Testing iOS Apps in 2015

"This talk covers topics about testing iOS apps. We'll start by discussing basic iOS app testing topics such as what to test and discussing the available tools. Then we'll' get into writing testable code, techniques for testing UI logic, and how to test asynchronous code. Then we're going to discuss how to you can add regression tests to a legacy codebase with Apple's testing framework, XCTest. Finally we'll end with a Q&A session."

Testing iOS Apps in 2015

@jerrymarino not a testing guy

Agenda

- Intro to testing
- What to test
- Available tools
- Techniques for testing iOS applications (Views, Controllers, Services, Models)
- Testing a legacy app
- Best practices

Intro to testing

- What is testing?
- Common objections
- Jusitifications

Testing

- Testing is a process of collecting empirical proofs by manipulating inputs and verifying outputs

Why Testing?

- For starters: ensure validation of software
- Improving code quality and asserting correctness

Tests?

When I test its in production

http://quotespics.com/wp-content/quote-images/i-dont-always-test-my-code.jpg

Tests?

- We have QA department
- Our app works great
- My code works AWESOME (rockstar)
- I don't have time for tests and neither do you!
- We run through it before every release

Tests

- Prevent regressions
- Prove shit works
- Are automateable
- *Robust* code design

Tests

The HotSpots app has several errors (not limited to):

- Improper use of KVO
- Improper use of NSNotificationCenter
- Unsafe use of external inputs
- Unsafe use of NSDictionary

Tests

- But it still works:

  Coincidentally safe memory management semantics

  API responses tested had expected values

- This class of errors can hit production

Tests

- iOS deployment is not `git push`
- Stakes of App Store development are high

No silver bullet

Test aren't a substitute for industry best practices

- Beta/internal release
- Code review
- Q&A

But, tests improve code quality on a level these can fail

No silver bullet

- Not everyone wants to test
- some people are opposed
- Testing works best when everyone is on the same page (Culture fit?)

TDD

- Dogma driven development
- Find the right balance for your project

How to write a basic test?

(plays video of app)

How to write a basic test?

UITableViewController tests

```
- (void)testItDisplaysModels
{
    [[NSNotificationCenter defaultCenter] postNotification:DidUpdateModel
                    object:_service];

    UITableViewCell *cell = [_controller.tableView
        cellForRowAtIndexPath:firstIndexPath];
    XCTAssertEqual(cell.titleLabel.text, @"@NSMeetup");
}
```

How to write a basic test?

```
@interface HotSpotCell : UITableViewCell

@property (nonatomic, readonly) UILabel *nameLabel;
- (void)displayLocation:(HSLocation *)location;

@end

@interface HSLocation : NSObject

@property (nonatomic, readonly) NSString *name;
- (instancetype)initWithAttributes:(NSDictionary *)attributes;

@end
```

```
@interface HotSpotCellTest : XCTestCase @end

- (void)testDisplaysLocationName {
//Subject
    HotSpotCell *cell = [[HotSpotCell alloc]
    initWithFrame:CGRectMake(0, 0, 46, 320)];

//Input (displayLocation message send)
    HSLocation *location = [[HSLocation alloc]
        initWithAttributes:@{@"name" : @"NSMeetup"}];
    [cell displayLocation:_location];

//Output
    XCTAssertEqualObjects(@"NSMeetup", cell.nameLabel.text);
}
```

The end

testing runs deeper

How to write tests (for real)?

I want to test my big data location based social app that has animations.

Next stop: the testing rabbit hole

How to write tests (Theoretical)?

- Inputs

  Identitify the expected inputs

- Outputs

  Verify the output for each input

Inputs and Outputs

Tests assert expected output for a given input

Inputs and Outputs

- Subject

  int fibannacci(int n);

- Input (5)
- Output (result)
- Assertions

  int result = fibannacci(5);

Inputs and outputs

- Public interface

This is how the code is used

How to write a test?

```
 @interface HSLocationsServices : NSObject

@property (nonatomic) NSArray *locations;
- (BOOL)fetch;

@end

@interface HotSpotsViewController : UITableViewController

@property (nonatomic, readonly) HSLocationsService *locationService;

- (id)initWithStyle:(UITableViewStyle)style locationService:(HSLocationsService *)locati

@end
```

How to write a test?

- stubbing: temporarily implment fake message
- spys (expection/rejection): invocation watching
- mock objects: proxy objects/gateway to mocking & stubbing

Mock objects

```
- (void)testDisplaysALoadingSpinnerAfterFetching {
    id mockLocationService = [HSLocationsServices newMock];
    [[[mockLocationService stub] fetch] andReturn:@YES];

    HotSpotsViewController *controller = [[HotSpotsViewController alloc] initWithStyle:U
    [controller viewWillAppear:YES];

    assert(controller.loadingView);
}
```

Stubbing

```
     //This method now returns yes!
    id mockLocationService = [HSLocationsServices newMock];
    [[[mockLocationService stub] fetch] andReturn:@YES];
```

