Islamic University of technology(IUT)

A Comparative Study of Static Code Metrics and Behavioural Metrics for Predicting Risk Scores in Android Apps

Authors

Fahim Arsad Nafis (170042004)

Maysha Afrin Munia (170042049)

Syeda Mishra Saiara (170042077)

Thesis Supervisor

Ashraful Alam Khan

Assistant Professor, Department of Computer Science and Engineering (CSE)

Thesis Co-Supervisor

S. M. Sabit Bananee, Imtiaj Ahmed Chowdhury

Lecturer, Department of Computer Science and Engineering (CSE)

Islamic University of Technology (IUT)

A thesis submitted to the Department of CSE

in partial fulfillment of the requirements for the degree of

Bachelor of Science in Software Engineering

Department of Computer Science and Engineering (CSE)

Islamic University of Technology (IUT)

Organization of the Islamic Cooperation (OIC)

Gazipur, Bangladesh

May 14, 2022

Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by Fahim Arsad Nafis, Maysha Afrin Munia, Syeda Mishra Saiara under the supervision of Ashraful Alam Khan, Assistant Professor, S. M. Sabit Bananee and Imtiaj Ahmed Chowdhury, Lecturer, Department of Computer Science and Engineering, Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither this thesis nor any part of it has been submitted anywhere else for any degree or diploma. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

Fahim Arsad Nafis	Maysha Afrin Munia	Syeda Mishra Saiara
Student ID: 170042004	Student ID: 170042049	Student ID: 170042077

Supervisor:

Ashraful Alam Khan	S. M. Sabit Bananee	Imtiaj Ahmed Chowdhury
Assistant Professor	Lecturer	Lecturer

Department of Computer Science and Engineering
Islamic University of Technology (IUT)

Abstract

Identifying security flaws and distinguishing non-susceptible code from vulnerable code is a difficult undertaking. Security flaws are usually inert until they are exploited. Software metrics have been widely utilized to forecast and signal a variety of software quality features. We investigate static code metrics and behavioural code metrics, their correlation, and their association with security vulnerabilities in Android applications. The aim of the study is to understand: (i) the comparison between static software metrics and behavioural code metrics; (ii) the ability of these metrics to predict security vulnerabilities, and (iii) which are the strongly correlated static code metrics and behavioural code metrics. From our study, we have found that even though static code metrics require higher computational power, it provides better results to predict the risky behaviour of android applications and Random Forest Regression provides more stable results with a better R2 score for this specified dataset which we create for our thesis.

Acknowledgement

We would like to start by expressing our deepest gratitude to Almighty Allah for allowing us to complete our research successfully. We are indeed grateful to be able to submit our thesis work by which are eventually going to end our Bachelor of Science study. Next we would like to express our honest appreciation to Ashraful Alam Khan, Assistant Professor, S. M. Sabit Bananee and Imtiaj Ahmed Chowdhury, Lecturer, Department of Computer Science and Engineering, Islamic University of Technology for being our adviser and mentor. His inspirations, ideas and observations for this particular research work were invaluable, and this thesis would have never been totally successful without his encouragement and proper guidance. His valuable opinion, time and input provided throughout the thesis work, from the first phase of thesis topics introduction, research area selection, proposition of algorithm, modification and implementation helped us to do our thesis work in proper way. We are grateful to him for his constant and energetic guidance and valuable advice.

We extend our appreciation to all the respected jury members of the thesis committee for their thoughtful comments and constructive criticism of our research work. They certainly helped us develop this work of science.

Finally, we would like to express our sincere gratitude to all the faculty members of the Computer Science and Engineering department of Islamic University of Technology for providing us with a congenial and supportive work environment.

Contents

List of Figures

Chapter 1

Introduction

We first provide an overview of our study in this section, which discusses the problem statement and the nature of the problem in depth. The research challenges pertinent to the entire scenario are also addressed on the basis of the description of the issue. We also note the thesis objectives, motivations and our contribution in separate subsections. At the end of this section we describe the organization of the thesis.

1.1 Overview

1.1.1 Static Analysis

Static analysis is a way of debugging apps that involves inspecting the code without executing the program. It provides an understanding of the code structure in more detail. This analysis method helps ensure that the code maintains the industry standards. In the context of static analysis, it focuses on scanning source code for certain coding patterns that are related with some type of warning or information.

