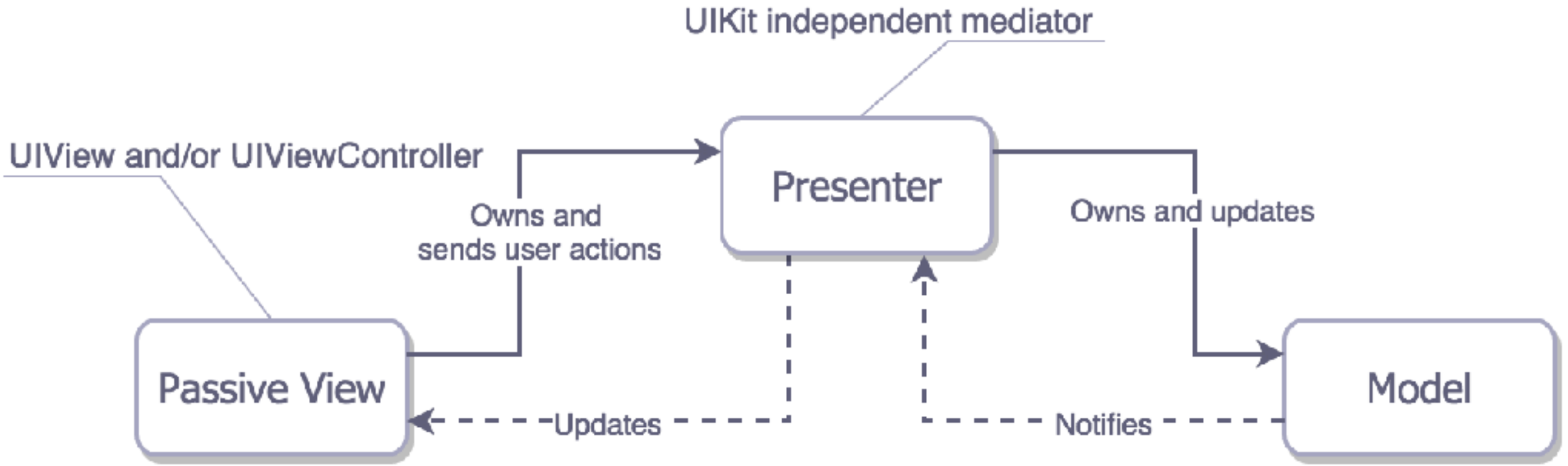
Apple MVC



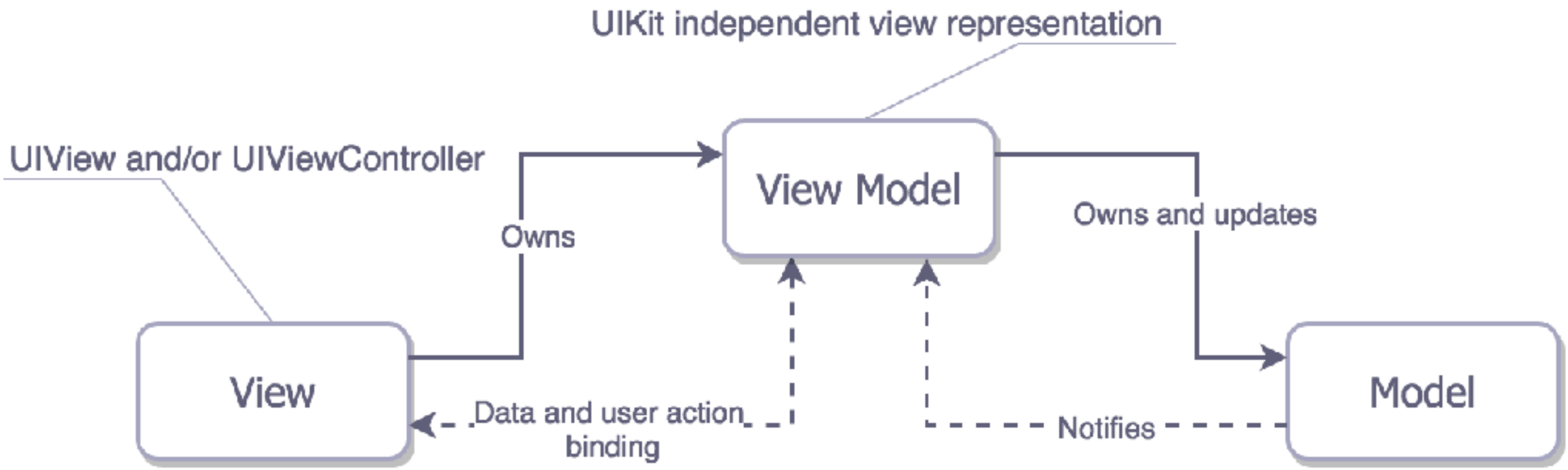
Apple MVC



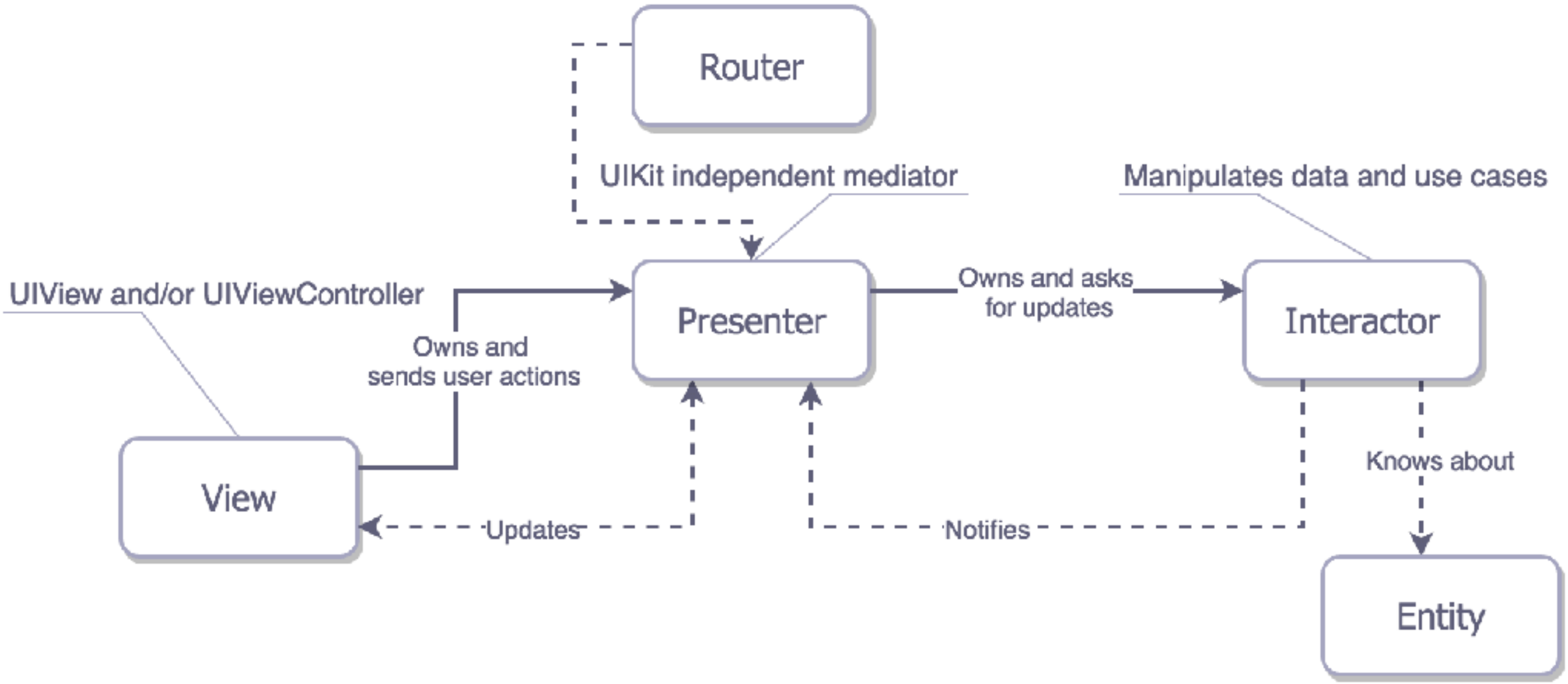
MVP



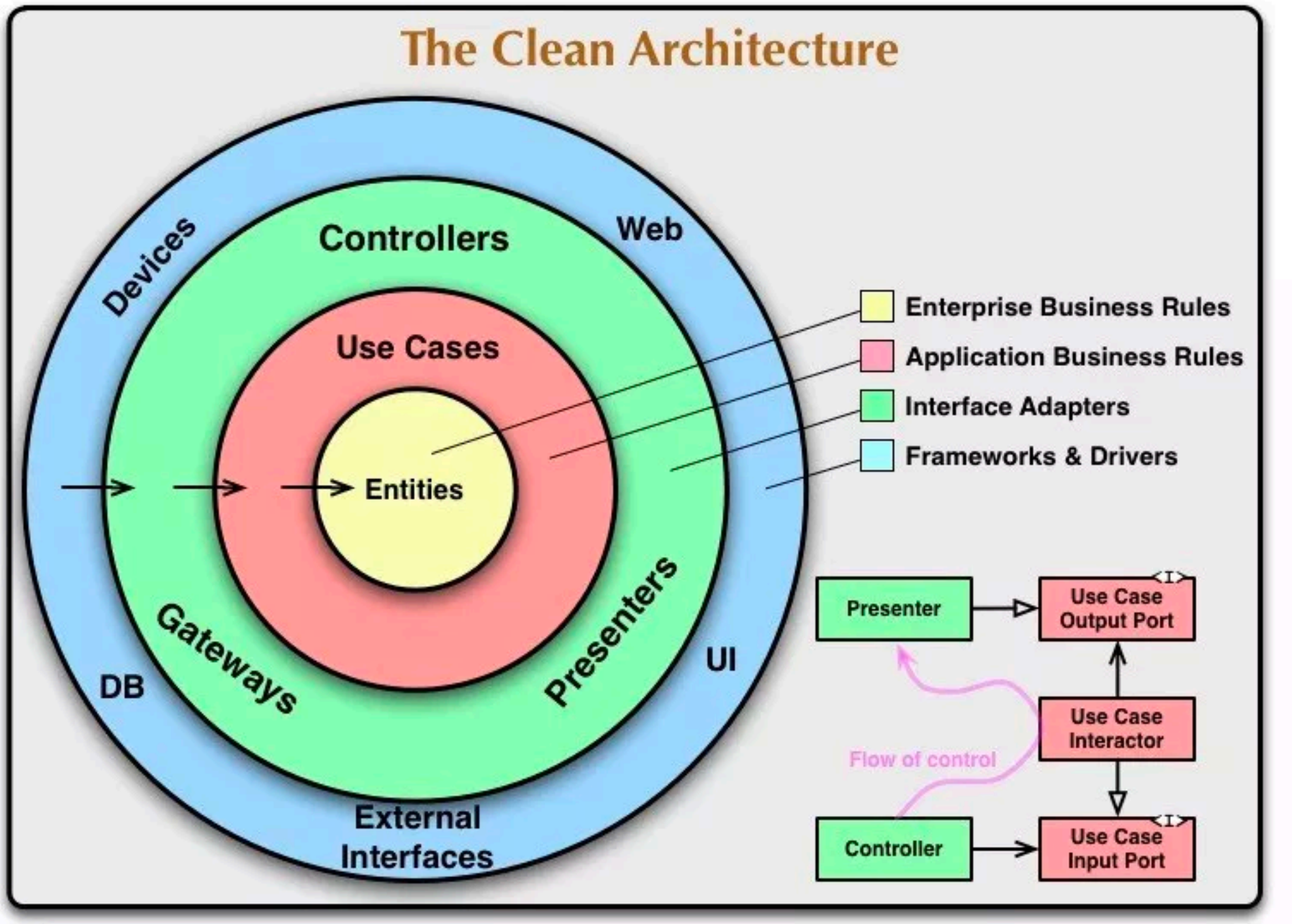
