Comparison Among Automatic Exploration Tools for Android Applications

Michael Osorio-Riaño
Department of Systems and Computing Engineering
University of Los Andes
Bogotá, Colombia
ms.osorio@uniandes.edu.co
Advised by: Mario Linares-Vásquez
m.linaresv@uniandes.edu.co

I. Abstract

TODO

***Index Terms*—Android, Mobile, Tests, Automatic Exploration,Empiric Software Engineering, Comparison**

II. Introduction

TODO

III. Related Work

TODO

IV. Objectives

- General Objective: Provide an objective point of view for the most widely used automatic exploration tools in order to facilitate developers the selection of the right tool that suits their needs.
- Specific Objectives:
  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?

V. Solution Design

In order to measure the method coverage reached by an exploration tool in one application, there is the need to know how many methods there are in the application and how many methods were called during the exploration. To achieve that, the application developers could count the number of methods in their project and write log lines at the beginning of every method. After that, they will need to compile, run the exploration tool against their application, measure the number of methods that were called during the exploration and compute statistics.

With that in mind, this study consists of nearly the same stages, i. instrumentation, ii. exploration, iii. coverage analysis, iv. summarize, and v. compute statistics, The stages ii. and iii. were repeated 10 times for every application that was selected, that leaded to the iv. stage. See Fig 1

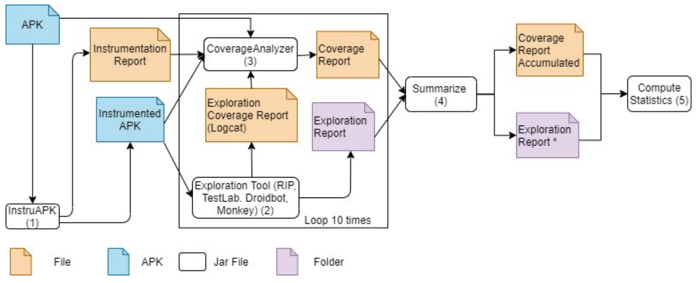
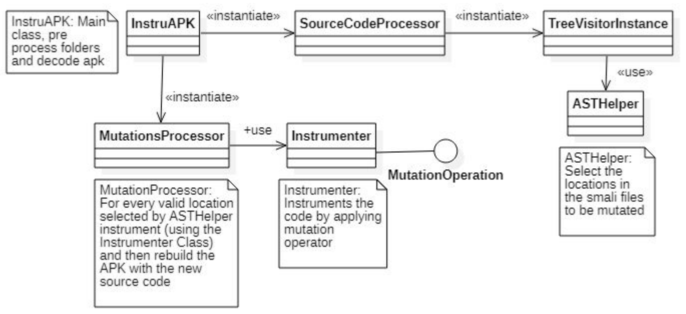


Fig. 1. Complete Study Workflow

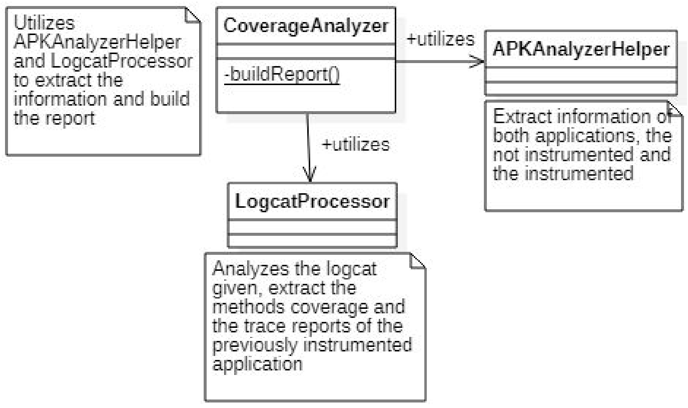
The different stages were completed as follow:

- Stage i: The applications' instrumentation was made by using InstruAPK. It is an instrumentation tool developed mainly for this study, its class diagram is presented in Fig 2. This tool 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 and the 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 and then, the log code is injected at the very beginning of each method. Its 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.
- Stage ii: The exploration was made by four different automatic exploration tools. 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



InstruAPK: Main class, pre process folders and decode apk
InstruAPK
«instantiate»
SourceCodeProcessor
«instantiate»
TreeVisitorInstance
«use»
ASTHelper
«instantiate»
MutationsProcessor
+use
Instrumenter
MutationOperation
MutationProcessor: For every valid location selected by ASTHelper instrument (using the Instrumenter Class) and then rebuild the APK with the new source code
Instrumenter: Instruments the code by applying mutation operator
ASTHelper: Select the locations in the smali files to be mutated

Fig. 2. Class Diagram InstruAPK Tool



Utilizes APKAnalyzerHelper and LogcatProcessor to extract the information and build the report
CoverageAnalyzer
-buildReport()
+utilizes
APKAnalyzerHelper
Extract information of both applications, the not instrumented and the instrumented
+utilizes
LogcatProcessor
Analyzes the logcat given, extract the methods coverage and the trace reports of the previously instrumented application

Fig. 3. Class Diagram CoverageAnalyzer Tool

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 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.

- Stage iii: The coverage measurement was made by CoverageAnalyzer. It is a Java Application created mainly for this study, its class diagram is presented in Fig 3. This tool analyses the resulting logcat of an Android phone, when executing an application that was instrumented by InstruAPK. It extracts different data such as, number of methods called, number of methods never called, number of error traces of the application being analysed, most called methods, less called methods, as well as the time stamp of all calls of every method. It is important to notice that the possibility of extracting all those information is because InstruAPK provides it.
- Stage iv: Due to the multiple executions of every APK per tool, it is important to summarize de data. The final report contains the total number of unique methods called during all ten executions, reporting only the first time they were called. The exploration reports are analysed and filtered as well.
- Stage v: This is the final stage of the study that involves, understanding data, computing statistics, creating graphs, extract insights and conclusions.

Finally, the set for this study is a set of 11 applications. 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.

VI. Methodology

The stages defined in Sec.V were defined to fulfilling the objectives defined in Sec.IV.

VII. Results

TODO

VIII. Conclusions

TODO

IX. Future Work

TODO

References

[1] Android monkey, http://developer.android.com/guide/developing/tools/monkey.html.
[2] Android testing framework, http://developer.android.com/guide/topics/testing/index.html.
[3] Amit Seal Ami, Md. Mehedi Hasan, Md. Rayhanur Rahman, and Kazi Sakib. 2018. Mobicomonkey: context testing of Android apps. In Proceedings of the 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft '18). ACM, New York, NY, USA, 76-79.
[4] Young-Min Baek and Doo-Hwan Bae. 2016. Automated model-based Android GUI testing using multi-level GUI comparison criteria. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016). ACM, New York, NY, USA, 238-249.
[5] Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, and Zhendong Su. 2017. Guided, stochastic model-based GUI testing of Android apps. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017). ACM, New York, NY, USA, 245-256.
[6] Nariman Mirzaei, Joshua Garcia, Hamid Bagheri, Alireza Sadeghi, and Sam Malek. 2016. Reducing combinatorics in GUI testing of android applications. In Proceedings of the 38th International Conference on Software Engineering (ICSE '16). ACM, New York, NY, USA, 559-570.
[7] Y. Zhauniarovich, A. Philippov, O. Gadyatskaya, B. Crispo and F. Massacci, "Towards Black Box Testing of Android Apps," 2015 10th International Conference on Availability, Reliability and Security, Toulouse, 2015, pp. 501-510.