Test automation and Model-Based Testing in agile dev cycle @ Spotify

Kristian Karl and Peng Ge

October 20, 2013
New recommended albums

Trending playlists near you

New releases

Offline playlists

Discover New Music

wearehunted.com
Why automate?

• We want to be faster
• Shorter release cycles
• Facilitate the testing in agile dev cycle

From: “Intelligent Test Automation” by Harry Robinson
What to automate?

Graphical user interface testing
Usability testing
Software performance testing
System testing
Functional testing
Load testing
Volume testing
Stress testing
Security testing
Scalability testing
Sanity testing

Unit testing
Smoke testing
Component testing
API testing
Regression testing
Installation testing
Maintenance testing
Recovery and failover testing.
Accessibility testing
Monkey testing
Integration testing
Our challenges

- Hard-to-test SUT
  
  *(Experiences of test automation: case study 1, An Agile Team’s Test Automation Journey: The First Year), Dorothy Graham & Mark Fewster)*

- Maintenance of automation
- Peoples expectations
- Flaky SUTs
- Flaky tests
- Re-prioritizations of non-critical but test hindering bugs
- Test data, test environments
- Supporting services
Our goals

• Create automated end-user regression tests on 4 major platforms
  1) Desktop – Windows and OSX
  2) iOS – iPhone and iPad
  3) Android
  4) Webplayer
  5) Some backend services

• To ease the workload for testers
• To deliver automated regression tests for a feature as a part of definition of done
• To deliver short feedback loops to teams using Dashboards
Model-based testing

• The models are the abstraction layer
• The testers designs the automation using models
• The developers implements the code of the automation
Test automator

- A test automator is a professional Java developer.
- Test experience is not mandatory.
- A test automator is embedded in the squad (team).
- Test automators form their own Guild.
GraphWalker: Model-Based Testing Light
What we needed:

• Easy to learn modelling syntax
• Open source, or freeware tools
• OS platform independency
Some words about GraphWalker

- GraphML [http://graphml.graphdrawing.org/]
- Simplistic syntax
- No exit or stop nodes
- Online generation
GraphWalker – Commands

• ANALYZE  
• GUI  
• HELP  
• LOG  
• MANUAL  
• MERGE  
• METHODS  
• OFFLINE  
• ONLINE  
• REQUIREMENTS  
• SOAP  
• SOURCE  
• XML
GraphWalker – Generators

- RANDOM
- SHORTEST_NON_OPTIMIZED
- A_STAR
- ALL_PATH_PERMUTATION
GraphWalker – Stop conditions

- REACHED_EDGE
- REACHED_VERTEX
- REACHED_REQUIREMENT
- EDGE_COVERAGE
- VERTEX_COVERAGE
- REQUIREMENT_COVERAGE
- TEST_LENGTH
- TEST_DURATON
- NEVER
GraphWalker – Vertex Keywords

• BLOCKED

• REQTAG

• SWITCH_MODEL
GraphWalker – Edge Keywords

- **BLOCKED**

```
  v_A
  |   |
  v_B  v_C  v_D
  |   |   |
  e_B e_C e_D
  |
  v_E
  |
  e_E
```

- **WEIGHT**

```
  v_A
  |   |
  v_B  v_C  v_D
  |   |   |
  e_B 0.5  e_D
  |   |
  v_C
```
GraphWalker – Java API

- Online generation
- Java Reflection
- Model <-> Java Class
- JavaDoc API

[http://graphwalker.org:8080/job/graphwalker/site/apidocs/index.html]
@Test
public void a_star() throws InterruptedException, StopConditionException, URISyntaxException {
    ModelHandler modelhandler = new ModelHandler();

    // Get the model from resources
    URL url = MultiModelTest.class.getResource("/model/ShoppingCart.graphml");
    File file = new File(url.toURI());

    // Connect the model to a java class, and add it to graphwalker's modelhandler.
    // The model is to be executed using the following criteria:
    // EFSM: Extended finite state machine is set to true, which means we are using the data domain
    // in the model
    // Generator: a_star, we want to walk through the model using shortest possible path.
    // Stop condition: Edge coverage 100%, we want to walk every edge in the model.
    modelhandler.add("Amazon", new Amazon(file, true, new A_StarPathGenerator(new EdgeCoverage(1.0)), false));

    // Start executing the test
    modelhandler.execute("Amazon");

    // Verify that the execution is complete, fulfilling the criteria from above.
    AssertassertTrue(modelhandler.isAllModelsDone(), "Not all models are done");

    // Print the statistics from graphwalker
    String actualResult = modelhandler.getStatistics();
    System.out.println(actualResult);
}
@Test
public void addMultipleGenerators() throws StopConditionException {
    ModelAPI model = new ModelAPI("graphml/org.graphwalker.multipleModels/a.graphml");
    model.setWeighted(false);
    model.setExtended(true);

    CombinationalCondition combinationalCondition = new CombinationalCondition();
    combinationalCondition.add(new RequirementCoverage(1.0));
    combinationalCondition.add(new EdgeCoverage(1.0));

    AlternativeCondition alternativeCondition = new AlternativeCondition();
    alternativeCondition.add(combinationalCondition);
    alternativeCondition.add(new TimeDuration(900));

    CombinedPathGenerator generator = new CombinedPathGenerator();
    generator.addPathGenerator(new A_StarPathGenerator(new ReachedVertex("C")));
    generator.addPathGenerator(new RandomPathGenerator(alternativeCondition));

    model.setGenerator(generator);
    Assert.assertTrue("Failed setting up the model", model.getMbt().hasNextStep());
}
Developers and developers

• Why not use the developers for TA?
• Why use developers for TA?
• Using Test API’s
• Defined by TA
• Implemented by developers
Before

android.view.View seekBarView =
solo.getView(com.spotify.mobile.android.ui.view.CancellableSeekBar.class, 0);
int[] xy = new int[2];
seekBarView.getLocationOnScreen(xy);
solo.clickOnScreen(xy[0] + 9 + (seekBarView.getWidth() - 18) * position, xy[1] + seekBarView.getHeight() / 2.0f);

After

page().seekTrack(position)
Live Tutorial

- **How to model a simple use case**
  Model an end user acceptance [regression] test of the Search functionality in the Spotify Webplayer. This is done by QA.

- **Refactor the model**
  Adapt the model so GraphWalker accept it’s syntax. This is done by the test automator, and then reviewed by QA.

- **Develop the test automation code**
  The test automator writes the code that implements model. GraphWalker is integrated into the code, and executes the test together with TestNG. This is don by the test automator.
Source code is provided at this Dropbox link.