MVVM



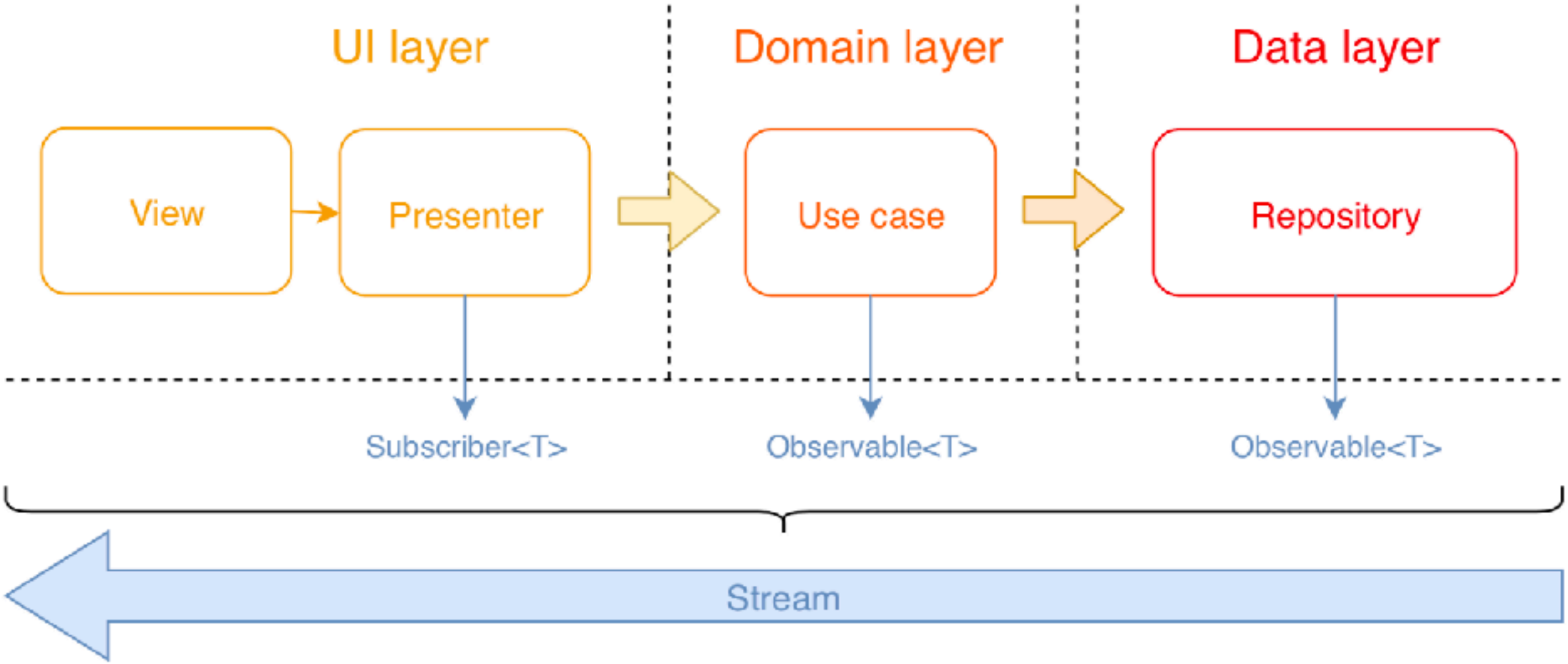
VIPER



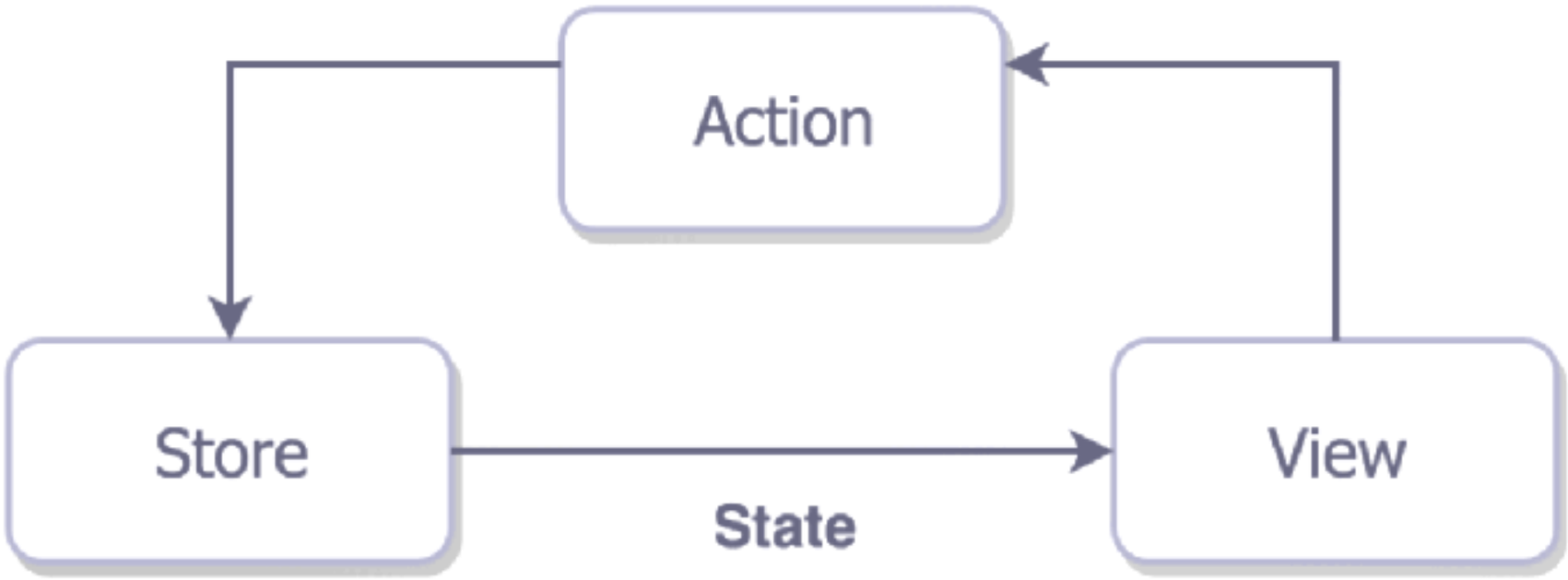
Clean Architecture



Clean Architecture



Redux-Like



No silver bullet

More testing types

