NTNU
Norwegian University of
Science and Technology

Thresholds for Software Quality Metrics in Open Source Android Projects

Mile Stojkovski

Applied Computer Science
Submission date: December 2017
Supervisor: Deepti Mishra, IDI
Co-supervisor: Mariusz Nowostawski, IDI

Norwegian University of Science and Technology
Department of Computer Science

Preface

The project was undertaken as a master thesis during autumn 2017 at the Department of Computer Science (IDI) at the Norwegian University for Science and Technology (NTNU).

This area of software quality and software quality metrics was introduced by Assoc. Prof. Deepti Mishra.

Having background in Computer Science or closely related field is preferable requirement for understanding this thesis.

15-12-2017

Acknowledgment

I would like to thank my supervisor Assoc. Prof. Deepti Mishra, for the support, advice and help she provided throughout the duration of this thesis and part of my master studies. My sincerest gratitude goes to Prof. Dr. Sofiya Ostrovska, for all the help she provided with probability distributions. I am also grateful to my co-supervisor, Assoc. Prof. Mariusz Nowostawski for the guidance.

A special thanks to my classmates for the time well spent together.

At last, i would like to thank my friends and family for their support throughout my studies.

M.S.

Abstract

Quality is an important aspect of every software development project. Different stakeholders are interested in different aspect of quality. For instance, from the users' point of view, it represents to what extent the application is satisfying their needs. On the other hand, developers may be more interested in the efforts needed to fix bugs, testing, maintaining existing functionalities provided by the application and adding new ones.

One aspect of measuring and ensuring quality can be achieved by utilizing software metrics. Multiple metrics suites have been proposed and they all capture different aspects of quality.

Product metrics, can reveal internal characteristics of an application and help developers to reach better quality. They indicate how maintainable, testable and extendable one application is based on inheritance characteristics, size and complexity of the applications' modules.

Since the metrics are represented by a number only and the guidelines for each metric specify the desired value, thresholds are needed that will define the lower and upper limit for each metric i.e., reference values that developers can relate to.

Five metrics, including Number Of Methods, Response For Class, Depth of Inheritance Tree, Number Of Children and Coupling Between Objects were successfully computed for 865 Android applications. By calculating the metrics on large number of applications, thresholds for 17 categories of applications have been proposed. These thresholds are different depending what category does the application belong to system, games, multimedia, etc. In addition, each category was divided into subcategories based on size and thresholds were proposed for the appropriate subcategories. By sub-categorizing we allow developers to have a reference values for their applications based on the size in addition to category.

Contents

List of Figures

List of Tables

Abbreviations

NOM - Number of Methods
RFC - Response for Class
DIT - Depth of Inheritance Tree
NOC - Number of Children
CBO - Coupling Between Objects
CF - Coupling Factor
LCOM - Lack of Cohesion of Methods
AC - Afferent Couplings
NPM - Number of Public Methods
NPF - Number of Public Fields
CC - Cyclomatic Complexity
WMC - Weighted Methods per Class
LOC - Lines of Code
NOA - Number of Attributes
IDE - Integrated Development Environment

1 Introduction

The introductory chapter provides a general overview of the topic covered in this thesis. The problem, the benefits and the contribution of this thesis are outlined below.

1.1 Topic

Apple's App Store and Google Play have accumulated in excess of 1 million downloadable mobile applications since their launch in 2008 [1]. This number shows that the stores are now first choice when it comes to publishing an application.

The overall success of app stores and the extensive number of mobile applications available is highly correlated to the mass adoption of smartphone devices by consumers. Even though smartphones existed prior 2008, it was only after the launch of the app stores that made the users truly exploit their computing power and versatility via downloadable applications. Before the app stores era, the problem of availability, compatibility, easy of use and easy of access were the biggest issues when distributing an application. [1]

One important aspect for the overall success of a mobile application, is considered to be its quality. Quality can be viewed from different perspectives. For a user it is very important that the application is stable and with proper affordance, i.e. that the usage is natural and consistent. From the developers' point of view, quality aspects include how maintainable, testable and extendable the application is. I.e., how complex a class is, how many classes does it inherit, what a particular class does, for how many jobs is it responsible, to what extend should a class be split or when does a class becomes too big to be maintained.

