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 14, 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 applications, by the 2020, in the two main markets together is 4407000. The 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 fulfill their needs or expectations. Therefore, every day more and more complex applications are released 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 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 is 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 more deep 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 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 MARIO ►[CITE]◄ . This tool uses a pseudo-random generation events strategy, 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. Another well known tool is Firebase Test Lab MARIO ►CITE◄ ; this tool can be used online. Accordingly to its documentation, it analyzes the UI of the applications and explores the apps by simulating users events. They claim to always explore the tool in the same order. Another exploration tool is RIP, an active project from The Software Design Lab at *Universidad de los Andes*; RIP explores applications using a model-based GUI testing technique MARIO ►CITAR RIP, EL PAPER QUE DIJO CAMILO◄ . It can lead to different exploration results due to its current implementation as well as to the 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 good decision. Besides, the project budget, and application complexity among others things can also affect the decision. The information available to make the decision is often based in 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 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 final products with poor quality. In short, the decision of the right automatic exploration tool should be easy 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 following specific objectives were proposed:

1. Compare exploration tools based on their code coverage capabilities
2. Compare exploration tool based on the number of unique error traces discovered while exploring an application.
3. Compare exploration tools by using qualitative aspects such as, is the tool a open source project? Is the tool free? Does the tool allow user to introduce login values? how useful is the tool report for developers 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

MARIO ▶*TERMINAR ESTA SECCIÓN*◀

// TODO aquí

1.4 Document Structure

This document has the following structure: In Chapter 2 related work is described, you will find information about automatic 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 was 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. On top of that, chapter 5 describes de conclusions using the data obtained in the previous chapter. Finally, chapter 6 depicts the future work, discoursing the 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 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., MARIO ►[CITE]◄ 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. MARIO ►[CITE]◄ study was made.

2.1 Crawldroid

MARIO ►CITE◄

This tool 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 tools 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 in order to know how good is their tool. Using a tool called ELLA for its coverage measurement. ELLA provides information about method and activity coverage. MARIO ►CITE 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

MARIO ▶CITE https://ieeexplore-ieee-org.ezproxy.uniandes.edu.co:8443/document/7965248◀
Droidbot 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 a open source project.

This tool is also a open source project and it makes part of a study, where their authors explains 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 a 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

MARIO ▶Cite https://firebase.google.com/◀

This is a Google product. This tool is different to the others because it does not make part of 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 number of visited activities.

2.4 RIP

https://ieeexplore.ieee.org/abstract/document/8530063/ MARIO ▶CITE◀

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

This tool is also part of a research named "Automated Extraction of Augmented Models for Android Apps". where their authors claim that all this new information is necessary owing to the complexity of mobile applications. Thus, when more information is recorded and less variables are unknown, bugs reproduction will be easier.

RIP does not contains any coverage metrics by default, 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 Shauvik Roy Choudhary, et al., MARIO ►[CITE]◄ work. Most probably because by the time they made the research, none of them existed. This creates a gap between the available information for developers and researches and the current automatic exploration tools. As a result, this study aims to provide information about some of the newest tools that are being used in the industry and the academy. Giving a better overview of nowadays automatic exploration tools.

Chapter 3

Solution Design

3.1 General Approach

With the purpose of achieving the objectives mentioned in Sec.1.2, a workflow was designed. This workflow contains five stages: (1) Instrumentation of the applications, (2) Exploration, (3) Coverage measurement, (4) Summarize data, and (5) Data Analysis. The workflow is depicted in Figure 3.1.

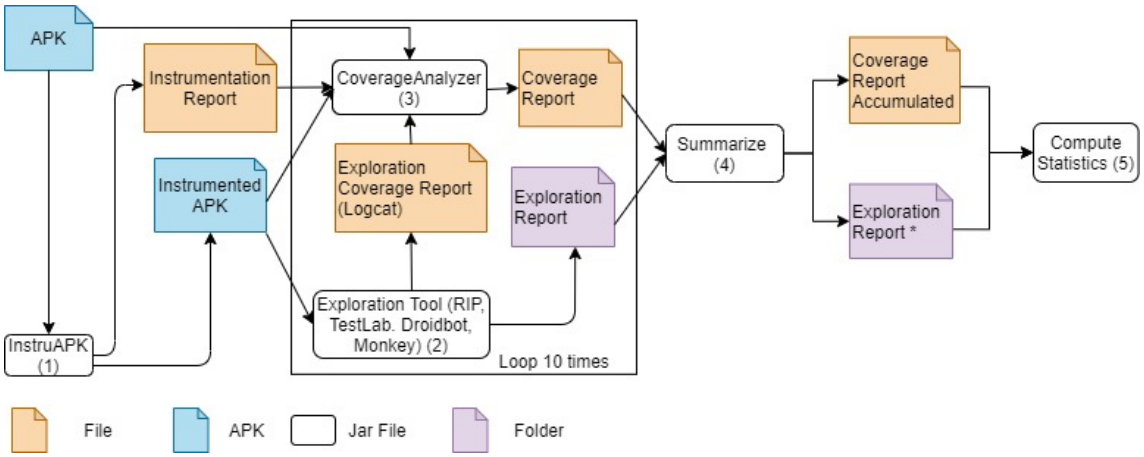

FIGURE 3.1: Main Workflow

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, 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 with four different exploration tools, (1) Droidbot, (2) Monkey, (3) Firebase Test Lab, and (4) RIP. MARIO ▶*Add citations for each tool*◀ 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 analyzed 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.