Stubbing

```
//This method now synchronously executes the completion block on the main thread
//when its called!

void (^theBlock)(NSInvocation *) = ^(NSInvocation *invocation) {
    /* code that reads and modifies the invocation object */
};
[[[mock stub] andDo:theBlock] someMethod:[OCMArg any]];
```

Expectations

```
 assert(view.badge == kAuthenticatedBadge);

[[mock expect] someMethod:someArgument]
```

Expectations

```
// Check if the account received [Account authenticateWithCompletionBlock:]
[[mock expect] someMethod:ANY_BLOCK];
[[mock expect] someMethod:someArgument]
[mock verify];
```

Available tools

- XCTest
- BDD Frameworks
- Mocking frameworks
- UIAutomation
- Continuous integration

Available tools: BDD Frameworks

- Write tests with pretty syntax
- Include mocking + stubbing

```
it(@"has a bar", ^{ [[foo.bar should] equal:@"bar anyone?"]; });
```

Kiwi is a world-class block based testing system with mocking, stubs, matchers and much more https://github.com/kiwi-bdd/Kiwi

Specta/Expecta is a popular setup https://github.com/specta/expecta

Available tools: OCMock

- Several years wise
- Extensive support for mocking, stubbing, spying
- Caveats: heavier duty syntax

Available tools: UIAutomation

- JavaScript
- Add accessibility identifiers & labels to use UIAutomation
- Real accessibility labels might not work
- Closed source: you **can't** extend it
- Runs in another process
- Subliminal adds an Objective-C interface to this (String factory)

You could probably disassemble it, use private APIs, code inject it?

Available tools: Continuous integration

- In 2015 there are several release targets
- Test production builds on multiple devices
- Travis CI + Github
- Jenkins for ultimate flexiblity
- Xcode bots are built in

Testing iOS apps

- View code
- UI code (controller)
- Model / Service layer code

What to test

- Used code paths
- Risky code
- *known* scenarios
- "Bugs" and programming errors

What to test (service layer)

- Model serialization code
- Network requests

What to test (service layer)

```
@interface HSLocation : NSObject

@property (nonatomic, readonly) NSString *guid;
@property (nonatomic, readonly) NSString *name;
@property (nonatomic, readonly) NSString *address;
@property (nonatomic, readonly) NSString *vicinity;
@property (nonatomic, readonly) NSUInteger hottness;
@property (nonatomic, readonly) CLLocationCoordinate2D coordinate;

- (instancetype)initWithAttributes:(NSDictionary *)attributes;

@end
```

What to test (service layer)

Unsafe serialization code

```
@implementation HSLocation
//Init with attributes from a serialized remote object i.e. NSJSONSerialization
- (instancetype)initWithAttributes:(NSDictionary *)attributes{
    if (self = [super init]) {
       _guid = [attributes objectForKey:@"place_id"];
       _name = [attributes objectForKey:@"name"];
       _vicinity = [attributes objectForKey:@"vicinity"];
       NSDictionary *geometry = [attributes objectForKey:@"geometry"];
       NSDictionary *location = [geometry objectForKey:@"location"];
       _coordinate.latitude = [[location objectForKey:@"lat"] doubleValue];
       _coordinate.longitude = [[location objectForKey:@"lng"] doubleValue];
   }
   return self;
}
@end
```

What to test (service layer)

Unsafe serialization code

```
 // Just don't even write this kind of code
NSDictionary *geometry = [attributes objectForKey:@"geometry"];
NSDictionary *location = [geometry objectForKey:@"location"];
```

What to test (service layer)

```
- (void)testItFetchesLocations {
    [NSURLProxy injectAllResponsesWithJSONFile:@"places_search_bar"
    responseCode:200];

    NSArray *locations = [HSLocation fetchLocations];

    HSLocation *firstLocation = locations.firstObject;
    assert([firstLocation.name isEqual:@"NSMeetup"]);
    //....
}
```

What to test (service layer)

```
[OHHTTPStubs stubRequestsPassingTest:^BOOL(NSURLRequest *request) {
    return [request.URL.host isEqualToString:@"mywebservice.com"];
} withStubResponse:^OHHTTPStubsResponse*(NSURLRequest *request) {
    // Stub it with our "wsresponse.json" stub file
    NSString* fixture = OHPathForFileInBundle(@"wsresponse.json",nil);
    return [OHHTTPStubsResponse responseWithFileAtPath:fixture
              statusCode:200 headers:@{@"Content-Type":@"application/json"}];
}]
```

What to test (service layer)

- When the endpoint changes, you can verify if the app will support this change in production, etc..

UI logic

- It's no different than model logic (in the regard that its logic)

Inputs and Outputs (UI)

- UIApplication handles dispatching events via UIResponder chain.

