Robots

The Best Testers You Will Ever Build

Alan Cooke

— Mobile Architect @ Zendesk

Test automation isn't without it's challenges

— Me

Mobile test automation is hard

low value

iPhone 6s/6s Plus

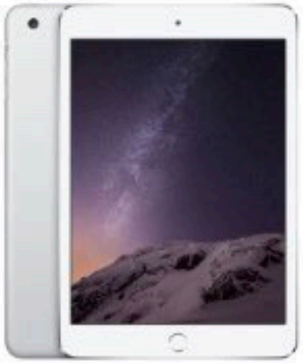
iPhone 7/7 Plus

iPhone 8/8 Plus

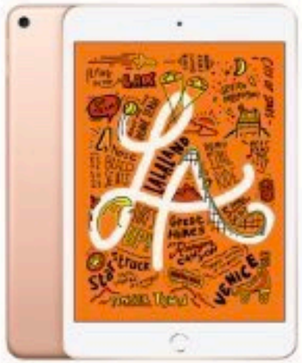
iPhone X

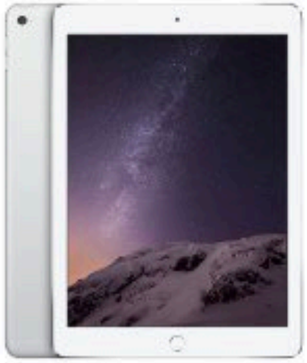
iPhone XS/ XS Max

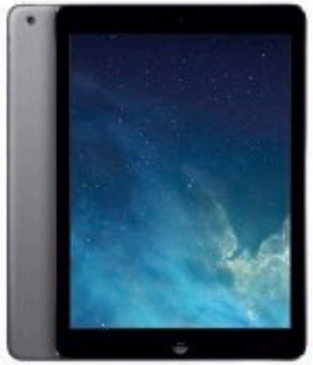
iPhone XR


iPad mini 3

iPad mini 4


iPad mini 5


iPad Air 2


iPad Air 3

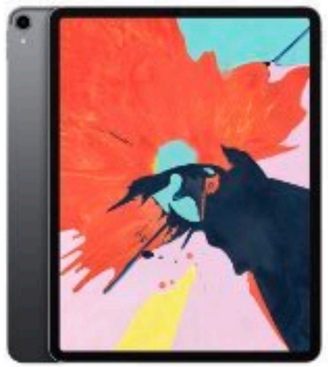
iPad 5

iPad 6

9.7-inch iPad Pro

10.5-inch iPad Pro

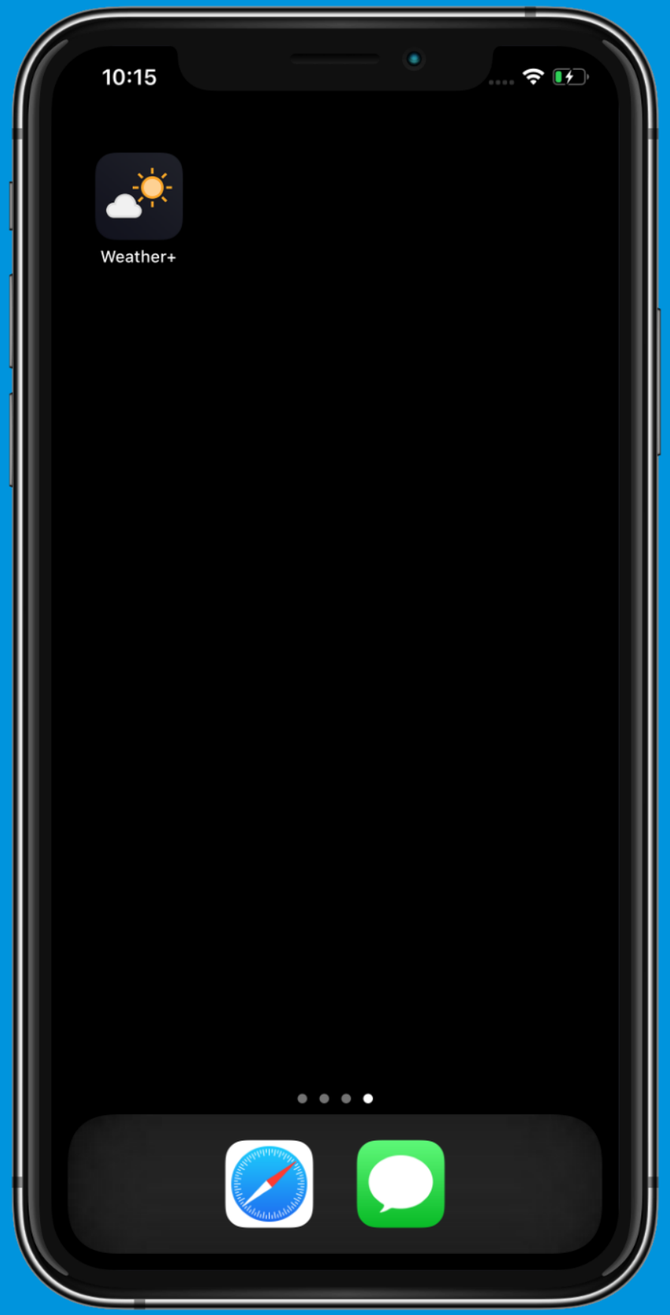
11-inch iPad Pro

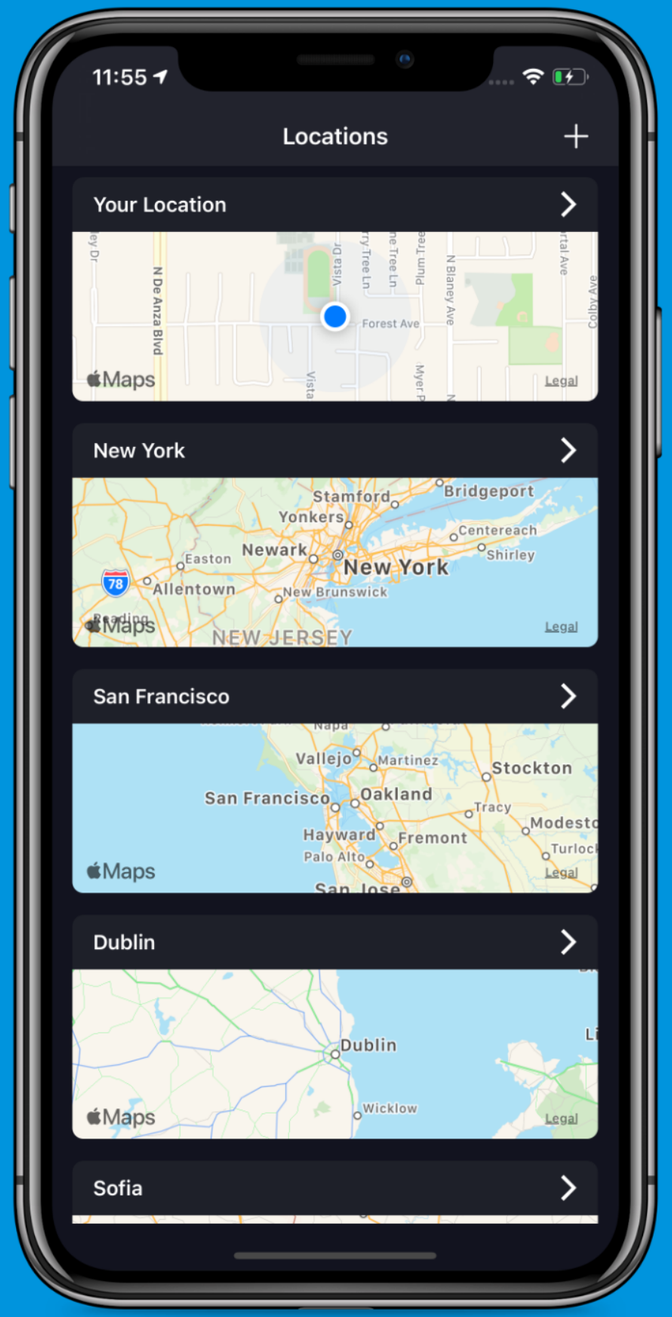

12.9-inch iPad Pro
1st, 2nd & 3rd gen

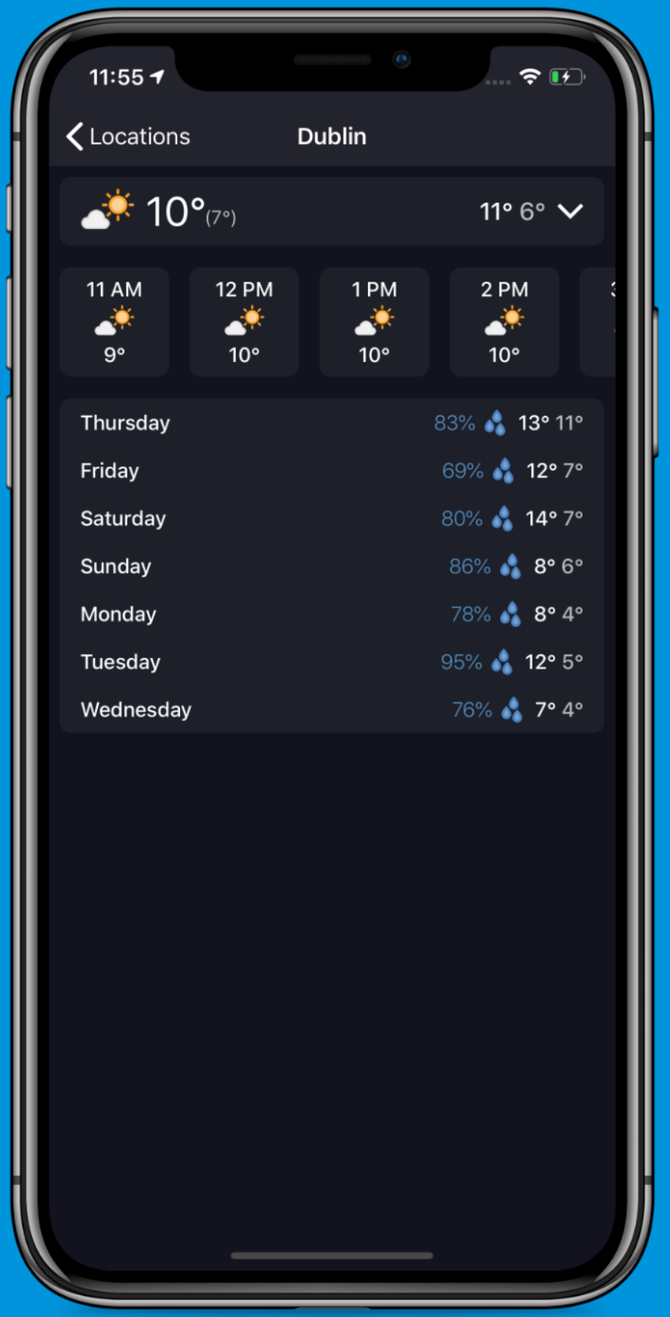
iPod Touch 6

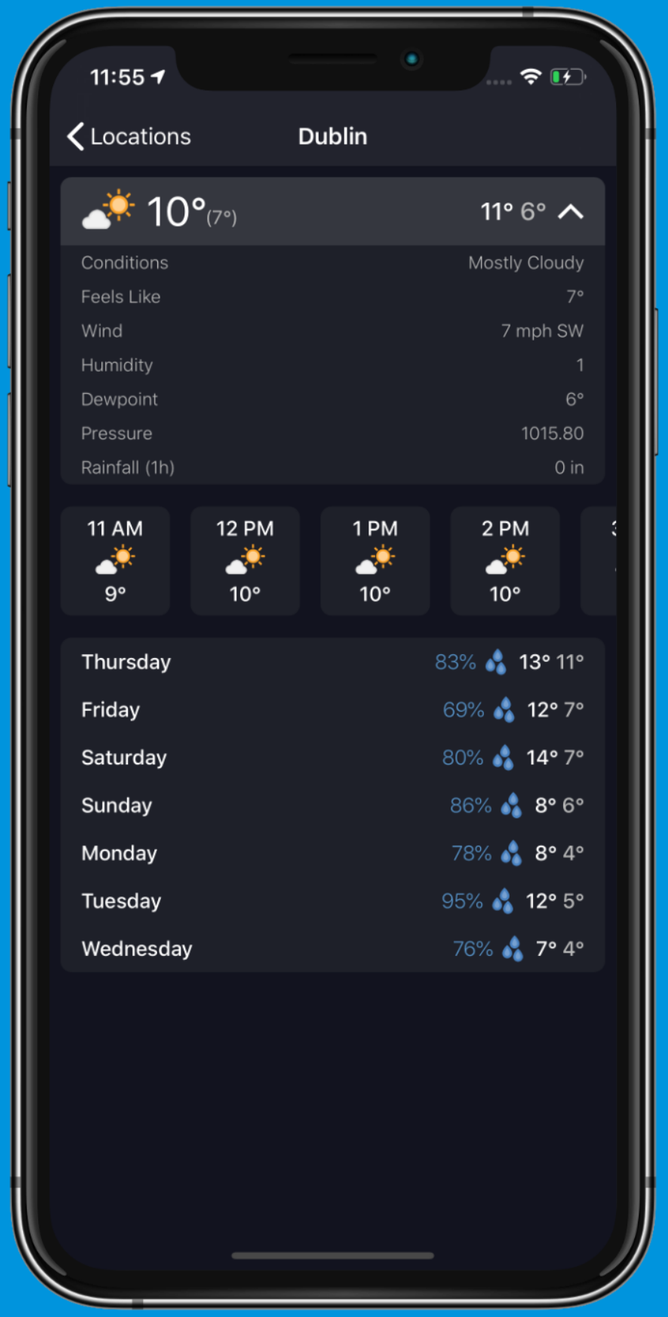
What makes a good UI test?

1. Easy to read