More testing type and tools

Snapshot testing
Monkey testing
Performance testing
Quick and Nimble

Performance testing

Function name start with **test**

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

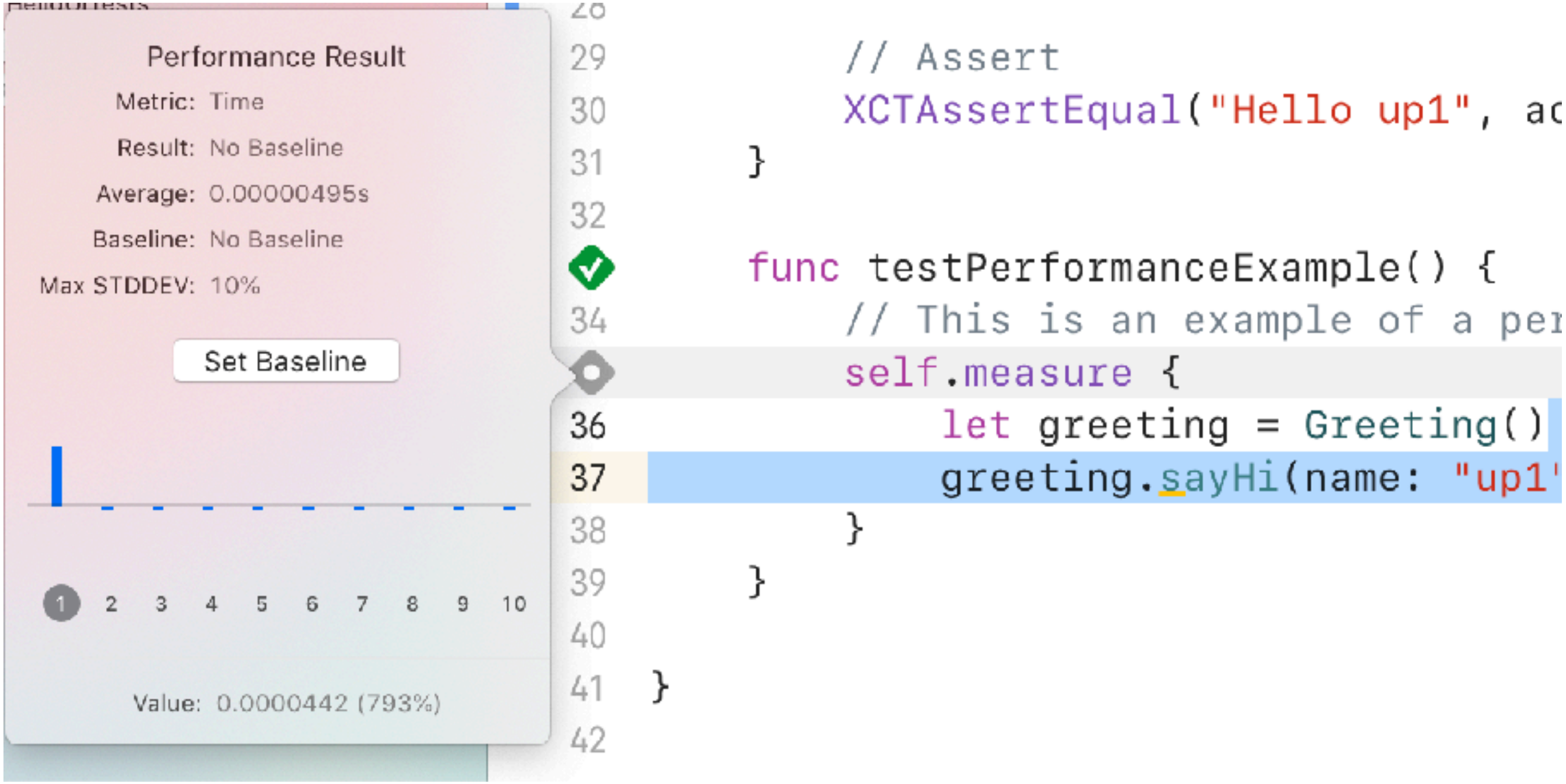
No assertions

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)
        })
    }
}
```

Working with

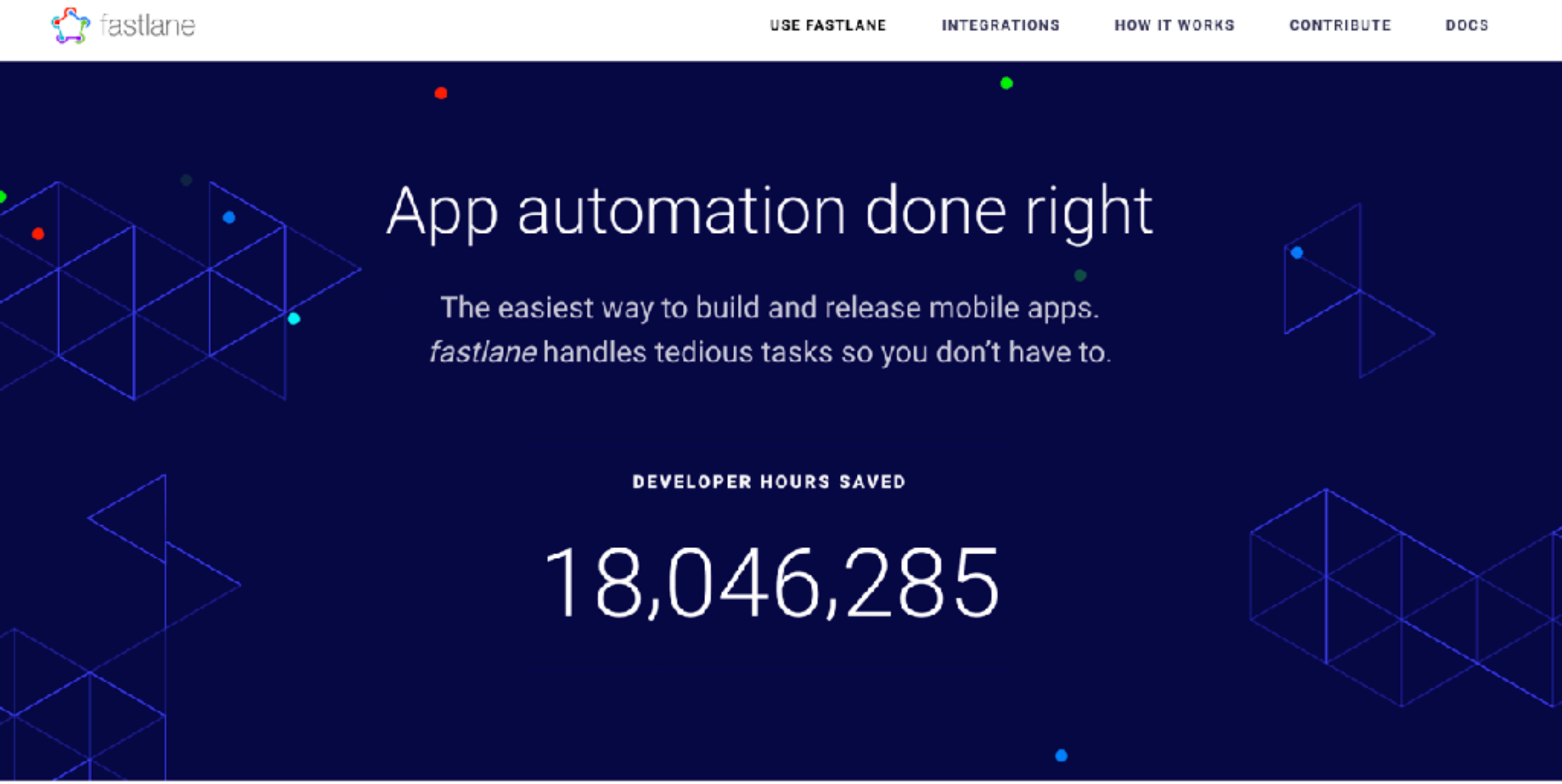