However, evaluating an application in order to improve the long term quality is not a trivial task. For this purpose there are multiple proposals that address the quality assurance model for mobile applications. For Android applications, a guide can be found on the Android developers website[1] which includes guidance on how to use user interface elements and how to manage permissions, among the other topics. The guide aids developers to choose the appropriate and Android specific features so the application is consistent in its own existence but also when compared with other Android applications. Additionally, Zahra et al. [2] and Franke et al. [3] have proposed two general quality models for mobile application which

[1] https://developer.android.com/develop/quality-guidelines/core-app-quality.html

depict the important attributes for mobile applications including: usability, portability, efficiency and flexibility. Somerville [4] proposed a set of properties that can be used to evaluate software quality. Software quality, according to Somerville, can be evaluated by external attributes, i.e. attributes perceived by users (efficiency, correctness, easy of use) and internal attributes such as attributes only perceived by the team developing the application e.g. coupling, cohesion, size and inheritance.

Since Android applications are written in Java, which is an object-oriented programming language, the well known and established object-oriented software quality metrics can be applied. Those quality metrics capture the quality aspects from the developers point of view, such as, maintenance efforts, inheritance and coupling.

Multiple software quality metrics have been proposed [5, 6, 7] in the literature. However, the most referenced are Chidamber and Kemerer's metrics suite [8, 9, 10] and Abreu and Carapuca's metrics suite [11].

Since their proposal, the metrics have been verified in multiple studies. The studies investigated the relationship between the metrics values in systems coded in object-oriented languages, mainly Java, C++ and C#. There is evidence that the metrics do expose a possible error in the modules of the systems. [12, 13, 5, 14, 15, 16, 17, 18, 19, 20, 21]

In addition, the metrics have been verified for Android applications [22] and there is evidence that they predict faults in modules.

1.2 Keywords

Metrics/Measurement, Software metrics thresholds, Android applications quality

1.3 Problem Description

Multiple metrics suites have been proposed and validated, showing their potential to indicate problems in the system. The values of the metrics are represented by a number only. However, all of the metrics have viewpoints or guidelines on what does the number represent and what could a increase/decrease of the metrics value cause. For instance, we know that the desired value for CBO is low and the lower it is the more modular the system becomes. Low CBO is also indicator of easier maintenance of the system.

Now the questions is how low should the value of CBO be? How do we define what is the acceptable upper limit for CBO?

In this thesis, we assess the threshold values for 5 metrics, NOC, RFC, DIT, NOC and CBO. The threshold values can be used as an indicator to where does a particular application stands compared to similar applications. They can help

developers relate and interpret the metrics meaning based and their values.

1.4 Justification, Motivation and Benefits

The motivation behind this thesis is based on the idea that software quality model for Android applications can benefit from having the already proposed and verified software quality metrics as a part. Moreover, by providing the threshold values for the metrics, developers can utilize them more efficiently and effectively, they will have a reference value that can relate to. Metrics can help understand where does the application being developed stands in terms of complexity, size, maintainability and cohesion. Metrics can be used to compare and rate software products, thus aid to define acceptance criteria between the developers and the application owners [23, 24].

Metric thresholds are defined by Lorenz and Kidd as:

> "Heuristic values used to set ranges of desirable and undesirable metrics values for measured software. These thresholds are used to identify anomalies, which may or may not be an actual problem." [6]

Since all software metrics are represented with a single number, the efficient usage by developers is tightly connected with the threshold values of a particular metric. Without knowing the metrics thresholds, applying the metrics in practice is limited [25]. As indicated by [26, 27], the possible reason of why object-oriented software quality metrics despite their importance, are not widely adopted in the software industry is because of the non-availability of threshold values.

Having the range or the desired values can stimulate the usage of metrics in developers day-to-day coding and can aid them in answering questions like "Which classes in the system have a large number of methods?" [27]

By proposing thresholds for different categories of application based on their functionality we contribute to more efficient and effective metric utilization by the developers.

Furthermore, by proving subcategories of applications based on their size and suggesting thresholds for each subcategory, we aid developers in locating their application and provide the appropriate thresholds based on the size in addition to category.

1.5 Research Questions

The aim of the thesis is to answer the following research questions:

RQ1: *To what extent, can the traditional object-oriented software quality metrics be applied to Android applications?*

Since Android applications are written in Java, this thesis will assess if and

which of the traditional object-oriented software quality metrics can be applied pragmatically on a large number of Android applications.

RQ2: *What are the threshold values for different categories of applications and the dataset as a whole?*

The aim is to separate the application into different categories based on their functionality and define thresholds for each category. Additionally, deriving thresholds for the dataset as a whole and comparing the values with the ones obtained for each category can potentially indicate were a general threshold can be proposed.

