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

THE SW DESIGN LAB

Systems and Computing Engineering Department

July 22, 2020

Abstract

Michael OSORIO-RIAÑO

Comparison and Analysis Between Automatic Exploration Tools for Android Applications

The number of different tools to explore Android applications has been increasing. Every tool has a different exploration strategy and clam to offer different benefits than others. The huge amount of tools and the lack of impartial information about them makes that developers and researchers have no basis and data to face a decision-making situation or data to compare their own new tools. Others studies have made different comparisons between exploration tools in the past, but most of those tools are no longer being used in the industry or in the academy, that is why there is a need of studies providing clear and unbiased information about the newest tools that allows the developers and researchers to acquire a better perspective of the modern exploration tools. That is the reason why in this study, four of the most used tools for automatic exploration of Android applications are analysed and compared according their progressive and achieved method coverage, and the max number of errors found in one exploration. Besides, a reproducible workflow is proposed for future studies of the same type as well as two tools for allowing faster and easier comparison are described.

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, and 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 that 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 this thesis.

Without the help of the mentioned people, 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,. Helping me in one way or another to reach this results. For that reason, all of them deserve the same feelings from my. One more time, thanks to all of them.

Contents

Abstract iii

Acknowledgements v

List of Figures ix

List of Tables xi

1 Introduction 1
- 1.1 Problem Statement . . . 2
- 1.2 Thesis Goals . . . 3
- 1.3 Thesis contribution . . . 3
- 1.4 Document Structure . . . 3

2 Related work 5
- 2.1 Crawldroid . . . 5
- 2.2 Droidbot . . . 6
- 2.3 Firebase Test Lab . . . 6
- 2.4 RIP . . . 6

3 Solution Design 9
- 3.1 General Approach . . . 9
- 3.2 InstruAPK . . . 11
- 3.3 CoverageAnalyser (CA) . . . 11

4 Empirical Study 13
- 4.1 Study Design . . . 13
- 4.2 Context of the Study . . . 13
- 4.3 Results . . . 15
  - 4.3.1 Method Coverage Results . . . 15
  - 4.3.2 Error Results . . . 17

5 Conclusions and Future Work 21

List of Figures

3.1 Main Workflow 9
3.2 Class Diagram InstruAPK 11
3.3 Class Diagram Coverage Analyser 12

4.1 Average Progressive Method Coverage by Tool According to the Number of Instrumented Methods Reported by InstruAPK 16
4.2 Average Progressive Method Coverage by Tool According to the Number of Methods Reported by APKAnalyzer 17
4.3 Coverage Achieved by Tool 17
4.4 Maximum Number of Errors Found by Tool in One Exploration . . 18
4.5 Average Number of Errors Found by Tool During all the explorations 19

List of Tables

4.1 Applications used for the study 14

Chapter 1

Introduction

Mobile applications market is a continuously growing market. According to Statista[1], the number of available applications, by 2020, in the two main markets together is 4407000. A large amount of applications means that there is a vast number of users willing to download them, but also it 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 released in the market, as well as the users, become 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.

A large amount of publicly available, their complexity, and also the more demanding users make this market a very competitive one.

In order to reach users quality expectations, improve release times and the sustainability needed for the companies, many approaches have been promoted, 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 automatic exploration tools; these tools aim to explore applications as deep as possible and find as many errors as possible. There are a plethora of exploration tools. Every year there are more of them, each one using different exploration strategies. Some of them use random inputs, others use image analysis, others use a mixture of these two, and new researches are starting using AI. It is easy to think that the deeper the exploration the more errors will be found, however, as will be shown in this text, that is not always the case. The number of errors detected

[1]Source visited on June 17, 2020. link

can vary due to the exploration strategy because of the nature of the errors and the apps. Furthermore, most of the exploration tools are developed for Android applications.

Examples of the automatic exploration tools are Monkey [2]. This tool uses a pseudo-random generation events strategy, leading to different exploration results unless the same seed is given for the generation of random events. This tool is the default one provided by Google. Another well-known tool is Firebase Test Lab [3]; this tool can be used on-line. Accordingly to its documentation, it analyses the UI of the applications and explores the apps by simulating users events. They claim to always explore the tool in the same order. One more tool is RIP [4], an active project from The Software Design Lab at *University of Los Andes*; RIP explores applications using a model-based GUI testing technique. It can lead to different exploration results due to its current implementation and comparison criteria.

