Technical Presentation Milestone

Comparing Testing Frameworks for Android Applications

Sahil Jain
Android Engineering Intern
Remind
2B Candidate for Bachelor of Software Engineering
July 22, 2015

Copyright © 2015 by *Sahil Jain*. All rights reserved.

Outline

- Goal of Presentation
- Background Information
  - Types of tests
  - How each framework works
  - Pull request workflow
- Motivation Behind Testing
- Current Problems
- Solution Criteria
- Evaluation
- Recommendation
- Conclusion
- Questions

Goal of Presentation

- Highlight importance of testing
- Illustrate how each testing framework works
  - Espresso vs. Robolectric
- Help audience decide which framework is right for their projects

Types of Tests

Unit Tests:

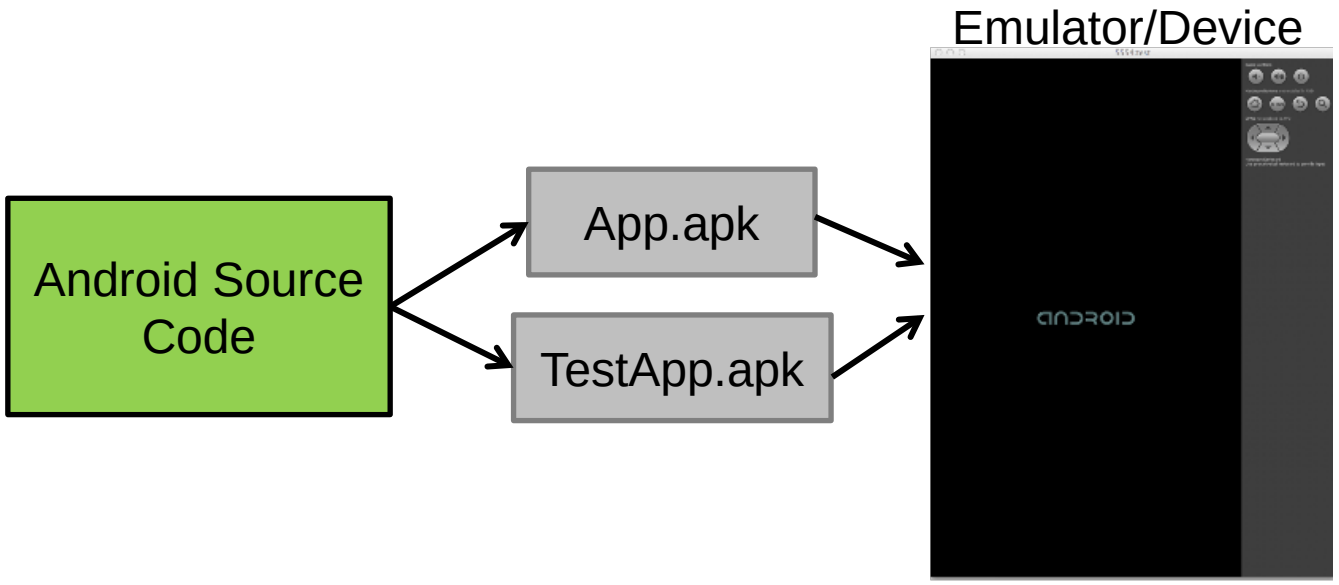
- Assert properties of single programmatical function
- Stub functionality not related to function being tested
- Isolated in terms of code

Functional Tests:

- Assert completeness and correctness of single feature
- Harder to stub functionality
- Isolated in terms of business logic