RQ3: *What are the threshold values for each of the subcategories of applications and the dataset as a whole?*

All applications that belong to specific category are not the same in terms of size and complexity. Thus, thresholds on a category level may not be quite useful for developers. Accordingly, it is be beneficial to separate the applications from each category into subcategories based on their size. Doing this can potentially yield different thresholds for different size of applications. Additionally, developers can locate their application both based on the category they belong and their size.

RQ4: *What are the quality implications of the thresholds values?*

What do thresholds values indicate for the applications being investigated and what are the quality implications of the defined thresholds.

1.6 Contributions

The main contribution of this thesis is to provide threshold values for software quality metrics that reveal the applications' complexity (NOM, RFC), inheritance (DIT, NOC) and the coupling between the modules (CBO). The aim is to help developers at using the metrics in their day-to-day coding. Thresholds are defined for categories of applications based on their functionality and on the dataset as a whole i.e. every application will be treated as part of one category. In addition, thresholds are suggested for subcategories of applications that are separated by size (number of modules). The viewpoint here is that thresholds might differ based on the size of the application. In order to provide the thresholds, first we need to demonstrate that the metrics can be applied pragmatically to a large number of Android applications.

To the best of our knowledge, no other study has investigated and proposed thresholds values for Android applications.

1.7 Thesis Structure

This thesis is structured in the following chapters:

Chapter 1 - Introduction introduces the topic covered by this thesis, and the problem area. Additionally, the motivation, benefits and the research questions are presented in this chapter.

Chapter 2 - Background and Related Work contains the previous research conducted on the area of software quality metrics, different methods of deriving thresholds and summarizes quality models for mobile application. This includes more in depth information about the field and importance of software quality metrics in predicting abnormalities.

Chapter 3 - Methodology and Implementation outlines the dataset used and its' justification. In addition tool selection and analysis plan are included. The implementation process is also elaborated in more details.

Chapter 4 - Results and Discussion presents the results for the threshold values. The results are visualized in tables. Whereas, the discussion section interpreters the obtained results, asses the validity of methods used and outlined the limitations.

Chapter 5 - Conclusion summarized our finding and addresses the research questions based on the results obtained from the analysis of the data.

2 Background and Related Work

McCall's quality model [28] depicted in figure 1, classifies all software requirements into eleven software quality factors. These factors are divided into three categories:

- **Product operation factors**: Correctness, Reliability, Efficiency, Integrity and Usability
  These quality factors deal with the requirements that affect the daily operation of the software.
- **Product revision factors** : Maintainability, Flexibility and Testability
  These quality factors refer to software maintenance activities.
- **Product transition factors** : Portability, Reusability and Interoperability
  These factors are included in the product transition, i.e., adaptation of the software to other environments and the interaction with other systems.

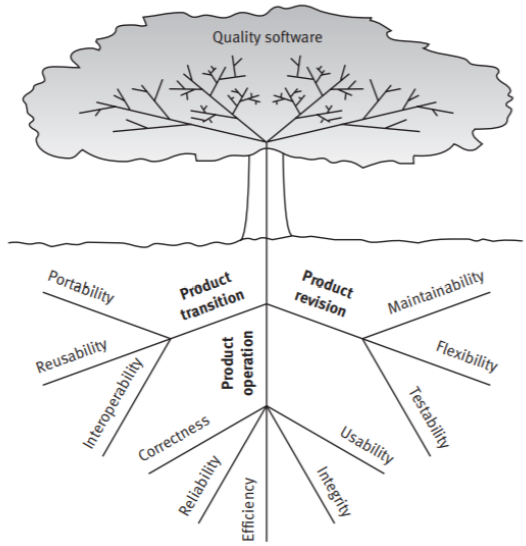

Figure 1: McCall's factor model tree [28]

2.1 Software Metrics

Software metrics can be used to locate possible issues and propose improvements in the quality of the software. There are multiple categories of software metrics, which are focusing in different aspect of the software life cycle [29, 30, 6].

Process metrics

Software process metrics provide measure of the development processes that the organization/team is performing. By monitoring and evaluating the success of the processes they can always be adjusted and improved. Examples of metrics include: the amount of time spent develop and design the product or particular feature, number of issues located/resolved, number of commits to a particular file and number of developers who changed particular file [14, 31, 32]. Since the development practices tend to change over time and due to specific language, lightweight processes are needed in order to adjust towards the merging trends and technologies. Agile development approaches tend to be very flexible and adjustable over time making it a suitable choice [33].

Project metrics

Project metrics allow the organization/team to define and measure the overall project flow. For example, how many developers are needed, do specialization testers are needed and what kind of hardware/software will be utilized by the specific project? Will the work be done remotely or on a physical location (offices)? Measuring these project specific characteristics and comparing them over time or over projects can give valuable information on how each project performed, hence making the decision for the future projects [6].