2. Easy to write
3. Easy to extend
4. Easy to debug


10:15
Weather+


11:55
Locations
Your Location
Forest Ave
Maps
Legal
New York
Stamford
Bridgeport
Yonkers
Centereach
Newark
Shirley
Easton
New York
Allentown
New Brunswick
NEW JERSEY
San Francisco
Vallejo
Martinez
Stockton
San Francisco
Oakland
Tracy
Hayward
Fremont
Palo Alto
Dublin
Dublin
Wicklow
Sofia


11:55
Locations
Dublin
10° (7°)
11° 6°
11 AM 9°
12 PM 10°
1 PM 10°
2 PM 10°
Thursday 83% 13° 11°
Friday 69% 12° 7°
Saturday 80% 14° 7°
Sunday 86% 8° 6°
Monday 78% 8° 4°
Tuesday 95% 12° 5°
Wednesday 76% 7° 4°


11:55
Locations
Dublin
10° (7°)
11° 6°
Conditions Mostly Cloudy
Feels Like 7°
Wind 7 mph SW
Humidity 1
Dewpoint 6°
Pressure 1015.80
Rainfall (1h) 0 in
11 AM 9°
12 PM 10°
1 PM 10°
2 PM 10°
Thursday 83% 13° 11°
Friday 69% 12° 7°
Saturday 80% 14° 7°
Sunday 86% 8° 6°
Monday 78% 8° 4°
Tuesday 95% 12° 5°
Wednesday 76% 7° 4°

