Towards Having a Cloud of Mobile Devices Specialized for Software Testing

Mehmet Cagri Calpur
Sabanci University
Faculty of Engineering and Natural Sciences
Orhanli, Tuzla, Istanbul 34956, Turkey
mehmetcagri@sabanciuniv.edu

Cemal Yilmaz
Sabanci University
Faculty of Engineering and Natural Sciences
Orhanli, Tuzla, Istanbul 34956, Turkey
cyilmaz@sabanciuniv.edu

ABSTRACT

This paper proposes a novel cloud testing platform specialized for software testing. Our novel approach aims to perform dynamic analysis on mobile application binaries, generate the model of the application, its test cases and test input sets on the run. Domain information generated via dynamic analysis and utilization of combinatorial interaction testing for test case and input set analysis will be used for improving the systems coverage capability. The system will be a self learning system in the sense that the lessons learned from testing one application will be used to test another application.

Keywords

Automated Software Testing, Cross Cutting Testing Concerns, Mobile Applications, Mobile Application Testing Environment,Cloud of Mobile Devices, Combinatorial Interaction Testing

1. INTRODUCTION AND BACKGROUND

Mobile platforms are becoming the predominant computing environments. Mobile devices come in various operating systems, sizes, computing powers, hardware settings and have many areas of use. This is commonly known as the "fragmentation problem". Extensive budgets are spent for procuring required hardware to increase device coverage.

The mobile device, its operating system, and the related programming APIs are often relatively new and untested. The working mechanics of the complete system is very complex. This aspect of mobile platforms increases the need for domain expertise.

We envision a cloud of mobile devices specialized for testing mobile applications. The first goal of the system is to automatically explore a given application, discover a behavioural model of the application, generate test cases according to the interaction and action library formed from the application under test and generate test case input equivalence classes. The second goal is to conduct test runs on any number of target operating systems and devices generating test output. The third goal is to refine the test cases by utilizing machine learning methods. Gathered information will build up a domain and expert knowledge pool and will be used to home in on vital testing concerns for future applications under test.

The "Fragmentation Problem" and the importance of application testing on many real devices has been addressed by Vilkomir et al. [8]. The empirical results obtained from this research strongly suggest that device-specific failures are commonplace. For example, at least 13 mobile devices were required to reveal all previously known defects.

Starov et al. [**?**] introduce the idea of multidirectional testing on cloud of devices and emphasize the importance of a combined testing effort from application developers, operating system developers, and hardware developers.

There are already some existing commercial clouds [2, 1], offering test environments on real mobile devices. These clouds typically provide capture and replay-type of services and runs the test cases on a spectrum of mobile devices.

Random testing is one of the most frequently used ways of testing mobile applications. Therefore, mobile platform typically offer random testing frameworks and tools, such as the Monkey tool [**?**] for Android. Many researchers, Amalfitano et al. [3] and Liu et al. [5] are just a few to name, have been working on improving random testing for mobile applications.

In another work, Amalfitano et al. [4] crawl the graphical user interface (GUI) of an application and generate test cases/inputs on the fly by utilizing GUI related events and actions. Evolutionary testing [7] of mobile applications is another emerging concept for testing mobile applications. These approaches typically extract a model of the application using static analysis and then the inferred model is leveraged to further test the application with the goal of achieving high code coverage [6].

2. APPROACH AND IMPLEMENTATION

The cloud testing platform that we propose can be decomposed into three main components and hardware infrastructure. The architectural representation of the system can be found in Figure 1.

The *Device Control* component facilitates the communication between the devices and the testing environment, sending in commands that will be executed on a device and returning the feedback about the actions performed. This component is based on the android debug bridge tools provided with the android platform, specifically the dalvik de-

MobileSoft'16 May 16-17 2016, Austin, TX, USA

ACM ISBN 978-1-4503-4178-3/16/05.

DOI: http://dx.doi.org/10.1145/2897073.2897109

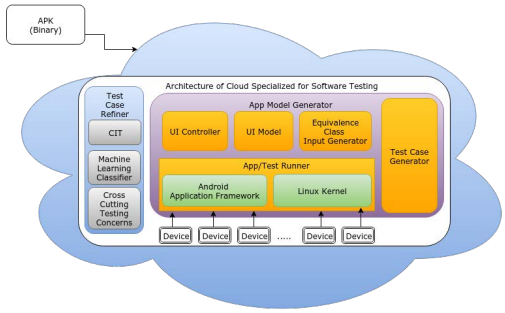

Figure 1: Architecture of the cloud testing system.

bug monitor api library. The android application framework includes required software packages and programming APIs to control a mobile device and/or an application running on the device. The programs and services from the linux kernel also enables interaction with the physical layer of the device for generating input and events.

