

Robotium

Easy Black-box Testing for Android

Agenda

- Vanilla vs Robotium
 - Instrumentation testing
 - Write readable tests, with 10x less code [DEMO]
 - (Still) access all the powers of Instrumentation [DEMO]
- Darker than Black box [DEMO]
- Robotium with Maven [DEMO]
- The future of testing on Android

Black box testing?



define:black box testing

Search

- ...doesn't know how an application is designed at the code level...
 - bughuntress.com/analytics/glossary.html
- ...done without reference to the source code...
 - www.sqablogs.com/jstrazzere/46/A+Glossary+of+Testing+Terms.html

Instrumentation testing

Vanilla
VS
Robotium

Instrumentation Testing

(vanilla)

What's instrumentation testing?

- Investigate interaction with UI
- Pressing buttons, navigating menus
- Blackbox method(?)
- Takes time
- Can use JUnit
- Good for acceptance or system tests
- Can be automated!

Android built-in support

- Runs on device/emulator
- JUnit 3 compatible framework
- Allows direct control of user interface
 - Touches
 - Taps
 - Scrolling in lists
- Can be started from Eclipse or shell

Android built-in support

- Tapping a view:

```
TouchUtils.tapView(someView)
```

- Pressing key (physical button):

```
getInstrumentation().
```

```
    sendKeyDownUpSync(KeyEvent.KEYCODE_MENU)
```

- Sending text:

```
sendKeys("some text")
```

Drawbacks and limitations

- Required to know implementation details
- You often have to manually add
`Thread.sleep(500)` to make tests work
- Large apps can be very complex to test
- Tests run slowly
 - Like if it would be done manually
 - Not suitable for TDD

Setting up instrumentation tests

- Runs in emulator/device using JUnit 3
- Separate test project as normal

Writing an instrumentation test

```
public class MyActivityTest extends ActivityInstrumentationTestCase2<MyActivity> {  
    public MyActivityTest() {  
        super("com.mycompany.myapp", MyActivity.class);  
    }  
  
    public void testSomething() {  
        // test code here  
    }  
}
```

Testing a Calculator GUI



Writing an instrumentation test

```
// Find views
EditText num1 = (EditText)
getActivity().findViewById(com.calculator.R.id.num1);
EditText num2 = (EditText)
getActivity().findViewById(com.calculator.R.id.num2);
Button addButton = (Button)
getActivity().findViewById(com.calculator.R.id.add_button);

// Perform instrumentation commands
TouchUtils.tapView(this, num1);
sendKeys(KeyEvent.KEYCODE_1, KeyEvent.KEYCODE_5);
TouchUtils.tapView(this, num2);
sendKeys(KeyEvent.KEYCODE_5);
TouchUtils.tapView(this, addButton);

// Fetch result and compare it against expected value
String actual = num1.getText().toString();
String expected = "20.0";
assertEquals("Addition incorrect", expected, actual);
```



Running instrumentation tests

- Started from Eclipse as “Android JUnit test”
- Can also be started from command line:

```
adb shell am instrument -w  
com.package/android.test.InstrumentationTestRunner
```

Instrumentation testing with Robotium

Robotium

- Facts
 - Instrumentation testing framework
 - Add-on jar
 - Open Source (Apache 2)
- Purpose
 - Simplify making tests

- Benefits

- Easy to write, shorter code
- Automatic timing and delays
- Automatically follows current Activity
- Automatically finds Views
- Automatically makes own decisions
 - when to scroll etc.
- No modification to Android platform
- Test execution is fast

- Current limitations

- Tied to ~~JUnit 3~~ Instrumentation on device
- Tied to one app process
- Needs initial Activity(?)



Writing tests with Robotium

- You use only one class: Solo
- Test class like for normal instrumentation:

```
public void setUp() {  
    solo = new Solo(getInstrumentation(), getActivity());  
}
```

Remember the Calculator GUI?