For developers, it is important to know the difference between these tools. They need to know the tools that will work the best in the new incoming project to make a better decision. Besides, the project budget, and application complexity among other things can also affect the decision. The information available to make the decision is often based on what the tools claim to do. This information is not completely reliable.

Additionally, for researchers, it 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 realize 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 turn easily in spending valuable time and at the end in making the wrong decision, resulting in final products with poor quality. In short, the choice of the right automatic exploration tool should be easy and it should rely in objective data, such as coverage reached, and the number of errors found.

[2] https://developer.android.com/studio/test/monkey
[3] https://firebase.google.com/

1.2 Thesis Goals

The main objective of this thesis is to provide quantitative 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. Thus, the following specific objectives were proposed:

1. Compare exploration tools based on their method coverage capabilities
2. Compare exploration tool based on the number of unique error traces discovered while exploring an application.

1.3 Thesis contribution

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

Alongside, this study shows that higher coverage does not guaranty a higher number of found errors, which, as demonstrated later in this work, is the case of Firebase Test Lab. This tool had the top method coverage both on average and achieved, but it is not the best one when regarding errors found.

All this information provides a better point of view to developers and researchers, giving fixed and objective comparison points that they can use in a decision-making situation, resulting in better projects.

Furthermore, even when the study does not have the objective of creating a reproducible workflow, it is created. New researchers can use such workflow as a basis for new comparison studies, as well as extend it and enhance it to create a new standard for the validation of new exploration tools for Android applications. Plus, two tools for helping the validation of new exploration tools were designed, InstruAPK (section 3.2) and CoverageAnalyzer (section 3.3). Such tools also can be improved, adding more features and gaining more precise information from the explorations.

1.4 Document Structure

exploration tools for android applications as well as information about researches that have already made comparisons between some tools. Next chapter, Chapter 3 describe the solution design which includes, the general approach (Section 3.1)

to get done the objectives described in section 1.2, besides a brief explanation of the tools created for this study (Sections 3.2 and 3.3). Now, in chapter 4 you will find the empirical study, here is clarify how the data were obtained, what tools and applications were used, the devices involved in the study and other important data to reproduce the research. Additionally, in the same chapter, you will find the results of the study together with their reasoning. Finally, chapter 5 describes conclusions using the data obtained in the previous chapter, as well as future work depicting new researches that could be made when taking this study as a base or as a reference.

Chapter 2

Related work

Every exploration tool provides different numbers, and comparisons made during their creation, this 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. S.R. Choundhary *et al* [2] gave information about the strengths and weaknesses of seven tools in their study *"Automated Test Input Generation for Android: Are We There Yet¿'*. They evaluated 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 [2] study was made.

2.1 Crawldroid

Crawldroid [1] uses a model-based GUI testing technique. Its purpose is to avoid local and repetitive exploration, by grouping widgets and then adjust the groups' priority depending on previous steps and the results of the widgets already actioned.

This tool makes part of a study named *"CrawlDroid: Effective Model-based GUI Testing of Android Apps"*where the authors made a comparison between this tool and some others to know how good is their tool. Using a tool called ELLA [1] for its coverage measurement. ELLA provides information about method and activity coverage.

[1] https://github.com/saswatanand/ella

Crawldroid was not used in this study because it was not possible to make it work. The setup of the tool was not possible. The were tries to contact with the authors but no answer was received from them.

2.2 Droidbot

Droidbot [3] is a lightweight exploration tool that does not need instrumentation in order to work. Droidbot is able to generate UI-guided test inputs using a state transition model that is generated while exploring the application. It is an open-source project.

This tool is also an open-source project and, it makes part of a study, where their authors explain that the tool can calculate different metrics by using the Android official profiling tool. Their approach can follow the trace of the methods that are triggered when a new widget is clicked or when an event is prompt. Besides, if the number of methods is available they will also calculate the coverage reached in the exploration.

2.3 Firebase Test Lab

Firebase Test Lab [2], is a Google product. This tool is different to the others because it does not make part of a research and also because it is available in the cloud, the users should create an account on it, create a project, upload the APK and the run the test. This is the unique exploration tool that is run in this way. Different from the other tools mentioned in this chapter, this is the only one that has a paid version. It reports no coverage values, but the number of visited activities.