Droidbot, Monkey and 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. Its output is a report containing two method coverage measurements, the first one, calculated using the number of methods reported by APKAnalyzer, a tool provided in the Android SDK Tools MARIO ▶*Citate APKAnalyzer*◀ , 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. The multiple executions are intending to get average values as well as comparable results along the different exploration tools. On top of that, the input for stage 4 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, i.e., Stage 5, encompasses data understanding, graphs creation, 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 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 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 representation of the app source code. These smali files are analyzed 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.

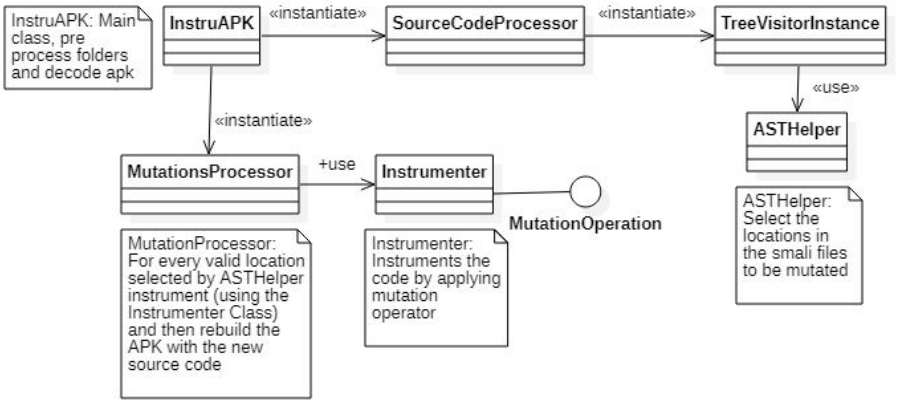

FIGURE 3.2: Class Diagram InstruAPK

Figure 3.2 only contains the main classes of the tool and offers a short explanation of what is doing 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.