Let's write our first test

1. Select a location
2. Check the weather displays
3. Check some known informations is displayed

```swift
func testSelectSavedCityDisplaysCorrectly() {

    //Given
    let city = "Dublin"
    let application = XCUIApplication()

    //When
    let collectionViewsQuery = application.collectionViews
    collectionViewsQuery.scrollViews.otherElements.staticTexts[city].tap()

    collectionViewsQuery.cells.otherElements.containing(.image, identifier:"Arrow").element.tap()

    //Then
    let predicate = NSPredicate(format: "label CONTAINS[c] %@", "Feels Like")
    let elementQuery = collectionViewsQuery.staticTexts.containing(predicate)
    XCTAssertTrue(elementQuery.count == 1)

}
```

```swift
func testSelectSavedCityDisplaysCorrectly() {

    //Given
    let city = "Dublin"
    let application = XCUIApplication()

    //When
    let collectionViewsQuery = application.collectionViews
    collectionViewsQuery.scrollViews.otherElements.staticTexts[city].tap()

    collectionViewsQuery.cells.otherElements.containing(.image, identifier:"Arrow").element.tap()

    //Then
    let predicate = NSPredicate(format: "label CONTAINS[c] %@", "Feels Like")
    let elementQuery = collectionViewsQuery.staticTexts.containing(predicate)
    XCTAssertTrue(elementQuery.count == 1)

}
```

```swift
func testSelectSavedCityDisplaysCorrectly() {

    //Given
    let city = "Dublin"
    let application = XCUIApplication()

    //When
    let collectionViewsQuery = application.collectionViews
    collectionViewsQuery.scrollViews.otherElements.staticTexts[city].tap()

    collectionViewsQuery.cells.otherElements.containing(.image, identifier:"Arrow").element.tap()

    //Then
    let predicate = NSPredicate(format: "label CONTAINS[c] %@", "Feels Like")
    let elementQuery = collectionViewsQuery.staticTexts.containing(predicate)
    XCTAssertTrue(elementQuery.count == 1)

}
```

```swift
func testSelectSavedCityDisplaysCorrectly() {

    //Given
    let city = "Dublin"
    let application = XCUIApplication()

    //When
    let collectionViewsQuery = application.collectionViews
    collectionViewsQuery.scrollViews.otherElements.staticTexts[city].tap()

    collectionViewsQuery.cells.otherElements.containing(.image, identifier:"Arrow").element.tap()

    //Then
    let predicate = NSPredicate(format: "label CONTAINS[c] %@", "Feels Like")
    let elementQuery = collectionViewsQuery.staticTexts.containing(predicate)
    XCTAssertTrue(elementQuery.count == 1)

}
```

Let's write another test

1. Swipe on a location
2. Select delete
3. Verify it is gone

```swift
func testDeleteSavedCity() {

    //Given
    let city = "Dublin"
    let application = XCUIApplication()

    //When
    let scrollViewsQuery = application.collectionViews.scrollViews
    scrollViewsQuery.otherElements.staticTexts[city].swipeRight()
    scrollViewsQuery.otherElements
      .containing(.staticText, identifier:city)
        .children(matching: .other)
        .element
          .children(matching: .other)
          .element
            .children(matching: .map)
              .element.swipeLeft()

    application.collectionViews
      .children(matching: .cell)
      .element(boundBy: 3)
      .scrollViews.otherElements
        .containing(.staticText, identifier:"Delete")
        .element.tap()

    let collectionViewsQuery = application.collectionViews

    let predicate = NSPredicate(format: "label CONTAINS[c] %@", city)
    let elementQuery = collectionViewsQuery.scrollViews.otherElements.staticTexts.containing(predicate)

    //Assert that there is no city
    XCTAssertTrue(elementQuery.count == 0)
}
```

```swift
func testDeleteSavedCity() {

    //Given
    let city = "Dublin"
    let application = XCUIApplication()

    //When
    let scrollViewsQuery = application.collectionViews.scrollViews
    scrollViewsQuery.otherElements.staticTexts[city].swipeRight()
    scrollViewsQuery.otherElements
      .containing(.staticText, identifier:city)
        .children(matching: .other)
        .element
          .children(matching: .other)
          .element
            .children(matching: .map)
              .element.swipeLeft()

    application.collectionViews
      .children(matching: .cell)
      .element(boundBy: 3)
      .scrollViews.otherElements
        .containing(.staticText, identifier:"Delete")
        .element.tap()

    let collectionViewsQuery = application.collectionViews

    let predicate = NSPredicate(format: "label CONTAINS[c] %@", city)
    let elementQuery = collectionViewsQuery.scrollViews.otherElements.staticTexts.containing(predicate)

    //Assert that there is no city
    XCTAssertTrue(elementQuery.count == 0)
}
```

```swift
func testDeleteSavedCity() {

    //Given
    let city = "Dublin"
    let application = XCUIApplication()

    //When
    let scrollViewsQuery = application.collectionViews.scrollViews
    scrollViewsQuery.otherElements.staticTexts[city].swipeRight()
    scrollViewsQuery.otherElements
      .containing(.staticText, identifier:city)
        .children(matching: .other)
        .element
          .children(matching: .other)
          .element
            .children(matching: .map)
              .element.swipeLeft()

    application.collectionViews
      .children(matching: .cell)
      .element(boundBy: 3)
      .scrollViews.otherElements
        .containing(.staticText, identifier:"Delete")
        .element.tap()

    let collectionViewsQuery = application.collectionViews

    let predicate = NSPredicate(format: "label CONTAINS[c] %@", city)
    let elementQuery = collectionViewsQuery.scrollViews.otherElements.staticTexts.containing(predicate)

    //Assert that there is no city
    XCTAssertTrue(elementQuery.count == 0)
}
```

```swift
func testDeleteSavedCity() {

    //Given
    let city = "Dublin"
    let application = XCUIApplication()

    //When
    let scrollViewsQuery = application.collectionViews.scrollViews
    scrollViewsQuery.otherElements.staticTexts[city].swipeRight()
    scrollViewsQuery.otherElements
      .containing(.staticText, identifier:city)
        .children(matching: .other)
        .element
          .children(matching: .other)
          .element
            .children(matching: .map)
              .element.swipeLeft()

    application.collectionViews
      .children(matching: .cell)
      .element(boundBy: 3)
      .scrollViews.otherElements
        .containing(.staticText, identifier:"Delete")
        .element.tap()

    let collectionViewsQuery = application.collectionViews

    let predicate = NSPredicate(format: "label CONTAINS[c] %@", city)
    let elementQuery = collectionViewsQuery.scrollViews.otherElements.staticTexts.containing(predicate)

    //Assert that there is no city
    XCTAssertTrue(elementQuery.count == 0)
}
```

```swift
func testDeleteSavedCity() {

    //Given
    let city = "Dublin"
    let application = XCUIApplication()

    //When
    let scrollViewsQuery = application.collectionViews.scrollViews
    scrollViewsQuery.otherElements.staticTexts[city].swipeRight()
    scrollViewsQuery.otherElements
      .containing(.staticText, identifier:city)
        .children(matching: .other)
        .element
          .children(matching: .other)
          .element
            .children(matching: .map)
              .element.swipeLeft()

    application.collectionViews
      .children(matching: .cell)
      .element(boundBy: 3)
      .scrollViews.otherElements
        .containing(.staticText, identifier:"Delete")
        .element.tap()

    let collectionViewsQuery = application.collectionViews

    let predicate = NSPredicate(format: "label CONTAINS[c] %@", city)
    let elementQuery = collectionViewsQuery.scrollViews.otherElements.staticTexts.containing(predicate)

    //Assert that there is no city
    XCTAssertTrue(elementQuery.count == 0)
}
```