Inputs and Outputs (UI)

Inputs Approach 1. "Unit test approach"

- Invoke callbacks

i.e. `[(UITableViewDelegate *)delegate tableView:view didSelectRowAtIndexPath:indexPath];`

Inputs and Outputs (UI)

Inputs Approach 2. "Integration approach"

- Mimic UIKit events by

  Manually instantiate UIEvent and UITouch

Inputs and Outputs (UI)

- Regardless of implementation, use an abstract interface to keep it flexible.

i.e. `[tableView tapAtIndexPath:indexPath];`

i.e. `[textView typeText:@"yoda"];`

What to test (UI)

- Presentation logic
- Layout logic
- UI <--> Model display logic

What to test (UI)

- Change propagation (KVO, NSNotification, delegate callbacks)
- Common programming errors

  (retain cycles, KVO, NSNotificationCenter, delegates, deallocated objects)...
- The best bugs are ones that made it through tests
- A production error is a new test case

Testable

- Only test public interfaces
- Testing trumps encapsulation

(Untestable) Code

```
//This interface isn't so bad
NSString *const HSLocationsServiceDidScoreLocationNotification;
NSString *const HSLocationsServiceDidFetchLocationsNotification;

@interface HSLocationsService : NSObject

@property (nonatomic, readonly) NSArray *locations;

- (BOOL)fetchMoreAfter:(id)sender;

@end
```

(Untestable) Code

```
- (id)init {
    if (self = [super init]) {
        _locationsInternal = [NSMutableArray array];
        [self.locationManager startUpdatingLocation];
    }
    return self;
}
```

Testability

```
NSString *const HSLocationsServiceDidScoreLocationNotification;
NSString *const HSLocationsServiceDidFetchLocationsNotification;

@interface HSLocationsService : NSObject

@property (nonatomic, readonly) NSString *query;
@property (nonatomic, readonly) CLLocation *location;
@property (nonatomic, readonly) CLLocationManager *locationManager;
@property (nonatomic, readonly) NSArray *locations;

- (instancetype)initWithQuery:(NSString *)query
           locationManager:(CLLocationManager *)locationManager;
- (BOOL)fetch;
- (BOOL)fetchMoreAfter:(id)sender;
- (void)scoreLocation:(HSLocation *)location;

@end
```

Multithreaded and Async functionality

- threads, `dispatch_async`, NSTimer, etc..

These are real things and can be tested

Async code

- Try refactoring and making things as synchronous as possible first.

Async code

- Main thread is the synchronization point in XCTest *mostly
- Busy waiting/runloop churn
- Wait for code to execute

Adding Integration Tests (Demo)

- Add tests to a 'Legacy app' UITableViewController
- Legacy?

UI test setup

- UIApplication
- Don't use the app delegate (probably)
- XCTest loads test bundle into Apps target
- Keep app delegate encapsulated and tight

UI test

```
 // What does this even do?
@interface HotSpotsViewController : UITableViewController

@end
```

UI test (testability)

```
@class HSLocationsService;

@interface HotSpotsViewController : UITableViewController

@property (nonatomic, readonly) HSLocationsService *locationService;

- (id)initWithStyle:(UITableViewStyle)style locationService:(HSLocationsService *)locati

@end
```

Cell test

```
@interface HotSpotCellTest : XCTestCase @end

@implementation HotSpotCellTest {
    HotSpotCell *_cell;
    HSLocation *_location;
}
```

Cell test

```
- (void)setUp
{
    [super setUp];

    _cell = [[HotSpotCell alloc] initWithFrame:CGRectMake(0, 0, 46.0, 320.0)];
    _location = [[HSLocation alloc]
        initWithAttributes:@{@"guid" : @"42", @"place_id" : @"NSMeetup"}];

    UIWindow *mainWindow = [[UIApplication sharedApplication].delegate window];
    [mainWindow addSubview:_cell];
}
```

Cell test

```
- (void)testChangePropagationSafety
{
    [_cell displayLocation:_location];
    [_cell removeFromSuperview];
    _cell = nil;
    //This test won't even if the observer wasn't removed
    XCTAssertNoThrow([_location setValue:@5 forKey:@"hottness"]);
}
```

Cell test

```
- (void)testDisplaysLocationHottnessScoreUponUpdate
{
    [_cell displayLocation:_location];
    [_location setValue:@5 forKey:@"hottness"];
    XCTAssertEqualWithAccuracy(1.0, _cell.accessoryLabel.alpha, 0.05);
    XCTAssertEqualObjects(@"5pts", _cell.accessoryLabel.text);
}
```

Best practices

- Keep your app working by testing (and implementing) real code paths
- Keep tests deterministic
- Test only public interfaces
- Build abstractions i.e. macros and test classes
- Keep logic out of tests
- Add good regression tests for each bug or error
- Stub network requests *deterministic*
- You'll implement a testing framework

Resources

- General testing: Google test blog http://googletesting.blogspot.com
- Apple WWDC 2014 "Testing in Xcode 6" https://developer.apple.com/videos/wwdc/2014/
- General testing: Martin Fowler posts i.e. http://martinfowler.com/articles/mocksArentStubs.html
- iOS Kiwi testing: https://github.com/kiwi-bdd/Kiwi/wiki
- HotSpots app & talk repo (tests patch in progress) PR welcome: https://github.com/jerrymarino/testing-ios-apps

Thank you

- @NSMeetup & Steve Derico community for hosting these events
- Julian Ramirez & Jim Hildensperger for rigorous feedback on this talk and test app

Questions and Answers

- open an issue on the talk repo
- ping me via @jerrymarino