Product metrics

Product metrics allow developers to evaluate the internal properties of the software they are building. They are very localized and specific towards what they are measuring and represent, thereby allowing them to directly examine and improve the quality of the software. Examples of metrics that reveal these properties include complexity coupling, reuse, size, inheritance and cohesion. [6, 14]

This thesis will focus on product metrics, more specifically their applicability and usage on Android applications.

2.1.1 Existing Object-Oriented Software Quality Suits

Multiple object-oriented software quality metrics have been proposed [34].

Some of the metrics proposed are:

Lorenz and Kidd [6] have proposed metrics that capture the size (of methods and classes), internal characteristics (of methods and class), external characteristics (of a class) and inheritance.

Li [5] has proposed metrics for class inheritance and method complexity.

Abreu and Carapuca [11] have proposed metrics that capture the reuse of class and methods, complexity and productivity.

Henderson-Sellers [7] has proposed metrics for inheritance and complexity of classes.

Chidamber and Kemerer [8] have proposed a metric suite for calculating complexity and coupling.

Table 1 summarizes the categories of the metrics proposed and their description.

Metric category	Description
Metrics by Lorenz and Kidd [6]	
Method size	Metrics focusing on the size of the methods expressed in lines of code.
Method Internals	Metrics focusing on complexity of the methods.
Class Size	Metrics focusing on the size of the classes expressed in number of methods and variables.
Class Inheritance	Metrics expressing the nesting of the class.
Method Inheritance	Metrics expressing the number of methods inherited, overridden and added.
Class Internals	Metrics focusing on the internal characteristics of a class expressed in cohesion, parameters per method, commented lined per method.
Class Externals	Metrics focusing on the external characteristics of a class expressed in coupling and reuse.
Metrics by Li [5]	
Class Inheritance	Metrics expressing the number of methods inherited by a subclass and the number of new methods in the subclass.
Method Complexity	Metrics expressing the internal structural complexity of all methods.
Metrics by Abreu and Carapuca [11]	
Class Complexity	Number of methods in a class.
Class/Method Reuse	Number of times that a class/method is inherited.
Class/Method Quality	Number of failures caused by a class/method during a specified time slot.
Class/Method Productivity	Effort to build new average class/method. "Average" stands for average size and average complexity. Units of effort may be human month, human year, etc.
Metrics by Henderson-Sellers [7]	
Class Size	Metrics exposing the size of class in lines of code.
Inheritance	Average inheritance - system level. Sum of DIT/number of classes.
Complexity	Complexity expressed in Number of Methods per module and Number of Attributes (or variables) per module.
Metrics by Chidamber and Kemerer [8]	
Complexity	Metrics that are measuring the number of children of a module and inheritance.
Coupling	Number of modules that a particular module is coupled with.

Table 1: Summary of existing metrics

Figure 2 is depicting Sommervilles'[4] example model on how quality attributes

(maintainability, reliability, reusability and usability) relate to metrics. The model is linking four important quality attributes to five metrics.

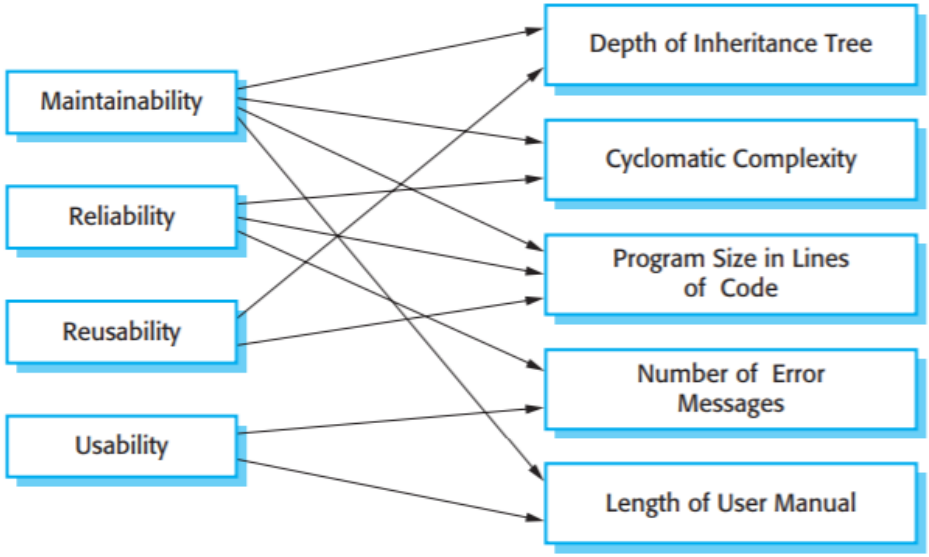

Figure 2: Relationships between quality attributes and metrics [4]

2.2 Metrics Validation

Multiple studies [12, 13, 5, 14, 15, 16, 17, 18, 19, 20, 21] have validated existing metrics. Validation and verification of metrics is an important step towards proving that indeed they are useful in development processes. By assessing if a particular metric or group of metrics are revealing the possible bugs of a module in a system, developers can obtain accurate knowledge on the complexity, maintainability and cohesion of a particular module in their system.

