Testing Android Application

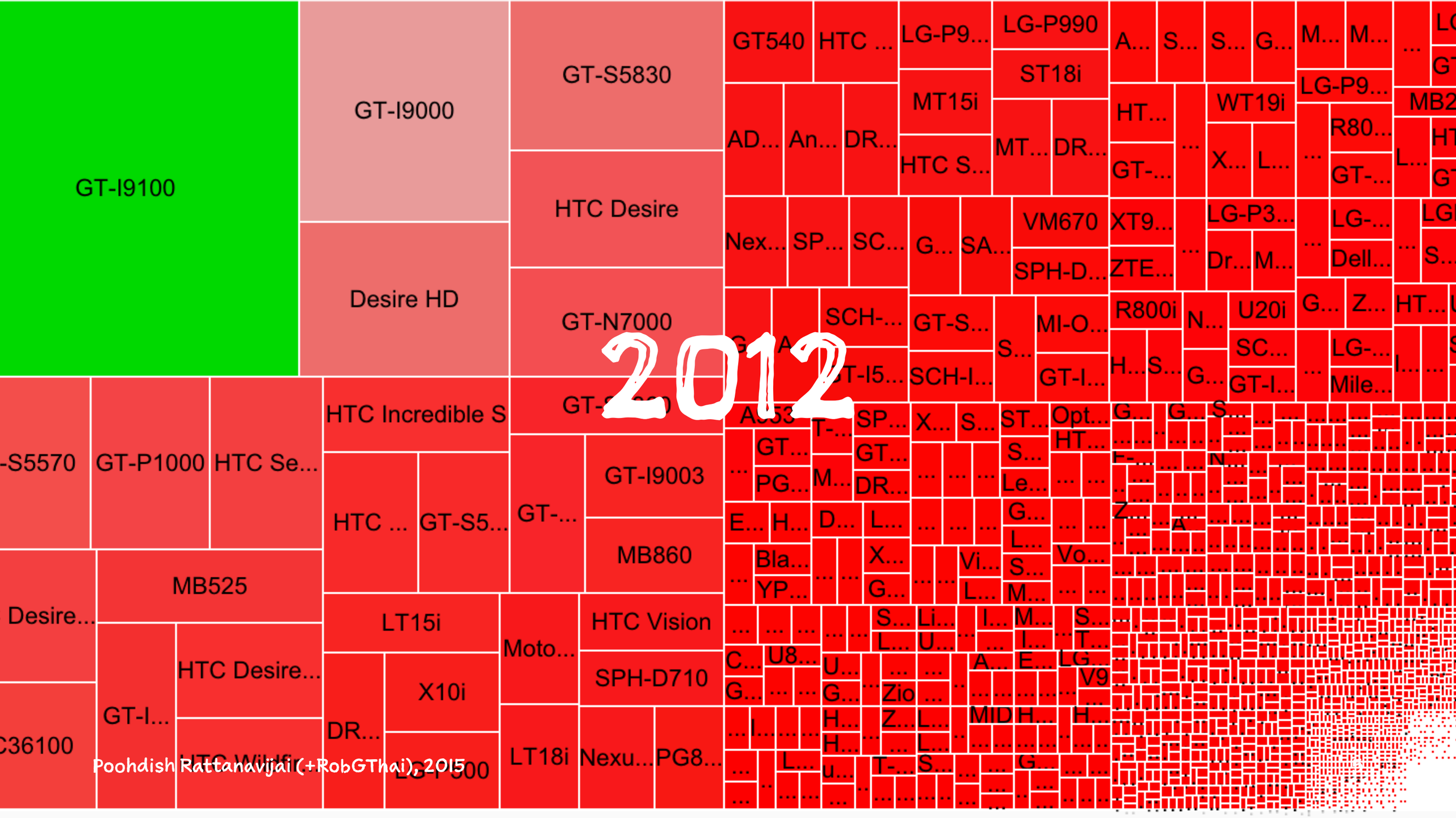
Machine Setup

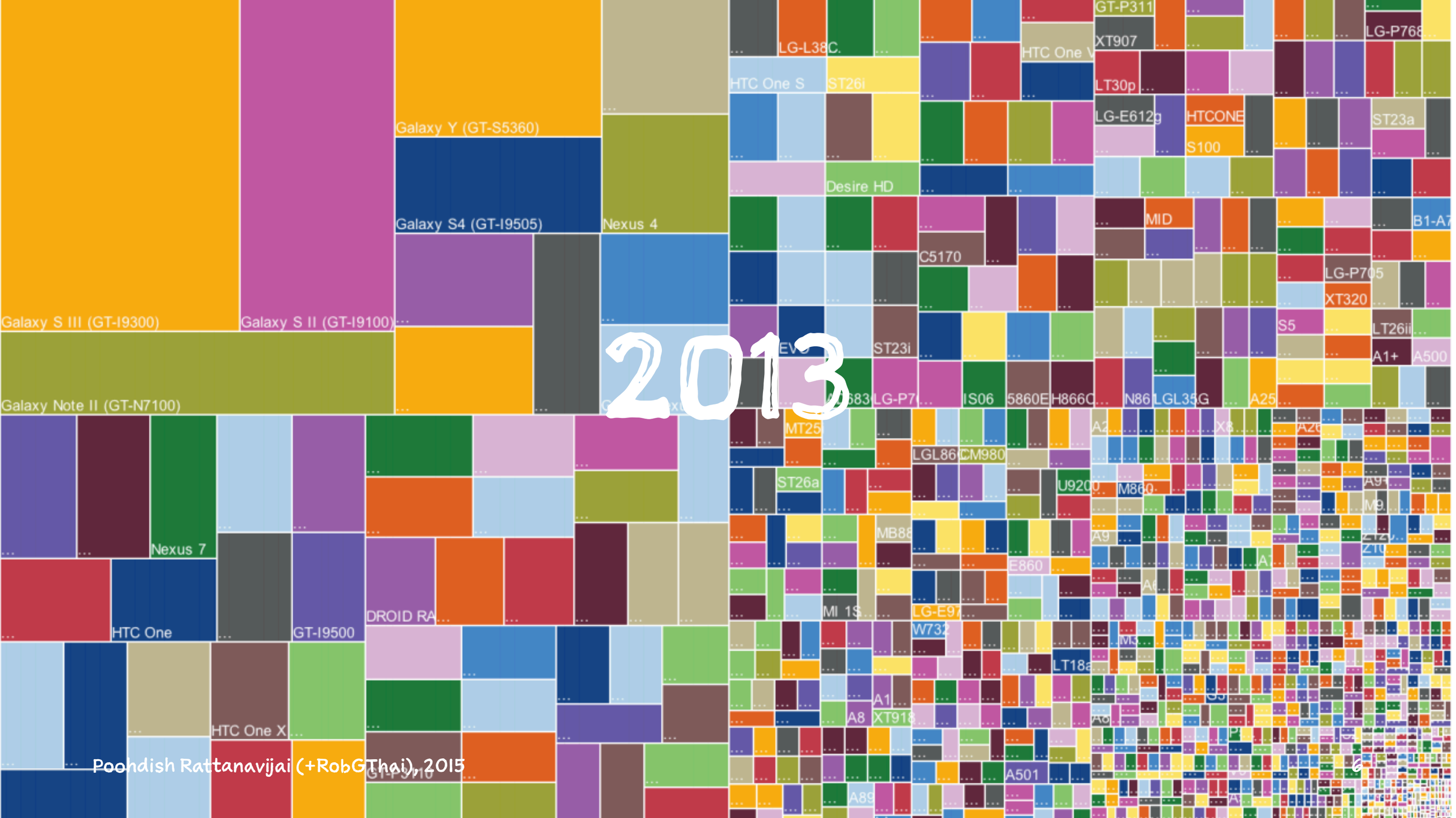
- JDK (1.7+)
- Android SDK
- Android Studio
- Gradle
- Genymotion (Optional)

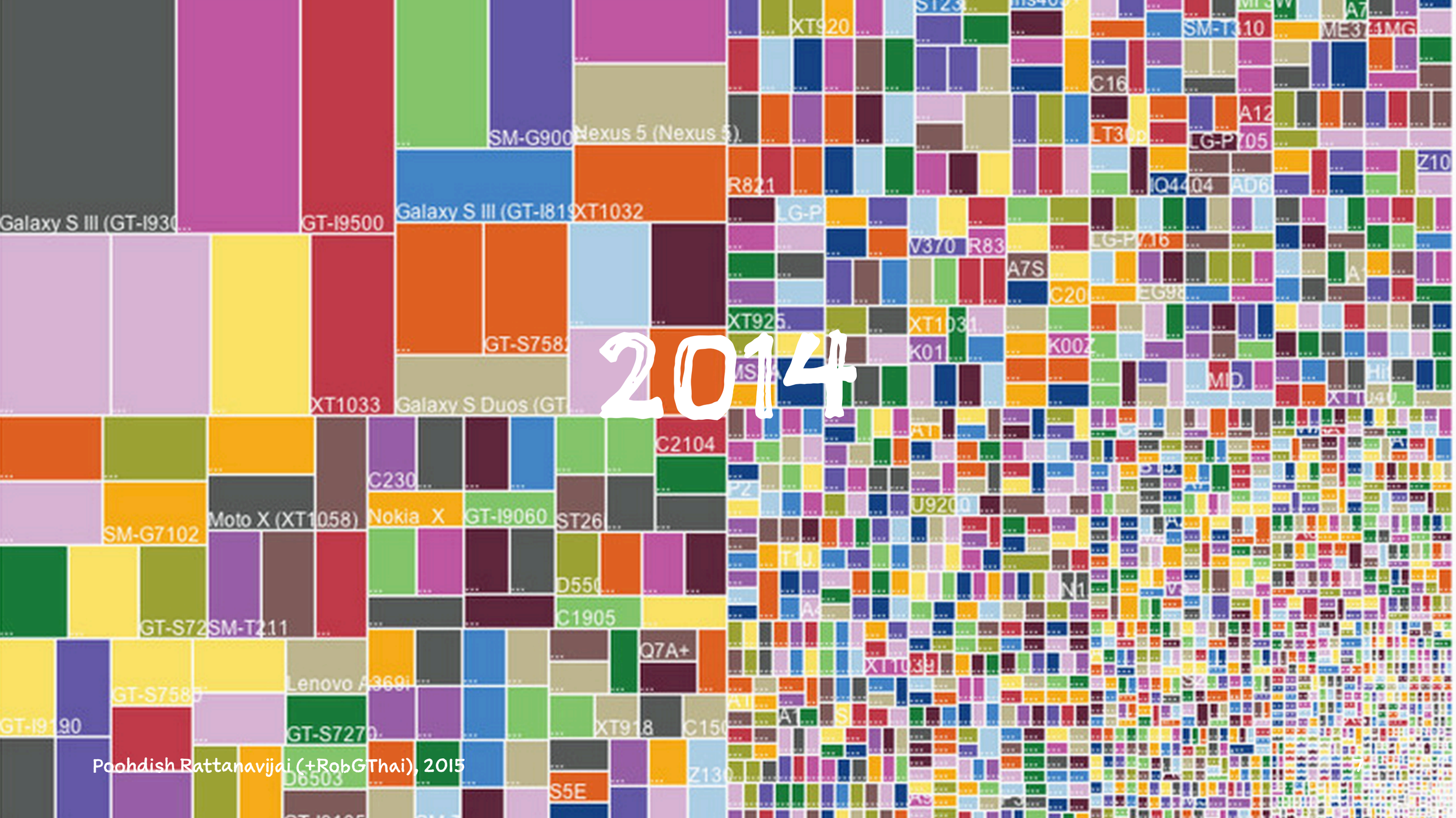
Objectives

- Understand the need for Instrumenting Acceptance Test
- Being able to write basic instrument test for Android application
- Generating consumable and sane report
- Learning about core component of Android Testing