How Espresso Works

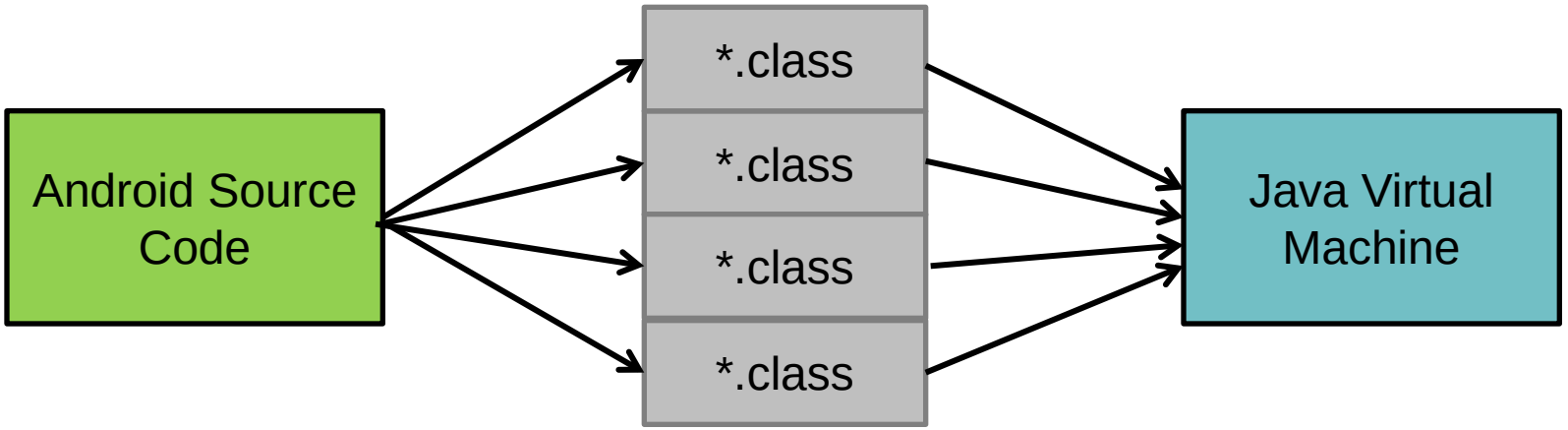
- Espresso decorates Android Instrumentation Test Framework
- Deploys packaged app to an Android device
- Performs UI commands based on timer



www.opencredo.com/2014/01/28

How Robolectric Works

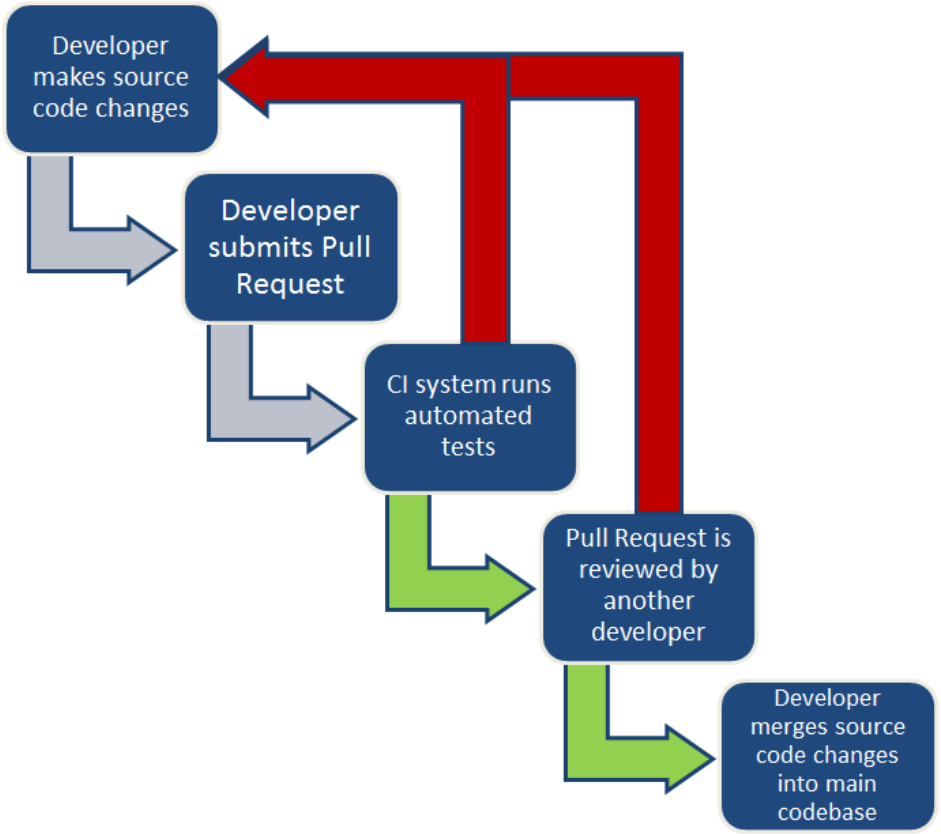
- Android OS is mocked in JVM
- OS-specific operations don’t happen, result is faked
- No real UI, screen operations are mocked
- Excels at unit testing



Pull Request Workflow

Pull Request: a proposed list of changes to the source code

Continuous Integration: upload source code multiple times/day



Motivation Behind Testing

- Flexibility
  - Change code without fear
- Maintainability
  - Tests act as documentation
- Reusability
  - Each component works

“Only judge detailed enough to test code is code.”