In their studies, Briand et al. [12, 13] empirically explored the relationships between metrics that measure coupling, cohesion, and inheritance and the probability of fault detection in system modules. Their goal was to understand the relationship between the metrics and the quality of the software developed.

Their analysis have shown that coupling and inheritance measures are strongly related to fault detection in a module. The depth of a module (for instance a class) appear to be an important quality factor since it can indicate the rate of change for a particular class.

"Multivariate analysis results show that by using some of the coupling and in-

> heritance measures, very accurate models can be derived to predict in which classes most of the faults actually lie. When predicting fault-prone classes, the best model shows a percentage of correct classifications about 80% and finds more than 90% of faulty classes." [12]

Li et al. [14] validated three groups of object-oriented metrics. The first group contains all the metrics proposed by Chidamber and Kemerer [8]. Whereas the second group of metrics focuses on the different forms of coupling and the last group discusses size metrics. They concluded that there is a strong relation between maintenance effort and metrics. The dataset in their study included two commercial software products, UIMS (User Interface System) and QUES (Quality Evaluation System). For the two systems' maintenance effort, data has been collected for the period of three years. The data is measured in number of lines changed per module. A line change is considered when performing addition or deletion, in addition a change in a line is counted as deletion and an addition. Estimates of the maintainability of object-oriented systems is based on the maintenance effort data. They further established that metrics values can help estimate maintenance efforts that would need to be invested in testing a module in the system. The limitation of their work is not investigating the abilities of one individual measure to predict maintainability. Therefore, the results must be used as a group of metrics and not as an individual meaning.

Zhou et al. [15], Aggarwal et al. [16], Elish et al. [35] and Li-jin et al. [17] have created different models to predict how maintainable a system is. Their dataset was the same used by Li et al. [14], and they reconfirmed the results. However, the limitation of Li et al. [14] apply to these studies as well.

Abreu et al. [18] and Harrison et al. [19] validated six metrics that focus on inheritance, coupling and are part of MOOD (Metrics for Object Oriented Design). Both papers conclude that MOOD metrics are indicators of error prone modules in a system, an additional observation is the fact that MOOD metrics are complementary to Chidamber and Kemerer [8]. I.e., both coupling metrics (CBO and CF) are revealing coupling characteristics but on a different level. CF treats the system as one and the metric is so called system-level metrics, whereas CBO is measured for each class. Accordingly, the metrics proposed by Chidamber and Kemerer can be used by developers on a day-to-day basis since they provide an overview of the project from class-level view [19, 36].

Dallal [20] explored if metrics that focus on size, cohesion and coupling can predict the maintainability of a class. The data set consisted of open source Java systems. The results demonstrate that modules with higher cohesion values and lower coupling have better maintainability which means that they are likely to be easily modified compared to those with opposite values. By looking at the practical

side of the results, they indicate that developers can improve the maintainability, i.e., decrease maintenance efforts of the developed modules by reducing the modules size and coupling and increasing the cohesion.

A more recent study by Dallal et al. [21] investigated the reuse-proneness of a class using metrics such as cohesion, coupling, and size. Their results show that most of the investigated metrics indeed are statistically significant predictors of the reuse-proneness of a class. Specifically, their results indicate that there is a positive relationship between coupling and class reuse-proneness.

Jabangwe et al. [10] have performed a systematic literature review on 99 papers that investigated the connection between metrics, and reliability, maintainability, effectiveness and functionality. They observe that Chidamber and Kemerer metric suite is the most popular across studies. Their results shows that complexity, coupling, size and cohesion metrics reveal a better understanding of reliability and maintainability for the systems when compared to the inheritance metrics.

By providing the metrics investigated, the dataset used and the main findings, table 2 summarizes the papers that have validated existing metrics.

