Automating State to State testing of mobile applications to mobile platforms

Jonathan Sanders
Dr. Justice

ABSTRACT

This paper outlines a new way to test a devices states' affect on an application.

KEYWORDS

automata for mobile device testing,automated state testing

ACM Reference Format:
Jonathan Sanders and Dr. Justice. 2017. Automating State to State testing of mobile applications to mobile platforms. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A common problem with automated mobile software testing is testing mobile applications when the state of the device changes. Testing the different states of an application has been enhanced with tools such as Espresso and Robotium [1]. The problem with this issue is that often, the state of a device will cause a catastrophic failure of a feature within an application when the state of a mobile device changes; therefore, it is more expensive to not test device states against application states at all. Since the state of an application can be easily saved and the state of a device can be easily changed programmatically, a solution can be made that automatically tests application states with different device states. For applications being used in different device states, such as a mapping application used outside of network range or medical applications using sensors for monitoring functions, application to device state testing this becomes an extremely important issue. Some of these issues can even be life safety critical. Additionally, the innovative ways applications are used with new hardware is only increasing. State testing of applications with new hardware devices is a critical area of research to ensure that applications can continue to operate well.

"The field of mobile specific, black-box testing, however, remains thin." [7] That is the reasoning behind this endeavor to make easier testing tools that easily test the interaction of device states with applications states.

We have developed a Mobile application called "TADS" (Test Application to Device State) that uses a Mobile development UI tester to test the Mobile application against multiple states and sub-states of the device [9]. TADS is primarily concerned with identifying errors caused by changing device states and not as concerned with identifying why the state change caused a failure.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RESEARCH CONTRIBUTIONS

This work demonstrates that a library for doing state testing of devices that can be run with espresso is highly beneficial and easy to make. The open source community could easily extend this set of tools started int his work that would compliment the automated testing tools that come standard with Android Studio. Or, a company could make a simple product that would be very inexpensive that could be used by developers to improve the quality of their android applications.

The second contribution this work makes is that it will hopefully motivate Google to enable manufacturers of hardware interfaces for Android devices to be able to submit Android Debug Bridge (ADB) commands, that change their device's states, to the ADB driver so that device state testing can become more robust, efficient, and commonplace. Currently, the inner workings of hardware devices must be understood in order to accomplish state changes and Google providing a way for state changes to be initiated via the ADB command will prove invaluable.

3 BACKGROUND

As of the writing of this paper, there is not any research available on building out an automatic device state testing utility. In order to understand this project a few items must be known ahead of time.

Espresso, an android testing application, has an excellent tool for testing Mobile application UI's [6]. The tool records what is happening at a code level while a user performs different actions using the UI. This enables a working application to have a test automatically generated that can then be run at a later time which can be used to create a regression test suite. This actually enables states of the application to be tested without having to use any form of state machines or modeling. This tool called "Test Recorder" will be foundational to TADS.

4 RELATED WORK

There are several tools developed that can capture various information of an application and store said information for later tests. These tools can be leveraged to create interesting app states instead of simple object states. For instance the authors of [8] built a tool that automatically captures UI information as a user utilizes an application. The researchers with[4] made a tool that can store UI information and then auto generate oracles that then get scripted into an Espresso test script for later use as a test case. This tool could be used to run test cases as the DVCs execute as the state instead of static properties being evaluated.

Many others, such as [4] have made tools for recording tests that can be later executed in ways that are plaform independent which is

a nice utility that we will not be concerned with in this work. Others have made unit level state testing models such as [5] in which the state of the application is tested by automatically building a model using a dynamic and static crawler and then running that created model against a verification algorithm. These researchers developed a way to use model testing to find security vulnerabilities [2]. Some have used machine learning to find the states of an application that are probably of value to test that have not been tested and make a model of those states for testing [3].

5 PAST RESEARCH CHALLENGES

There are numerous challenges to this research. The main challenge is that I am not up to speed on how espresso works from a low level perspective. The next issue is that I am not aware of how Android Studio works at a very low level either. In order to exploit architectural features one must be aware of how these items work precisely.

For instance, my first attempt was to make a set of functions in a name space that would be called iteratively with a loop added to the test case. This had numerous issues and I could not make it work. The problem was that I had to force the application back to the beginning each iteration. There was not a good function that was easily called to do so because of the nature of the espresso and AndroidJUnit4 test runners. The code that I was embedding was actually inside of the test and so it was impossible to change states and then rerun the application tests from within the tests; otherwise it would have become recursive and app state would have created a whole host of new problems.