1.1.2 Static Code Metrics

Static Code Metrics are a set of software metrics that provide quantitative insight into the code. Some popular static code metrics are - Line of Code (LOC), Number of Bugs, Number of Classes, Methods and Functions, Technical Debt, Cyclomatic Complexity, Cognitive Complexity etc. Common tools like SonarQube [14], Raxis [1], QARK [7] etc. are used to find out static code metrics from the source code.

1.1.3 Behavioural Code Metrics

Behavioural Code Metrics is a set of software metrics that provide qualitative insight into the code. Some popular used behavioural metrics are - Tele-

phony services abuse, Remote connection establishment, PIM data leakage, Audio/video flow interception, Denial of Service, Arbitrary code execution, Geo location information leakage etc. There are some common tools like AndroWarn [8], Android Lint [6] etc which are used to figure out the behavioural code metrics from the source code

1.1.4 Fuzzy Risk Score

This risk score is an estimate of the security and privacy risks of your Android application. Androrisk assigns each app a risk value from 0 to 100, depending on the accesses and settings of the application. The higher the risk score, the greater the vulnerability of the application. Each authorization has a weight assigned to it based on its sensitivity and potential risk (i.e. access to the Internet, geolocation, or payment systems). Androrisk looks at 21 different risk categories that an app might provide to end-users in terms of security and privacy. Androrisk uses fuzzy logic[9] to calculate the application's security and privacy risk based on the mapping of actions to the 21 categories.

1.2 Problem Statement

There has been lots of work in the field of static analysis of the Android application in the past, but most of them have been geared towards a static code metrics. The field of behavioural code metrics in the static analysis domain is relatively new and is quite promising. Having explained the various applications of static analysis and our motivation for this domain, we can finally declare our problem statement as follows:

> The primary objective of our thesis is to conduct a comparative study of static code metrics and behavioural metrics for predicting risk scores in Android applications.

1.3 Organization of the Thesis

In Chapter 2, we discuss the background knowledge needed to understand the concept of our thesis.We talk about the different tools and methods we have used. In Chapter 3, we dive into the existing literature reviewed for our work. In Chapter 4, we present the methodology of our work and the preparation processes of our dataaset.In Chapter 5, we present our results and discussion followed by the conclusion in Chapter 6.

Chapter 2

Background Knowledge

2.1 Android APK Decompilation

Figure 1: Process of Android APK Decompilation

Android APK decompilation is the method of conducting reverse- engineering the android APK files to recover the almost similar version of Java source code.[10] This is the most effective way to recover a similar version of the source code. It gives us an idea about the vulnerability of any application. It is considered to be a good approach to detecting the threat of the application. In case modding any app, APK decompilation is required. There are certain steps which are followed during the Android APK decompilation. APK is basically a zipped file where encoded resources and dex files are stored. Apktool [3] decodes the APK and converts them into SAMLI files. SAMLI files are basically assembly files. The SAMLI files are then passed to the dex2jar [11] converter in order to convert the SAMLI files into jar files. This jar file contains the Java byte codes in a zipped file. These Java byte codes are not readable. To extract the source code from the byte code, Luyten [4], JD GUI can be used. Luyten helps to convert the byte code into the readable format of the source code.

2.1.1 Apktool

Apktool [3] is a reverse engineering tool for 3rd party, closed, binary Android apps which decode resources to the nearly original form and rebuild them after making some modifications. Working with this app is easier than with other tools because of the file structure. It helps us in the automation of some repetitive tasks like building APK etc.

2.1.2 dex2jar

dex2jar [11] is a code converter that converts a .dex file to .class files. The output is provided as a zipped file. It is designed to read the Dalvik Executable (.dex/ .odex) format. smali/baksmali disassembles dex and reassembles dex from smali files.

2.1.3 Luyten

Luyten [4] is an open-source Java decompiler GUI and converts bytecode to source code.

2.2 SonarQube

SonarQube [14] is an open-source platform for continuous code quality inspection and static code analysis to uncover static code metrics like LOC, Code Smells, Cyclomatic Complexity, Code Duplications etc. Analyzing an android APK requires certain steps to be followed. At first the APK is decompiled into Java code. Then the environment is setup for the project. Through the terminal instructions, the Java code is analyzed and it provides the desired static code metrics.

2.3 Androwarn

Androwarn [8] is an open-source tool for detecting and warning users about potentially dangerous Android app behavior. Static analysis of the application's Dalvik bytecode, which is created throughout the process from the APK, is used to make the detection. When an APK is run through Androwarn, a report is created based on the user's chosen technical detail level. Essential, Advanced, and Expert are the three detail levels. The report is available in three formats: HTML, JSON, and txt.