Source	Metrics	Dataset	Main findings/conclusion
Briand et al. [12, 13]	Coupling, Cohesion and Inheritance	C++, total 180 classes	Coupling and inheritance strongly related to fault detection.
Li et al. [14]	Chidamber and Kemerer metrics, coupling, size	Two commercial software products with maintenance effort data available	Metrics can help estimate maintenance efforts.
Zhou et al. [15], Aggarwal et al. [16], Elish et al. [35] and Li-jin et al. [17]	Same metrics as Li et al. [14]	Same data set as Li et al. [14]	Same conclusion as Li et al. [14].
Abreu et al. [18],	Inheritance and Coupling	Data gathered in a controlled experiment performed at the University of Maryland on C++ systems [37]	Indications that metrics predict maintainability.
Harrison et al. [19]	Inheritance and Coupling	Three releases of laboratory electronic retail system (ERS)	Metrics are indicators of error prone modules in a system. MOOD metrics are complementary to Chidamber and Kemerer's metrics.
Dallal [20]	Size, Cohesion and Coupling	Open source Java systems	Modules with higher cohesion values and lower coupling have better maintainability.
Dallal et al. [21]	Cohesion, coupling, and size	Six Java open source systems	Positive relationship between coupling and class reuse-proneness, cohesion has negative impact on the reuse-proneness.

Table 2: Summary of papers validating existing metrics

2.3 Deriving Thresholds for Software Quality Metrics

From the literature review, there are different approaches on how to derive thresholds values for software quality metrics. These approaches are not language specific

as they are applied to object-oriented languages in general.

2.3.1 Deriving Thresholds Using Traditional Techniques

Erni et al. [38] proposed to use mean (μ) and standard deviation (σ) to suggest a threshold from project data. Moreover, they suggest threshold values for three metrics, coupling, complexity and cohesion by using one project with three versions. Mean (μ) represents the average value of the metric for a particular project and standard deviation (σ) is calculated from the same project for the same metric. $T_{min} = \mu - \sigma$ is used to find the lower limit, whereas, $T_{max} = \mu + \sigma$ is used to find the upper limit. Using statistical techniques is common for data that is not heavily-tailed or strongly skewed. However, previous research shows that when it comes to software quality metrics, the values usually follow right-skewed or heavily-tailed distribution. DIT being and exception. [26, 27]

Lanza and Marinescu [25] defined thresholds for four metrics that include inheritance, coupling, size and complexity. The project used to derive the values included in total of 82 systems in both, Java and C++. The mean (μ) was presented as a typical value and standard deviation (σ) as an upper limit.

2.3.2 Deriving Thresholds Using Error Models

Error data is another way to derive thresholds, by comparing the publicly available error data from different sources.

Shatnawi et al. [39] explored the use of error data from three releases of Eclipse[1] (versions 2.0, 2.1 and 3.0). Eclipse was chosen because it is an open source system and the error data is available. The error data is stored in Bugzilla which is an error-archive database for Eclipse.

Every error that is registered in Bugzilla has a severity of low, medium and high depending on the type and impact of the error. Using this data, the authors goal was to separate the modules in Eclipse into two categories: modules that have errors and modules that do not have errors. In addition to separating them into these two categories, they investigated whether they could classify each module into four additional categories: no error, low impact error, medium impact error and high impact error. The goal is to do all the categorization based on specific metric values.

Shatnawi et al. [39] further concluded that there was not enough data to derive threshold values to categorize the modules into error and no error. However, they managed to derive threshold values that can help developers locate high impact error prone modules and no error modules. The main drawback of using error models, as indicated by the authors is mainly data collection and its effects on generalizability of the derived thresholds.

[1] https://www.eclipse.org

Furthermore, Benlabri et al. [40] analyzed two C++ telecommunication systems and examined the relationship between thresholds and software bugs for DIT NOC, CBO, RFC and WMC, all of which are Chidamber and Kemerer metrics. The authors created two error probability models, one with and one without thresholds. The model that has threshold value for a metrics, has a zero probability of error to exist in the module if the metric value is below the threshold. Authors found no empirical evidence that would give more value to the model with thresholds.

The work of Benlabri at al. [40] and El Eman et al. [41] indicate that no empirical evidence was found in order to support a model that can predict faults based on threshold values. Additionally, they concluded that no optimal class size can be defined based on their study that compared class size and bugs. However, these results are tightly connected to the specific model that predicts the bugs. Other models potentially can give contrasting results.

Herbold et al. [42] used machine learning algorithm in order to determine a method that would derive threshold values, their method is independent of any language. The metric categories that were the focus of their study included metrics related to size, coupling, complexity and inheritance. They have concluded that their method is able to improve thresholds values of existing metrics, but the biggest drawback is that the method can only create a binary classification and therefore only categorize between good or bad. The lack of "shades of gray" makes this approach limited.

