Comparison and Analysis Between Automatic Exploration Tools for Android Applications

Author:
Michael OSORIO-RIAÑO

Advisor:
Mario
LINARES-VÁSQUEZ

A thesis submitted in fulfillment of the requirements
for the degree of Bachelor in Software and Computer Engineering
in

Systems and Computing Engineering Department

July 12, 2020

Abstract

Michael OSORIO-RIAÑO

Comparison and Analysis Between Automatic Exploration Tools for Android Applications

Acknowledgements

First, I want to express my deepest thanks to Professor Mario Linares-Vásquez for helping me with the development of this final work, giving me the necessary feedback for getting this project to this final version. Giving me new ideas and ways to solve the presented problems while executing this research.

Second, I would like to give my thanks to all the members of The Software Design Lab, for sharing with me their experiences and knowledge which were very important for developing this thesis. Especially to Camilo Escobar for his great help giving the main concept of InstruAPK, for helping with its implementation, besides giving me feedback about the figures in this text as well as solving some extra questions and doubts that I had during the process of developing this thesis.

Third, I want to say thanks to my mother and sister for keeping me motivated within all my major, till the last moment of it. For their unconditional support and for being there in the moments I needed them the most.

At last, but not least important, I want to say thanks to all my friends for sharing their knowledge with me and contributing with that to the final product of this thesis.

Without the help of the people mentioned, this work would not be possible.

I wish to clarify that the order in the mention does not reflect the level of thankfully I feel for the people mentioned in this statement. All of them supported this work in different ways, and under their capabilities, and helping me in one way or another to reach this results. For that reason, all of them deserves the same feelings from my. One more time, thanks to all of them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mobile applications market is a continuous growing market. According to Statista, the number of available application, by the 2020, in the two main markets together is 4407000. The great amount of applications means that there is a vast number of users willing to download them, but also means that they can change an application, or even the platform, whenever they desire to, or when something does not fulfil their needs or expectations. Therefore, every day more and more complex applications are launch in the market as well as the users becomes more demanding. As a result, every new application or new functionality should have a good quality, they should work as expected in all different scenarios the distinct users will put them in, they have to be developed very fast and also, they have to be cheap in order to be sustainable for the company who develops them.

The great amount, the complexity, and also the more demanding users make this commerce a very competitive one.

In order to reach users quality expectations, the frequently releases time of the market and the sustainability needed for the companies, many things have been developed, among them, automated testing. Automated testing has been of high interest for researchers and companies because it lowers the costs of production and it allows better quality products. One of the branches of automated testing is the automatic exploration tools, these tools aim to explore the application as deep as possible and find as many errors as possible. There is a plethora of exploration tools. Every year there are more of them, each one using different exploring strategies. Some of them use random inputs, others use image analysis, others a mixture of these two, and new researches are starting using AI. It is easy to think that the more deep exploration the more errors will be found, but, as will be shown in this text, that is not always the case. The number of errors detected can vary due to the exploration strategy because of the nature of the

errors. Furthermore, most of this exploration tools are developed for Android applications.

Examples of the automatic exploration tools are Monkey, this tool uses a semi-random generation events strategy to explore the applications, leading to different exploration results unless the same seed is given for the generation of the random events. This tool is the default one provided by Google; Firebase Test Lab, this tools is the only one which is online. Accordingly to its documentation, it analyzes the UI of the applications and explores it simulating users events. They claim to always explore the tool in the same order. Another exploration tool is RIP, an active project inside the University of Los Andes at the research group The Software Design Lab, it explores applications using a model-based GUI testing with multiple comparison criteria. It can lead to different exploration results due to its current implementation as well as to the comparison criteria.

For developers is important to know the difference between these tools. They need to know the tools that will match the best in the new incoming project to make a good decision. Besides, the project bugged, and application complexity among others things can also affect the decision. The information available to make the decision is most of the time based in what the tools claim to do and if the developers have enough working with more than one then they have some extra information. Though, this information is not completely reliable, it is not objective and can be slanted.