2.4 RIP

RIP [4] is an active project in charge The Software Design Lab at The University of Los Andes. This tool is designed to take into account multiple variables during the exploration. Variables such as the context of the application while exploring an activity, the GUI elements presented in one activity. The purpose of this extra information is to provide better quality testing.

This tool is also part of research named *"Automated Extraction of Augmented Models for Android Apps"*. where their authors claim that all this new information is

[2]https://firebase.google.com/

necessary owing to the complexity of mobile applications. Thus, when more information is recorded and fewer variables are unknown, bugs reproduction will be easier.

RIP does not contain any coverage metrics by default. So, its coverage should be calculated by using a different tool. It only provides information about what activities visited, what events triggered to get from one state to another, and some other extra information later discussed.

As can be seen, non of the aforementioned tools were discussed in S.R. Choudhary *et al* [2] work. Most probably because by the time they did the research, none of them existed. That creates a gap between the available information for developers and researchers and the current automatic exploration tools. As a result, this study will provide information about some of the newest tools that are in use within the industry and the academy. The study will give a better overview of nowadays automatic exploration tools.

Chapter 3

Solution Design

3.1 General Approach

The objectives mentioned in section Sec.1.2 were achieved by following the workflow shown below in figure 3.1. This workflow contains five stages: (1) Instrumentation of the applications, (2) Exploration, (3) Coverage measurement, (4) Summarize data, and (5) Data Analysis.

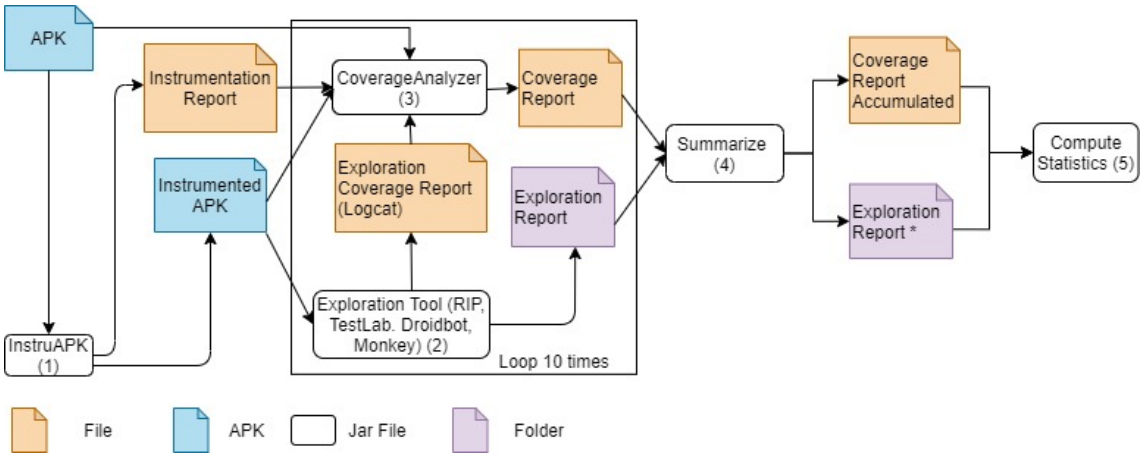

FIGURE 3.1: Main Workflow

For the first stage, **InstruAPK** (Section 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 instrumented. The input for this stage is the original APK file. 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.

In the stage number two, the exploration was made with four different exploration tools, (1) Droidbot 2.2, (2) Monkey [1], (3) Firebase Test Lab 2.3, and (4) RIP 2.4. Every tool was executed to explore apps 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 analysed applications. 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. Most of the time, the report contains the device logcat after the exploration. In the case of Monkey, it was extracted by using an adb command [2] because it is not extracted by the tool itself.

Droidbot, Monkey and Firebase Test Lab were selected because of their high use in the industry, and RIP was selected because it is an active project from The Software Design Lab.

In stage 3, **CoverageAnalyzer (CA)** (Section 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. The output is a report containing two method coverage measurements, the first one, calculated using the number of methods reported by APKAnalyzer [3], a tool provided in the Android SDK Tools, and the second one with the number of instrumented methods reported by InstruAPK.