2.3.3 Deriving Thresholds Using Clustering Algorithms

Olivera et al. [43] proposed a method by using K-means clustering algorithm in order to group modules of the system into groups automatically. The approach is based on similarity between two systems.

> ”For example, suppose two systems, S1 and S2. Suppose that S1 was developed in accordance with the best principles/concepts of Software Engineering. Moreover, it was developed by a team of skilled developers, with high experience. Thus, the probability that S1 has a high degree of internal quality is high. Suppose also that using the approach proposed in this paper, we identify that S2 and S1 are similar. Thereby, we claim that this similarity is a strong indication that S2 also has high levels of internal quality.” [43]

Yoon at al. [44] investigated the usage of K-means clustering algorithm to identify outliers when measuring the values of the metrics. By observing the distribution, outliers can be defined as external or internal. External outliers appear in isolated clusters while internal appear far away from other observation withing the same cluster.

Both studies indicated the important shortcomings of using K-means clustering algorithm. The algorithm requires an input parameter that affect both, per-

formance and accuracy of the result. The outliers are identified manually and the algorithm must be executed again after excluding them. In addition the most limiting aspect is that different thresholds can be extracted for the same dataset and metric.

2.3.4 Deriving Thresholds from Repositories

The following method includes large number of systems as part of a dataset. The dataset is divided into categories based on the type of application and its functionality. Further sub-categorization is done by defining a size parameter (lines of code, number of classes, number of modules). The values for metrics are calculated for each application in the appropriate category/subcategory

Ferreira et al. [26] have derived thresholds for six metrics, including, LCOM, DIT, CF, AC, NPM, and NPF. The dataset contained 40 open source Java systems varying from 18 to 3500 classes. The metrics were calculated from the bytecode of the targeted systems. In order to derive the thresholds, a scatter plot was generated that revealed the frequency of the metric values.

Filo et al. [27] extended and improved the work of Ferreira et al. [26] by extracting the thresholds for 17 metrics and using 111 open source Java systems. The first improvement is on how the ranges of the thresholds are named, as mentioned, Ferreita et al. [26] are using "good", "regular" and "bad" while this work is using "good/common", "regular/casual" and "bad/uncommon" ranges in order to express better the concept of frequency when deriving the thresholds. The second improvement relates to the improvement on how to read the frequency chart, where two percentiles were defined based on their own analysis and the analysis of Alves at al. [24].

In their work, Alves at al. [24] have derived thresholds using 100 systems developed in Java and C#. From the data set, 60 Java systems were proprietary and 22 open source while, 17 C# systems were proprietary and 1 open source. It is worth mentioning that the authors used a tool[2] that is developed by the company that they are working for and 77 systems out of 100 are commercial and developed for their customers.

The general limitation of this approach is obtaining large number of relevant systems.

2.3.5 Deriving Thresholds from Experience

Multiple studies defined the thresholds based on their experience [25, 45, 46, 47]. According to Lanza and Marinescu [25], generally accepted thresholds that are based on information which are common and generally accepted knowledge is one of the ways to define threshold values. McCabe [45], developed the concept of CC

[2] https://www.sig.eu

and based on his experience "the particular upper bound that has been used for CC is 10 which seems like a reasonable, but not magical, upper limit". The idea is to show developers that if a module exceeds CC of 10, it probably needs splitting.

Nejmeh [46] proposed a metrics that is building on top of McCabe's CC. According to the author, NPATH threshold value is 200. The value is derived from his experience from former studies done at AT&T[3].

Coleman et al. [47] have worked on maintainability index metric. The metric is computed using the values of CC, lines of code and lines of comments. According to their experience, the value ranges are from 65 to 85. Methods that have values less than 65 are hard to maintain. Methods that have values between 65 and 85 are moderately maintainable and those who have value above 85 are highly maintainable.

The general weakness of deriving thresholds based on experience is the fact that reproduction of the result is very limited. In addition, since usually there is no information about the sample size, generalizability is disputable. The final disadvantage is the lack of scientific bases that can lead to disagreement about the validity of the thresholds/values.

2.4 Mobile Software Quality Model

Two software quality models have been proposed for mobile applications. They capture the important aspects that are contributing to increase the quality.

Franke et al. [3] proposed a model that captures usability, efficiency, data persistence, adaptability, portability and extensibility. Figure 3 depicts the model. In a follow up study, Franke et al. [48] explored the possible metrics that can represent the attributes depicted by the model. Including CC, WMC and LCOM.