2.4 AndroGuard

Androguard[5] is a python tool to analyze Android applications. It is mainly used to conduct Mobile Forensics, Malware Detection and Security Detection The Androrisk module from the Androguard tool was used to calculate the Fuzzy Risk Score. AndroGuard can work with:

- DEX, ODEX
- APK
- Android's binary XML
- Android resources
- Disassemble DEX/ODEX bytecodes
- Decompiler for DEX/ODEX files

It can be used from the CLI or graphical frontend for AndroGuard, or use AndroGuard purely as a library for your own tools and scripts. It is also available in a specialized Linux environment,SANTOKU [13]

Chapter 3

Literature Review

3.1 Predicting Android Application Security and Privacy Risk With Static Code Metrics

The goal of this article is to help Android app developers examine the security and privacy risks connected with their apps. As predictors, they employ static code metrics. They assessed the security and privacy risk of an Android application by determining how vulnerable it is to leaking end-user private information and exposing vulnerabilities. They look into how well static code metrics derived from the source code of Android applications can be utilized to predict the security and privacy risk of those apps. They used SonarQube [14] to collect 21 static code metrics from 1,407 Android applications and used the data to forecast the security and privacy risk of the apps. They used a radial-based support vector machine (r-SVM) [15]. They received a precision of 0.83 for r-SVM.[12]

3.2 Empirical Analysis of Static Code Metrics for Predicting Risk Scores in Android Applications

They explore which software static metrics have a stronger link to source code security vulnerabilities. They theorized that certain metrics may be used to identify vulnerable from non-vulnerable code. The researchers looked at a dataset of 1407 Android apps with various static code metrics. The goal of this project is to create an empirical study that will look for correlations between these measures, as well as their impact on finding vulnerabilities in source code, and to produce a collection of metrics that will help anticipate security problems. Some of the static measures revealed a significant link in the outcomes of this empirical study.[2]

Chapter 4

Proposed Method

4.1 Dataset Preparation

- **APK Selection** :We have scraped 119 android applications from Google Playstore. In the case of choosing the applications, we have tried to focus on the most used applications in Bangladesh.

- **Extracting Static Code Metrics** :Following the APK selection process, we have collected the required metrics from SonarQube. The static code metrics we have extracted for this process are mentioned in the following table.

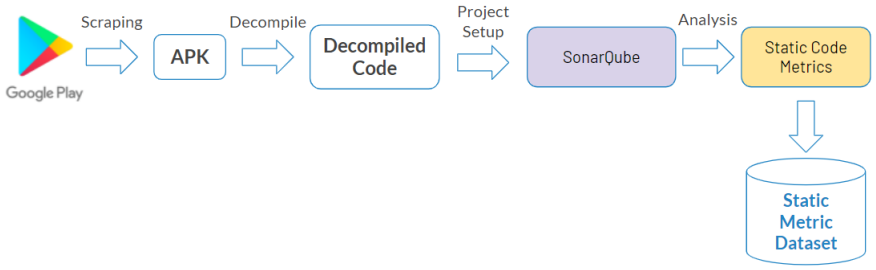

Figure 2: Extracting Static Code Metrics

- **Extracting Behavioral Code Metrics** :After scrapping the apps from the google store we pass them through Androwarn. Then we get a featured report generated for each application. And the report provides 10 behavioural features which are as follows:

  - Device settings exfiltration
  - Geolocation information leakage
  - Connection interfaces information exfiltration
  - Telephony services abuse

- Audio/video flow interception
- Remote connection establishment
- PIM data leakage
- Arbitrary code execution
- Denial of Service
- Telephony identifiers exfiltration

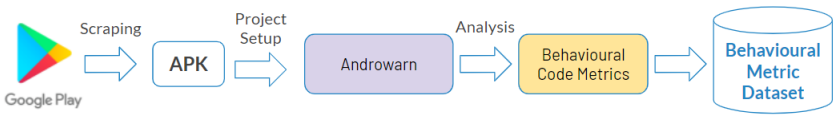

Figure 3: Extracting Behavioural Code Metrics

Telephony identifiers exfiltration means the theft or leakage of any device information like IMEI, IMSI, MCC, MNC, LAC, CID, operators name etc. Device settings exfiltration includes software version leakage, usage statistics, systems settings, logs leakage. Geolocation information leakage is about the GPS/WiFi geolocation information theft or reading the information from the application. Connection interfaces information exfiltration is theft of WiFi credentials, Bluetooth MAC address etc. Denial of Service defines event notification deactivation, file deletion, process killing, virtual keyboard disable, terminal shutdown/reboot.

