Class

1st

Date

05•Sep•18

iOS Engineer

Capital One

Route

Swift Testing

Twitter

@RobRWAPP

From

Rob Whitaker

Email

rw@rwapp.co.uk

To

iOSDevUK

Website

rwapp.co.uk

SWIFT TESTING JOURNEY

Indie Testing

F**k it,
That'll do

Test Example

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

7:51

Capital One

Card ending 2346 | Credit limit: £5,550

Current balance
£100.99

Available to spend
£456.44

Payment details

Payment due by
Friday 18th December

Minimum amount
£12.00

Last statement balance
£785.45

Pay off your balance faster
A small change to your Direct Debit could make a big difference. →

Your **Direct Debit** will come out of your bank account on or after **23rd September**.

Last payment

Your last payment of £11.52 on Friday 12th December was accepted.

Recent payments may take 2 working days to appear here.

Recent transactions

DIRECT DEBIT PAYMENT + £62.57

At a glance | Transactions | Payment | My card | More

7:53

Recent transactions

All | In | Out | Pending

Sunday 9th October

DIRECT DEBIT PAYMENT
THANK YOU
+ £62.57

PAYMENT ADJUSTMENT
£62.57

Tuesday 7th April

DIRECT DEBIT PAYMENT
THANK YOU
+ £401.55

Friday 3rd April

LINCOLN SPORT PARACHUTE CLUB NEBRASKA
UNITED STATES OF AMERICA
£300.00
Pending

Wednesday 4th February

WILKO RETAIL LIMIT
ILKESTON 176
£23.22

DUNELM (SOFT FURNISHING) - MAKE YOUR HOME LOVELY
ILKESTON
£9.99

Tuesday 3rd February

At a glance | Transactions | Payment | My card | More

7:49

My card

Capital One CREDIT
•••• •••• •••• 2346
EXPIRES ••/•• CVV •••
mastercard

My card

View my PIN →

Manage my card

My card has arrived, activate it →

My card has been lost or stolen →

At a glance | Transactions | Payment | My card | More

@RobRWAPP

We're Hiring

@RobRWAPP

Testing

Writing tests is like eating vegetables

Boosts confidence in your code

Prevents regressions

Means any code that's been through your pipeline is releasable

Helps to document code

Unit Testing

'Poking & prodding'

Objective-C

Dependency injection (Objection)

Mocking framework (OCMock)

http://objection-framework.org/

http://ocmock.org/

Swift

How do you unit test in Swift?

How do you test a combined Swift/Obj-C app?

How do you inject dependancies?

Obj-C mocking libraries don't work/are limited with Swift

System Under Test

```
class TestObject {
    let manager = Manager()

    func methodUnderTest() -> String {
        var counter = 0

        while counter < 100 {
            counter += 1
        }

        let value = manager.methodToCall(value: counter)
        return "\(value)!"
    }
}
```

Protocols

Protocols make it simple to create manual mocks

Protocols can bridge Swift & Obj-C

Protocol all the things

```
protocol ManagerProtocol {
    func methodToCall(withValue: Int) -> String
}
```

Manual Mocks

Coding a mock class that conforms to the protocol

Methods capture arguments received

Tests can set the value to return

Count how many times the method is called

Manual Mocks

```
class MockManager: ManagerProtocol {
    private(set) var methodCalled = 0
    var returnValue: String?
    private(set) var receivedValue: Int?

    func methodToCall(value: Int) -> String {
        methodCalled += 1
        receivedValue = value
        return returnValue!
    }
}
```

Manual Mocks

```
let mockManager = MockManager()
    mockManager.returnValue = "Hooray"
```

```
    XCTAssertEqual(mockManager.methodCalled, 1)
    XCTAssertEqual(result, "Hooray!")
    XCTAssertEqual(mockManager.value, 100)
```

Dependency Injection with init()

Inject objects into classes through default values in inits

```
class ExampleObject {
   init(manager: ManagerProtocol = Manager()) {
```

```
let object = ExampleObject()
```

In tests, pass in mocks

```
let mockManager = MockManager()
let testObject = ExampleObject(manager: mockManager)
```