Stages 2. and 3. were repeated ten times for every application that was selected. The multiple executions are intending to get average values as well as comparable results along with the different exploration tools. On top of that, input for stage 4 is all the method coverage reports form stage 3 as well as the exploration report from stage 2. The output is 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, i.e., Stage 5, encompasses data understanding, graphs creation, a comparison using the graph and analysis of different qualitative aspects of every exploration tool. Providing the figures presented in chapter 4, as well as leading

[1] https://developer.android.com/studio/test/monkey
[2] https://developer.android.com/studio/command-line/logcat
[3] https://developer.android.com/studio/command-line/apkanalyzer

to the conclusions in chapter 5 and helping to fulfil reach the objectives proposed at the beginning of this document.

3.2 InstruAPK

This tool [4] was developed for this study. It uses APKTool, a known Java application that allows inverse engineering of Android apps, allowing applications' instrumentation without the need of recompiling their source code. APKTool decodes the APK file and its result is the smali [5] 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 searches for methods following the android project structure that uses the application package name to store the application source code.

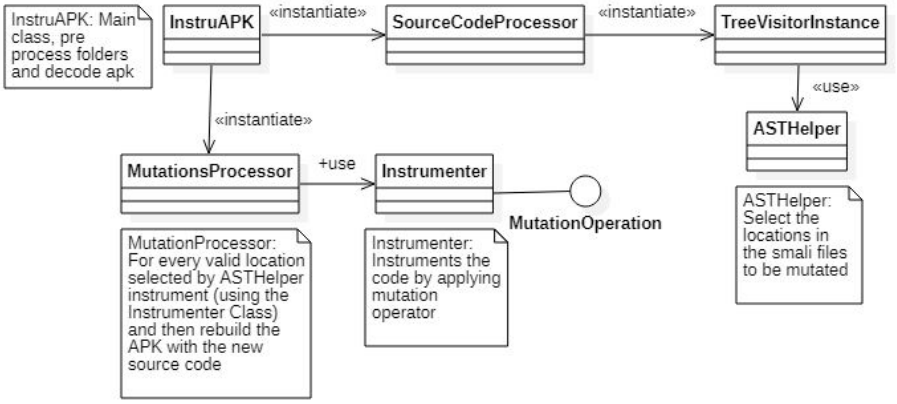

FIGURE 3.2: Class Diagram InstruAPK

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

3.3 CoverageAnalyser (CA)

This tool [6] 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

[4] https://github.com/TheSoftwareDesignLab/InstruAPK
[5] https://github.com/JesusFreke/smali/wiki
[6] https://github.com/TheSoftwareDesignLab/CoverageAnalyzer

methods reported by APKAnalyzer [7] 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.

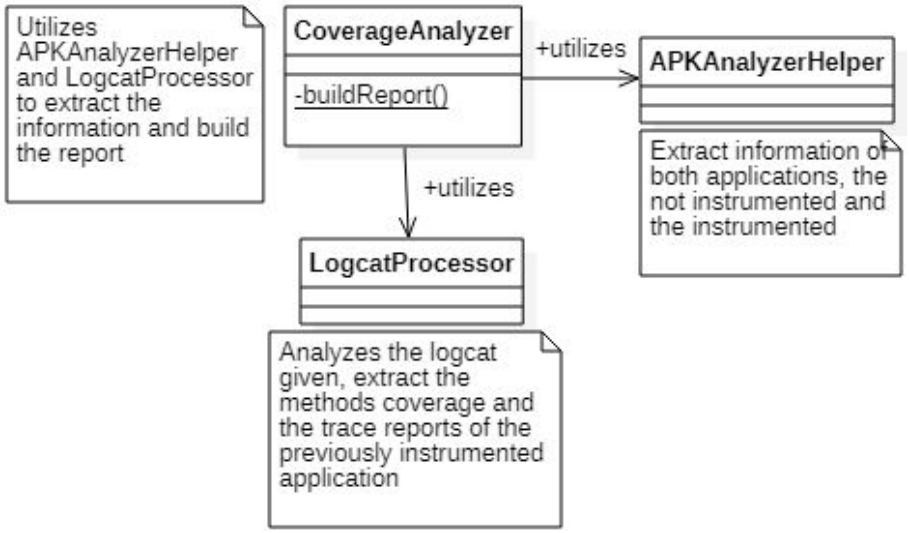

FIGURE 3.3: Class Diagram Coverage Analyser

These two tools are the main basis of this study, but its further review is left for previous studies. Both tools are now open source projects that can be found on Github.