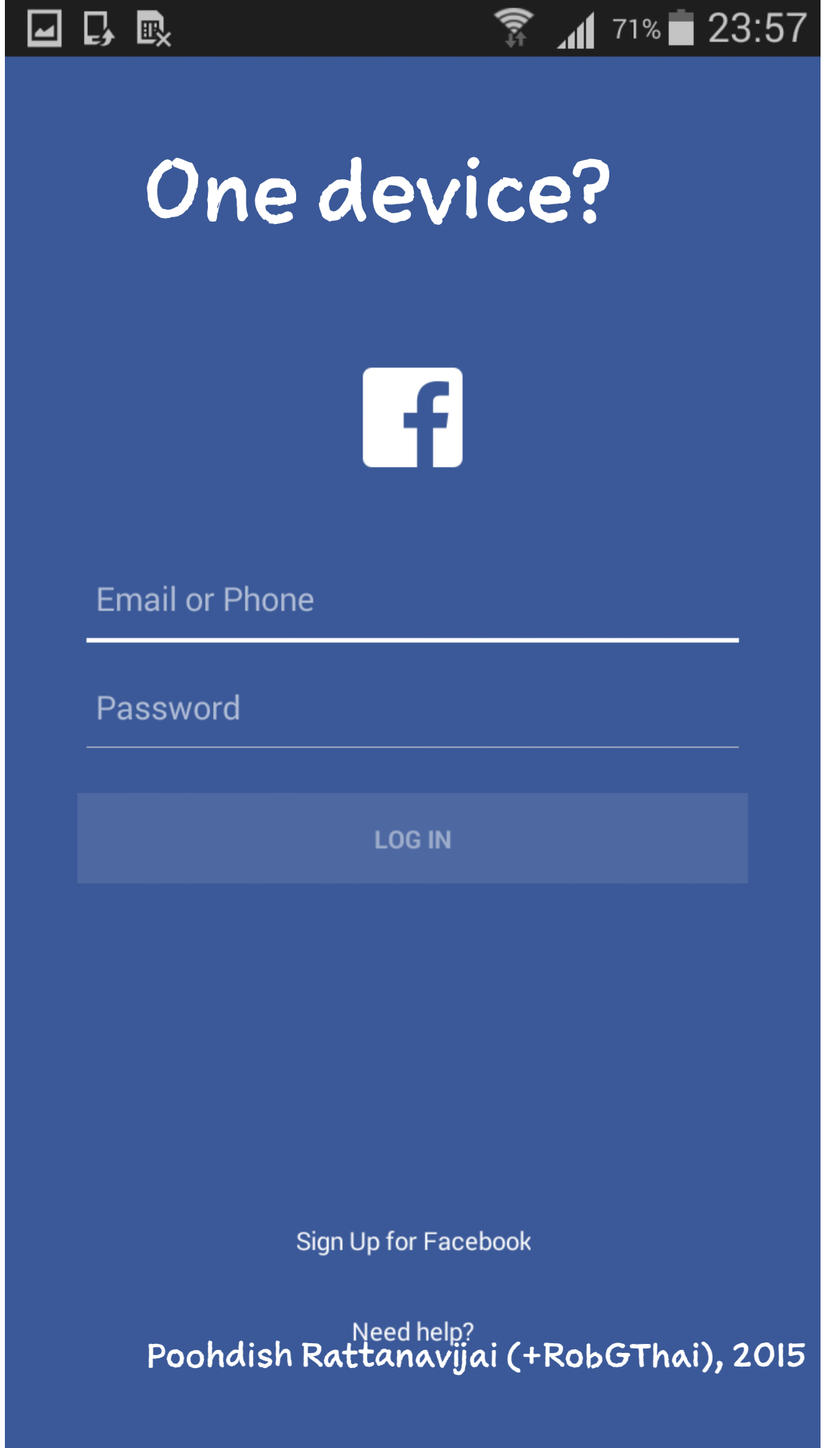
Why Automate Test?

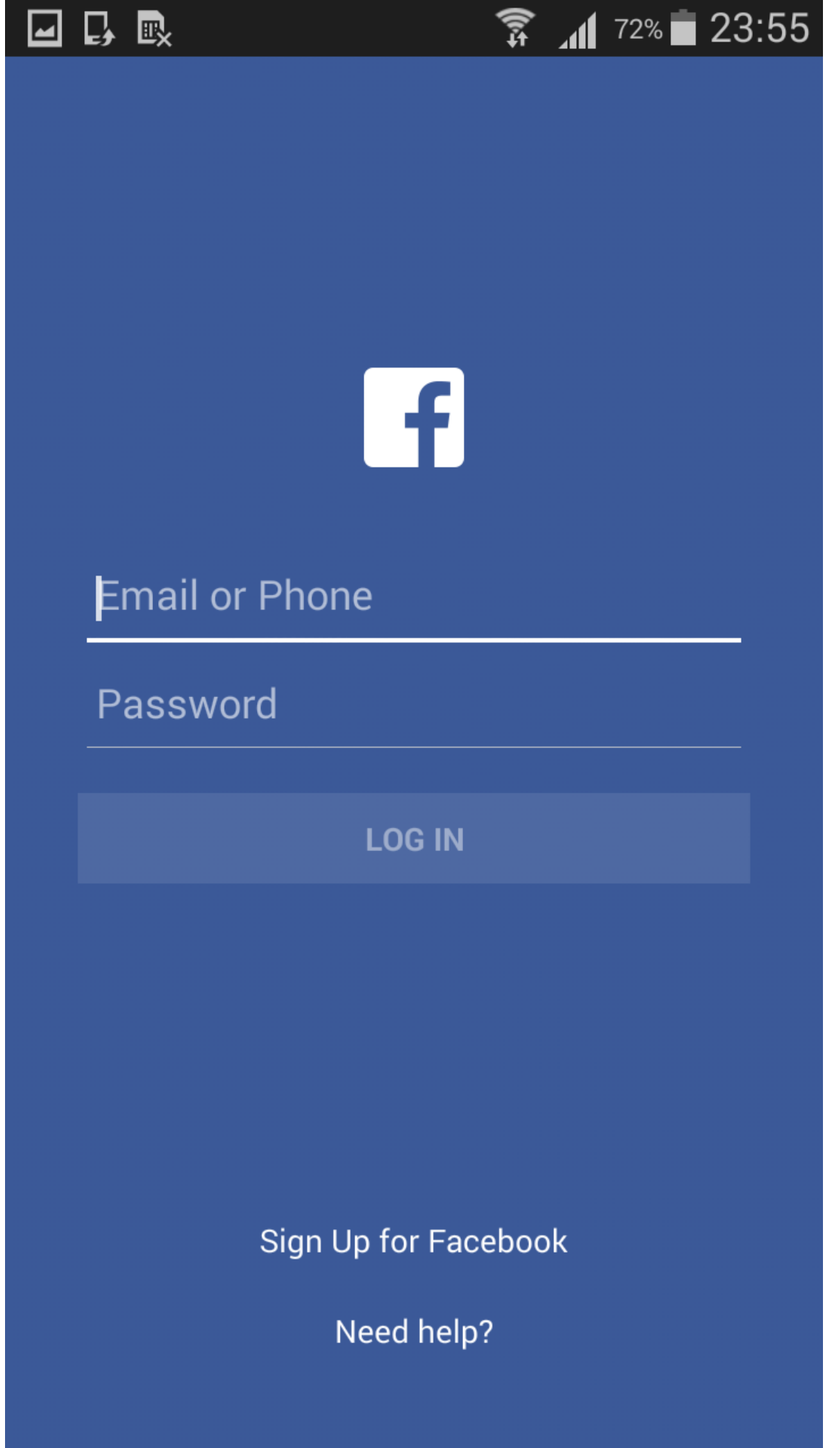

GT-I9100
GT-I9000
Desire HD
GT-S5830
HTC Desire
GT-N7000
2012
GT540
HTC ...
LG-P9...
LG-P990
ST18i
MT15i
HTC S...
AD...
An...
DR...
MT...
DR...
A...
S...
S...
G...
M...
M...
LG-P9...
HT...
GT-...
WT19i
X...
L...
R80...
GT-...
MB2
Nex...
SP...
SC...
G...
SA...
VM670
SPH-D...
XT9...
ZTE...
LG-P3...
Dr...
M...
LG-...
Dell...
SCH-...
GT-S...
SCH-I...
S...
MI-O...
GT-I...
R800i
H...
S...
N...
G...
U20i
SC...
GT-I...
G...
Z...
HT...
LG-...
Mile...
SP...
X...
S...
ST...
Opt...
HT...
S...
Le...
GT...
PG...
M...
DR...
E...
H...
D...
L...
G...
L...
Vi...
S...
L...
M...
Vo...
Bla...
YP...
X...
G...
S...
Li...
I...
M...
S...
L...
U...
I...
T...
C...
U8...
U...
A...
E...
LG...
V9
G...
G...
Zio
H...
H...
Z...
L...
MID
H...
H...
L...
L...
u...
T-...
S...
G...
HTC Incredible S
HTC ...
GT-S5...
GT-I9003
MB860
GT-...
-S5570
GT-P1000
HTC Se...
Desire...
MB525
LT15i
HTC Desire...
GT-I...
C36100
DR...
X10i
LG-P500
Moto...
LT18i
HTC Vision
SPH-D710
Nexu...
PG8...
Poohdish Rattanavijai (+RobGThai), 2015