Let's write more tests

1. Select add location
2. From Search screen, enter "Sofia, Bulgaria"
3. Confirm it displays on the list

```swift
func testSearchForCityAndSave() {

    //Given
    let citySearch = "Sofia, Bulgaria"
    let cityTitle =  "Sofia"
    let application = XCUIApplication()

    //When
    let addButton = application.navigationBars["Locations"].buttons["Add"]
    addButton.tap()

    let searchForLocationsSearchField = application.navigationBars["SimpleWeather.SearchView"].searchFields["Search for locations"]
    searchForLocationsSearchField.tap()
    searchForLocationsSearchField.typeText(citySearch)

    let searchResult = application.collectionViews.staticTexts["Sofia, Bulgaria"]
    searchResult.tap()

    //Then
    let collectionViewsQuery = application.collectionViews
    let predicate = NSPredicate(format: "label CONTAINS[c] %@", cityTitle)
    let elementQuery = collectionViewsQuery.scrollViews.otherElements.staticTexts.containing(predicate)

    //Assert that there is no city
    XCTAssertTrue(elementQuery.count == 1)

}
```

```swift
func testSearchForCityAndSave() {

    //Given
    let citySearch = "Sofia, Bulgaria"
    let cityTitle =  "Sofia"
    let application = XCUIApplication()

    //When
    let addButton = application.navigationBars["Locations"].buttons["Add"]
    addButton.tap()

    let searchForLocationsSearchField = application.navigationBars["SimpleWeather.SearchView"].searchFields["Search for locations"]
    searchForLocationsSearchField.tap()
    searchForLocationsSearchField.typeText(citySearch)

    let searchResult = application.collectionViews.staticTexts["Sofia, Bulgaria"]
    searchResult.tap()

    //Then
    let collectionViewsQuery = application.collectionViews
    let predicate = NSPredicate(format: "label CONTAINS[c] %@", cityTitle)
    let elementQuery = collectionViewsQuery.scrollViews.otherElements.staticTexts.containing(predicate)

    //Assert that there is no city
    XCTAssertTrue(elementQuery.count == 1)

}
```

```swift
func testSearchForCityAndSave() {

    //Given
    let citySearch = "Sofia, Bulgaria"
    let cityTitle =  "Sofia"
    let application = XCUIApplication()

    //When
    let addButton = application.navigationBars["Locations"].buttons["Add"]
    addButton.tap()

    let searchForLocationsSearchField = application.navigationBars["SimpleWeather.SearchView"].searchFields["Search for locations"]
    searchForLocationsSearchField.tap()
    searchForLocationsSearchField.typeText(citySearch)

    let searchResult = application.collectionViews.staticTexts["Sofia, Bulgaria"]
    searchResult.tap()

    //Then
    let collectionViewsQuery = application.collectionViews
    let predicate = NSPredicate(format: "label CONTAINS[c] %@", cityTitle)
    let elementQuery = collectionViewsQuery.scrollViews.otherElements.staticTexts.containing(predicate)

    //Assert that there is no city
    XCTAssertTrue(elementQuery.count == 1)

}
```

```swift
func testSearchForCityAndSave() {

    //Given
    let citySearch = "Sofia, Bulgaria"
    let cityTitle =  "Sofia"
    let application = XCUIApplication()

    //When
    let addButton = application.navigationBars["Locations"].buttons["Add"]
    addButton.tap()

    let searchForLocationsSearchField = application.navigationBars["SimpleWeather.SearchView"].searchFields["Search for locations"]
    searchForLocationsSearchField.tap()
    searchForLocationsSearchField.typeText(citySearch)

    let searchResult = application.collectionViews.staticTexts["Sofia, Bulgaria"]
    searchResult.tap()

    //Then
    let collectionViewsQuery = application.collectionViews
    let predicate = NSPredicate(format: "label CONTAINS[c] %@", cityTitle)
    let elementQuery = collectionViewsQuery.scrollViews.otherElements.staticTexts.containing(predicate)

    //Assert that there is no city
    XCTAssertTrue(elementQuery.count == 1)

}
```

```swift
func testSearchForCityAndSave() {

    //Given
    let citySearch = "Sofia, Bulgaria"
    let cityTitle =  "Sofia"
    let application = XCUIApplication()

    //When
    let addButton = application.navigationBars["Locations"].buttons["Add"]
    addButton.tap()

    let searchForLocationsSearchField = application.navigationBars["SimpleWeather.SearchView"].searchFields["Search for locations"]
    searchForLocationsSearchField.tap()
    searchForLocationsSearchField.typeText(citySearch)

    let searchResult = application.collectionViews.staticTexts["Sofia, Bulgaria"]
    searchResult.tap()

    //Then
    let collectionViewsQuery = application.collectionViews
    let predicate = NSPredicate(format: "label CONTAINS[c] %@", cityTitle)
    let elementQuery = collectionViewsQuery.scrollViews.otherElements.staticTexts.containing(predicate)

    //Assert that there is no city
    XCTAssertTrue(elementQuery.count == 1)

}
```

SimpleW...erUITests 〉 iPhone 11 Pro Max | Finished running Weather+ on iPhone 11 Pro Max

- SimpleWeatherTests 0 tests
- SimpleWeatherUITests 7 tests
  - SimpleWeatherUIBuilderTests
  - SimpleWeatherUIFunctionBuilderTest
    - testSelectSavedCityDisplaysCorrectly()
  - SimpleWeatherUITests
    - testSelectSavedCityDisplaysCorrectly()
    - testDeleteSavedCity()
    - testSearchForCityAndSave()

SimpleWeather

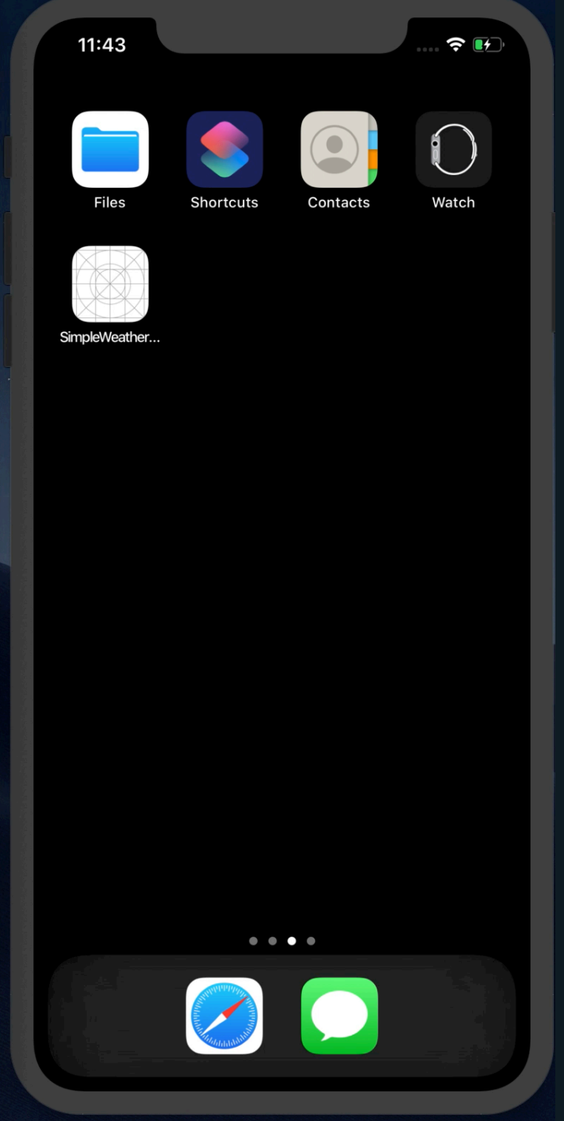
General | Signing & Capabilities | Resource Tags | Info | Build Settings | Build Phases | Build Rules

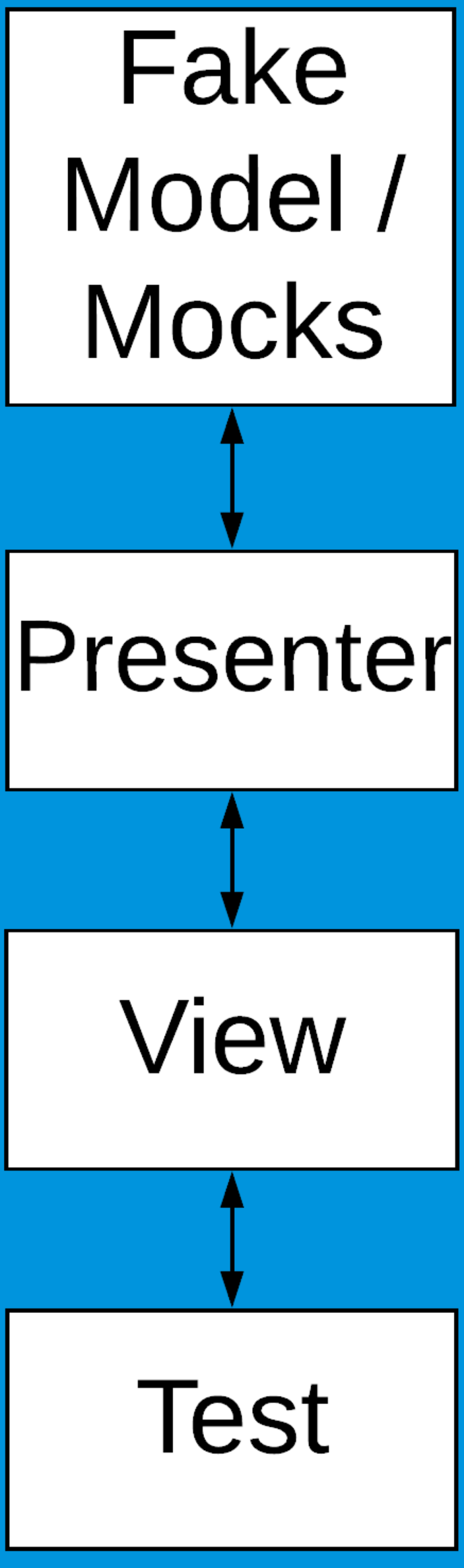
PROJECT
- SimpleWeather

TARGETS
- SimpleWeather
- SimpleWeatherTests
- SimpleWeatherUIT...