System Under Test

```
class TestObject {
    let manager: ManagerProtocol

    init(manager: ManagerProtocol = Manager()) {
        self.manager = manager
    }

    func methodUnderTest() -> String {
        var counter = 0
        while counter < 100 {
            counter += 1
        }

        let value = manager.methodToCall(value: counter)
        return "\(value)!"
    }
}
```

Test

```
func test_methodCalled() {
    // given
    let mockManager = MockManager()
    mockManager.returnValue = "Hooray"
    let testObject = TestObject(manager: mockManager)

    // when
    let result = testObject.methodUnderTest()

    // then
    XCTAssertEqual(mockManager.methodCalled, 1)
    XCTAssertEqual(result, "Hooray!")
    XCTAssertEqual(mockManager.value, 100)
}
```

Expectations

How do we know we didn't call a function we weren't expecting?

Set expectations, then consume them when called

MockBase class that all mocks can extend

```
public class MockBase<MockProtocol> {
        public lazy var expect: MockProtocol = type(of:
self).init(receiver: self.receiver.expect()) as! MockProtocol

    internal var receiver: ExpectationReceiver

    public required init(receiver: ExpectationReceiver =
ExpectationConsumer()) {
        self.receiver = receiver
    }

    public func verify(file: StaticString = #file, line: UInt = #line)
{
        receiver.verify(file: file, line: line)
    }

    internal func accept(_ call: String, file: StaticString = #file,
line: UInt = #line) {
        receiver.accept(call: call, file: file, line: line)
    }
}
```

```
private class ExpectationSetter: ExpectationReceiver {
    private let consumer: ExpectationConsumer

    init(consumer: ExpectationConsumer) {
        self.consumer = consumer
    }

    override func accept(call: String, file: StaticString, line: UInt)
{
        guard !consumer.expectations.contains(call) else {
            XCTFail("duplicate expectation: \
(String(describing:call))", file: file, line: line)
            return
        }
        consumer.expectations.append(call)
    }
}
```

```
public class ExpectationConsumer: ExpectationReceiver {
    public override init() {}

    var expectations = [String]()

    override func expect() -> ExpectationReceiver {
        return ExpectationSetter(consumer: self)
    }

    override func accept(call: String, file: StaticString, line: UInt)
{
        guard let index = expectations.index(of: call) else {
            XCTFail("unexpected call: \(String(describing:call))",
file: file, line: line)
            return
        }
        expectations.remove(at: index)
    }
```

```
    override func verify(file: StaticString, line: UInt) {
        if expectations.count > 0 {
            for expectation in expectations {
                XCTFail("unsatisfied expectation: \
(String(describing:expectation))", file: file, line: line)
            }

            expectations = []
        }
}
```

Manual Mock with Expectations

```
class MockManager: MockBase<ManagerProtocol>, ManagerProtocol {

    var returnValue: String?

    func methodToCall(value: Int) -> String {
        accept("methodToCall(value: \(value)")
        return returnValue!
    }
}
```

Test with Expectations

```
func test_methodCalled() {
    // given
    let mockManager = MockManager()
    mockManager.returnValue = "Hooray"
    let testObject = TestObject(manager: mockManager)

    // expect
    mockManager.expect.methodToCall(value: 100)

    // when
    let result = testObject.methodUnderTest()

    // then
    XCTAssertEqual(result, "Hooray!")
    mockManager.verify()
}
```

Obj-C Interop

How to test Swift code that calls out to Obj-C?

How to test Obj-C code that calls out to Swift?