Additionally, for researchers is also important to know what is the advantages of all the different exploration strategies, as well as their disadvantages. They need to compare them to know what is the next step to make the field go further.

1.1 Problem Statement

Accordingly to the previous section, the number of automatic exploration tools is increasing annually. Thus, with no objective information about them, developers will need to explore new tools and compare them to find the best one. It can turns easily in spending valuable time and at the end in making the wrong decision, resulting in poor quality final products. In short, the decision of the right automatic exploration tool should be easy, quickly and rely in objective data, such as coverage reached, and number of errors found.

1.2 Thesis Goals

The main objective of this thesis, is to provide quantitative and qualitative information of the most widely used automatic exploration tools for Android mobile applications, to facilitate developers in the selection of the right tool that suits their needs. Under those circumstances, the next specific objectives were proposed.

1. Compare the tools by their exploration coverage
2. Compare the tools by the number of unique error traces discovered while exploring an application.
3. Compare the tools using qualitative aspects such as, is the tool a open source project? Is the tool free? Is the tool allowing introduce login values? how useful is the tool report for developer to reproduce, find and fix bugs?

1.3 Thesis contribution

The main contribution of this thesis is to provide developers with enough and objective information, to decide which automatic exploration tool suits their projects' needs the most, making this process easy and less time consuming.

Furthermore, even when the study does not have the objective of create a reproducible work flow, it was created and is

// TODO aquí

1.4 Document Structure

//TODO aquí

This document has the following structure: In Chapter 2 the

Chapter 2

Related work

Every exploration tool provides different numbers, and comparisons made during their creation, this in order to show their advantages, but not for sure their weaknesses. This information does not allow developers neither researchers to know what are the best tools as of now or how good a tool matches their projects.

Shauvik Roy Choudhary, et al., give about the strengths and weaknesses of seven tools in their study "Automated Test Input Generation for Android: Are We There Yet?". They evaluate the tools using four metrics i. ease of use, ii. ability to work on multiple platforms, iii. code coverage, and iv. ability to detect faults. 14 Tools and 68 applications were used in total in their study. Running 10 times every application in seven of the 14 tools for a maximum of 60 minutes.

Moreover, different tools have been developed since Shauvik Roy Choudhary, et al. study was made. One example is Firebase Test Lab, which started to allowing the automatic testing in the cloud. This type of application was not taken into account.

Chapter 3

Solution Design

3.1 General Approach

With the aim of complete the objectives mentioned in Sec.1.2, a workflow was designed. This work flow contains five stages.

1. Instrumentation of the applications
2. Exploration
3. Coverage measurement
4. Summarize data
5. Data Analysis

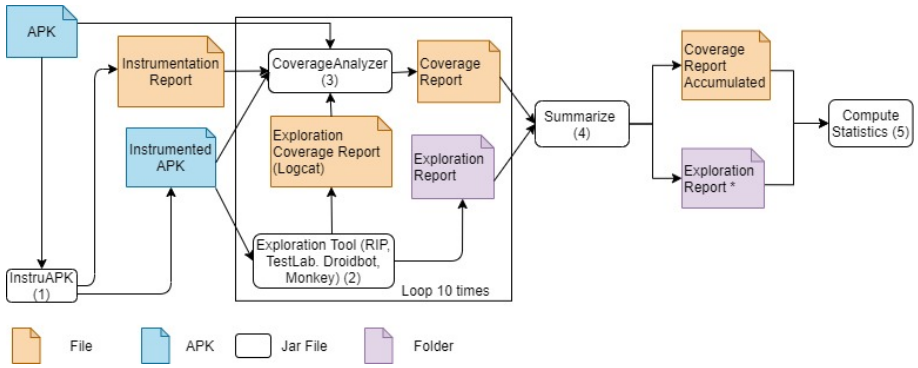

FIGURE 3.1: Main Workflow

For the first stage, **InstruAPK** 3.2 was used to make the instrumentation of the applications. This tool only takes into account the methods under the package name of the application that is being analysed. As a result, methods from different libraries are not being taken into count. The input for this stage is the original APK file, and the output is the instrumented APK together with the instrumentation report containing general information such as the file path, method's name,

file name, and the method arguments, as well as a sequential number that will help us to know the total amount of instrumented methods and will work as their unique identifier.