Basic | Customized | All | Combined | Levels | Swift

Localized String Swift UI Support	Yes
Interface Builder Storyboard Compiler - Options	
Setting	SimpleWeather
Default Module	SimpleWeather
Swift Compiler - Code Generation	
Setting	SimpleWeather
Compilation Mode	<Multiple values>
Debug	Incremental
Release	Whole Module
Disable Safety Checks	No
Exclusive Access to Memory	Full Enforcement (Run-time Checks in All Builds)
Optimization Level	<Multiple values>
Debug	No Optimization [-Onone]
Release	Optimize for Speed [-O]
Swift 3 @objc Inference	Default
Swift Compiler - Custom Flags	
Setting	SimpleWeather
Active Compilation Conditions	<Multiple values>
Debug	DEBUG
Release	
Other Swift Flags	
Swift Compiler - General	
Setting	SimpleWeather
Install Objective-C Compatibility Header	Yes
Objective-C Bridging Header	**SimpleWeather/SimpleWeather-Briding-Header.h**
Objective-C Generated Interface Header Name	SimpleWeather-Swift.h
Precompile Bridging Header	Yes
Reflection Metadata Level	All
Swift Compiler - Language	
Setting	SimpleWeather
Swift Language Version	**Swift 5**
Swift Compiler - Search Paths	
Setting	SimpleWeather
Import Paths	
Swift Compiler - Warnings Policies	
Setting	SimpleWeather
Suppress Warnings	**Yes**
Treat Warnings as Errors	No




Fake Model / Mocks
Presenter
View
Test

THIS IS FINE.

With proper design, the features come cheaply.

— Dennis Ritchie

Patterns for writing UI Tests

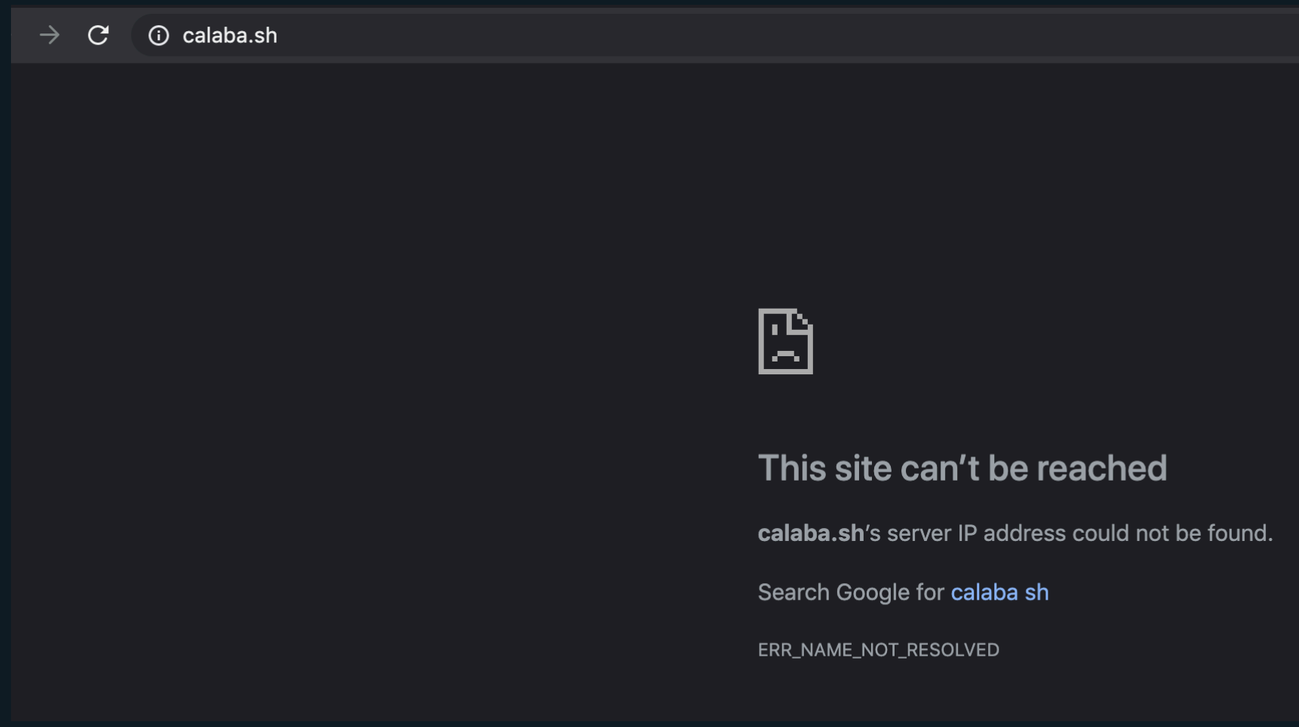
1. No pattern
2. BDD style
3. Page Objects
4. Robot