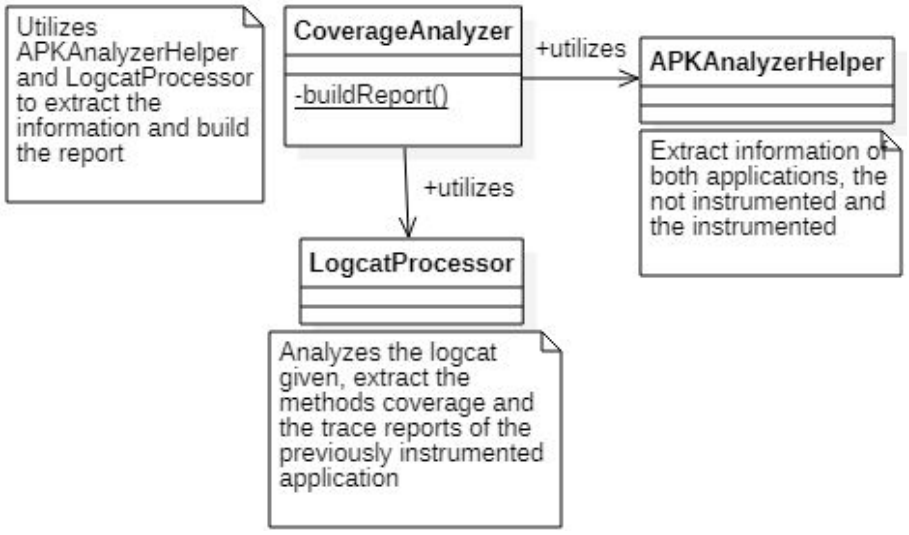

FIGURE 3.3: Class Diagram Coverage Analyser

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

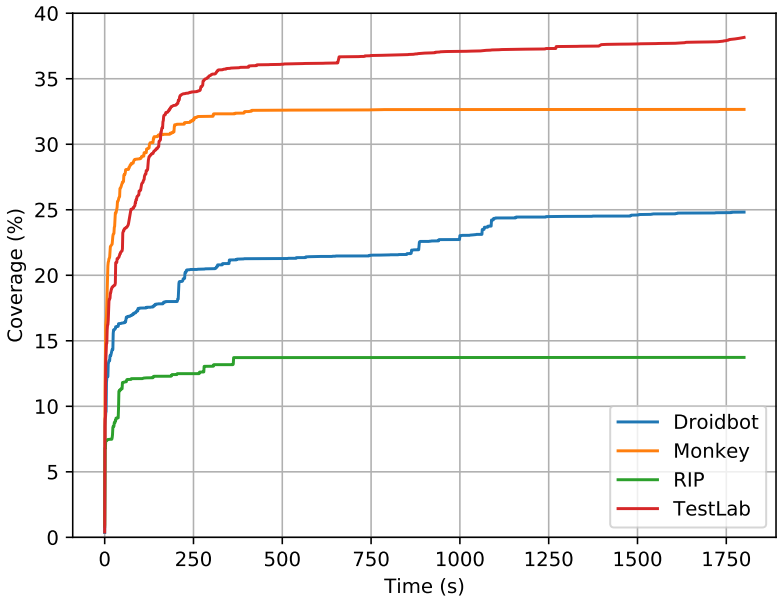

FIGURE 4.1: Average Method Coverage by Tool

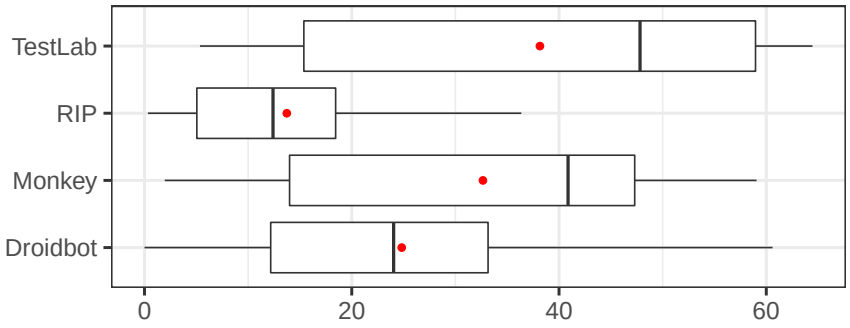

FIGURE 4.2: Boxplot of Accumulated Coverage by Tool

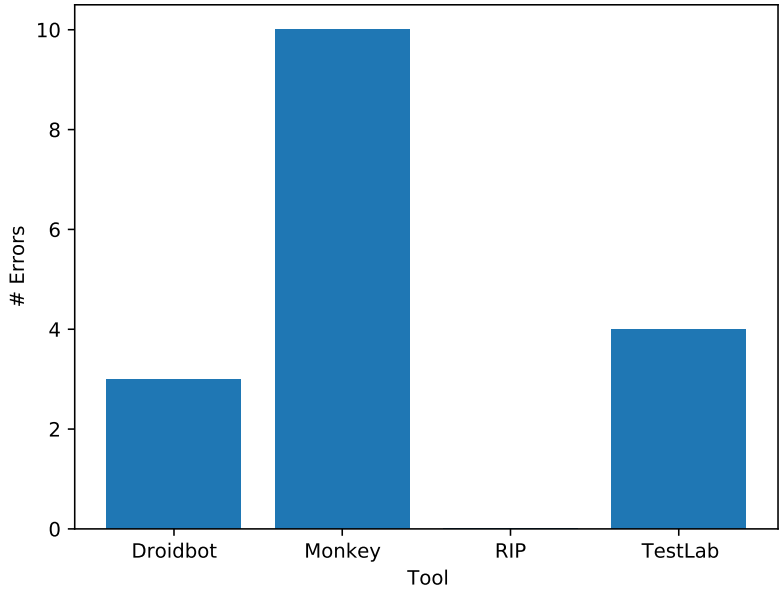

FIGURE 4.3: Maximum Number of Errors Found by Tool

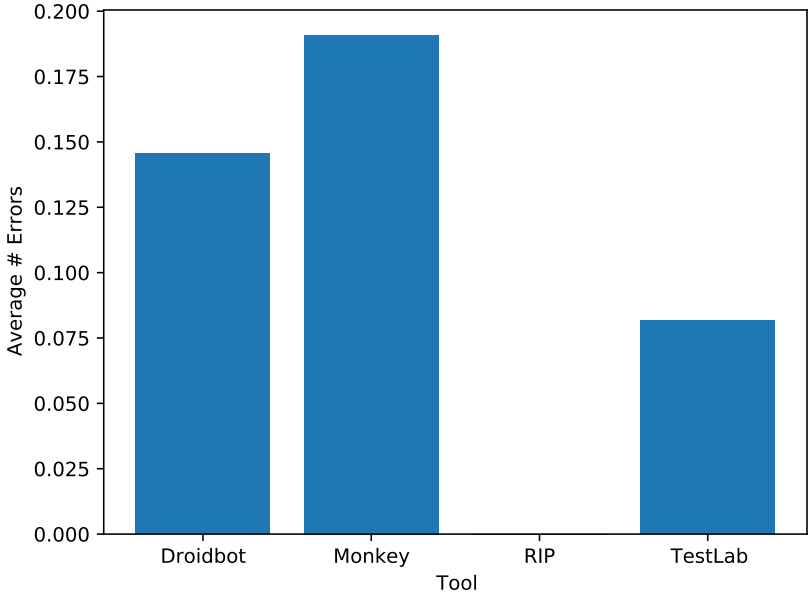

FIGURE 4.4: Average Number of Errors Found by Tool

Chapter 5

Conclusion

Chapter 6

Future Work