- **Calculating Fuzzy Risk from Androrisk**: As we can see in Figure 1, we can calculate the Fuzzy Risk score from The Androrisk module of AndroGuard. Each authorization has a weight assigned to it based on its sensitivity and potential risk. Some assigned weights are: MONEY RISK: 5, SMS RISK: 5, PHONE RISK: 5, INTERNET RISK: 2, PRIVACY RISK: 5, DYNAMIC RISK: 5, etc. These values are used to calculate a risk score.

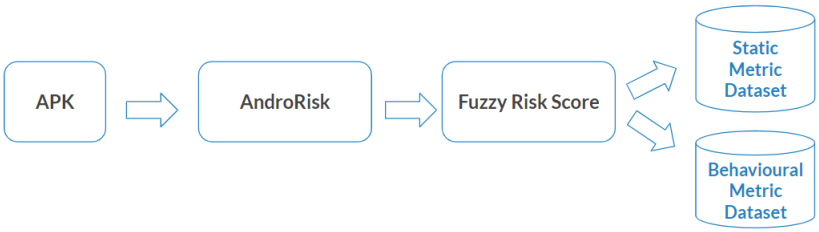

Figure 4: Fizzy Risk Score Calculation

- **Dataset 1 (Static Code Metrics)** : After analyzing the apps in SonarQube, we have figured out the required metrics and noted them down. In this dataset, there is a total of 16 features. Androrisk Fuzzy Risk Score has been used as label data for this dataset.

- **Dataset 2** : After the report generation the data was stored as a binary dataset. The columns contained the metrics and the rows had the application names. The apps which showed the behaviours had the value of 1, and the others had 0 in them. The last column contained Androrisk Fuzzy Score which we used as the variable in our regression model.