The exploration was made by four different exploration tools, i. Droidbot, ii. Monkey, iii. Firebase Test Lab, and iv. RIP. Every tool was run to explore with a maximum time of 30 minutes. It is important to notice that, even when the max execution time seems to be short, it was enough for most of the application. This was because of the size of the applications, if the application is small, then the coverage will increase rapidly, because a bug was found during the exploration, or even because the tool marks the exploration as done.

This stage input is only the instrumented APK file, and its output is the exploration report that every tool provides. I took the logcat from the exploration report, but when it was not available in it, as in the case of Monkey, it was extracted by using the adb shell command. The other three tools got the information at the end of their exploration.

Some of these tools, Droidbot, Monkey and Test Lab, were selected because of their high use in the industry, and the remaining one was selected because of personal interest owing to It is an active project at the University of Los Andes inside the research group The Software Design Lab which I am part of.

In stage 3, **CoverageAnalyzer (CA)** 3.3 was used to make the coverage measurement and search for error lines. This stage inputs are the original APK, the instrumentation report and the instrumented APK, both from stage 1, and the logcat from stage 2. Its output is a report containing two method coverage measurements, the first one, calculated using the number of methods reported by APK-Analyzer, a tool provided for Google, and the second one with the number of instrumented methods reported by InstruAPK.

Stages 2. and 3. were repeated 10 times for every application that was selected, that leaded to the stage 4. The multiple executions are intending to get average values as well as comparable results along the different exploration tools. The input for this stage are all the method coverage reports from of the stage 3 as well as the exploration report from stage 2. The output are the accumulated method coverage by each tool for each application, which was calculated taking the unique methods called over all the ten executions, the average accumulated method coverage over time, as well as the number of errors and its average found per application.

The final stage encompasses data understanding, graphs creation, comparison using the graph and analysis of different qualitative aspects of every exploration tool. Thus,

3.2 InstruAPK

This tool was developed for this study. It uses APKTool, a known Java application that allows inverse engineering in Android apps, allowing applications' instrumentation without the need of recompiling their source code. APKTool decodes the APK file and its result is the smali representation of the app source code, These smali files are analysed in order to find all the methods to be instrumented at the very beginning of each method. It is important to notice that no external libraries methods are instrumented. InstruAPK only search for methods following the android project structure that uses the application package name to store the application source code.

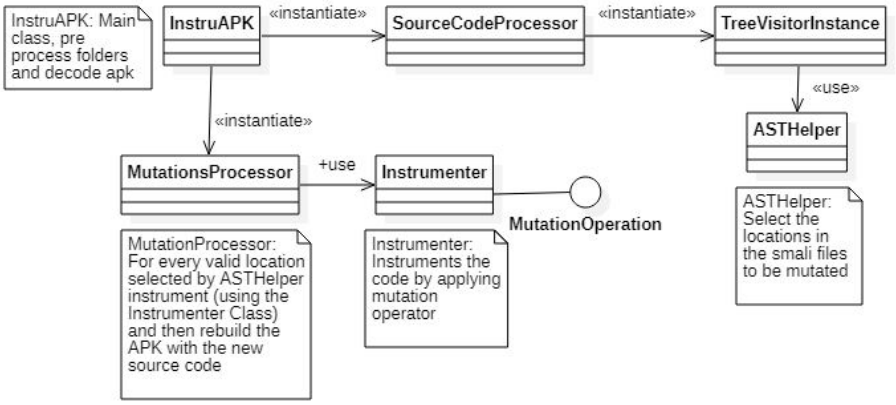

FIGURE 3.2: Class Diagram InstruAPK

Figure 3.2 only contains the main classes of the tool and offers a short explanation of what is doing every one to understand it in more detail.