Current Problems with Espresso

- Cannot run Espresso tests in the cloud
  - Espresso requires an emulator/device
- Test results unreliable
  - Hypothesis: built-in timer is inaccurate
- Tests take too long to run
  - Hypothesis: packaging app takes a long time

circleci.com

Travis CI

travis-ci.com

Solution Criteria

Quantitative	Qualitative
Speed (3 minutes or less)	Code complexity should not increase
Reliability (<5% false results)	Significant code coverage

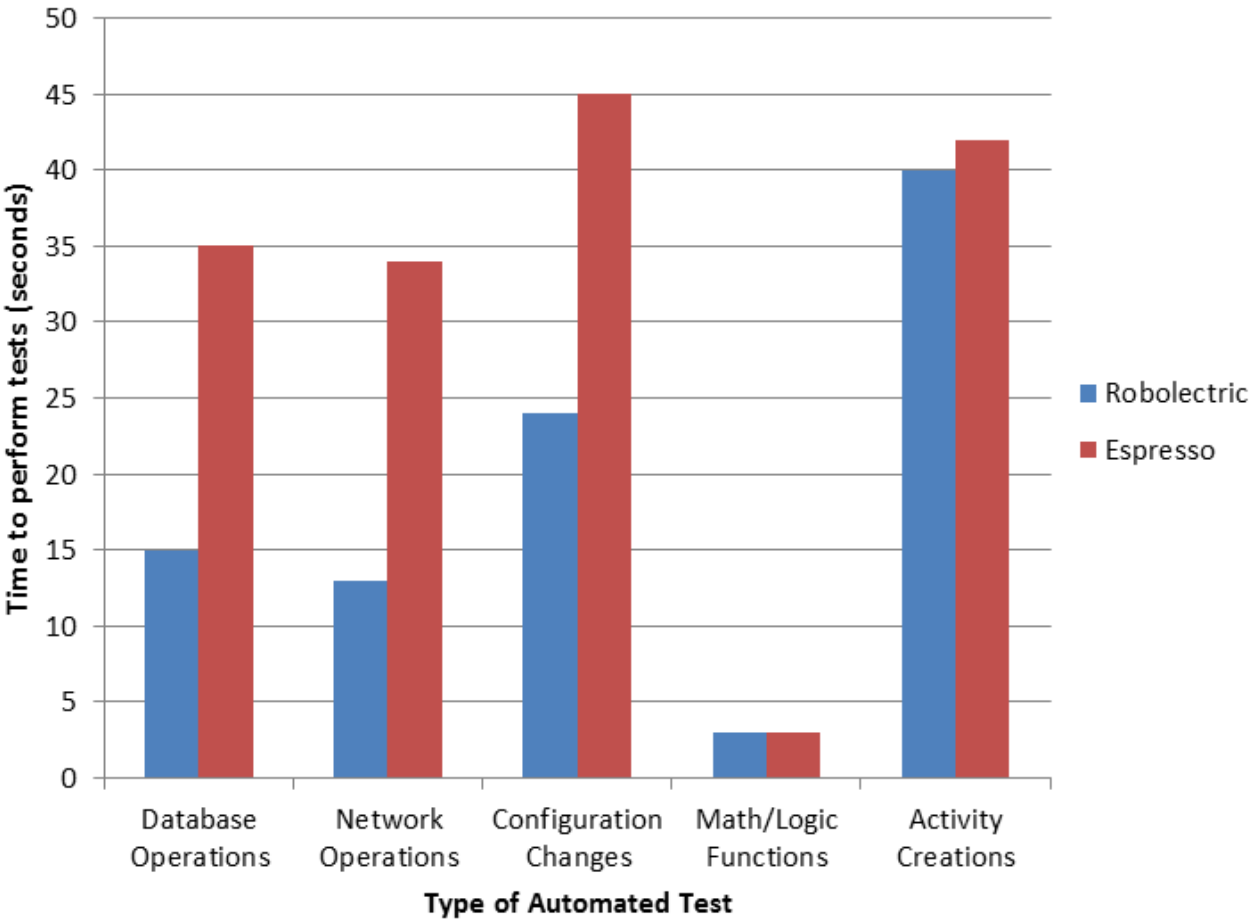
developer.android.com

vs.