Any person who desires to compare different exploration tools can reproduce this workflow. Even can make use of the same tools for the instrumentation and the coverage measurement, allowing easy and fast comparisons. Consequently, decision-making starts to be easier for developers and researchers. Also, gives the possibility to researchers to compare their exploration tools effortlessly and quickly.

[7] https://developer.android.com/studio/command-line/apkanalyzer

Chapter 4

Empirical Study

4.1 Study Design

This paper aims to provide information about some of the most used automatic exploration tools for android applications inside the industry and the academy. This information is going to be useful for developers and researchers when they face a decision-making situation related to the selection of the right exploration tool that fits their needs. In consequence, an empirical study was designed and will provide answers for the following research questions:

- RQ-1 What tool reaches the highest accumulated method coverage?
- RQ-2 What tool finds the largest number of failures during one exploration?
- RQ-3 what tool has the top average value of errors found across different apps during all the explorations?

4.2 Context of the Study

To answer the research questions, 11 applications were selected to be executed. The list of selected apps can be seen in Figure.4.1. This set is a subset of a set of open source applications widely used by The Software Design Lab for other studies and tests. Every APK in the subset should be successfully instrumented by InstruAPK, it should compile without any problem after instrumentation and it should be launched in an emulator without any issue after the instrumentation process.

Equally important, four exploration tools were selected, two from the industry and two from the academic side. The first tool is Firebase Test Lab (Section 2.3). It was selected for being widely used in industry and for also being a Google

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

product. The second one, Monkey [1], was selected as a baseline because it is the state-of-the-art and practice tool for inputs generation; it is by default included in the Android SDK Tools. The third one, Droidbot (Section 2.2), was selected from the academic side. Droidbot has been a point of study for many pieces of research. Many other tools have based their functionality on this tool. The last one is RIP (Section 2.4). It was selected for being of special interest for us. It is our exploration tool, and it is currently an active project inside the Software Design Lab at the University of Los Andes.

Each tool was executed ten times per application and with a maximum time of 30 minutes. The number of executions and the maximum time was arbitrary decisions that were made because of time limitations for the study. During the study, we noticed that most of the tools ended the exploration or reached their maximum coverage within the first 15 minutes. That means that the maximum exploration time was more than enough in almost all cases.

The same emulator was used for all the exploration tools, except for Firebase Test Lab because this tool offers its own set of emulators. In all the cases a Pixel 2 XL was used, but in the case of Monkey, Droidbot and RIP there was more control over the device specifications. For the last-mentioned tools, the specifications of the emulator used within the experiment are Google APIs Intel Atom (x86), API level 27, SD card size of 512MB, and RAM size of 4096MB. The specifications are unknown for the case of Test Lab.

Finally, the workflow specified in section 3.1 was followed for each application in table 4.1 by using the aforementioned exploration tools.

[1] https://developer.android.com/studio/test/monkey

4.3 Results

4.3.1 Method Coverage Results

As mentioned before, some explorations ended before the max execution time allowed (30 minutes) giving no data for the upcoming seconds. To solve those scenarios, the coverage reached by the tool at the moment when the exploration end was kept the same until completing the total time. Once this has been done, the results are comparable second by second.

As a result of the instrumentation made by InstruAPK, the timestamp of every called method is known, thus, the coverage reached by a tool in a specific second can be calculated. The aforementioned information was used to calculate the average accumulated method coverage reached for every tool across the 11 analysed apps. Such results are presented in figure 4.1 and figure 4.2. The data were calculated as follow:

Let's take A as the set of applications; N as the number of explorations; t as time in seconds (for this study $0 < t < 1800$, and T as the exploration tool, then we have:

$$g(a_{T,t}) = \frac{\sum_{i=1}^{n} Coverage(T,t,a)}{N} \tag{4.1}$$

Where $Coverage(T,t,a)$ is the instantaneous coverage of tool T in second t for the application a. Which means, the new methods called by tool T in second t.

$$f(T_t) = \frac{\sum_{a \in A} g(a_{T,t})}{|A|} \tag{4.2}$$

$$H(T_t) = \begin{cases} H(T_{t-1}) & \text{if the exploration has end,} \\ f(T_t) + H(T_{t-1}) & \text{if still exploring,} \\ f(T_0) & \text{if } t = 0. \end{cases} \tag{4.3}$$

Thus, equation 4.1 is the average coverage of tool T for application a in time t for the N executions; equation 4.2 is the average coverage of tool T in time t across the N executions and all the apps; and, 4.3 is what is called *accumulative average coverage* of tool T in time t across all apps and the N executions.