3.3 Coverage Analyser (CA)

This tool is a Java Application created for this study. It analyses the resulted logcat file after an exploration. It searches for the log lines injected by InstruAPK, as well as for errors, filtering the results using the package name of the application under the analysis. For the coverage measurement, the tool uses the number of methods reported by APKAnalyzer as well as the number of methods reported by InstruAPK, resulting in two different method coverage values. As mentioned

before, CA searches for the log lines injected by InstruAPK making CA depend on it. For that reason, CA can be seen as a complement of InstruAPK, rather than a separate application.

Figure 3.3 contains the main classes of CoverageAnalyzer besides a short explanation of its functionality.

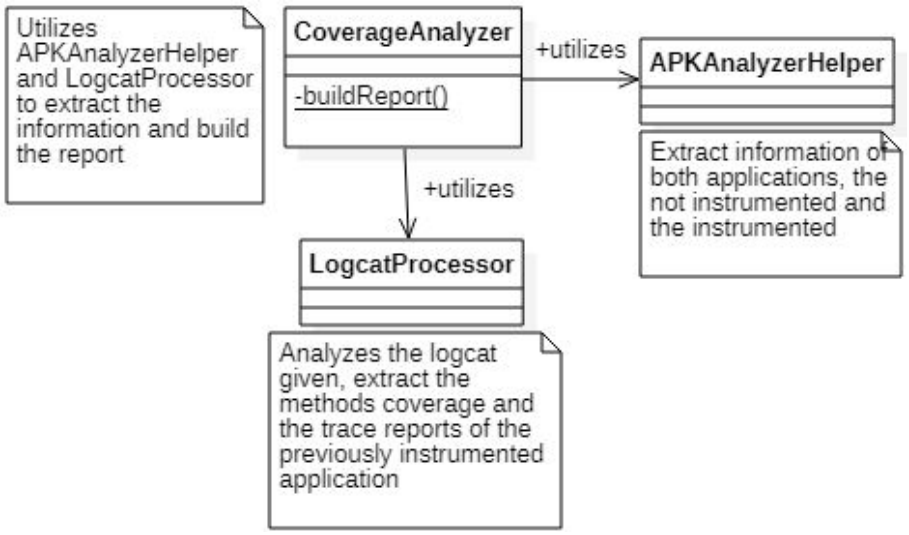

FIGURE 3.3: Class Diagram Coverage Analyser

These two tools were the main basis of this study, but its further review its leave for previous studies.

Certainly, all the stages were necessary in order to complete every objective. The stages design was made regarding the specific objectives and as a result achieving the general one.

Any person who desires to compare different exploration tools, can reproduce this work flow. Even can make use of the same tools for the instrumentation and the coverage measurement, allowing easy and fast comparisons. Consequently, the decision-making starts to be easier for developers and researchers. Also,gives the possibility to researchers of compare their own exploration tools in a effortless and quick way.

Chapter 4

Empirical Study

4.1 Study Design

As expected, para cumplir con el objetivo general de esta tesis se deben cumplir con los objetivos especificos, una vez estos sean completados a cabalidad entonces se tiene el objetivos general completado.

Explicar por qué se seleccionaron las herramientas (RIP, TestLab, etc) Explicar las apps utilizadas en el estudio.

La idea es que estos objetivos especificos lleven o ayuden a llegar al cumplimiento de este objetivo general.

Se debe explicar cómo se cumplió con cada uno de ellos y al final explicar cómo se llegó a cumplir con el objetivo general

mostrar los resultados y analizarlos.