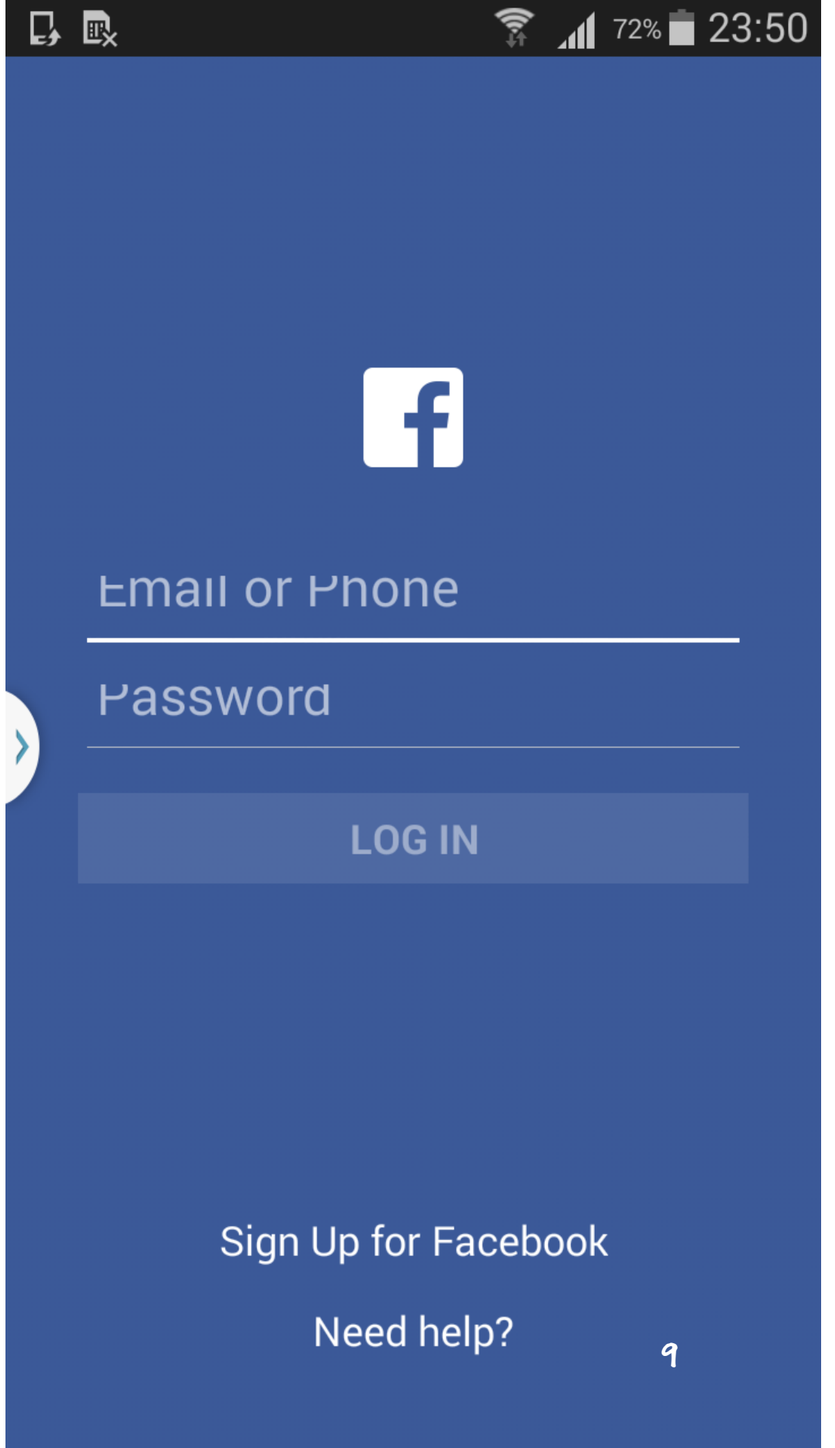

2013
Galaxy S III (GT-I9300)
Galaxy S II (GT-I9100)
Galaxy Note II (GT-N7100)
Galaxy Y (GT-S5360)
Galaxy S4 (GT-I9505)
Nexus 4
Nexus 7
HTC One
GT-I9500
HTC One X
DROID RA
HTC One S
ST26i
Desire HD
C5170
ST23i
MT25
ST26a
MB88
MI 1S
LGL86
CM980
E860
W732
LT18a
A1
A8
XT918
A501
GT-P311
XT907
LT30p
LG-E612g
HTCONE
S100
MID
LG-P705
XT320
S5
ST23a
LT26ii
A1+
A500
IS06
5860E
N86
LGL35G
A25
U920
M860
A9
Poohdish Rattanavijai (+RobGThai), 2015


2014
Poohdish Rattanavijai (+RobGThai), 2015

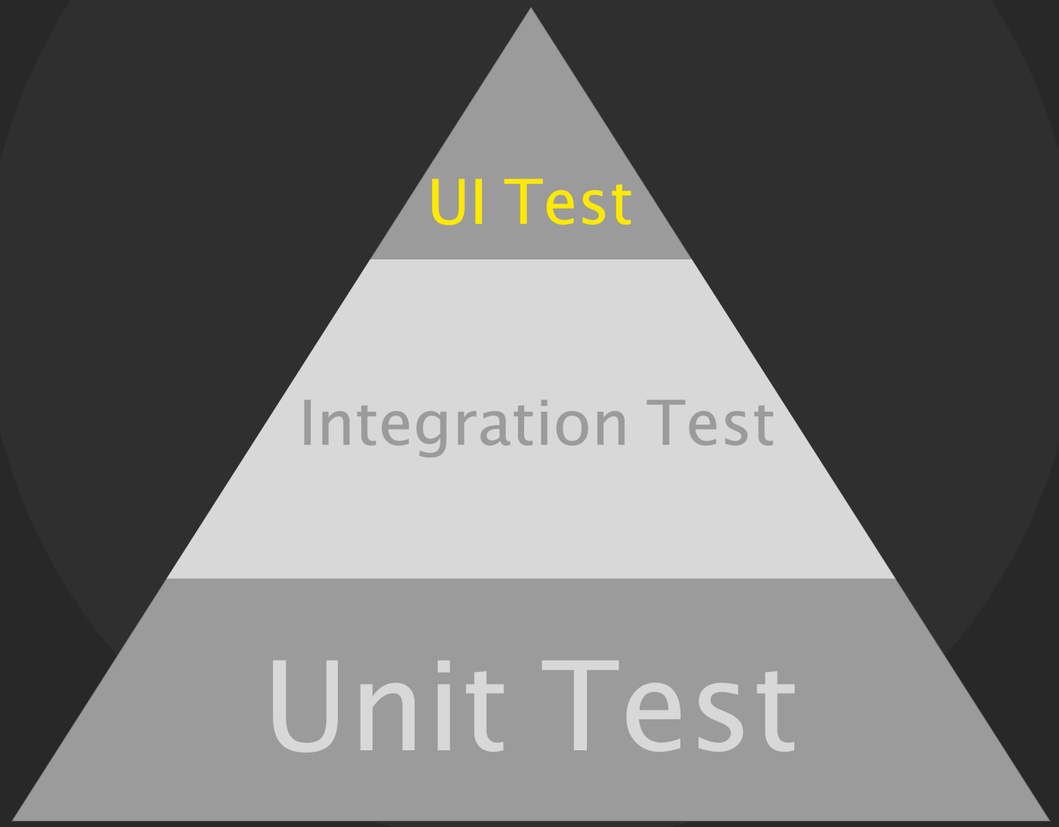
What is your scope?


71% 23:57
One device?
Email or Phone
Password
LOG IN
Sign Up for Facebook
Need help?


72% 23:55
Email or Phone
Password
LOG IN
Sign Up for Facebook
Need help?


72% 23:50
Email or Phone
Password
LOG IN
Sign Up for Facebook
Need help?

What are there to test?



A brief history of testing on Android

Testing on Android (Dark Ages)

- Robolectric
- Robotium
- JUnit
- Android Mock

Robolectric

- Setup
- Installation
- Quick Start
- Eclipse Quick Start
- Downloads
- User Guide

Robolectric

Test-Drive Your Android Code

Running tests on an Android emulator or device is slow! Building, deploying, and launching the app often takes a minute or more. That's no way to do TDD. There must be a better way.

Wouldn't it be nice to run your Android tests directly from inside your IDE? Perhaps you've tried, and been thwarted by the dreaded `java.lang.RuntimeException: Stub!`?