Thus, equation 4.3 is what is shown in figures 4.1 and 4.2. For coverage in figure 4.1, equation 4.1 was calculated using the number of instrumented methods by InstruAPK and for figure 4.2 the number of methods reported by APKAnalyzer. The curves have the same behaviour, as expected, but in Figure 4.2 gap between tools is more visible.

Additionally, for every execution the called methods are stored, every method has a unique id that was given during the instrumentation. Thus, after the 10 executions of an application, every execution result could be analysed searching for the methods that have been called neither during previous explorations nor during the current one. Therefore, the number of unique methods called for one application during all 10 explorations can be obtained. That is what is named as *achieved coverage* and can be seen in figure 4.3. The coverage presented in figure 4.3 was calculated only by using the number of instrumented methods by InstruAPK.

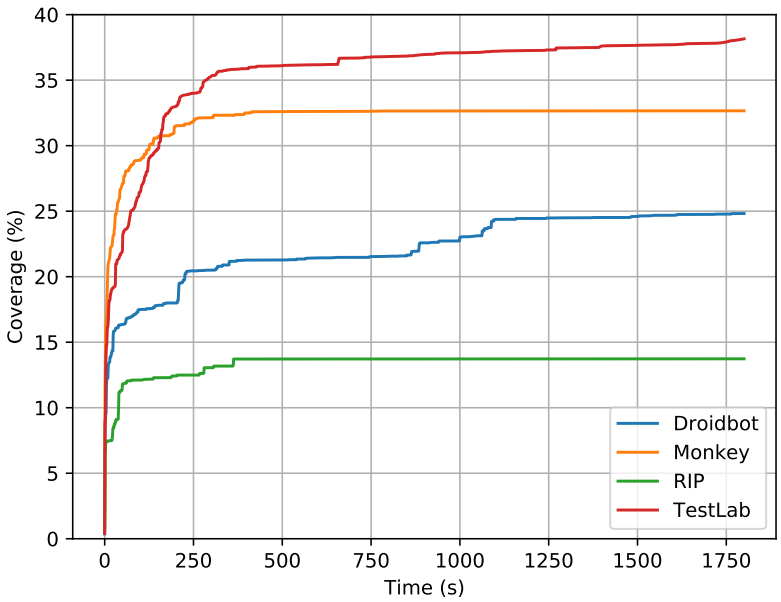

FIGURE 4.1: Accumulated Average Method Coverage by Tool According to the Number of Instrumented Methods Reported by InstruAPK

Hence, RQ-1 was responded using graphs 4.1, 4.2, and 4.3. Firebase Test Lab is the tool with the highest progressive average method coverage reached. Surprisingly, followed by Monkey, even when Monkey has no complex architecture nor exploration strategy. In the case of achieved coverage, Firebase Test Lab keeps the

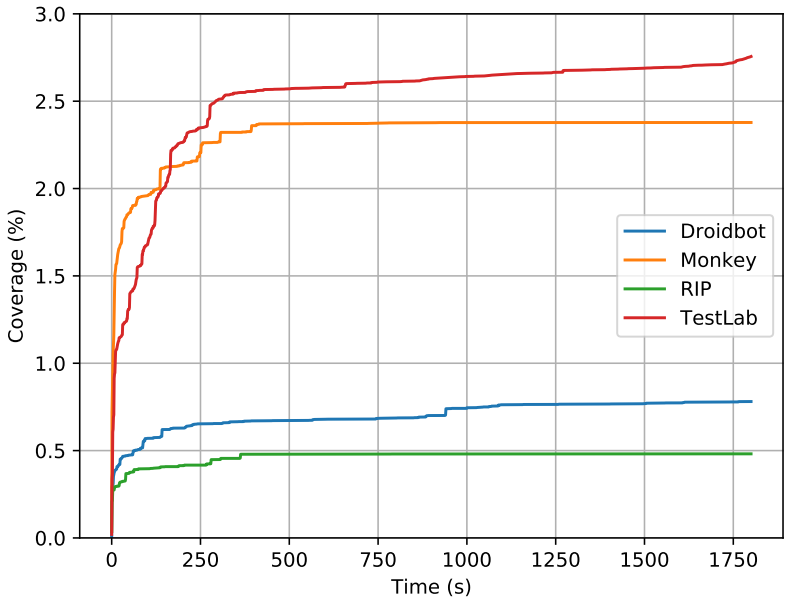

FIGURE 4.2: Accumulated Average Method Coverage by Tool According to the Number of Methods Reported by APKAnalyzer