My second approach involved making a separate test class that would call the test class that espresso made. There were several problems with this. The biggest problem was that there are these preconditions that must be set using annotations. A "rule" gets set and that must come before the test. The rule must be scoped to the test class and cannot be scoped to the function being called. The way that AndroidJUnit executes the tests is that it sees the "@RunWith" annotation which is tied to a class, then the "@Rule" annotation, then the "@Test" annotation. It executes in that order. So, when a new test class is generated, those idioms must be used. I discovered that trying to loop through one test class within another was not easily accomplished and could possibly be impossible. I shifted my approach to the current implementation.

6 IMPLEMENTATION

The main purpose of this Application is to simplify testing of different device states with the application. There are preloaded "device state changes" (DSCs) that can be used in a test suite to test the app states that are chosen for testing. A DSC is a test case that starts with a device state then runs the test cases then changes the Devices state then runs the tests again. A few examples of DSCs are: airplane mode off, sample the app state, switch airplane mode to on, re-sample the app state to evaluate.

I shifted the approach to be able to implement the DSCs to a BATCH script using ADB from the command line. This approach works well because intents and actions can be called from the command line. The command to run the tests can be changed to either run all tests in the project or to run individual tests.

Currently the script will run the test, change the state of the device, and then rerun the test. When there is a failure the cause of the failure is captured in the command window.

7 FUTURE RESEARCH CHALLENGES AND POSSIBLE SOLUTIONS

Scripting does not offer the robust programming capabilities that true object oriented architectures offer. Building classes to iterate different device states and enforcing a true DRY and SOLID scripting library may prove to be a challenge. Additionally, the automated testing tools available for BATCH files are limited. One possible solution to this problem is to implement a .NET WPF project that calls the ADB API and can be used to execute the automated device state tests.

Another challenge is that commands do not exist for ADB to change proprietary device states. Figuring out how to instantiate hardware devices from the command interface will prove extremely difficult and more manual testing will be required in those instances.

A third challenge to overcome will be making the logging and debugging apparatus in a way that it is easy to use and robust. Currently only the DSC that was in play during the execution of the tests will be recorded and debug information from the ADB driver. It is unknown to the research team at this time how much of the Android debugging information actually comes through the ADB driver. The research team will have to tap into android logs and capture pertinent information or some other solution to capturing pertinent information will have to be developed. This task is outside the scope of this current project (Note to class members, this will probably get moved to future research section).

Timeline	
10/16	Investigate Classes scripting
10/23	Figure out more ADB device commands to change states
10/30	Implement with more commands
11/6	Capture Logging
11/13	Well defined Metrics
11/20	Finding apps to test library with
11/27	Testing and recording
12/4	Refining
12/11	Refining / turn in

REFERENCES

[1] 2016. Top 10 Mobile Testing Tools. (Nov 2016). http://www.optimusinfo.com/top-10-mobile-testing-tools/

[2] G. Bai, Q. Ye, Y. Wu, H. Merwe, J. Sun, Y. Liu, J. S. Dong, and W. Visser. 2017. Towards Model Checking Android Applications. *IEEE Transactions on Software Engineering* PP, 99 (2017), 1–1. https://doi.org/10.1109/TSE.2017.2697848

[3] Wontae Choi, George Necula, and Koushik Sen. 2013. Guided GUI Testing of Android Apps with Minimal Restart and Approximate Learning. *SIGPLAN Not.* 48, 10 (Oct. 2013), 623–640. https://doi.org/10.1145/2544173.2509552

[4] M. Fazzini, E. N. D. A. Freitas, S. R. Choudhary, and A. Orso. 2017. Barista: A Technique for Recording, Encoding, and Running Platform Independent Android

Tests. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 149–160. https://doi.org/10.1109/ICST.2017.21

[5] Amin Milani Fard, Mehdi Mirzaaghaei, and Ali Mesbah. 2014. Leveraging Existing Tests in Automated Test Generation for Web Applications. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*. ACM, New York, NY, USA, 67–78. https://doi.org/10.1145/2642937.2642991

[6] Godfrey Nolan. 2015. *Agile Android*. Apress Disctributed to the Book trade worldwide by Springer Science+Business Media, Berkeley, CA New York.

[7] Fernando Paulovsky, Esteban Pavese, and Diego Garbervetsky. 2017. High-coverage testing of navigation models in Android applications. In *Proceedings of the 12th International Workshop on Automation of Software Testing*. IEEE Press, 52–58.

[8] F. Paulovsky, E. Pavese, and D. Garbervetsky. 2017. High-Coverage Testing of Navigation Models in Android Applications. In *2017 IEEE/ACM 12th International Workshop on Automation of Software Testing (AST)*. 52–58. https://doi.org/10.1109/AST.2017.6

[9] C. D. Turner and D. J. Robson. 1993. The state-based testing of object-oriented programs. In *1993 Conference on Software Maintenance*. 302–310. https://doi.org/10.1109/ICSM.1993.366932