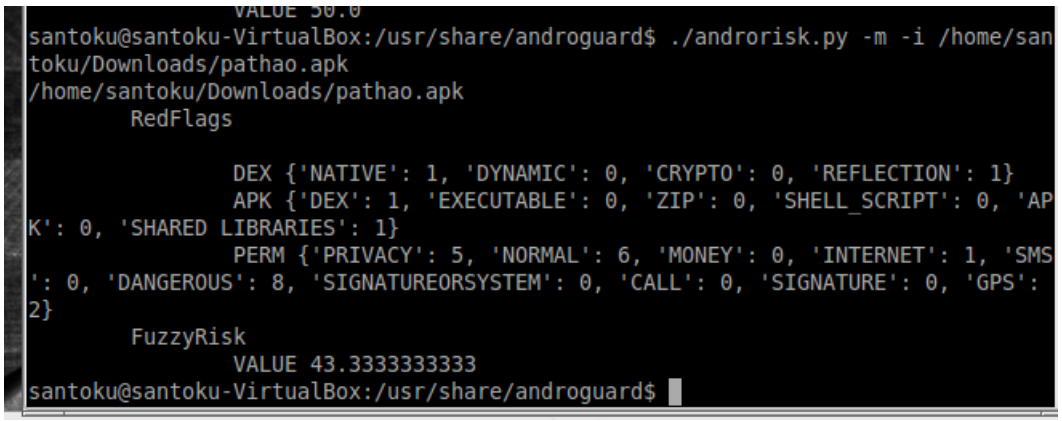

Figure 5: Fuzzy Risk Value Result of a Bangladeshi App

Index	App Name	Duplicated Blocks	Duplicated Files	LOC	Functions	Classes	Files	Comments Coverage (%)	Cyclomatic Complexity	Cognitive Complexity	Androrisk Fuzzy Score
1	Bondhu	30	18	14179	2449	415	27287	26.4	3190	1497	51.11111111
2	BHBFC	2453	103	150115	12069	2134	1022	2.3	26873	29091	50
3	Bangabandhu	1049	73	120639	10448	1708	826	1.8	23019	24863	51.11111111
4	Family Planning	548	31	48971	5126	620	270	3	10960	10904	2
5	Bangladeshi Fire Service	7628	945	343393	31527	5311	3597	13.2	58679	52190	100
6	BMET Info	150	30	44087	4823	574	241	3.4	9308	9548	51.11111
7	Village Court	7663	937	343165	31680	5333	3627	6.5	58623	51919	92
8	Green Life	469	24	25480	2694	352	154	0.9	6006	5960	3
9	Health Service	564	44	49899	5156	636	286	2.9	11163	11019	1
10	Immunization Alert	142	23	43361	5762	558	230	2.7	9184	9491	4
11	Mobile Health Service	4863	530	189746	18560	3009	1872	3.8	33163	27916	99
12	PDS Recovery	117	19	28477	3324	251	196	1.5	6124	6241	51.11111
13	Textile Calculator	132	38	35368	3912	409	189	2.9	7508	7658	5
14	ShornoKishori	2820	309	153975	15577	2259	1344	3.8	27825	25096	92
15	Care Satisfaction	1524	143	106452	9092	1112	558	3.5	16163	14337	50
16	ERD	2590	304	130038	12946	1859	1844	3.9	23267	21221	93
17	Rabies	500	32	36781	3884	403	184	2.7	8319	8289	1
18	Grontho Kendro	517	20	30706	3348	456	202	1.4	7090	6303	0
19	Disaster Info	357	159	25867	2714	357	159	0.9	6007	5955	52
20	BTEB Info	388	101	89110	9729	1146	822	8.3	19311	23222	50

Figure 6: Dataset 1

	A	B	C	D	E	F	G	H	I	J	K	L
		Telephony identifiers	Device settings	Geolocation information	Connection interfaces	Telephony services	Audio/video flow	Remote connection	PIM data leakage:	External memory	PIM data modification:	Arbitrary code execution:
1	Khabar Koi	1	1	1	1				1			1
2	Corona Tracer BD		1		1			1				
3	THE AGE OF DENG		1						1			1
4	24th BCS Admin As		1		1							1
5	PPR-2008		1									
6	ভূমি উন্নয়ন কর		1		1	1		1	1			
7	NID Wallet	1	1		1			1	1			1
8	Bangladesh VPN -	1	1		1	1						1
9	Bangladesh Televis	1	1		1	1		1				1
10	The Daily Star - Ban	1	1		1	1			1			1
11	বাংলাদেশের মানচিত্র	1	1		1	1		1	1			
12	Bangladesh Train T	1	1			1			1			
13	Meena Game	1	1		1	1						1
14	dominos	1	1		1	1		1				
15	ভূমিসেবা		1		1	1		1				
16	All Laws Of Banglad	1	1		1	1			1			
17	Bangladeshi Matr	1	1		1	1	1	1	1			1
18	Azan Bangladesh	1	1		1	1		1	1			1
19	My Airtel - Banglade		1		1	1						
20	Bangladesh Flag	1	1		1	1	1		1			1
21	Bangla Calendar (	1	1		1	1		1	1			
22	সরকারি ওয়েবসাইট	1	1		1	1						
23	Bangladesh Directory	1	1		1	1		1	1			1
24	Bangladesh Pratidi	1	1		1			1	1			

Figure 7: Dataset 2

4.2 Workflow

The workflow is divided into two parts for the two categories of analyzers. One describes the SonarQube analyzer workflow which takes the decompiled APK as input rather than APK. And the other one is for Androwarn which takes direct APK as an input object. From the discussion above, it is visible that datasets were also prepared considering the two separate analyzers our study has. The separate metric datasets are passed through Regression models separately. Then the predicted fuzzy score from each analyzer model is compared with one another. Comparison is carried out based on the evolution metric values that the three models individually generate for each of the analyzers. The models were trained by three different regression models. We used Linear Re-

gression, SVR (Support Vector Regression), and Random Forest Regression. We used these regression models to find the best-fit lines and values for our model. For extracting the correlation value between the features of the sample data, we used Spearman's Rank Correlation value gained after the model training.

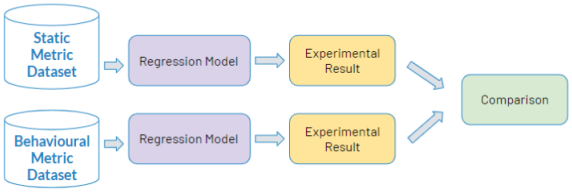

Figure 8: Workflow of the proposed comparison model

Chapter 5

Results and Discussion

The three regression models reflect a very keen observation of the dataset we have. The evolution metrics our models have are MAE(Mean Absolute Error), MSE(Mean Squared Error), RMSE(Root mean squared error) and R2 score. The MAE value for static code metrics for linear regression, SVR, and Random Forest was, respectively, 41.8122, 19.5646, and 12.6839. That means the error value decreased significantly from each regression model used. It can be inferred that, as linear regression minimizes the error between the actual and predicted values through best line fit, it gives a higher error value for the static code metrics in our dataset. Whereas, in the case of the behavior metric model, the MAE values from the regression models, 19.0226, 16.6093, and 18.2660, respectively maintain consistent values. But the error value increased in the Random Forest model. This inconsistency is a reason to infer static code metrics better than the behavioural ones.