Why fastlane ?

CI/CD for iOS app

To help deployment and testing easier

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

Fastlane

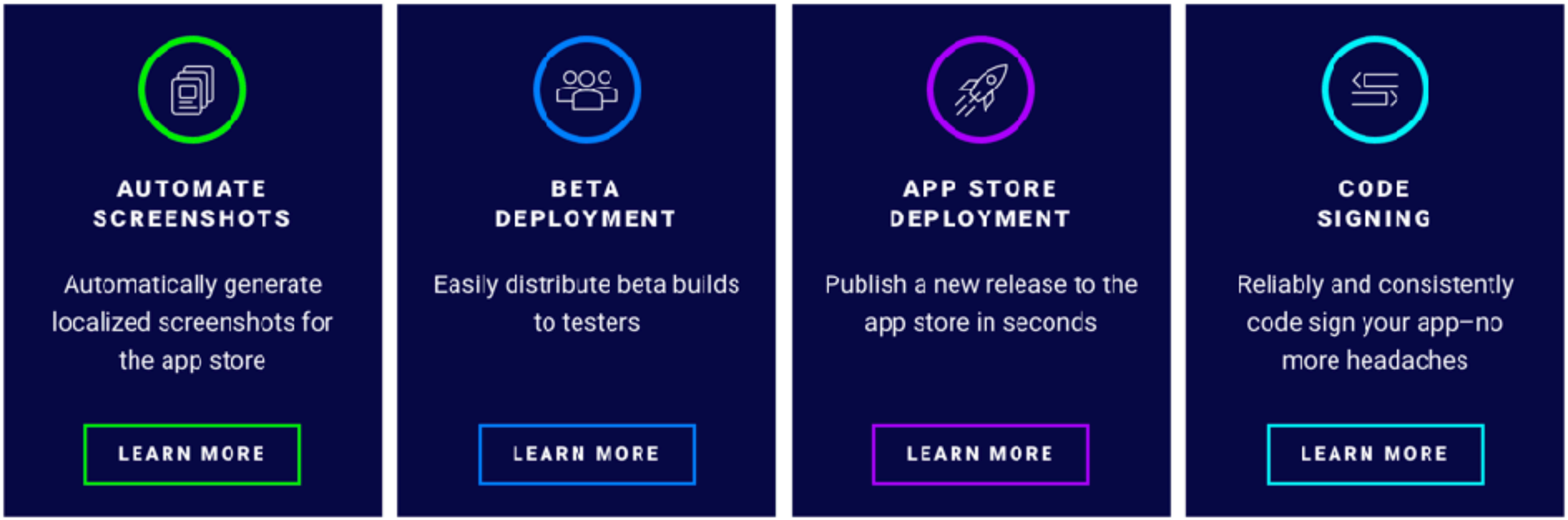