Remember standard instrumentation test?

```
// Find views
EditText num1 = (EditText)
getActivity().findViewById(com.calculator.R.id.num1);
EditText num2 = (EditText)
getActivity().findViewById(com.calculator.R.id.num2);
Button addButton = (Button)
getActivity().findViewById(com.calculator.R.id.add_button);

// Perform instrumentation commands
TouchUtils.tapView(this, num1);
sendKeys(KeyEvent.KEYCODE_1, KeyEvent.KEYCODE_5);
TouchUtils.tapView(this, num2);
sendKeys(KeyEvent.KEYCODE_5);
TouchUtils.tapView(this, addButton);

// Fetch result and compare it against expected value
String actual = num1.getText().toString();
String expected = "20.0";
assertEquals("Addition incorrect", expected, actual);
```

Test Calculator with Robotium

```
public void testAdd() {  
    solo.enterText(0, "5");  
    solo.enterText(1, "3");  
    solo.clickOnButton("Add");  
    assertTrue(solo.searchEditText("8.0"));  
}
```

Some Robotium commands

- `clickOnButton(String regex)`
- `clickInList(int line)`
- `enterText(int index, String text)`
- `searchText(String regex)`
- `clickOnMenuItem(String regex)`
- `getCurrentActivity()`
- `goBack()`, `goBackToActivity(String name)`

Writing Tests with Robotium

+ still access standard Instrumentation

[DEMO]

Darker than Black Box



Black box testing?

Google

define:black box testing

Search

- ...doesn't know how an application is designed at the code level...
 - bughuntress.com/analytics/glossary.html
- ...done without reference to the source code...
 - www.sqablogs.com/jstrazzere/46/A+Glossary+of+Testing+Terms.html

The usual “Black box” way

- Two projects in Eclipse
 - App project
 - Test project
- Write test *without looking at* app's internals

The Darker than Black box way

- Any application's apk file
- One project in Eclipse
 - ~~App project~~
 - Test project
- No access to original project
- Write test *without any access* to app's internals

Darker than Black Box

Test any APK w/o its source code

[DEMO]

Robotium with Maven

Maven

- Project management + Build tool
- Based on sane defaults
 - Can be overridden with configuration
 - Only configure what's different
- Plugins add functionality
 - [maven-android-plugin](#): build Android projects + run Instrumentation tests
- Dependencies downloaded automatically
 - [Declare need for Robotium](#)
 - Maven downloads jar

Declare need for Robotium

in test project

```
<dependency>  
  <groupId>com.jayway.android.robotium</groupId>  
  <artifactId>robotium-solo</artifactId>  
  <version>1.8.0</version>  
</dependency>
```

Perform build and Run all Tests

including Robotium tests

```
$ mvn install
```

Maven with Robotium

Automating test execution

[DEMO]

The future of Testing on Android

Google's Roadmap



Google's Roadmap

<http://source.android.com/roadmap/>

- Last entry “Beyond Q1 2009”:
 - Support for WVGA and QVGA...
- Not keen on disclosing what they will / will not release.

Google's Roadmap

~~<http://source.android.com/roadmap/>~~

- Last entry “Beyond Q1 2009”:
 - Support for WVGA and QVGA...
 - Page deleted
- Not keen on disclosing what they will / will not release.

Robotium's Roadmap for Instrumentation Testing

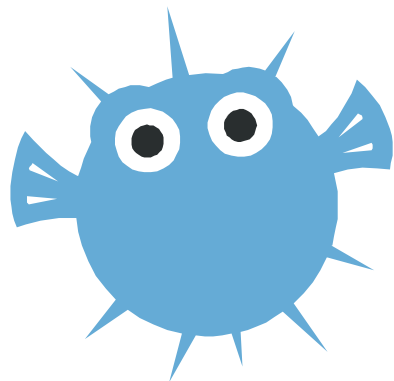
- Features we want to implement next
 - Remote control
 - Similar to Selenium RC
 - Cucumber integration
 - Generate screenshot on failure
 - UI test coverage
- Features further down the line?
 - Multidevice support

Robotium resources

Getting Started +
Robotium Developers discussion group:

www.robotium.org

Questions?



JAYWAY