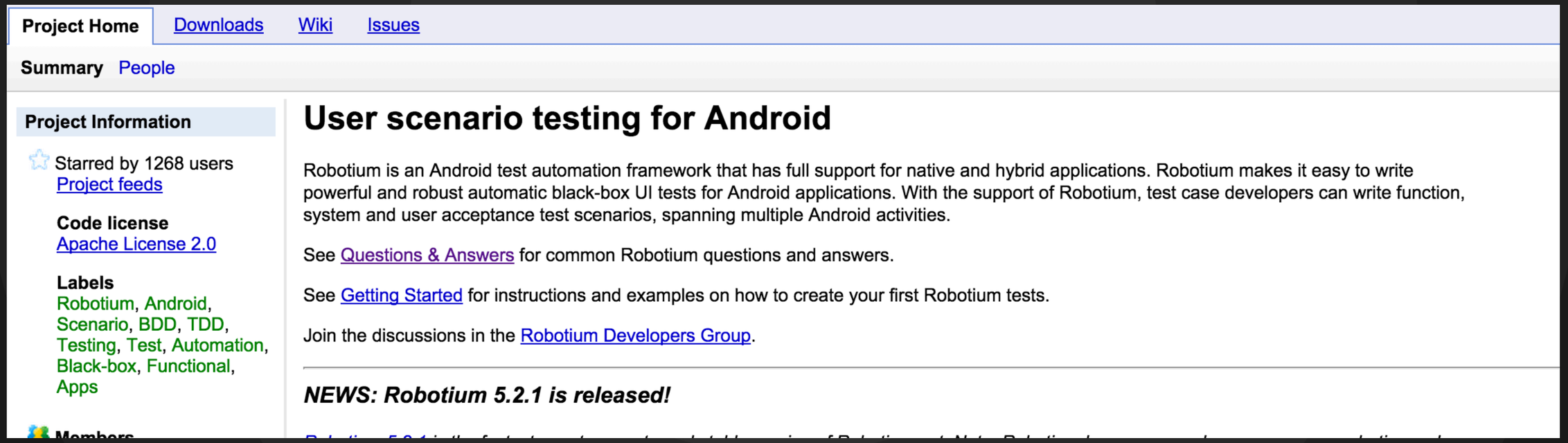
Robolectric is a unit test framework that de-fangs the Android SDK jar so you can test-drive the development of your Android app. Tests run inside the JVM on your workstation in seconds. With Robolectric you can write tests like this:

```
// Test class for MyActivity
@RunWith(RobolectricTestRunner.class)
public class MyActivityTest {

  @Test
  public void clickingButton_shouldChangeResultsViewText() throws Exception {
    Activity activity = Robolectric.buildActivity(MyActivity.class).create().get();
```

Robotium

<<Recheck this>>



JUnit

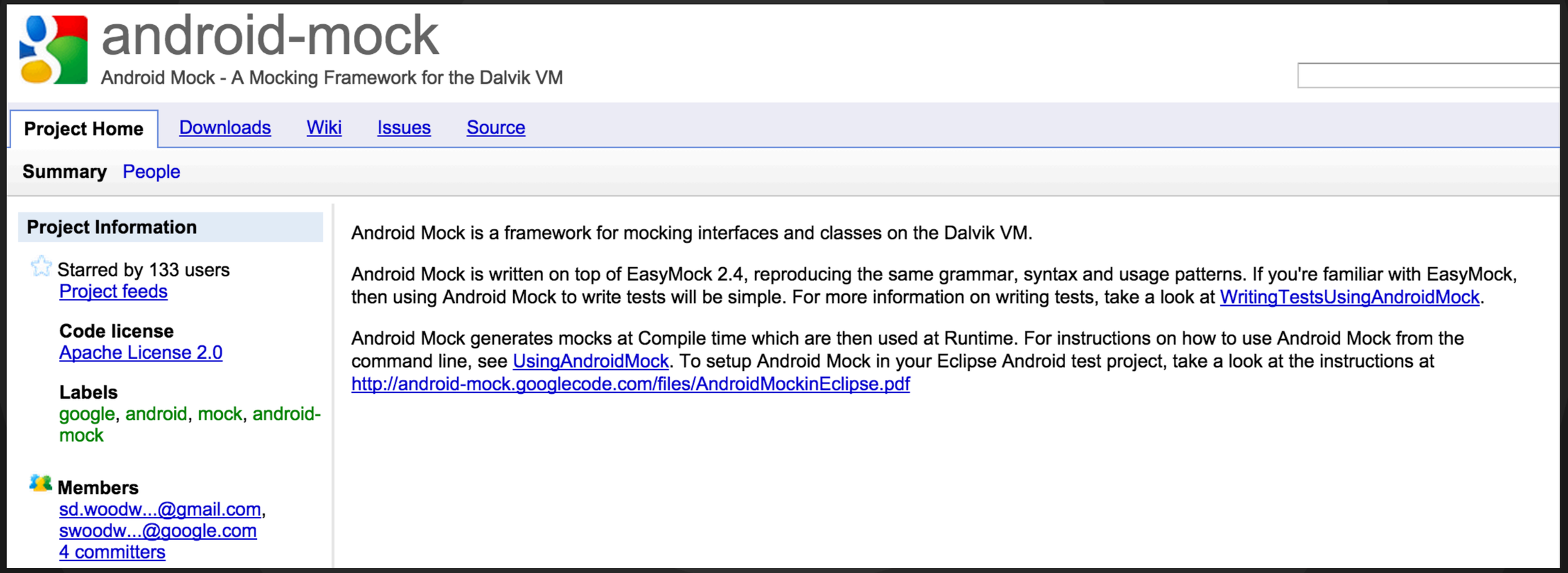
JUnit / About Version: 4.12 | Last Published: 2014-12-04

JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.

```
@Test
public void lookupEmailAddresses() {
    assertThat(new CartoonCharacterEmailLookupService().getResults("looney"), allOf(
        not(empty()),
        containsInAnyOrder(
            allOf(instanceOf(Map.class), hasEntry("id", "56"), hasEntry("email", "roadrunner@fast.org")),
            allOf(instanceOf(Map.class), hasEntry("id", "76"), hasEntry("email", "wiley@acme.com"))
        )
    ));
}
```

Hamcrest matchers

Android Mock



Testing on Android (Middle Ages)

- Robolectric
- Calabash
- UiAutomator
- Espresso 1.1

Calabash

Calabash enables you to write and execute automated acceptance tests of mobile apps. Calabash is cross-platform, supporting Android and iOS native apps. It is open source and free, and has a company, Xamarin, backing and developing it.

With Xamarin Test Cloud you can automatically test your app on hundreds of mobile devices.

Calabash consists of libraries that enable test-code to programmatically interact with native and hybrid apps. The interaction consists of a number of end-user actions. Each action can be one of

Gestures

Touches or gestures (e.g., tap, swipe and rotate).

Cucumber

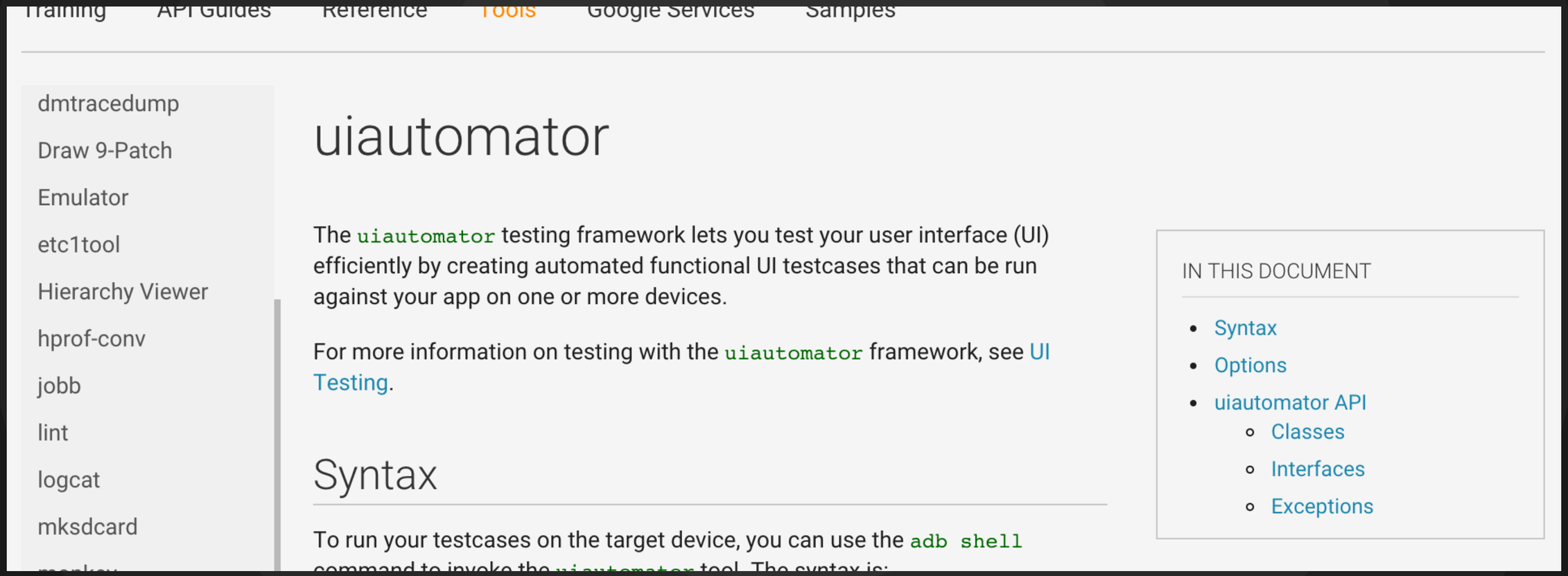
Calabash supports Cucumber. Cucumber lets us express the behavior of our app using natural language that can be understood by business experts and non-technical QA staff. Here is an example.

```
Feature: Rating a stand
  Scenario: Find and rate a stand from the list
    Given I am on the foodstand list
    Then I should see a "rating" button
    And I should not see "Dixie Burger & Gumbo Soup"

    When I touch the "rating" button
    Then I should see "Dixie Burger & Gumbo Soup"

    When I touch "Dixie Burger & Gumbo Soup"
```