Model Generator component is responsible for crawling the application via the device control system, exploring the application and generating the model. System states and transition functions compiled from input and performable action possibilities form up the component and the output is a formal finite state machine representation of the system. Input equivalence classes and concrete input set generation is handled according to the model/fsm information.

Test Case Refiner component utilizes combinatorial interaction testing (CIT) to analyse the input set for impossible or negligible input combinations to reduce the size of the test input space. The model and test input information aggregated are processed with machine learning techniques to infer domain and expert knowledge information about the classified applications, which will be used by the testing system to improve the testing capability.

We envision a system that performs dynamic analysis and testing of mobile applications. The dynamic nature of the system removes the source code requirement. The user uploads the binary into the system and the application is installed in real devices on the cloud.

2.1 Combinatorial Interaction Testing (CIT)

The test input combinations will be generated by covering arrays specialized for the system for identifying testing concerns. The output of application runs with these combinations will be shaping up the FSM model of the application. CIT methods will be employed to distinguish test input and test run combinations that are improving the test coverage.

2.2 FSM and CIT

The finite state machine and the information from combinatorial interaction testing will be the training input of the machine learning methods that will be used for classificatiob. The motivation behind employing machine learning is automating the generation of domain and expert information pool for applications that are classified in same group because of their GUI, execution and FSM model. The test cases and testing concerns that have been used on previous applications of a class will be directly used for new software-under-test for improving the testing success.

2.3 Cross Cutting Testing Concerns

Cross Cutting Testing Concerns is the term that we coined to address the special testing concerns that needs to be included in test cases according to the domain knowledge. Prioritization and injection of these concerns along with the regular testing artifacts for the actual functionality of the system is very similar to the well known aspect oriented programming concept of cross cutting concerns.

A very preventable, yet easily ignored defect generating aspect of activity mechanic in the Android devices is the destruction and restarting of applications when the screen orientation is changed, applications are switched or physical buttons like back button are pressed. The restarting behaviour causes the loss of system state. Our proposed system aims to utilize its domain knowledge, such as the previously mentioned property for activity restarts, to automatically insert test conditions into suitable locations in the running test flow, which is known to cause manifestation of such defects. The accumulated domain and expert knowledge pool from previously tested applications will be the basis for such precise interventions.

3. CONCLUSION

The mobile computing ecosystem still lacks full automation of mobile application testing and intelligent testing systems. Current testing practices relies heavily on the contribution and commitment of the development team to the testing process. However, the mobile ecosystem is still in its infancy stage and expert and domain knowledge is scarce. We believe that the cross cutting testing concern concept will be the key for complete test coverage.

4. REFERENCES

[1] Keynote. http://www.keynote.com.

[2] Perfecto mobile. http://www.perfectomobile.com.

[3] D. Amalfitano, N. Amatucci, A. R. Fasolino, P. Tramontana, E. Kowalczyk, and A. M. Memon. Exploiting the saturation effect in automatic random testing of android applications. In *Mobilesoft15 Conference Proceedings*, pages 33–43. ACM, May 2015.

[4] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. Memon. Using gui ripping for automated testing of android applications. In *ASE12 Conference Proceedings*, pages 258–261. IEEE/ACM, September 2012.

[5] Z. Liu, X. Gao, and X. Long. Adaptive random testing of mobile application. In *ICCET10 Conference Proceedings Volume 2*, pages V2–297 – V2–301. IEEE, April 2010.

[6] A. Machiry, A. Tahiliani, and N. Naik. Dynodroid: An input generation system for android apps. In *ESEC/FSE13 Conference Proceedings*, pages 224 – 234. ACM, August 2013.

[7] R. Mahmood, N. Mirzaei, and S. Malek. Evodroid: Segmented evolutionary testing of android apps. In *FSE14 Conference Proceedings*, pages 599–609. ACM, November 2014.

[8] S. Vilkomir, K. Marszalkowski, C. Perry, and S. Mahendrakar. Effectiveness of multi-device testing mobile applications. In *MobileSoft15 Conference Proceedings*, pages 44–47. ACM, May 2015.