//TODO two from the industry and two from the academic side. The first tool was Firebase Test Lab. it was selected for being widely used in industry and for also being a Google product. The second one, Monkey, was selected for being the most basic one and because it is also included in the SDK for developing Android Apps. The third one, Droidbot, was selected from the academic side. Droidbot has been a point of study for many researches. Many others tools have based their functionality on this tool. The last one is RIP, this tool was selected for being of special interest for us. It is our own exploration tool and is is currently an active project inside the Software Design Lab at University of Los Andes.

Every tool was executed ten times per application, and every execution with a maximum time of 30 minutes. Some tools ended its exploration before the max time. The number of executions and the maximum time were arbitrary decisions that were made because of time limitations for the study. Although, during the

TABLE 4.1: Applications used for the study

App ID	Package Name	# Methods Reported by APKAnalyzer	# Methods Instrumented by InstruAPK
1	appinventor.ai_nels0n0s0ri0.MiRutina	61993	9351
2	com.evancharlton.mileage	4000	1162
3	com.fsck.k9	18799	7003
4	com.ichi2.anki	32370	2209
5	com.workingagenda.devinettes	19274	66
6	de.vanitasvitae.enigmandroid	13083	574
7	info.guardianproject.ripple	19429	100
8	org.connectbot	20606	1145
9	org.gnucash.android	75473	504
10	org.libreoffice.impressremote	14691	649
11	org.lumicall.android	45784	540

study was notice that most of the tools ended the exploration or reached their maximum coverage within the first 15 minutes. Which means that the maximum time for exploration was more than enough in almost all cases.

//TODO Besides that, for this study, a set of 11 applications was used. This set is a subset of a set of open source applications utilised inside The Software Design Lab research group for other studies and tests, including RIP. Every APK in the subset should be successfully instrumented by InstruAPK, it should compile without any problem after instrumentation and it should be launch in an emulator without any issue after instrumentation.