UiAutomator

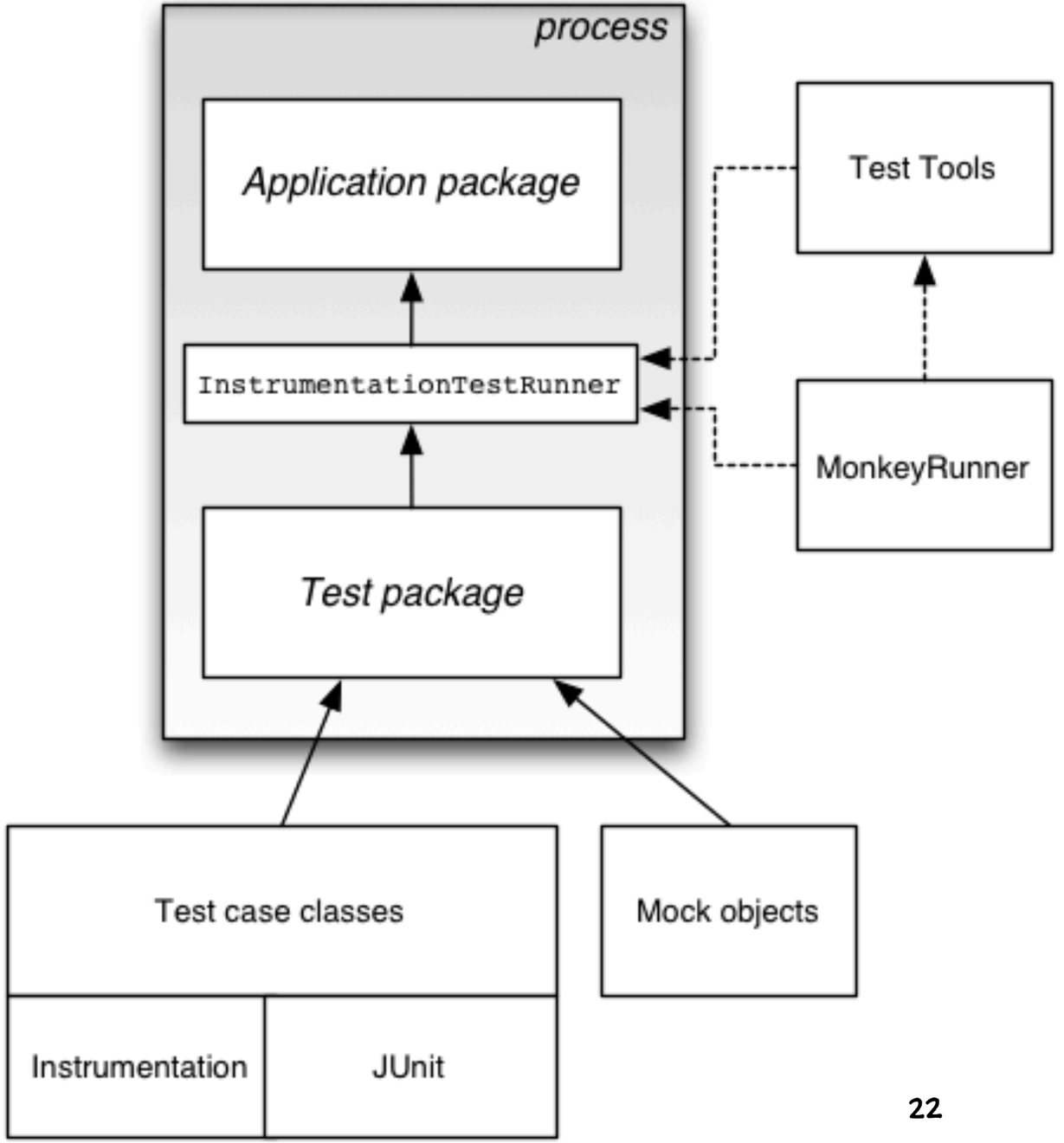


Testing on Android (Golden Ages)

1. Testing Framework
2. Spoon
3. Espresso 2.0
4. Burst

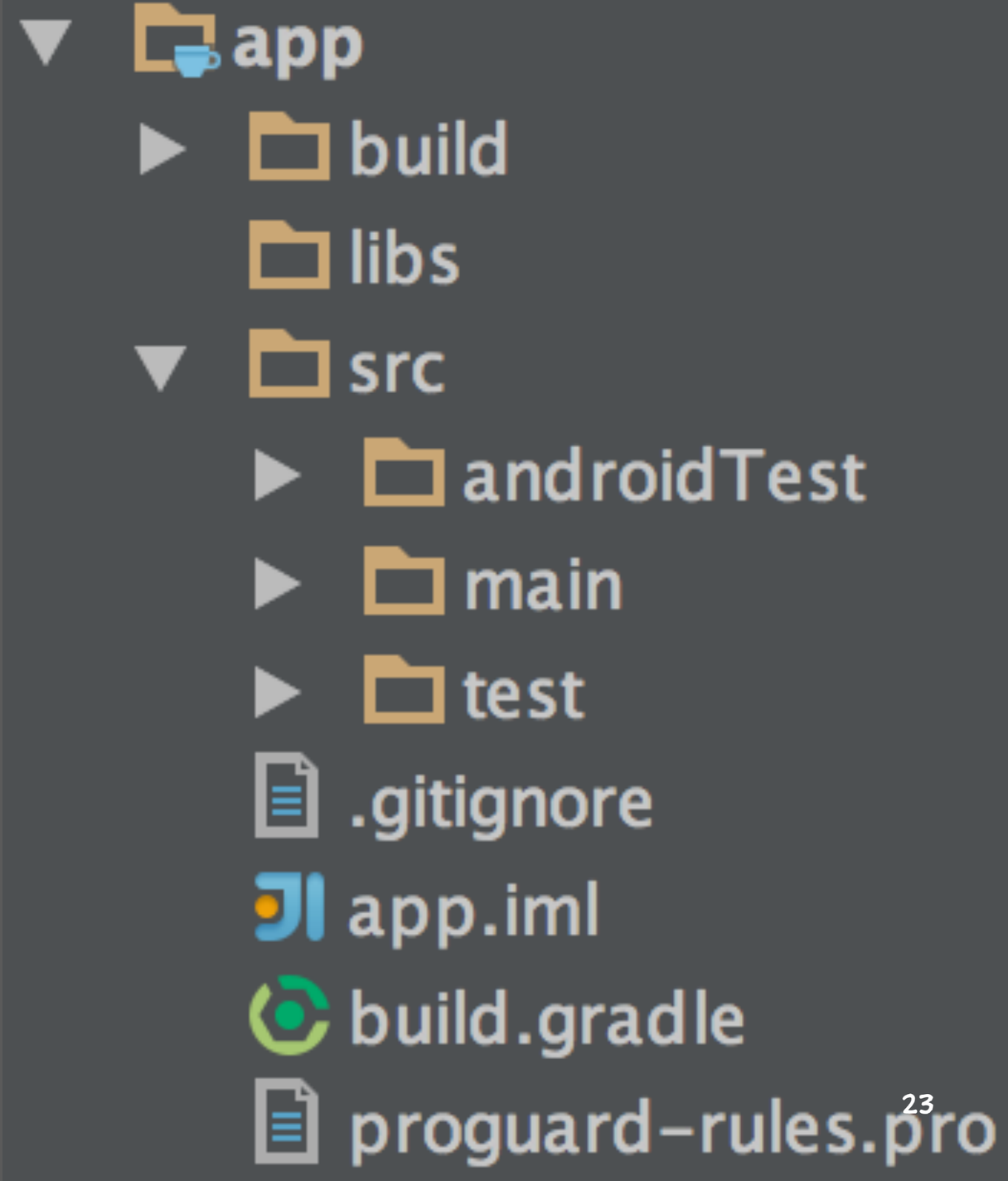
Begin Android Testing

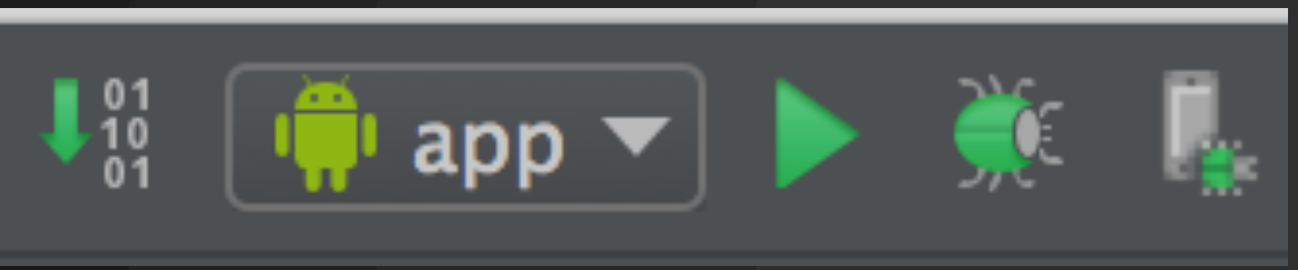
1. Testing Framework



Creating HelloTest

1. Create **New Project** in Android Studio
2. Name it **HelloTest**
3. Minimum SDK 16 (Android 4.1 JB)
4. Blank Activity
5. Make some coffee while waiting





Act like an expert

Check if project can be built

```
./gradlew build
```

Check if device connected

```
adb devices
```

Install application

```
./gradlew installDebug
```

Run your first test

```
./gradlew connectedCheck
```

or use Gradle shortcuts

```
./gradlew cC
```

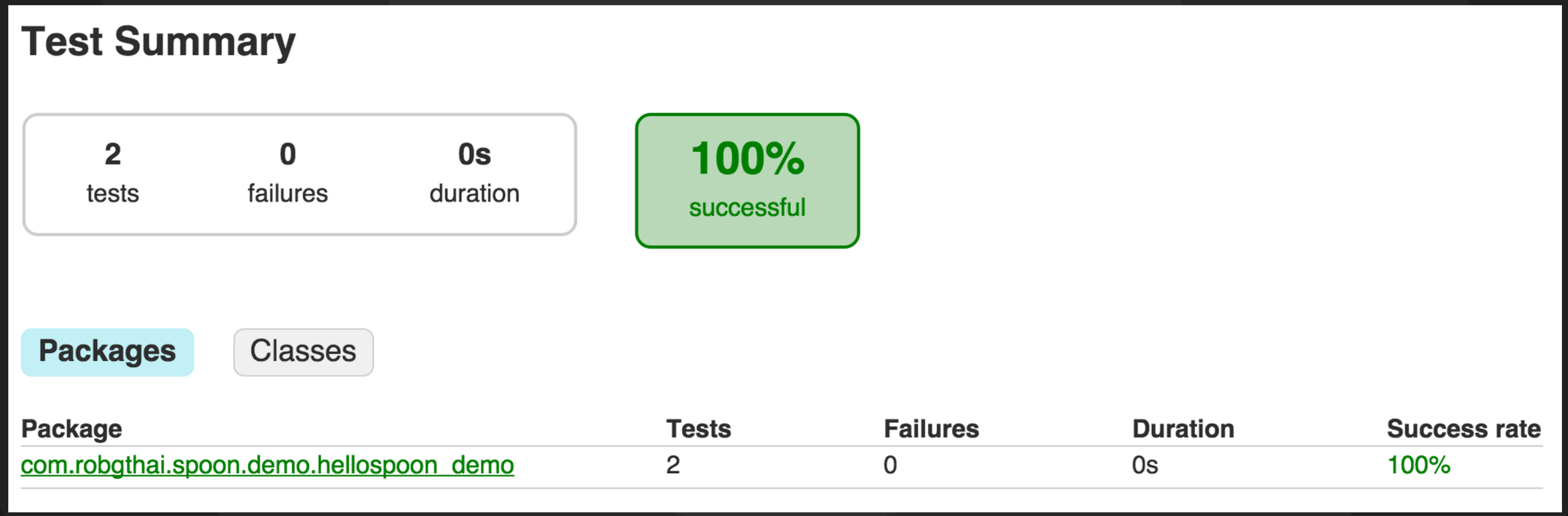
```
:app:connectedCheck

BUILD SUCCESSFUL

Total time: 13.156 secs
```

Test report?

```
open app/build/outputs/reports/androidTests/connected/index.html
```



Test Summary

2 tests	0 failures	0s duration

100% successful

Packages | Classes

Package	Tests	Failures	Duration	Success rate
com.robgthai.spoon.demo.hellospoon_demo	2	0	0s	100%

Create your 1st test case

1. Create **MainActivityTest** Class in the same directory with **ApplicationTest**.
2. Extends ActivityInstrumentationTestCase**2**

```
public class MainActivityTest
extends ActivityInstrumentationTestCase2<MainActivity>{

  public MainActivityTest() {
    super(MainActivity.class);
  }
}
```

Create your 1st test method

```
public void testActivityIsAvailable() {
  assertNotNull("Activity is not available", getActivity());
}
```

Rerun the test

```
./gradlew cC
```

Looks boring?