Next, if the RMSE values and MSE values are analyzed, a consistent decrement in the case of static code metrics is observed from the values (63.8278, 29.5648, and 15.2452 respectively). Whereas for the behavioural metric, the error value was somewhat in the same range, which means the data samples were resistant to any changes and had inconsistency for any changes. As observed from the values, 22.2731, 26.8461, and 21.4201, it is visible that the RMSE value increased for the SVR model, and the last one was for Random Forest. But it was likely that SVR would give a lesser RMSE value than linear regression. This inconsistency is also a reason to denote behavioural metrics as worse than static code metrics.

The R2 score shows some interesting insights from the result. Before that, if the values are observed we see, for static code metrics the values are -3.7932, -0.0284, and 0.7265. Here the convergence of the data plots got better gradually from linear regression to Random Forest regression. But if the scores for behavioural metrics are observed from the table, (0.2608, -0.0739, and 0.3163)

	LR		SVM		RF Regression	
	S. M	B. M	S. M	B. M	S. M	B. M
MAE	41.8122	19.0226	19.5646	16.6093	**12.6839**	18.2660
MSE	4073.99	496.0891	874.0825	720.7143	232.4183	458.8199
RMSE	63.8278	22.2731	29.5648	26.8461	15.2452	21.4201
R2 score	-3.7932	0.2608	-0.0284	-0.0739	**0.7265**	0.3163

Table 1: Results and Comparison

the inconsistency is again observed for these types of metrics. Though random forest gives a better convergence visualization for behavioural metrics, it is low compared to the R2 score for static code metrics.

The R2 score is another strong proof that SonarQube gives better feature metrics than Androwarn. In other words, static code metrics implies stronger results in predicting risk scores in applications than behavioural metrics.

