2015 IEEE/ACM 37th IEEE International Conference on Software Engineering

A Combined Technique of GUI Ripping and Input Perturbation Testing for Android Apps

Gennaro Imparato

University of Naples "Federico II", Naples, Italy
genn.imparato@studenti.unina.it

***Abstract* — Mobile applications have become an integral part of the daily lives of millions of users, thus making necessary to ensure their security and reliability. Moreover the increasing number of mobile applications with rich Graphical User Interfaces (GUI) creates a growing need for automated techniques of GUI Testing for mobile applications.**

In this paper, the GUI Ripping Technique is combined with the Input Perturbation Testing to improve the quality of Android Application Testing. The proposed technique, based on a systematic and automatic exploration of the behavior of Android applications, creates a model of the explored GUI and then uses it to generate the perturbed text inputs. The technique was evaluated on many Android apps and its results were compared with random input tests.

***Index Terms* — Android Application Testing, GUI Ripping, Testing Automation, Input Perturbation Testing.**

I. Motivation

In the last years, mobile applications helped millions of users to connect to social networks, to search information on the web, to check emails, to edit documents and pictures, etc. as demonstrated by over 1 Million applications available in the Google Play Store [1].

Several mobile applications are written by developers with restricted time and resources budget, being often released under intense *time to market* pressure. The mobile applications must correctly handle a wide variety of system and user actions and the testing of these actions takes much time.

Unsurprisingly, the apps often react badly to unexpected actions, seriously degrading user experience [2]. Therefore it is necessary, in order to improve the security and reliability of mobile applications, to define and to apply new testing techniques specifically designed for them.

In this work two tools, called SlumDroid and GUIAnalyzer, are presented to support Android Application Testing (AAT). SlumDroid is a modified version of *AndroidRipper* [3] that implements the GUI Ripping Technique [4] and creates a model of the Application Under Testing (AuT). This model is used by GUIAnalyzer to produce the perturbed text inputs that will be used in the new test session.

II. Related work

Mobile Application Testing represents a challenging activity with several open issues. The Android SDK includes Monkey [5], a fuzz testing tool, that generates sequences of random events to apps on the Android Virtual Device (AVD).

Fuzz testing generates redundant events, thus it is not suited for generating inputs and events that require the human intelligence (e.g., playing and winning a game) [6] and cannot ensure effective failure detection.

An alternative iterative GUI testing approach was presented in [7] where the GUI Ripping Technique is applied to mobile applications. This approach is similar to approaches adopted for Web Crawling and it is more efficient and effective with respect to standard random testing tools.

The aim of this study is to improve the effectiveness of AAT combining the GUI Ripping Technique with the Input Perturbation Testing.

III. Approach Overview

The GUI Ripping Technique is implemented by a tool, called GUI Ripper, that constructs a navigation model of the application by dynamically interacting with its user interface and observing changes of the GUI state [4].

The main challenge of the GUI Rippers is providing specific inputs for text fields without instructions from the users [8]. Usually some human interventions, such as providing a valid username and password for a login screen, are needed to reach all parts of the AuT and achieve better values of the code coverage [9, 10].

In order to support the testing process, the SlumDroid prototype tool has been developed. SlumDroid is a configurable GUI Ripper, implemented in Java language, that is able to interact with user interface elements of Android apps and to emulate physical interactions (such as press Menu button, press Back button and orientation changes of the device).

It uses heuristics to determine if the displayed screen is similar to the previously visited GUI states in order to avoid redundant explorations.

SlumDroid uses the Robotium framework [11] to emulate the user interactions with Android devices and to extract the executable tasks which are stored in a task list.

SlumDroid adopts different traversal strategies (such as Breadth First, Depth First and Random First) to pick the next task to perform. The iterative ripping process proceeds until the task list is empty.

A separate tool, called GUIAnalyzer, was developed to perform Input Perturbation. It receives as input the GUITree (the GUI model of AuT) and screenshots captured during the ripping process.