4.2 Context of the Study

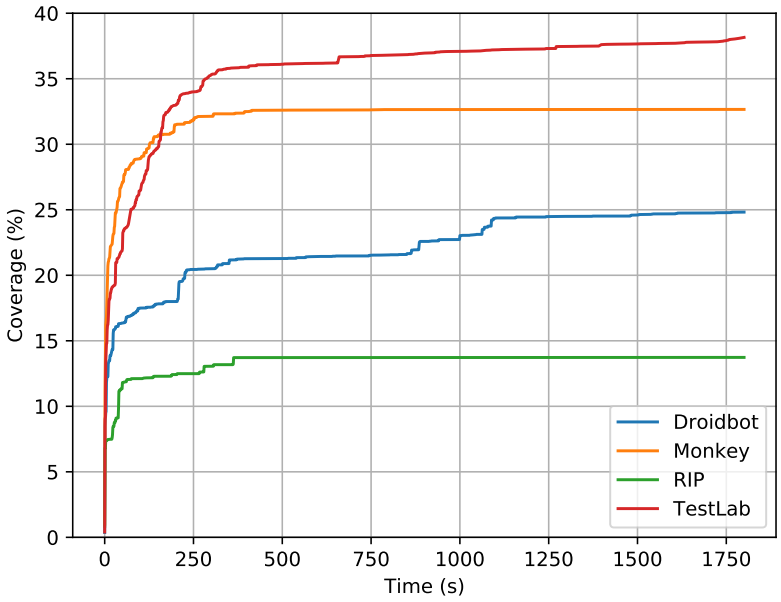

FIGURE 4.1: Average Method Coverage by Tool

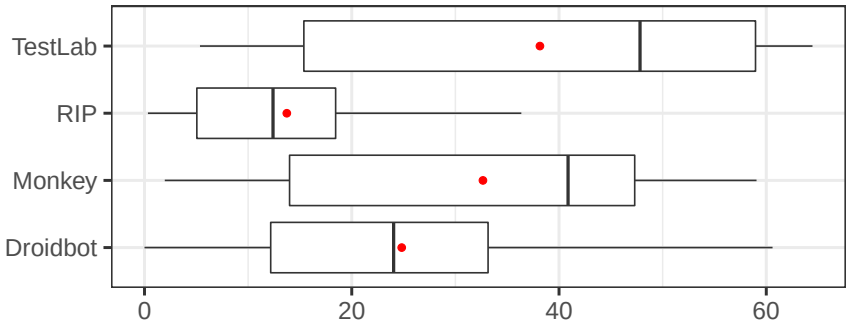

FIGURE 4.2: Boxplot of Accumulated Coverage by Tool

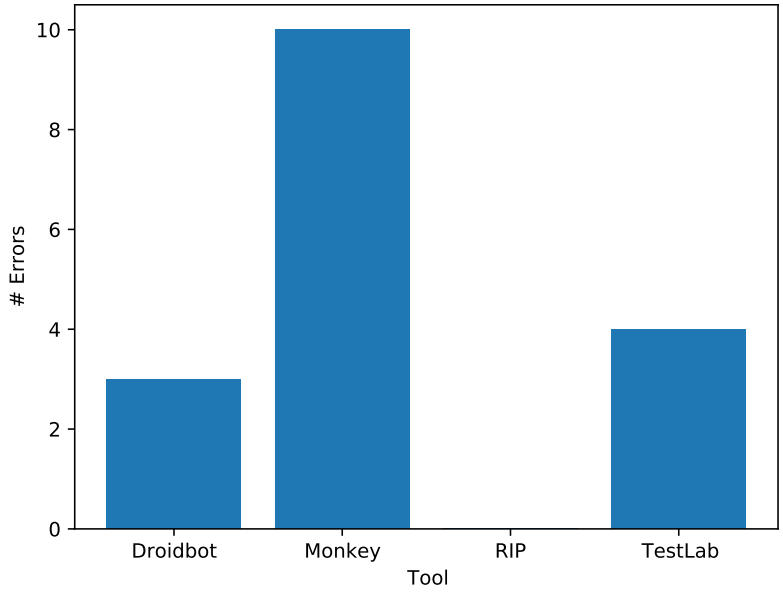

FIGURE 4.3: Maximum Number of Errors Found by Tool

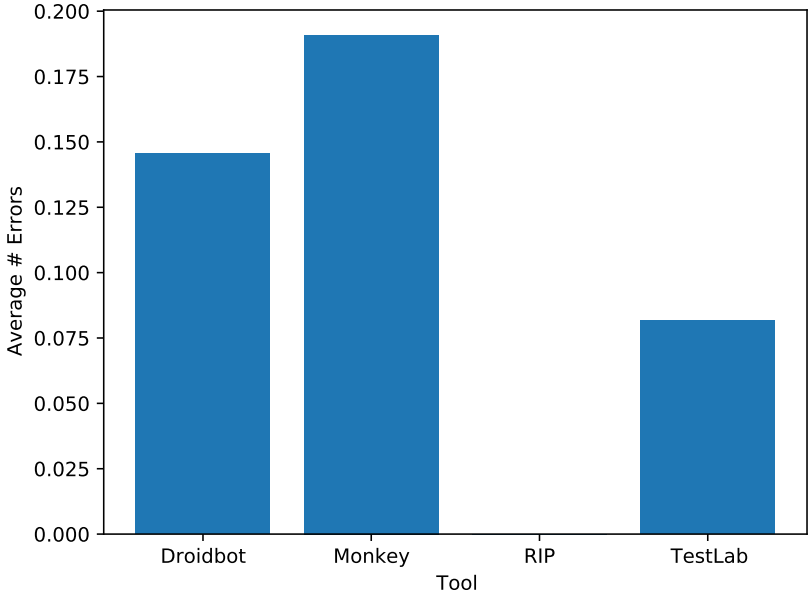

FIGURE 4.4: Average Number of Errors Found by Tool

Chapter 5

Conclusion

Chapter 6

Future Work