Introduction to Spoon [1]

[1] http://square.github.io/spoon/

Spoon

Spoon improve upon the existing instrumentation tests to make it more useful by visualizing the result.

Spoon run tests on all connected devices simultaneously as long as **adb** can see it.

Integrating Spoon

- Add Spoon client to project.

```
androidTestCompile 'com.squareup.spoon:spoon-client:1.1.2'
```

- Capture screenshot inside test.

```
Spoon.screenshot(activity, "initial_state");
```

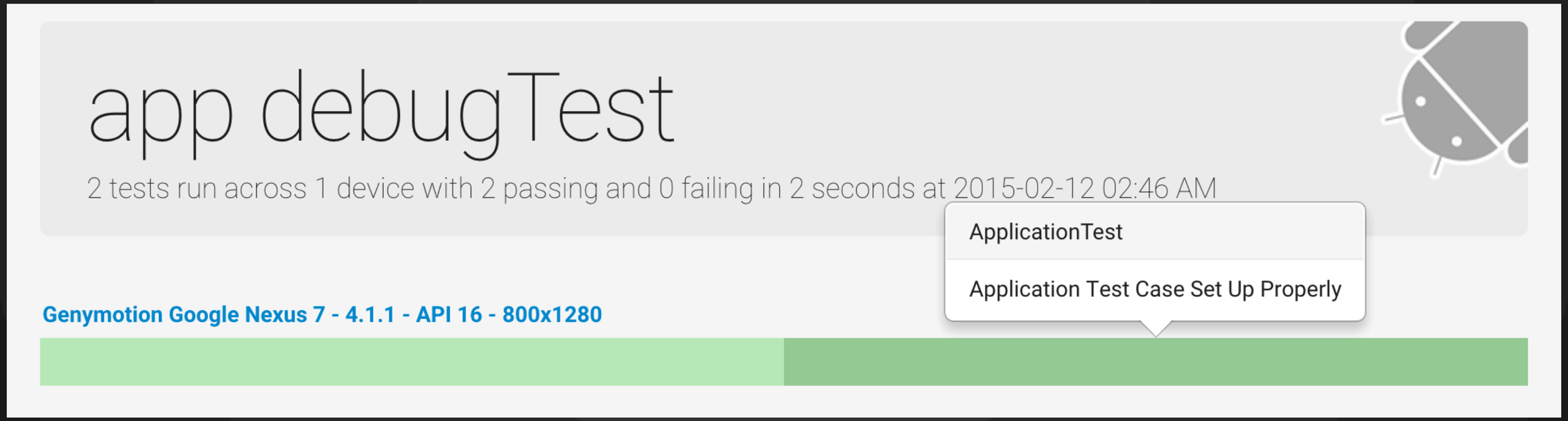
That's it.

Running Spoon

```
java -jar spoon-runner-1.1.2-jar-with-dependencies.jar
 --apk debug.apk
 --test-apk test.apk
 --sdk /usr/local/Cellar/android-sdk/23.0.2
```

Open spoon report

```
open app/build/spoon/debug/index.html
```



Too hard?
Let's find some help

Integrate Spoon-gradle-plugin

Add buildscript to the top of App's **build.gradle** and apply plugin.

```
buildscript {
  repositories {
    jcenter()
  }
  dependencies {
    classpath 'com.stanfy.spoon:spoon-gradle-plugin:0.14.1'
  }
}

apply plugin: 'spoon'
```

Using Spoon-gradle-plugin

spoon-gradle-plugin allow Gradle to start spoon instrumentation testing.

```
./gradlew spoon
./gradlew spoon -PspoonClassName=fully.qualified.TestCase
```

Just like that

Config spoon plugin

```
spoon {
  // for debug output
  debug = true

  // Change report output directory
  baseOutputDir = file("$buildDir/outputs/reports/spoon/")

  // Allow specifying single test via CLI
  if (project.hasProperty('spoonClassName')) {
    className = project.spoonClassName

    if (project.hasProperty('spoonMethodName')) {
      methodName = project.spoonMethodName
    }
  }
}
```

Let's "see" the report

```
public void testScreenshotIsWorking() {
  assertNotNull("Activity is not available", getActivity());
  Spoon.screenshot(getActivity(), "start");
}
```

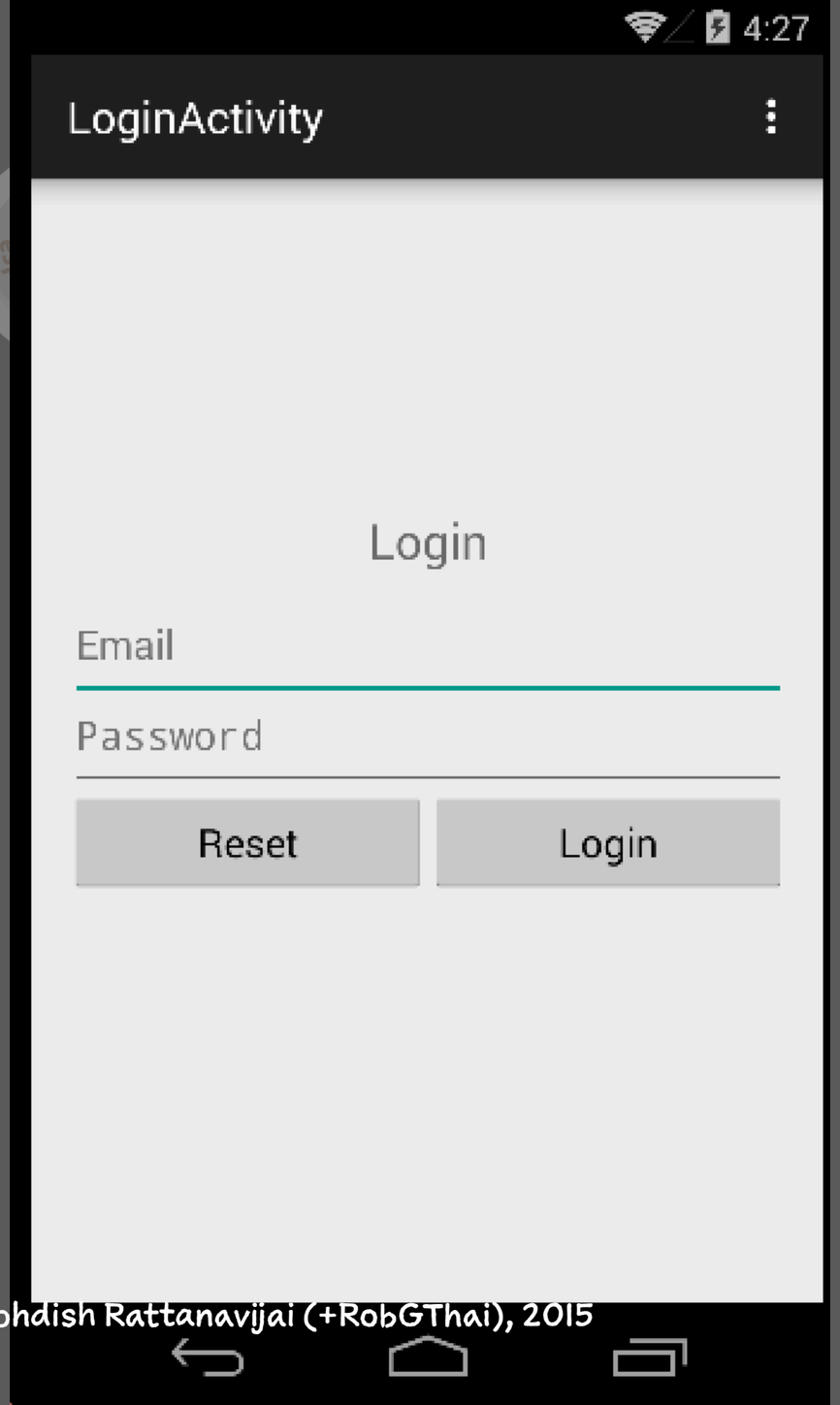
Run it using Spoon Runner

```
./gradlew spoon
```

Let's do some real testing

Instrumentation Testing

1. Design how actual user would interact with your application.
2. Then write a script imitating those actions.
3. Validating if the outcome match the expectation.



What are the Test Cases?

Let's check if user see "Hello World!"

Back to MainActivityTest

```
public void testHelloWorldVisibleByDefault() {
  ViewGroup root = (ViewGroup) getActivity().findViewById(android.R.id.content);
  int total = root.getChildCount();

  for(int i = 0; i < total; i++) {
    View v = root.getChildAt(i);
    if(v instanceof TextView) {
      assertEquals("TextView contain Login"
      , ((TextView) v).getText().toString(), "Hello Login");
    }
  }
}
```


Do not want.


UI TESTING FOR ANDROID
espresso

Espresso is a simple API for writing beautiful UI test

Let's Begin

Espresso.onView()

Use to find view inside the current hierarchy. It is to be used in conjunction with **ViewMatcher** or any **org.hamcrest.Matcher** implementation.

Example:

```
onView(withId(R.id.name_field))
onView(withText("Hello Steve!"))
```

```java
public void testHelloWorldVisibleByDefault() {
    ViewGroup root = (ViewGroup) getActivity()
            .findViewById(android.R.id.content);
    int total = root.getChildCount();

    for(int i = 0; i < total; i++) {
        View v = root.getChildAt(i);
        if(v instanceof TextView) {
            assertEquals("TextView contain Hello World!"
                    , ((TextView) v).getText().toString()
                    , "Hello World!");
        }
    }
}
```

```java
public void testHelloWorldVisibleByDefault() {
    onView(withText("Hello World!"))
            .check(matches(isDisplayed()));
}
```

That's ~~voodoo~~ Espresso

```
onView(Matcher)
      .perform(ViewAction)
      .check(ViewAssertion)
```

1. Espresso's ViewMatcher save the day.

- isDisplayed() // Match displayed view
- isEnabled() // Match enabled view
- hasSibling() // Match view with sibling
- withId // Match view with id
- Many more ...