robolectric.org

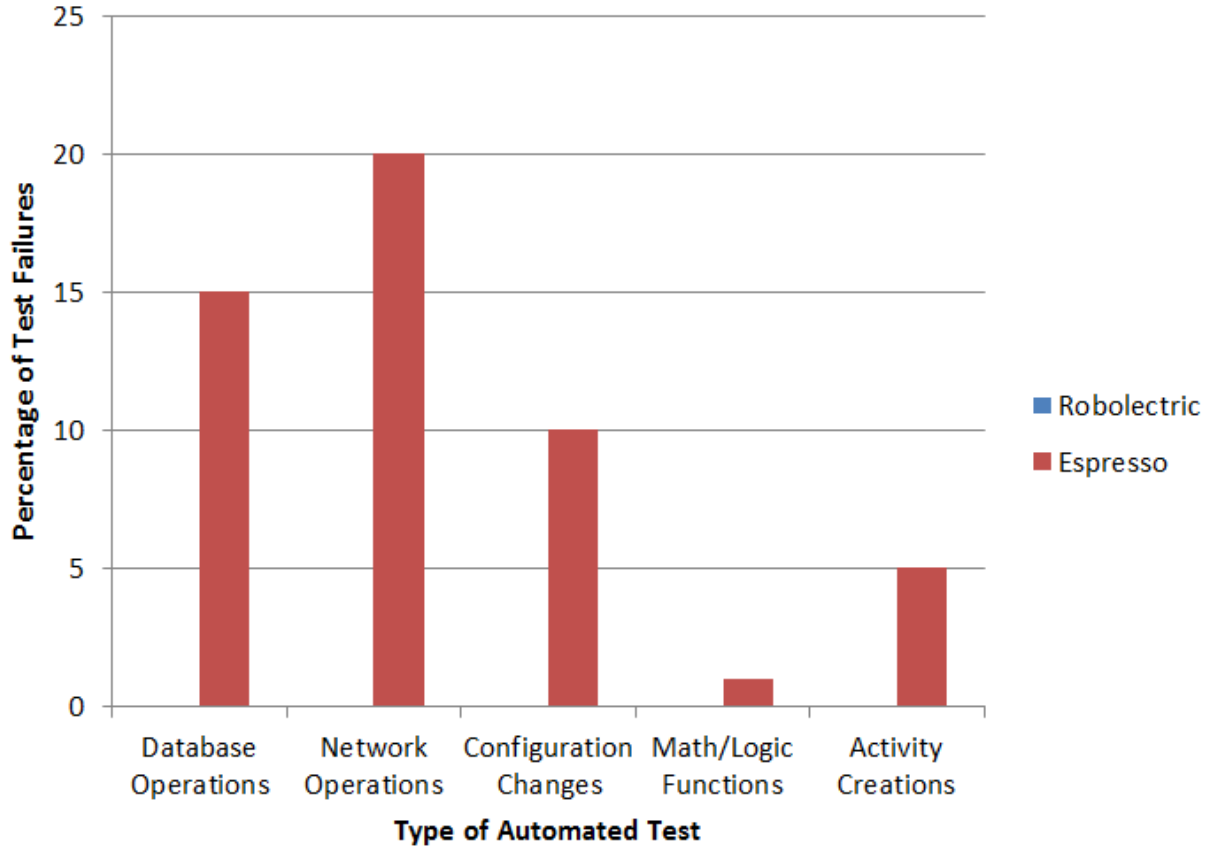
Speed Evaluation

– Long operations are mocked by Robolectric



Reliability Evaluation

- Espresso unreliable due to timer-based system
- Robolectric reliable because operations are mocked out



Code Coverage Evaluation

- Robolectric performs tests using shadow objects
- Robolectric ships new shadow objects regularly, but still behind
- Perpetual lag of shadow objects vs. actual objects

- Espresso has no such limitation
- Espresso favoured in this category

Code Complexity Evaluation

Similarities	Differences
• Number of lines of code • Readability of code • Same set of assertions • Both frameworks founded on JUnit	• Robolectric requires boilerplate to set up activities

Overall, both frameworks are very similar

Recommended Testing Framework

Multi-Criteria Decision Analysis:

Criterion	Weighting	Robolectric	Espresso
Reliability	40%	0.40	0.08
Speed	30%	0.30	0.18
Code Coverage	20%	0.15	0.20
Code Complexity	10%	0.10	0.10
Total (x100)	-	**95**	**56**

- Robolectric better with reliability and speed
- Espresso better with code coverage
- Overall, **Robolectric is the favourable solution**

Conclusions

- Robolectric performs faster than Espresso
- Robolectric is more reliable than Espresso
- Robolectric lags behind Espresso in code coverage
- Trivial difference in code complexity

Above results due to:

- Robolectric runs in JVM, Espresso runs on device
- Robolectric mocks Android OS

Summary

- Highlighted importance of testing
- Illustrated how each testing framework works
- Evaluated each framework in context of CI
- Recommended framework based on results

Questions?

“If you have tests, you do not fear making changes to the code! Without tests every change is a possible bug.” – Robert C. Martin, Clean Code

References

[1] N. M. Fraser and E. M. Jewkes, *Engineering Economics: Financial Decision Making for Engineers*. Toronto, Canada: Pearson Canada, 2012.

[2] D. T. Milano and P. Blundell, Learning Android Application Testing. Birmingham, UK: Packt Publishing, 2015.

[3] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. Westford, MA: Prentice Hall, 2008.

All diagrams and photos that are not cited are my own work