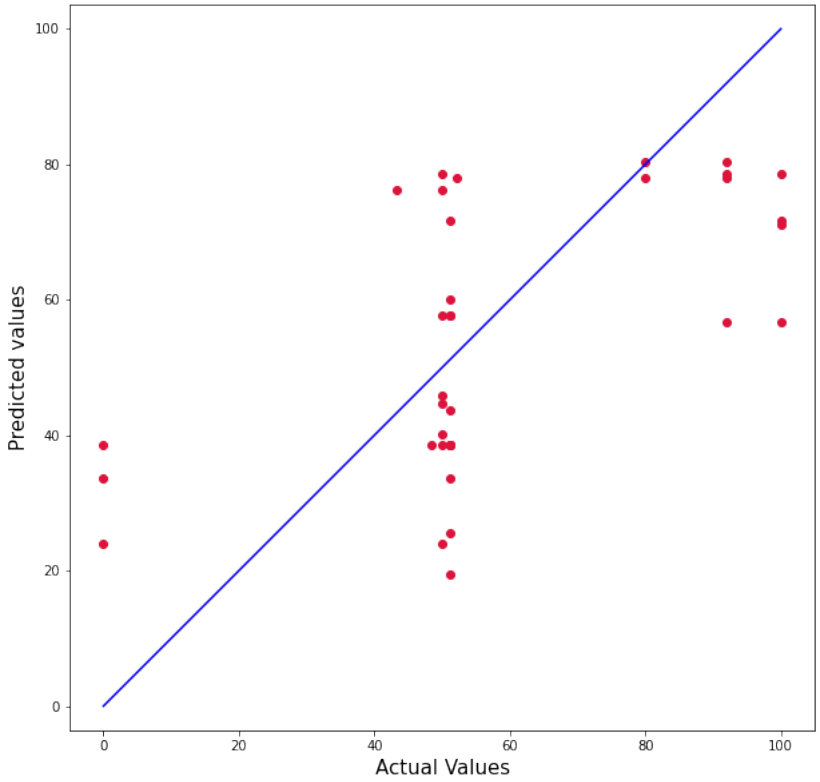

Figure 9: Regression line for Androwarn

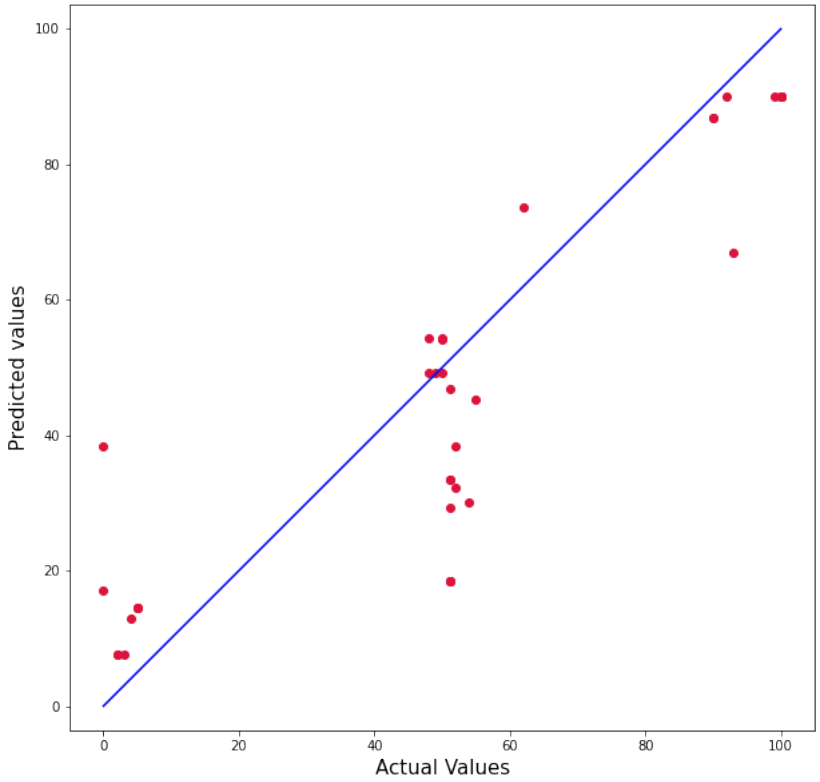

Figure 10: Regression line for SonarQube

The spearman's correlation coefficient for the features for static code metrics gave a good and clear visualization value, whereas for behavioural metrics it gave negative values and maximum values were below 80

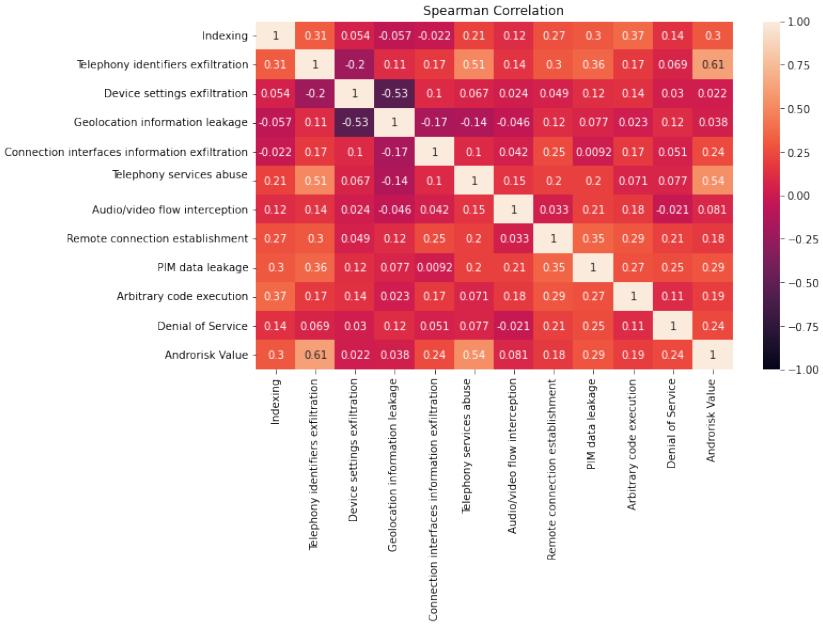

Figure 11: Heatmap depicting spearman's correlation between features in Androwarn