Patterns for writing UI Tests

1. ~~No pattern~~
2. BDD style
3. Page Objects

Patterns for writing UI Tests

1. ~~No pattern~~
2. ~~BDD style~~
3. Page Objects



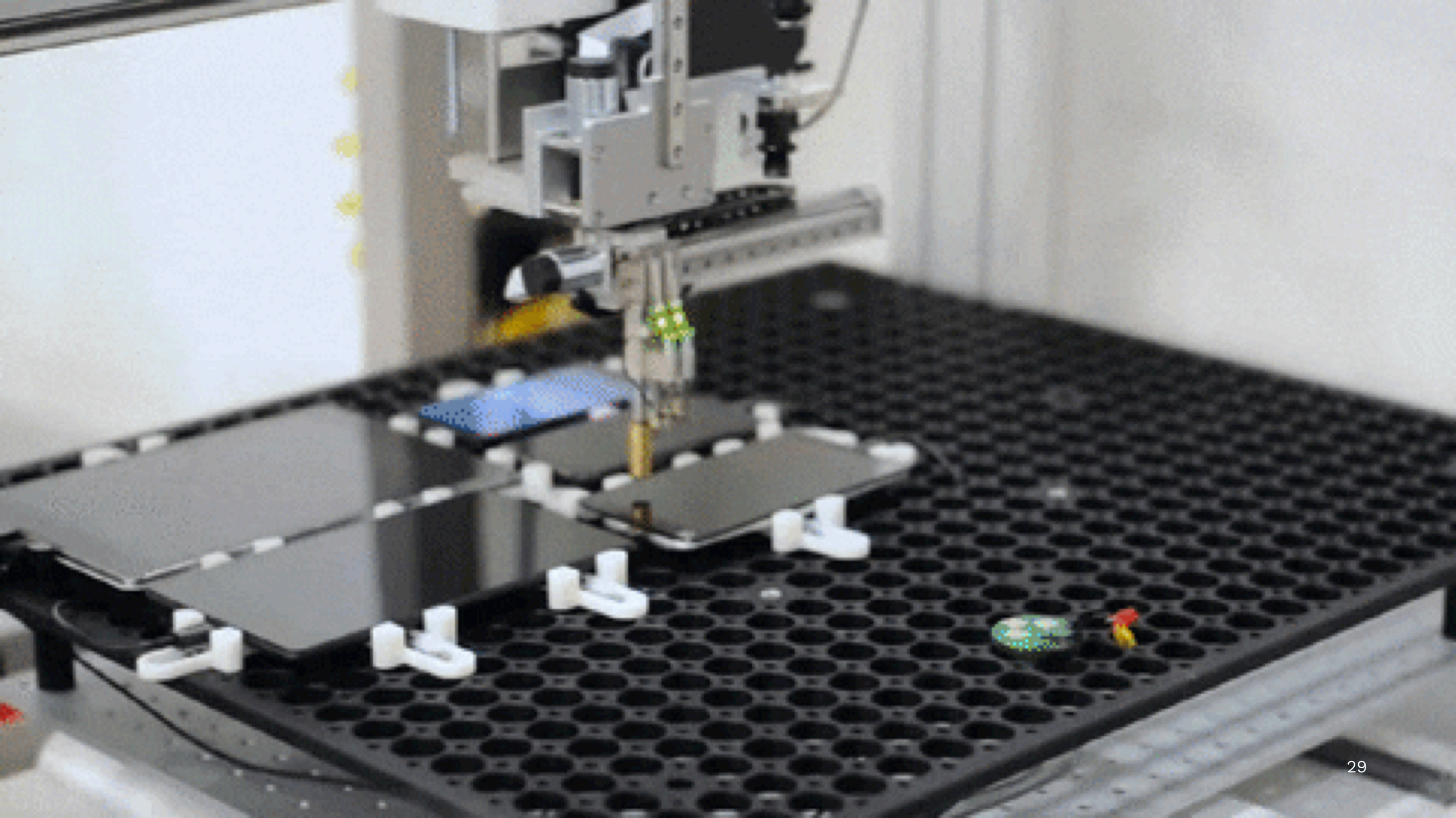
Patterns for writing UI Tests

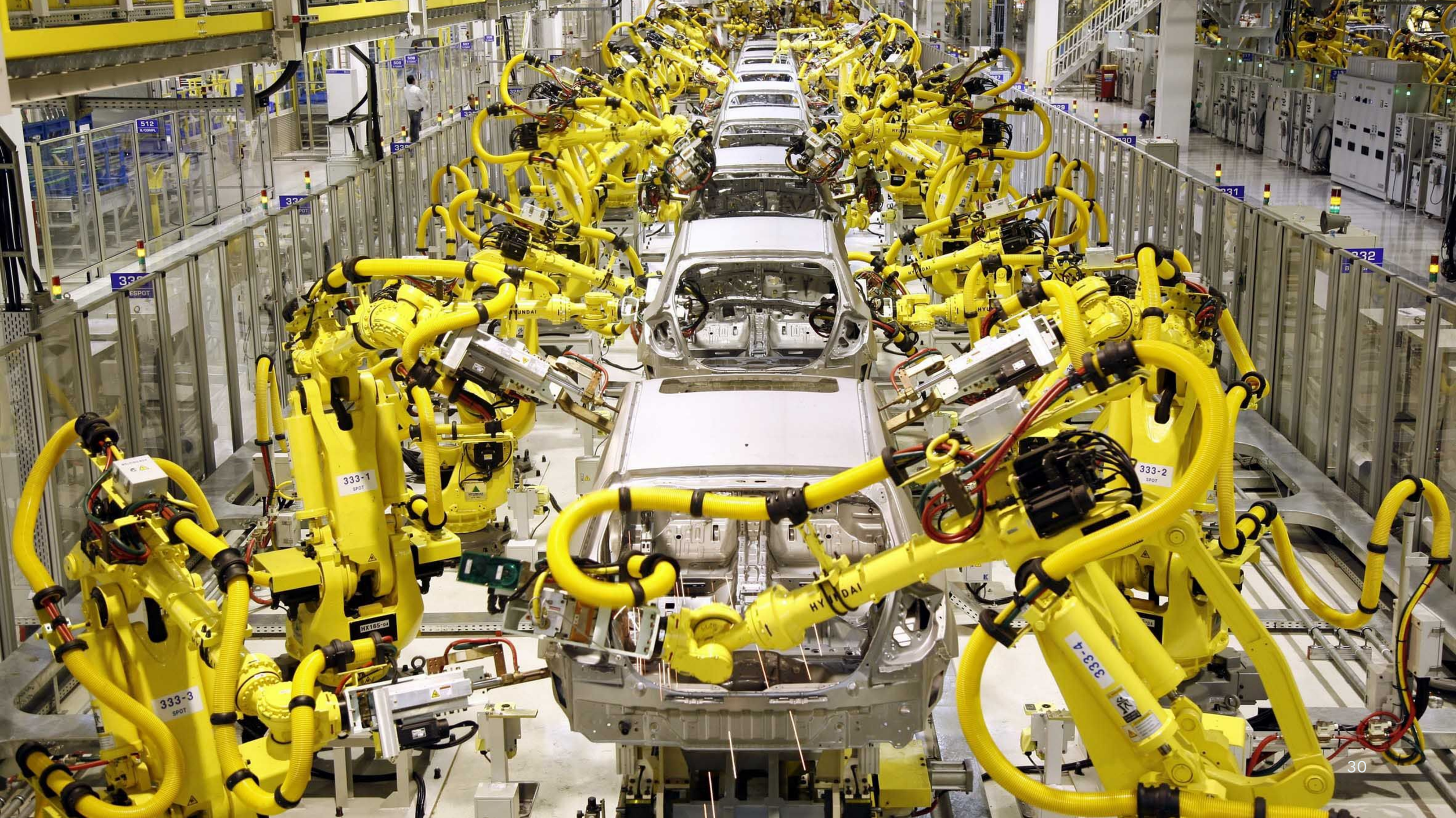
1. ~~No pattern~~
2. ~~BDD style~~
3. ~~Page Objects~~

Patterns for writing UI Tests

1. ~~No pattern~~
2. ~~BDD style~~
3. ~~Page Objects~~

Robots




333-1
SPOT
333-2
SPOT
333-3
SPOT
333-4
HYUNDAI
HX165-04

BOOKSTORE
Design

Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Constructs complex objects using step-by-step approach

— Builder Pattern

Constructs complex objects using **step-by-step** approach

— Builder Pattern

Lets build a robot

```
class WeatherListRobot {
    private var application: XCUIApplication { XCUIApplication() }
}
```

Lets build a robot

```
class WeatherListRobot {

    private var application: XCUIApplication { XCUIApplication() }

    @discardableResult
    func selectLocation(city: String) -> WeatherDetailsRobot {
        let collectionViewsQuery = XCUIApplication().collectionViews
        collectionViewsQuery.scrollViews.otherElements.staticTexts[city].tap()

        return WeatherDetailsRobot()
    }

}
```

```swift
class WeatherListRobot {

    private var application: XCUIApplication { XCUIApplication() }

    //...

    @discardableResult
    func selectSearch() -> WeatherSearchRobot {
        let addButton = application.navigationBars["Locations"].buttons["Add"]
        addButton.tap()
        return WeatherSearchRobot()
    }

    //...
}
```

```swift
class WeatherListRobot {

    private var application: XCUIApplication { XCUIApplication() }

    //...

    @discardableResult
    func deleteLocation(city: String) -> Self {

        let scrollViewsQuery = XCUIApplication().collectionViews.scrollViews
        scrollViewsQuery.otherElements.staticTexts[city].swipeRight()
        scrollViewsQuery.otherElements.staticTexts[city].swipeLeft()

        XCUIApplication().collectionViews.children(matching: .cell)
            .element(boundBy: 3)
              .scrollViews.otherElements
                .containing(.staticText, identifier:"Delete").element.tap()

        return self
    }

    //...
}
```

```swift
class WeatherListRobot {

    private var application: XCUIApplication { XCUIApplication() }

    //...

    @discardableResult
    func deleteLocation(city: String) -> Self {

        let scrollViewsQuery = XCUIApplication().collectionViews.scrollViews
        scrollViewsQuery.otherElements.staticTexts[city].swipeRight()
        scrollViewsQuery.otherElements.staticTexts[city].swipeLeft()

        XCUIApplication().collectionViews.children(matching: .cell)
            .element(boundBy: 3)
              .scrollViews.otherElements
                .containing(.staticText, identifier:"Delete").element.tap()

        return self
    }

    //...
}
```

```swift
class WeatherListRobot {

    private var application: XCUIApplication { XCUIApplication() }

    //...

    @discardableResult
    func deleteLocation(city: String) -> Self {

        let scrollViewsQuery = XCUIApplication().collectionViews.scrollViews
        scrollViewsQuery.otherElements.staticTexts[city].swipeRight()
        scrollViewsQuery.otherElements.staticTexts[city].swipeLeft()

        XCUIApplication().collectionViews.children(matching: .cell)
            .element(boundBy: 3)
              .scrollViews.otherElements
                .containing(.staticText, identifier:"Delete").element.tap()

        return self
    }

    //...
}
```