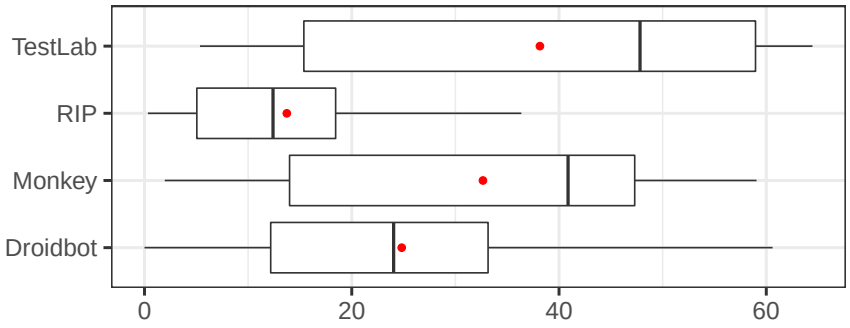

FIGURE 4.3: Coverage Achieved by Tool

lead with the highest values, but now Droidbot is the tool in second place. Nevertheless, the results for Droidbot are very fluctuating, making again Monkey, the second tool with the second-best results.

As can be seen, the dot for every tool in figure 4.3 are the max values reached in figure 4.1. So, the last graph mentioned will be the description of how each tool reached its value through time.

4.3.2 Error Results

Thanks to CoverageAnalyzer, it is possible to establish the number of error traces found by an exploration tool. As explained before, CA analyses the logcat and

extracts the error traces and filter them using the package name of the app under analysis. That is how the results for figure 4.4 and figure 4.5 were obtained. The first graph shows only the top value found after manually analysing the results for every exploration, and the second one represents the average number of errors found across all the executions.

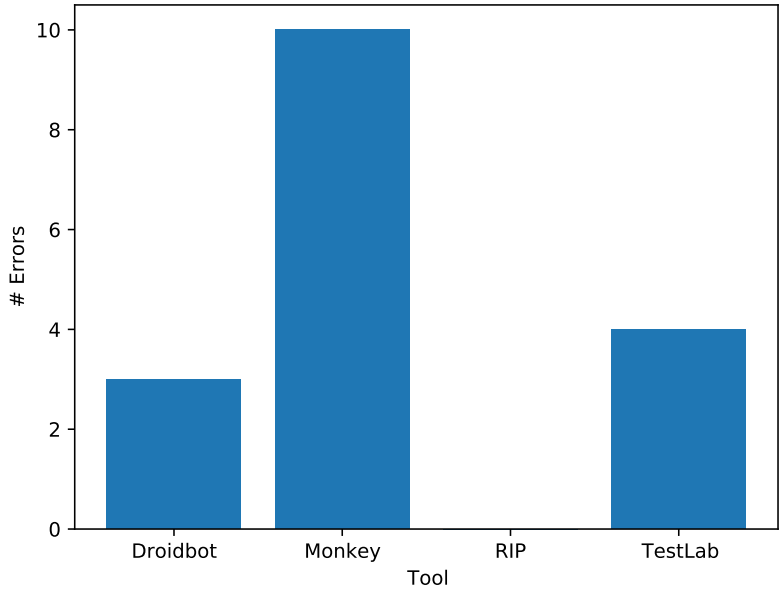

FIGURE 4.4: Maximum Number of Errors Found by Tool in One Exploration

Thus, Monkey is the tool with the top number of failures found during one exploration (RQ-2), as well as the one with the highest average errors found during all explorations (RQ-3).