978-1-4799-1934-5/15 $31.00 © 2015 IEEE
DOI 10.1109/ICSE.2015.241

ICSE 2015, Florence, Italy
ACM Student Research Competition

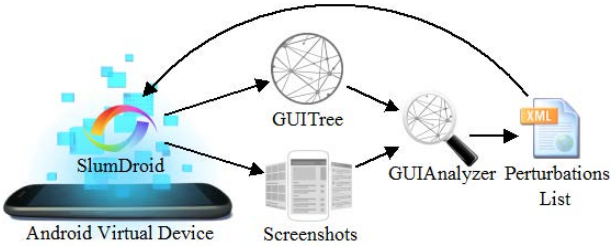

Fig. 1. Reference Schema

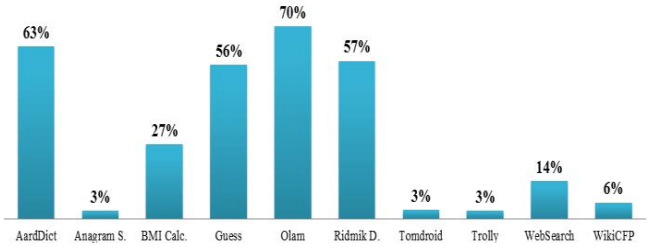

Fig. 2. Improvement of Code Coverage

GUIAnalyzer shows the properties of the text fields (such as the input text type) and the GUI image that contains them.

It generates text inputs by perturbing the regular expressions that are associated to each input text type. It performs three types of operations on regular expressions:

- remove the mandatory sets;
- disorder the sequence of sets;
- insert invalid and dangerous characters, such as an empty string or extremely long strings.

For example, an invalid email address can be a string without *"@"* or with some invalid characters, such as *"$"*, *"&"* and *"="* [12].

GUIAnalyzer output is a SlumDroid configuration file, in XML format, with the list of perturbable widgets and the associated perturbed inputs. This list will be used in a new test session. Fig. 1 shows reference schema of proposed technique. The overall process is semi-automatic. The manual intervention of a tester is only required for configuring the options, while the most expensive steps are performed automatically.

IV. Results

The benchmark-suite consists of ten applications downloaded from Google Play Store and F-Droid Repository. Their development projects are accessible through web-based hosting services that offer a bug report system.

The case study consisted in the execution of two test sessions that were sequentially executed on Genymotion [13], a fast and easy-to-use Android emulator. Each test begins from the same starting condition (app newly installed) and the AuT is restored to its initial state at each task. In the first one, SlumDroid inserts random numeric values into text fields.

TABLE I. Comparison Experiments

	Random Inputs			Perturbed Inputs		
Apps	**PLoC**	**Crashes**	**Bugs**	**PLoC**	**Crashes**	**Bugs**
AardDict	43%	1	1	70%	1	1
Anagram S	70%	0	0	72%	1	1
BMI Calc.	74%	0	0	94%	4	2
Guess	50%	0	0	78%	4	1
Olam	40%	0	0	68%	6	2
Ridmik D.	47%	0	0	74%	0	0
Tomdroid	64%	1	1	66%	8	3
Trolly	72%	0	0	74%	2	1
WebSearch	66%	1	1	75%	1	1
WikiCFP	85%	0	0	90%	3	2
	Total	3	3	Total	30	14

In the second test, SlumDroid uses the perturbations list provided by GUIAnalyzer. If during the ripping process, the current GUI contains more perturbable widgets, SlumDroid selects an editable text widget, inserts its valid input and combines the perturbed inputs for the other text fields. This operation is repeated for each perturbable widget of the layout.

The coverage metrics have been computed by Emma [14] and are reported in the PLoC (Percentage of Lines of Code Coverage) columns of Table 1.

For some apps, the PLoC is not close to 100% and there are several reasons for this. For example, applications may have dead code or code which executes only under special circumstances, such as specific device configurations and different operating system versions [15, 16].

By considering the perturbed inputs tests, there is an increase of the PLoC between 3% and 70% (see Fig. 2), with an average improvement of about 30%.