Lets go back to the tests

Test 1: Select, Display & Verify

```
func testSelectSavedCityDisplaysCorrectly() {
    //Given
    let listRobot = WeatherListRobot();
    let verifyDetailsRobot = WeatherDetailsVerifierRobot()
    //When
    listRobot.selectLocation(city: "Dublin")
    .expandWeatherDetails()
    //Then
    verifyDetailsRobot.verifyWeatherDetailsExpanded()
}
```

Test 1: Select, Display & Verify

```
func testSelectSavedCityDisplaysCorrectly() {
    //Given
    let listRobot = WeatherListRobot();
    let verifyDetailsRobot = WeatherDetailsVerifierRobot()
    //When
    listRobot.selectLocation(city: "Dublin")
    .expandWeatherDetails()
    //Then
    verifyDetailsRobot.verifyWeatherDetailsExpanded()
}
```

Test 1: Select, Display & Verify

```
func testSelectSavedCityDisplaysCorrectly() {
    //Given
    let listRobot = WeatherListRobot();
    let verifyDetailsRobot = WeatherDetailsVerifierRobot()
    //When
    listRobot.selectLocation(city: "Dublin")
    .expandWeatherDetails()
    //Then
    verifyDetailsRobot.verifyWeatherDetailsExpanded()
}
```

Test 1: Select, Display & Verify

```
func testSelectSavedCityDisplaysCorrectly() {
    //Given
    let listRobot = WeatherListRobot();
    let verifyDetailsRobot = WeatherDetailsVerifierRobot()
    //When
    listRobot.selectLocation(city: "Dublin")
    .expandWeatherDetails()
    //Then
    verifyDetailsRobot.verifyWeatherDetailsExpanded()
}
```

Test 1: Select, Display & Verify

```
func testSelectSavedCityDisplaysCorrectly() {
    //Given
    let listRobot = WeatherListRobot();
    let verifyDetailsRobot = WeatherDetailsVerifierRobot()
    //When
    listRobot.selectLocation(city: "Dublin")
    .expandWeatherDetails()
    //Then
    verifyDetailsRobot.verifyWeatherDetailsExpanded()
}
```

Test 2: Delete & Verify

```
func testDeleteSavedCity() {
    //Given
    let listRobot = WeatherListRobot();
    let verifierRobot = WeatherListVerifiierRobot();
    //When
    listRobot.deleteLocation(city: "Dublin")
    //Then
    verifierRobot.verifyLocationGone(city: "Dublin")
}
```

Test 2: Delete & Verify

```
func testDeleteSavedCity() {
    //Given
    let listRobot = WeatherListRobot();
    let verifierRobot = WeatherListVerifiierRobot();
    //When
    listRobot.deleteLocation(city: "Dublin")
    //Then
    verifierRobot.verifyLocationGone(city: "Dublin")
}
```

Test 3: Search, Select, Save & Verify

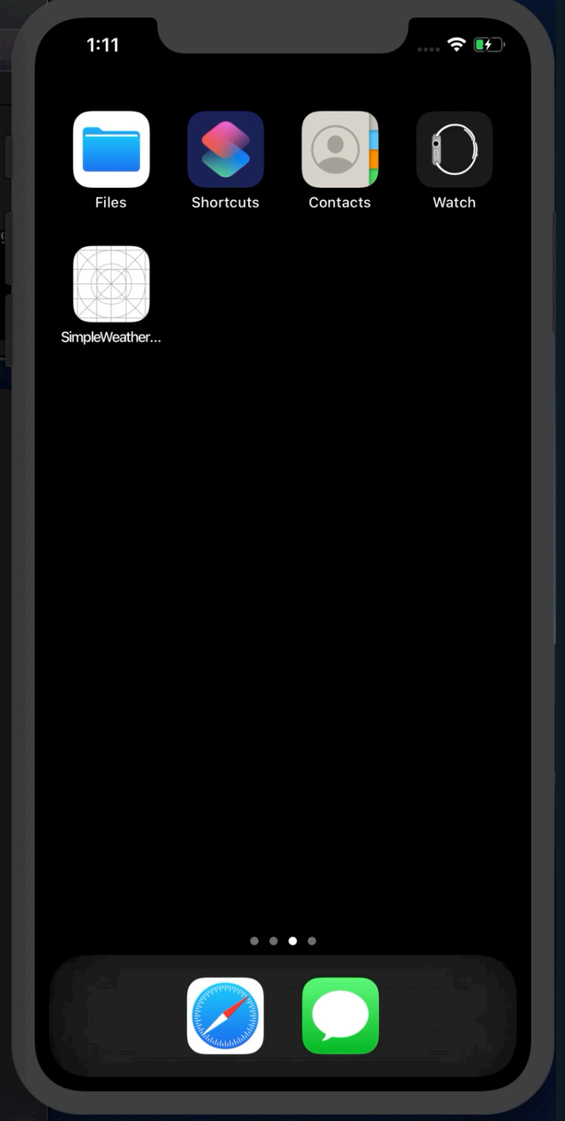
```
func testSearchForCityAndSave() {
    //Given
    let listRobot = WeatherListRobot();
    //When
    listRobot.selectSearch()
    .searchForLocation(query: "Sofia, Bulgaria")
    .selectLocation(location: "Sofia, Bulgaria")
    //Then
    let verifierRobot = WeatherListVerifiierRobot();
    verifierRobot.verifyLocationsVisible(cities: "Sofia")
}
```

SimpleW...erUITests 〉 iPhone 11 Pro Max | Finished running Weather+ on iPhone 11 Pro Max

SimpleWeather 〉 Si...ests 〉 SimpleWeatherUIBuilderTests.swift 〉 testSelectSavedCityDisplaysCorrectly()

- SimpleWeatherTests 0 tests
- SimpleWeatherUITests 7 tests
  - SimpleWeatherUIBuilderTests
    - testSelectSavedCityDisplaysCorrectly()
    - testDeleteSavedCity()
    - testSearchForCityAndSave()
  - SimpleWeatherUIFunctionBuilderTest
  - SimpleWeatherUITests

```
//
//  Created by Alan Cooke on 11/20/19.
//  Copyright © 2019 Ryan Nystrom. All rights reserved.
//

import XCTest

class SimpleWeatherUIBuilderTests: XCTestCase {

    override func setUp() {
        super.setUp()

        continueAfterFailure = false

        let app = XCUIApplication()
        app.launch()
    }

    func testSelectSavedCityDisplaysCorrectly() {
        //Given
        let listRobot = WeatherListRobot();
        let verifyDetailsRobot = WeatherDetailsVerifierRobot()
        //When
        listRobot.selectLocation(city: "Dublin")
        .expandWeatherDetails()
        //Then
        verifyDetailsRobot.verifyWeatherDetailsExpanded()
    }

    func testDeleteSavedCity() {
        //Given
        let listRobot = WeatherListRobot();
        let verifierRobot = WeatherListVerifiierRobot();
        //When
        listRobot.deleteLocation(city: "Dublin")
        //Then
        verifierRobot.verifyLocationGone(city: "Dublin")
    }

    func testSearchForCityAndSave() {
        //Given
        let listRobot = WeatherListRobot();
        //When
        listRobot.selectSearch()
        .searchForLocation(query: "Sofia, Bulgaria")
        .selectLocation(location: "Sofia, Bulgaria")
        //Then
        let verifierRobot = WeatherListVerifiierRobot();
        verifierRobot.verifyLocationsVisible(cities: "Sofia")
    }
}
```