2. Perform action on view using ... perform()

You can do all sort of stuff using ViewAction.

Click, scroll, type, swipe, hardware button, etc.

3. Check view state using ... check()

Check whether the view match the specified state.

ViewAssertion contains **matches()** method that adapt Matcher to use when checking. ViewAssertions also contain **doesNotExist()** method for check such that.

```
onView(Matcher)
      .perform(ViewAction)
      .check(ViewAssertion)
```

Matchers

USER PROPERTIES
```
withId(…)
withText(…)
withTagKey(…)
withTagValue(…)
hasContentDescription(…)
withContentDescription(…)
withHint(…)
withSpinnerText(…)
hasLinks()
hasEllipsizedText()
hasMultilineText()
```

UI PROPERTIES
```
isDisplayed()
isCompletelyDisplayed()
isEnabled()
hasFocus()
isClickable()
isChecked()
isNotChecked()
withEffectiveVisibility(…)
isSelected()
```

COMMON HAMCREST MATCHERS
```
allOf(Matchers)
anyOf(Matchers)
is(...)
not(...)
endsWith(String)
startsWith(String)
```

HIERARCHY
```
withParent(Matcher)
withChild(Matcher)
hasDescendant(Matcher)
isDescendantOfA(Matcher)
hasSibling(Matcher)
isRoot()
```

INPUT
```
supportsInputMethods(…)
hasImeAction(…)
```

CLASS
```
isAssignableFrom(…)
withClassName(…)
```

ROOT MATCHERS
```
isFocusable()
isTouchable()
isDialog()
withDecorView(…)
isPlatformPopup()
```

SEE ALSO
Preference matchers
Cursor matchers

View Actions

CLICK/PRESS
```
click()
doubleClick()
longClick()
pressBack()
pressImeActionButton()
pressKey([int/EspressoKey])
pressMenuKey()
closeSoftKeyboard()
openLink(…)
```

GESTURES
```
scrollTo()
swipeLeft()
swipeRight()
swipeUp()
swipeDown()
```

TEXT
```
clearText()
typeText(String)
typeTextIntoFocusedView(String)
replaceText(String)
```

View Assertions

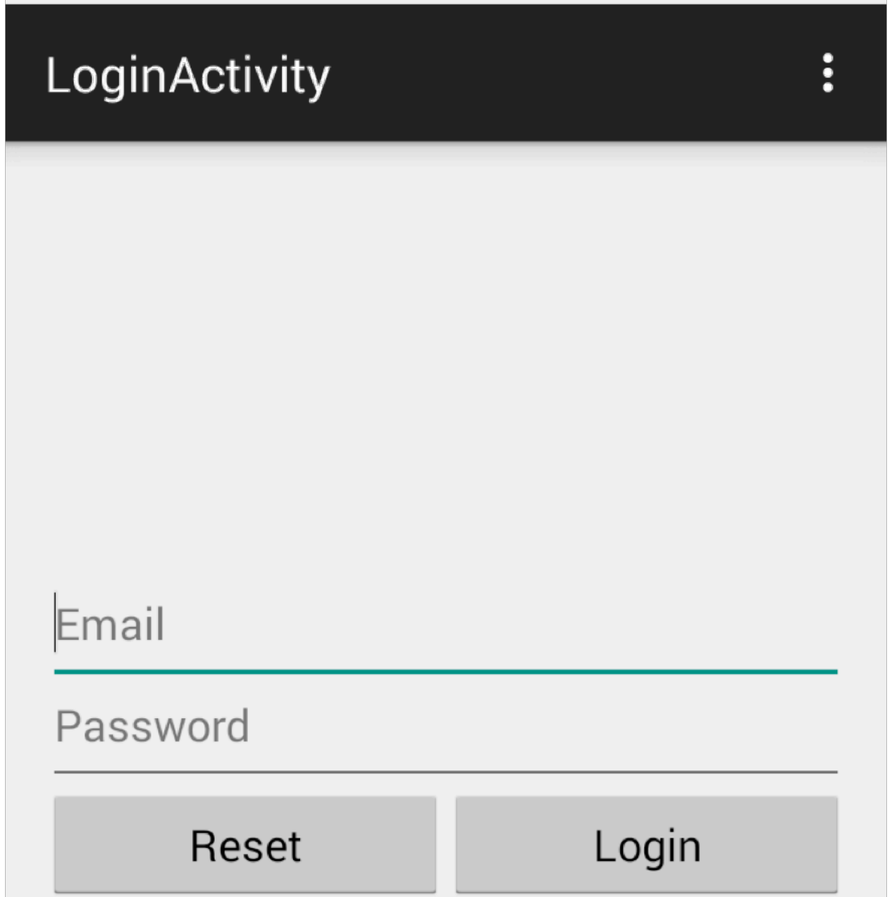
```
matches(Matcher)
doesNotExist()
selectedDescendantsMatch(…)
```

LAYOUT ASSERTIONS
```
noEllipsizedText(Matcher)
noMultilineButtons()
noOverlaps([Matcher])
```

POSITION ASSERTIONS
```
isLeftOf(Matcher)
isRightOf(Matcher)
isLeftAlignedWith(Matcher)
isRightAlignedWith(Matcher)
isAbove(Matcher)
isBelow(Matcher)
isBottomAlignedWith(Matcher)
isTopAlignedWith(Matcher)
```

Let's try this

https://code.google.com/p/android-test-kit/wiki/EspressoV2CheatSheet



Give it a shot

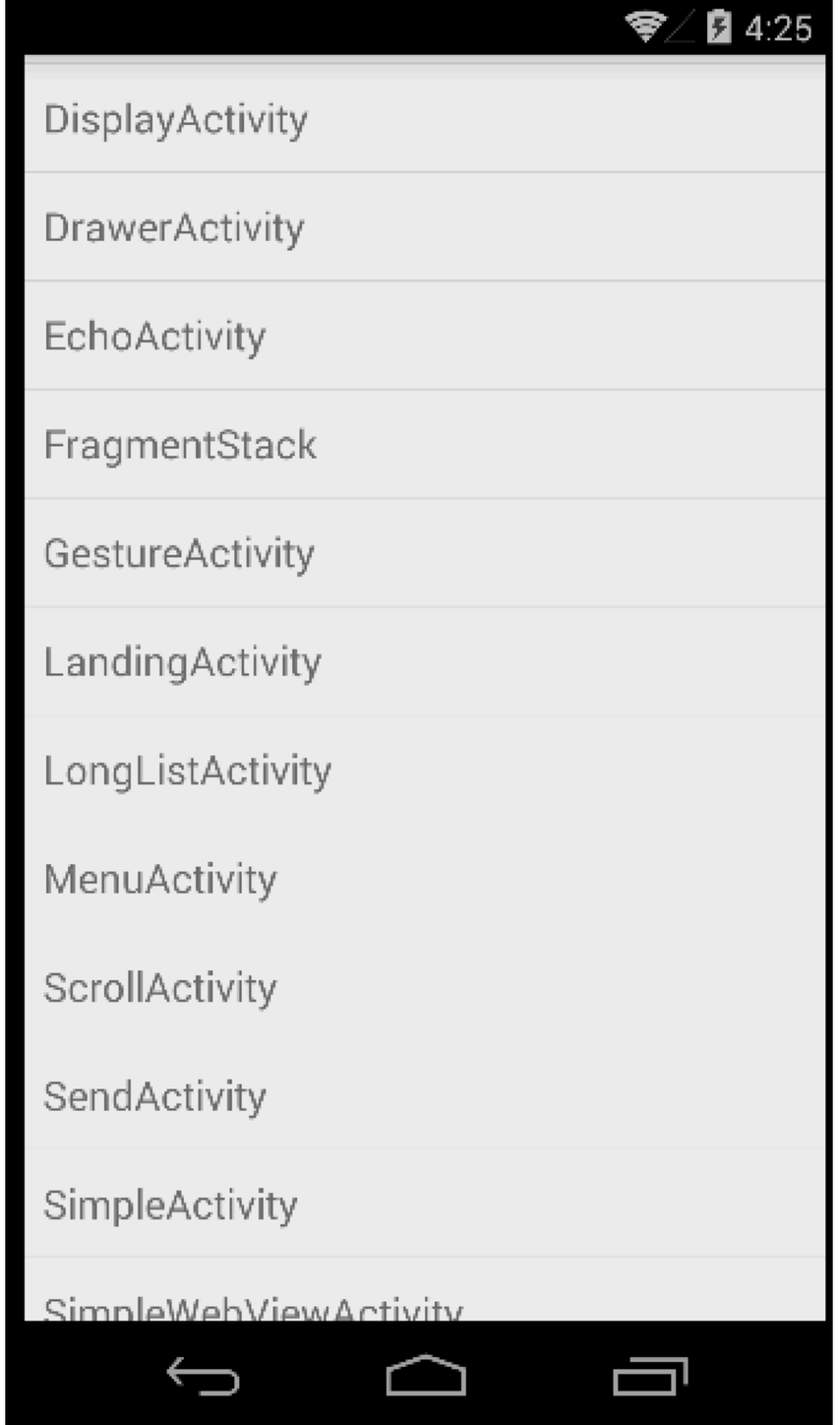
Wild CEO appeared!

The CEO use "Where is my logo?"
It's super effective.

Congratulations you can now use Espresso!

Recap

What about ListView



How?