Obj-C Interop

Test swift classes in Swift, Obj-C in Obj-C

Swift classes can be mocked in Obj-C by regular mocking libraries

Obj-C classes can be extended in Swift to conform to Swift protocols

Mocking Obj-C

```objc
@interface Manager : NSObject
- (NSString)methodToCallWithValue:(Int)value;
@end
```

```swift
protocol ManagerProtocol {
    func methodToCall(withValue: Int) -> String
}

extension Manager: ManagerProtocol {}
```

UI Testing

We unit test all non-UI code

End-to-end UI tests

Calabash

Runs the same tests on Android & iOS

Calabash is no longer supported

How do we ensure our UI testing will work with iOS 12+

Native UI Testing

XCUI allows us to write native UI tests in Swift

Tests in the same project, written in parallel

Its provided by Apple, so should have long term support

Native UI Testing

NSURLProtocol intercepts requests & return

Mocks specified by launch arguments

Robot pattern replaces Cucumber, making tests easy to read and understand for non-iOS Devs

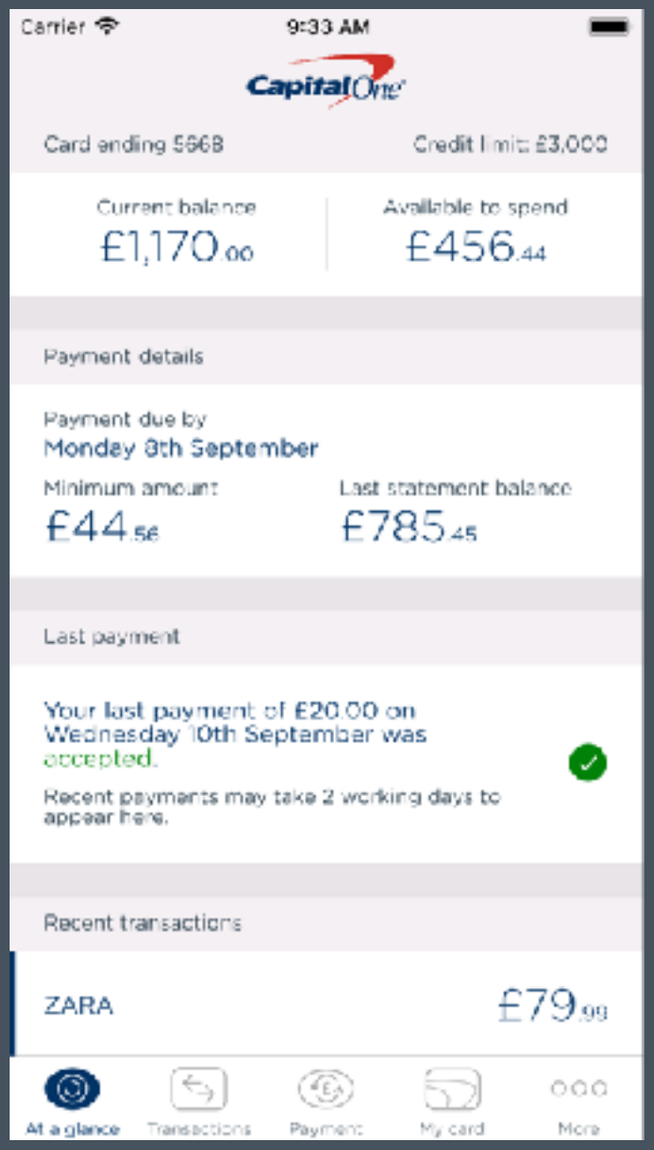
Native UI Testing

```
func test_login() {
    app.launchWithDefaultMocks()

    LoginRobot()
        .login()
        .checkAtAGlanceScreenDisplayed()
}
```

```swift
@discardableResult
func login(code: String = "121212") -> AtAGlanceRobot {
    app.typeTextWithKeyboard(text: code)
    return AtAGlanceRobot()
}
```

```swift
@discardableResult
func checkAtAGlanceScreenDisplayed() -> AtAGlanceRobot {
    assertExists(cardEnding, timeout: 10)
    return self
}
```



Native UI Testing

Tests are more reliable and faster

Engineers are happier

Journey Progress

UI tests have been fully replaced with native tests

UI test time has been cut significantly & results are more reliable

All new app code is written, unit tested & UI tested in Swift

Where are we heading?

Writing manual mocks can be a chore. Can Sourcery help?

iOS 12 has introduced 'flakyness' to UI tests

Mono = One

Rail = Rail

Rob Whitaker
@RobRWAPP
rwapp.co.uk
rob.whitaker@capitalone.com