Test
Robot
View
Presenter
Fake
Model /
Mocks

Swift 5

Let's apply some of Swift 5's language features and see how where go

Property ~~Delegates~~ Wrappers

- Introduced in Swift 5.x
- Offers the ability to extend the behaviour of properties
- Similar to Java Annotations but solely focused to properties, for now..

```swift
@propertyWrapper
struct ConstructionWrapper<Value: XCUIApplication> {

    var wrappedValue: Value

    init() {
        wrappedValue = Value()
    }

}
```

```swift
@propertyWrapper
struct ConstructionWrapper<Value: XCUIApplication> {

    var wrappedValue: Value

    init() {
        wrappedValue = Value()
    }

}
```

```swift
@propertyWrapper
struct ConstructionWrapper<Value: XCUIApplication> {

    var wrappedValue: Value

    init() {
        wrappedValue = Value()
    }

}
```

```swift
@propertyWrapper
struct ConstructionWrapper<Value: XCUIApplication> {

    var wrappedValue: Value

    init() {
        wrappedValue = Value()
    }

}
```

```swift
class WeatherListRobot {

  @ConstructionWrapper
  private var application: XCUIApplication

}
```

```
class WeatherListRobot {

  @ConstructionWrapper
  private var application: XCUIApplication

}
```

Trailing Closures

```
extension EvolvedWeatherDetailsVerifierRobot {
    convenience init(_ closure: (EvolvedWeatherDetailsVerifierRobot) -> Void) {
        self.init()
        closure(self)
    }
}
```

Trailing Closures

```
extension EvolvedWeatherDetailsVerifierRobot {
    convenience init(_ closure: (EvolvedWeatherDetailsVerifierRobot) -> Void) {
        self.init()
        closure(self)
    }
}
```

Trailing Closures

```
extension EvolvedWeatherDetailsVerifierRobot {
    convenience init(_ closure: (EvolvedWeatherDetailsVerifierRobot) -> Void) {
        self.init()
        closure(self)
    }
}
```

Trailing Closures

```
extension EvolvedWeatherDetailsVerifierRobot {
    convenience init(_ closure: (EvolvedWeatherDetailsVerifierRobot) -> Void) {
        self.init()
        closure(self)
    }
}
```

Revisit the tests

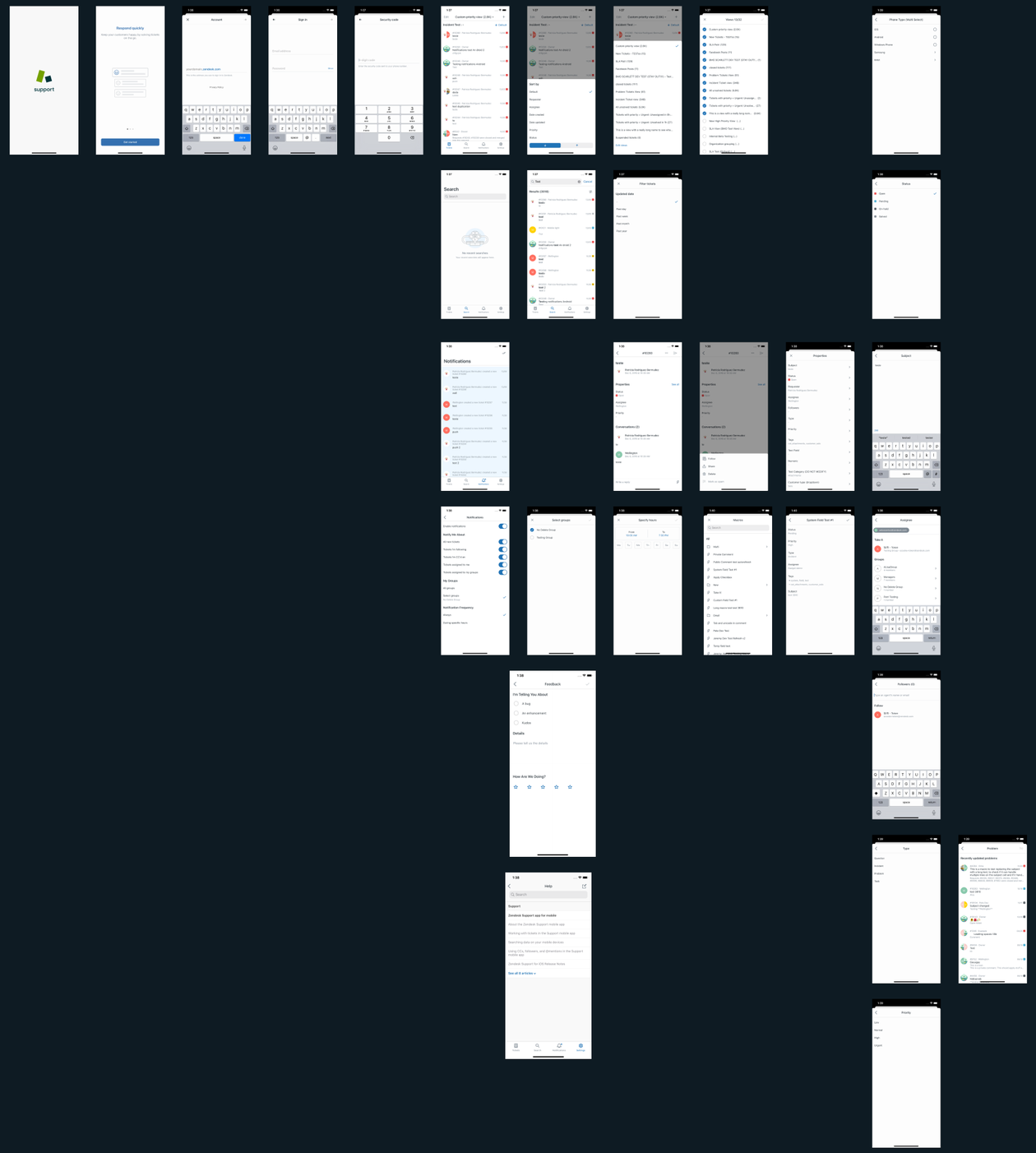
```
func testSelectSavedCityDisplaysCorrectly() {

    EvolvedWeatherListRobot {
        $0.selectLocation(city: "Dublin")
        .expandWeatherDetails()
    }

    EvolvedWeatherDetailsVerifierRobot {
        $0.verifyWeatherDetailsExpanded()
    }

}
```

What makes a good UI test?

1. Easy to read ✅
2. Easy to write ✅
3. Easy to extend ✅
4. Easy to debug ✅

Case study: Zendesk Support

1. Universal app (iOS & iPadOS)
2. Predominately written in Swift
3. EarlGrey 2
4. OHHTTPStubs
5. Run on every Pull Request


support
Respond quickly
Account
Sign In
Security code
Search
Filter tickets
Status
Notifications
Properties
Subject
Select groups
Specify hours
Macros
Assignee
Feedback
I'm Telling You About
Details
How Are We Doing?
Help
Support
Zendesk Support app for mobile
Followers (0)
Type
Problem
Priority

Kotlin tests follow same approach

```kotlin
@Test
fun viewIdShouldSelectView() {
    Preferences.setLastView("invalid view id")

    activity.launch()

    TicketListRobot
            .verifyViewSelected("Your unsolved tickets")
            .openViewSelector()
            .selectView(position = 1)

    val selectedViewId = Preferences.getLastView()
    assertEquals(selectedViewId, knownSelectedViewID)
}
```

Open Source

- https://github.com/a1cooke/codemonster2019
- Simple Weather - https://github.com/rnystrom/SimpleWeather
- Testing Robots

Thank you!

Twitter: @a1cooke
Github: a1cooke