Testing Mobile Apps
Testing!

• The most important thing we never teach you to do!

• Consider how most projects go in your undergrad career:
  – Requirements elicitation? We give you a page with the info.
  – Team management? You’ve been in the same classes for years.
  – Documentation? What documentation?
  – Testing often falls to the wayside
Where’s the testing?

- Depending on which version of each course you’ve seen, you may have seen some testing material in:
  - CS 2110: JUnit
  - CS 3240: More unit testing + descriptions of other forms
  - Upper level electives: Perhaps some domain specific testing
Types of Testing

• Black box test – don’t care what happens in the function; just check the result
• White box test – looks at what lines are executed
• Unit Testing – testing individual functions / modules; black box or white box
• System Testing – usually high-level; black box
• Regression Testing – testing previous functionality; usually black box
So, what are we focusing on?

• We can’t really do regression testing

• System testing is (theoretically) what you do when you open your app on the device and verify everything works before submitting it to me

• Thus, we will focus on unit testing!
Unit Testing in Android

• Unit testing in Android is based around the same stuff as any unit testing in Java

• JUnit
  – Built around “assert” statements
  – JUnit 4 uses annotations to setup tests (JUnit 3 used method naming conventions)
  – setUp() and tearDown() open and close tests
Unit Testing in Android

• Great! That sounds simple!
• And it is... for testing all of your basic logic
• But, of course, we know that’s not all we have to worry about

• How do we test:
  – UI interactions?
  – Sensor data?
  – Starting/stopping activities?
Android Testing Components
Test Tools

• android.test.*

• Contains:
  – TouchUtils: simulate screen touches
  – ViewAsserts: verify visual components are on screen
  – MoreAsserts: Android-specific tests
  – PerformanceTests: for testing speed and memory
Monkey and MonkeyRunner

- Monkey: a package for simulating pseudo-random key strokes, gestures, etc. on a device
- MonkeyRunner: allows you to write tests in Python
- (They’re not related...)
Instrumentation

• Instrumentation provides functionality “hooks” to manipulate an app’s lifecycle

• You can directly call onStop() or onResume() for instance

• You can fully destroy an app and bring it back

• You can also do some basic UI interface (but better to just use TouchUtils)
Mock Objects

• A mock object is a fake/testing version of a system object (or service) that provides testing data instead of real data

• For example:
  – Switching out a content provider for one with known testing data
  – Switching out the LocationManager with one that follows a set path
  – Simulating Intents
Testing Multiple Devices

• It is exceptionally difficult for an Android developer to test against even a reasonable subset of available platforms
• Or is it? Can the cloud help us with this?
• https://aws.amazon.com/device-farm/
On to iOS!

- The idea is basically the same
- Testing changed dramatically with Xcode 6
- Before 6, automation tests were written in Javascript
- With Swift, a new unit testing framework was added
  - Regular test
  - Performance tests
  - UI tests
XCTestCase

• All your basic asserts are available
  – XCTAssertTrue, XCTAssertEqual, XCTAssertNil...

• Performance test allow you to set aside some code in a special codeblock for execution

• UI tests are actually pretty interesting...
What do you test?

• Key functionality

• Key use cases
  – Think of the operational profile of the devices and users of your app

• Basic phone interactions
  – What happens when a call comes in? A text message?
  – What happens if you lose network connectivity?
Consider Your Apps

• Write down 5 key things that your app needs to test
• Leave out the “common phone” stuff
  – Incoming phone call, device rotate, etc.
• What are the major functionality tests?
• Any UI tests?
• Any mock objects?
• Any instrumentation?