TABLE II. Bug Detection

Exceptions	**Apps**
IllegalArgumentException	AardDict, Tomdroid, WikiCFP.
IllegalStateException	WebSearch
NullPointerException	WikiCFP
NumberFormatException	BMI Calculator, Guess.
SQLiteException	Anagram Solver, Olam, Tomdroid, Trolly.
StringIndexOutOfBoundsException	Olam

Moving the focus on the bugs, with respect to random input tests five new exceptions (such as *IllegalArgument*, *NullPointer*, *NumberFormat*, *StringIndexOutOfBounds and SQLite*) were detected when using perturbed inputs.

Further bugs were detected during both test sessions of *AardDict*, *WebSearch* and *Tomdroid* apps. Such bugs occur when the device is rotated while an AsyncTask is running and a ProgressBar is shown.

Table 2 reports all fatal exceptions encountered during the test executions. Each bug was manually verified on real devices in order to eliminate any false positives and the new bugs were confirmed after being reported to developers.

Concluding, a novel approach for the AAT has been proposed, based on Input Perturbation. Tested on different apps, the methodology has shown, with respect to classical random test approach, to be particularly effective both in terms of code coverage and in bugs detection.

REFERENCES

[1] Android Operating System Statistics – AppBrain: http://www.appbrain.com/stats/stats-index

[2] G. Hu, X. Yuan, Y. Tang, and J. Yang. Efficiently, Effectively Detecting Mobile App Bugs with AppDoctor. *In Proceedings of the 9th European Conference on Computer Systems (EuroSys)*, April 13-16, 2014, Amsterdam, The Netherlands.

[3] D. Amalfitano, A.R. Fasolino, P. Tramontana, S. De Carmine, and G. Imparato. A Toolset for GUI Testing of Android Applications. *In Proceedings of the 28th IEEE International Conference on Software Maintenance* (ICSM), September 23-30, 2012, Riva del Garda, Trento, Italy.

[4] A. M. Memon, I. Banerjee, B. Nguyen, and B. Robbins. The First Decade of GUI Ripping: Extensions, Applications, and Broader Impacts. *In Proceedings of the 20th Working Conference on Reverse Engineering* (WCRE), October 14-17, 2013, Koblenz, Germany.

[5] UI/Application Exerciser Monkey | Android Developers: http://developer.android.com/tools/help/monkey.html

[6] A. Machiry, R. Tahiliani, and M. Naik. Dynodroid: An Input Generation System for Android Apps. *In Proceedings of the ACM Symposium on Foundations of Software Engineering* (FSE), August 18-26, 2013, Saint Petersburg, Russia.

[7] D. Amalfitano, A.R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon. Using GUI Ripping for Automated Testing of Android Applications. *In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (ASE), September 3-7, 2012, Essen, Germany.

[8] A. Kull. Automatic GUI Model Generation: State of the Art. *In Proceedings of the 2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops* (ISSREW), November 27-30, 2012, Dallas, Texas, USA.

[9] P. Aho, N. Menz, and T. Räty. Enhancing generated Java GUI models with valid test data. *In Proceedings of the 2011 IEEE Conference on Open Systems* (ICOS), September 25-28, 2011, Langkawi, Malaysia.

[10] T. Azim and I. Neamtiu. Targeted and Depth-First Exploration for Systematic Testing of Android Apps. *In Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications* (OOPSLA), October 29-31, 2013, Indianapolis, Indiana, USA.

[11] Robotium - The World's Leading Android™ Test Automation Framework, available at: http://code.google.com/p/robotium/

[12] N. Li, T. Xie, M. Jin, and C. Liu. Perturbation-based user-input-validation testing of web applications, *Journal of Systems and Software,* v.83 n.11, p.2263-2274, November, 2010.

[13] Genymotion, available at: https://www.genymotion.com/

[14] Emma: a Free Java Code Coverage Tool, available at: http://emma.sourceforge.net/

[15] V. Rastogi, Y. Chen, and W. Enck. AppsPlayground: Automatic Security Analysis of Smartphone Applications. *In Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy* (CODASPY), February 18-20, 2013, San Antonio, Texas, USA.

[16] S. Salva and S.Z. Zafimiharisoa. Model Reverse-engineering of Mobile Applications with Exploration Strategies. *In Proceedings of the 9th International Conference on Software Engineering Advances* (ICSEA), October 12-16, 2014, Nice, France.