https://fastlane.tools/

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.



https://fastlane.tools/

Fastlane for iOS app

deliver snapshot frameit

pem sigh cert

produce gym scan

https://fastlane.tools/

Installation

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

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

Fastlane Action

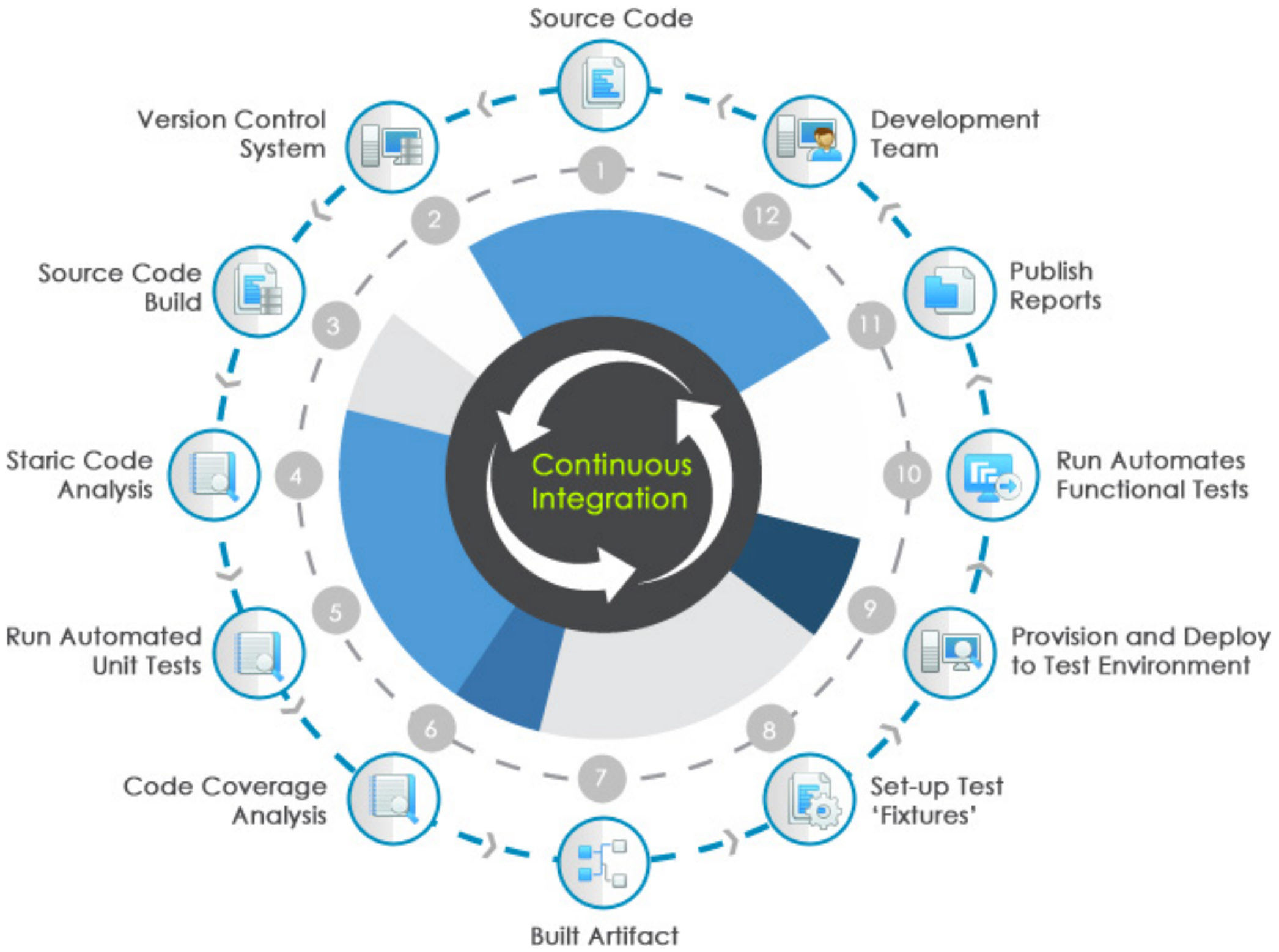
Testing => scan, xcov

Screenshot => snapshot, capture_ios_screenshots

https://docs.fastlane.tools/actions/

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**

Demo CI/CD with Jenkins