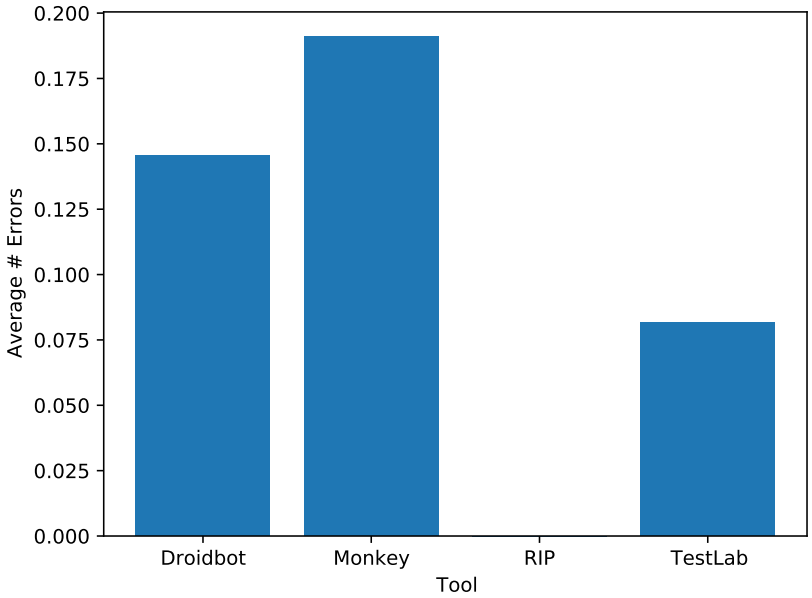

FIGURE 4.5: Average Number of Errors Found by Tool During all the explorations

Chapter 5

Conclusions and Future Work

With the results shown in sections 4.3.1 and 4.3.2 there are enough data to conclude that: The best tool for reach high method coverage is Firebase Test Lab, which also is capable of finding some errors while exploring.

Nevertheless, in a testing environment, the high method coverage is not that important if no errors are found, for that reason, Monkey becomes a better option. Even when this tool has no complex architecture nor complicated exploration strategy, and is the default tool provided with Android SDK Tools. In this study is shown that the relation between method coverage reached and the number of errors found of Monkey is better than for the other tools presented in this document.

Another conclusion for this study is that a more complex exploration strategy not always leads to better coverage and higher numbers of errors discovered.

Thus, it is visible that the main objective together with the specific objectives proposed in section 1.2 were fulfilled within this document. The specific objective number one was satisfied in section 4.3.1, and the specific objective number two was achieved in section 4.3.2. The results shown in the aforementioned sections, allow developers and researcher to validate their decisions when selecting an automatic exploration tool for Android applications, as well as give them an idea of what is still missing when regarding the automatic exploration of Android applications. In consequence, the main objective was accomplished.

Furthermore, as more and more exploration tools are designed and implemented, as well as exploration strategies there is the need of repeat this research periodically owing to provide information of the newest tool to developers and researchers. This study also will allow researchers to know what are the next steps for reaching better testing tools regarding automatic exploration.

Besides, the workflow designed for this study and detailed in section 3.1 can be refined, extended and enhanced so as to achieve the standardization of the validation of new exploration tools for Android apps.

Additionally, because of time limitations, and resources, the number of applications used in this study was not as high as was expected at the beginning. So, in order to extend this study, more applications and different tools can be used, as well as use human exploration to compare the results of a human-being against the automatic exploration tools can be made.

Another validation that can be made, is to validate that the tools are orthogonal to each other. That means that all the methods visited by one tool were visited for the other. Could be interesting because can be the case of a tool reaching less coverage, but it visited the most complex methods, and one with high coverage but, it only reached the methods that do not present any difficulty.

Finally, the tool designed for this study InstruAPK (3.2) and CoverageAnalyzer (3.3) can have several improvements. InstruAPK can only instrument methods that are being called inside the source code, avoiding the instrumentation of dead code, which for now could be rising the number of instrumented methods and limiting the accuracy of the coverage reports.

Bibliography

[1] Yuzhong Cao et al. “CrawlDroid: Effective Model-based GUI Testing of Android Apps”. In: Sept. 2018, pp. 1–6. ISBN: 978-1-4503-6590-1. DOI: 10.1145/3275219.3275238.

[2] S. R. Choudhary, A. Gorla, and A. Orso. “Automated Test Input Generation for Android: Are We There Yet? (E)”. In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2015, pp. 429–440.

[3] Yuanchun Li et al. “DroidBot: a lightweight UI-Guided test input generator for android”. In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 2017, pp. 23–26.

[4] S. Liñán et al. “Automated Extraction of Augmented Models for Android Apps”. In: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2018, pp. 549–553.