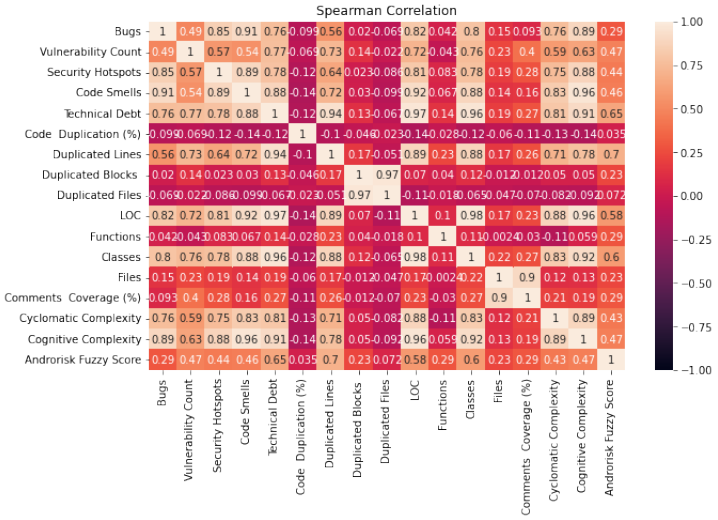

Figure 12: Heatmap depicting spearman's correlation between features in SonarQube

Thus from the discussion above, it can be concluded that 1) Static code metrics give more consistent and better performance metric values compared to behavioural metrics. 2) The behavioural metrics have less relevance to each other in one particular android application.

Chapter 6

Conclusion

As per discussed in former sections, the goal was to come to a distinctive conclusion about the two types of metrics used in the study. After analysing the results and discussing the explanations afterwards, it is clear that there are some unique differences and distinctions between the performance of predicting risk scores between the two metrics. It is evident that set of static code metrics is better in predicting risk scores than behavioural metrics. The value of our evolution metrics proved our problem statement. But surely the work can be improved in future. Furthermore, while this experiment measured only between the two categories of metrics, more comparative studies are required to gain more insight into different patterns of metrics clusters, to investigate, for instance, whether any other groups of metrics give better results than static code metrics. Or behavioural metrics if associated with different other metrics gives a better prediction in risk scores than it gives a single entity of metric set.

References

[1] Apr 8 et al. *Raxis: Penetration testing, Red Teaming*. URL: https://raxis.com/.

[2] Mamdouh Alenezi and Iman Almomani. "Empirical Analysis of Static Code Metrics for Predicting Risk Scores in Android Applications". In: Jan. 2018, pp. 84–94. ISBN: 978-3-319-78752-7. DOI: 10.1007/978-3-319-78753-4_8.

[3] *Apktool*. URL: https://ibotpeaches.github.io/Apktool/.

[4] Deathmarine. *Luyten: An open source java decompiler GUI for Procyon*. URL: https://github.com/deathmarine/Luyten.

[5] Ken Dunham et al. *Android malware and analysis*. CRC Press, 2014.

[6] *Improve your code with Lint checks nbsp;: nbsp; Android developers*. URL: https://developer.android.com/studio/write/lint.

[7] Linkedin. *Linkedin/qark: Tool to look for several security related Android application vulnerabilities*. URL: https://github.com/linkedin/qark.

[8] Maaaaz. *Androwarn: Yet another static code analyzer for malicious Android Applications*. URL: https://github.com/maaaaz/androwarn.

[9] Claudio Moraga. "Introduction to Fuzzy Logic". In: *Facta universitatis - series: Electronics and Energetics* 18 (Sept. 2005), pp. 319–328. DOI: 10.2298/FUEE0502319M.

[10] Ya Pan et al. "A Systematic Literature Review of Android Malware Detection Using Static Analysis". In: *IEEE Access* 8 (2020), pp. 116363–116379. DOI: 10.1109/ACCESS.2020.3002842.

[11] pxb1988. *dex2jar: Tools to work with Android .dex and java .class files*. URL: https://github.com/pxb1988/dex2jar.

[12] Akond Rahman et al. "Predicting Android Application Security and Privacy Risk with Static Code Metrics". In: *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. 2017, pp. 149–153. DOI: 10.1109/MOBILESoft.2017.14.

[13] *Santoku Linux*. URL: https://santoku-linux.com/.

[14] *SonarQube: Code quality and code security*. URL: https://www.sonarqube.org/.

[15] Karl Thurnhofer-Hemsi et al. "Radial basis function kernel optimization for Support Vector Machine classifiers". In: *CoRR* abs/2007.08233 (2020). arXiv: 2007.08233. URL: https://arxiv.org/abs/2007.08233.