Espresso.onData()

The ugly sister

Espresso.onData()

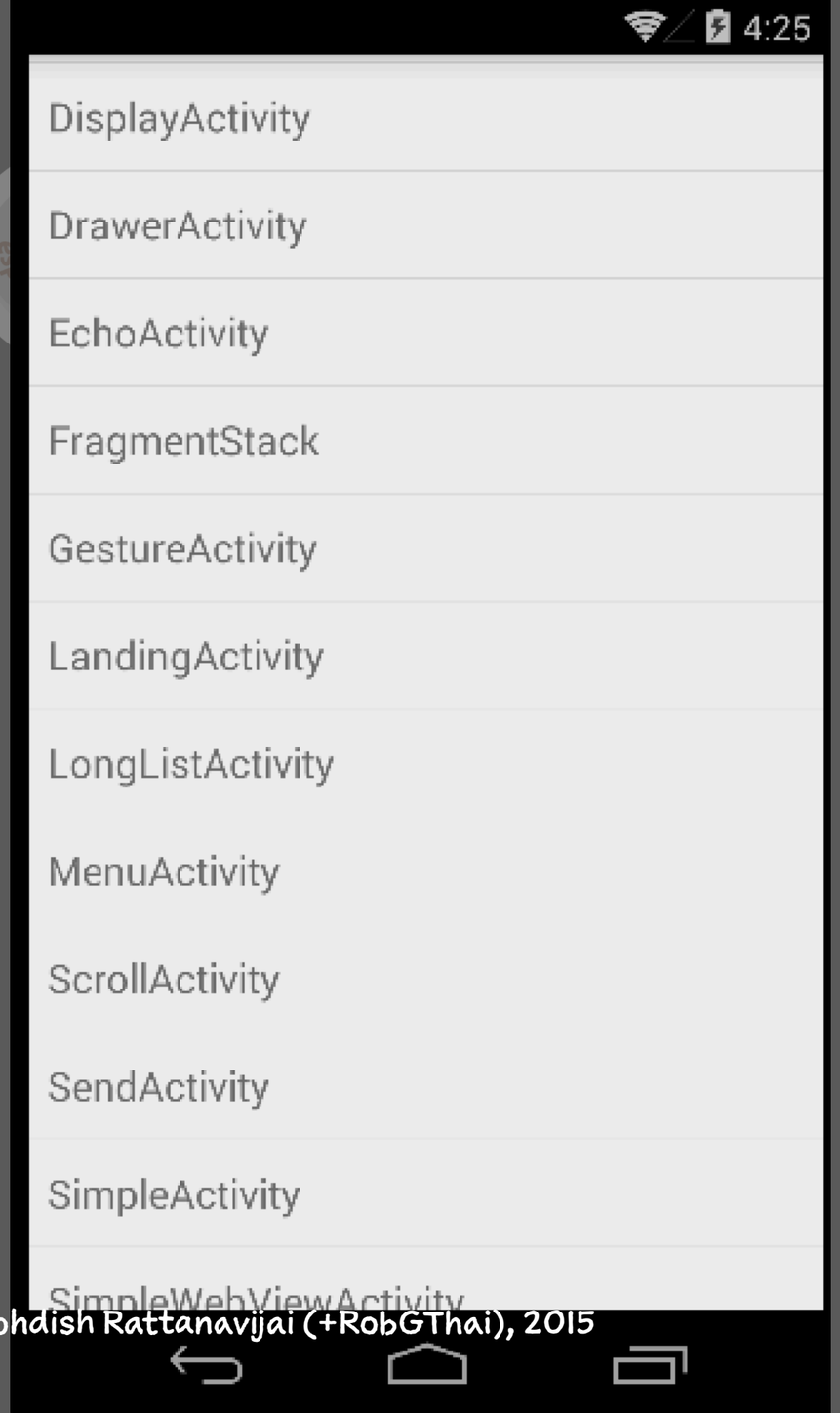
onData is very similar to **onView** but is used to interact with data inside AdapterView. You can specify root or adapter or the view at given position.

- `onChildView()` • `atPosition()` • `inRoot()`
- `inAdapterView()` • `usingAdapterViewProtocol()`
- `check()` • `perform()`

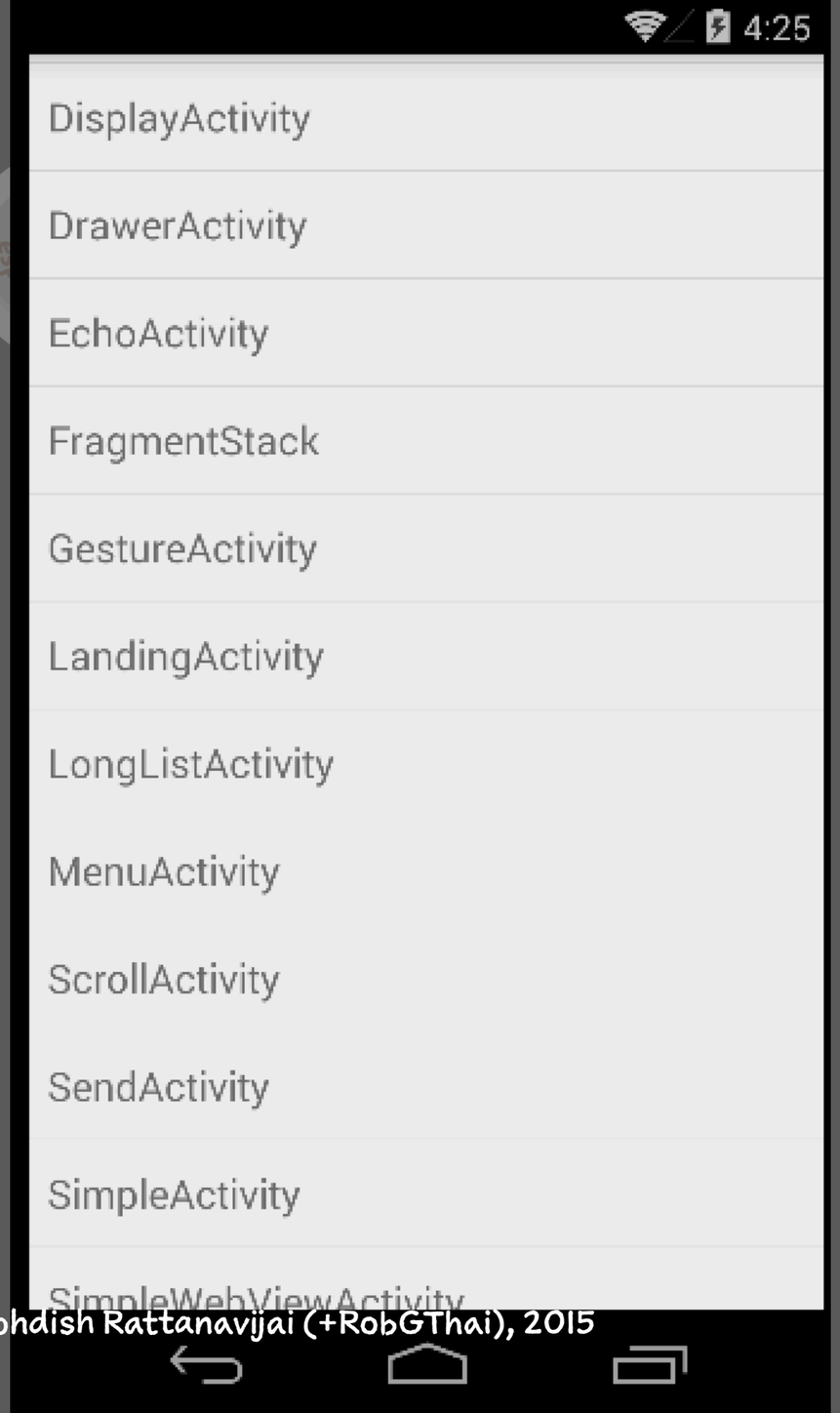
Espresso.onData()

```
onData(Matchers.is(LongListMatchers.withItemSize(8)))
.atPosition(0)
.perform(click());

onView(allOf(withText("7"), hasSibling(withText("item: 0"))))
.perform(click());
```



Hands On with onData Spinner



Hands on with ListView


```
public void onRequestButtonClick(/unused/ View view) {
  Thread t = new Thread() {
    @Override
    public void run() {
      final String helloworld = helloWorldServer.getHelloWorld();
      runOnUiThread(new Runnable() {
        @Override
        public void run() {
          setStatus(helloworld);
        }
      });
    }
  };
  t.start();
}
```

Hands On with Asynchronous task

registerIdlingResources

Espresso recap

- onView()
- onData()
- registerIdlingResources()
- setFailureHandler()
- closeSoftKeyboard()
- pressBack()

What else can it do?

- Open/Close NavigationDrawer
- Option Menu
- Contextual ActionBar
- etc.

Introduction to Burst

Burst

Burst is a library for testing module that required varying data to test.

Sample Case:

A month factory handle creation of month object from the given number of month.

i.e. 1 -> January, 2 -> February, ... , 12 -> December

Standard UnitTest method

Test 1: VerifyJanuary
Test 2: VerifyFebruary
Test 3: VerifyMarch
Test 4: VerifyApril
...
Test 12: VerifyDecember

Let's write that test

You can use Burst in Instrumentation Test with ActivityRule

Integrating Burst

Dependencies

```
androidTestCompile 'junit:junit:4.11'
androidTestCompile 'com.squareup.spoon:spoon-client:1.1.2'
androidTestCompile ('com.android.support.test.espresso:espresso-core:2.0')
androidTestCompile ('com.android.support.test.espresso:espresso-contrib:2.0') {
  exclude group: 'com.android.support', module: 'appcompat'
  exclude group: 'com.android.support', module: 'support-v4'
  exclude module: 'recyclerview-v7'
}
androidTestCompile ('com.squareup.burst:burst-junit4:1.0.2')
```

Integrating Burst

Force Hamcrest versioning

```
configurations.all {
  exclude module : 'junit-dep'

  resolutionStrategy {
    force 'org.hamcrest:hamcrest-core:1.1'
    force 'org.hamcrest:hamcrest-integration:1.1'
    force 'org.hamcrest:hamcrest-library:1.1'
  }
}
```

Challenge Time
Let's automate Github app[9]

[9] https://github.com/github/android

Extras

AndroidTestCase & AndroidUnitTest

Hamcrest Matcher [h]

Hamcrest is a framework for writing matcher objects allowing **match rules** to be defined declaratively.

[h] https://code.google.com/p/hamcrest/wiki/Tutorial

Q&A

Thank you

Poohdish Rattanavijai
thisisrobg@gmail.com
+RobGThai

testHiEspressoIsDisplayed

create a new test to check TextView with "**Hi Espresso**" is displayed.

testTextHelloIsDisplayed

create a new test to check TextView with id **R.id.txtHello** is displayed.

testTextHelloIsDisplayed

create a new test to check TextView with id **R.id.txtHello** is displayed.

Project setup

- espresso-core:2.0
- testing-support-lib:0.1
- espresso-contrib:2.0
- junit:4.11