[3] https://www.att.com

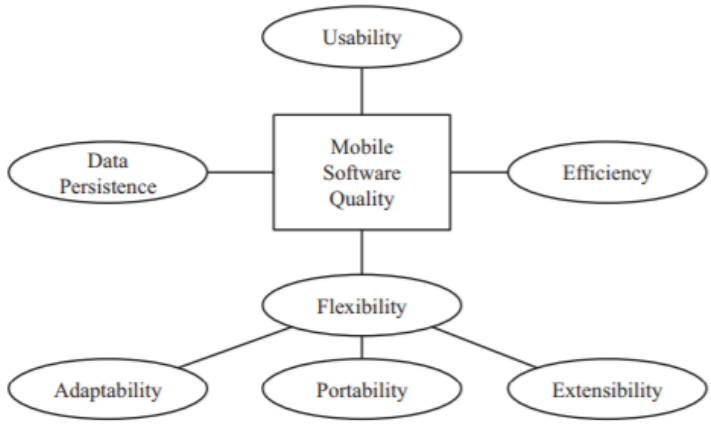

Figure 3: Mobile software quality model proposed by Franke et al. [3]

Zahra et al. [2] have proposed a slightly more detailed model that captures features depicted in figure 4.

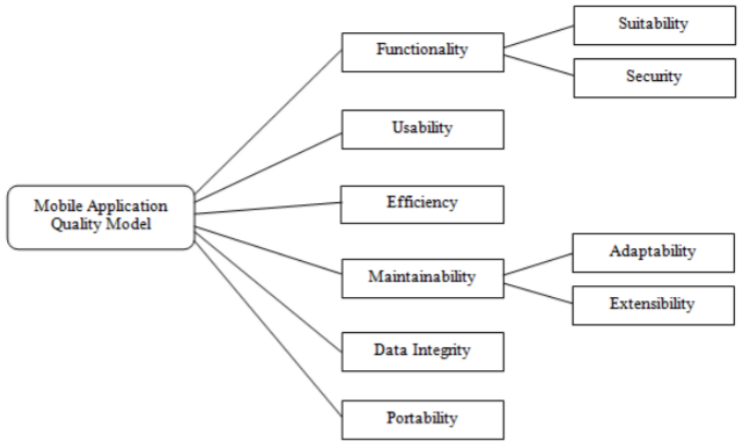

Figure 4: Mobile software quality model proposed by Zahra et al. [2]

Usability

In both models, usability is defined as the extend to which the application is understood, learned, used and be attractive to a user. Usability is all about improving the interaction between the user and the application. [2, 3]

Efficiency

Efficiency is seen from the resources point of view, where the optimal usage of the rather limited resources is ideal. The importance of battery usage is the main focus. Additionally, memory and processing power are also taken into consideration. [2, 3]

Extensibility
Extensibility focuses on improving the functionality of an application both when adding new functionalities and improving the usability. Adding new functionality includes new features and/or adjusting the application to the ever changing operating system versions. [2, 3]

Data integrity/persistence
Data persistence and data integrity are both focusing on the application's ability to keep information even after interruption from a third party app. For instance, when the phone is receiving a phone call, the application should pause and resume after the call is finished. [2, 3]

Portability
Portability as described by both models, takes into consideration the different mobile devices and their performance. The ultimate goal is to make the application run on as many devices as possible and yet keep all other attributes of the quality model on a high level. [2, 3]

Adaptability
Both models are describing the idea that mobile phones are compact devices that people carry with them almost all the time. This leads to changes in what kind of environments they are used, different connectivity possibilities. Applications are expected to adapt on the orientation of the device and the different types of inputs. [2, 3]

Functionality
Functionality is split into suitability and security on the model proposed by Zahra et al. [2]. It reveals the extend to which the application accomplishes the requirements specified in the planning phase. Further, suitability is checking to what extend a particular app/functionality fulfills the needs of the users.

The limitation of these models is mainly the lack of providing and testing suitable metrics that would represent these important and specific aspect of mobile applications. However, they do indicate that the traditional metrics including metrics like McCabe CC, WMC or LCOM can be used for some of the quality aspects. This

still does not complete the range of metrics. More specialized metrics are needed to be able to measure all aspects proposed by both models [48, 2, 3].

A step forward towards specific metrics has been done by Ryan at al. [49], who proposed metrics that measure resource utilization in terms of network, memory and processor.

2.5 Applicability of Object-Oriented Software Quality Metrics in Android Applications

When researching for papers that have used metrics in Android applications specifically, three papers came out as most relevant. The first one [50] applied the metrics on prototype application. The second [51] provided a dataset with computed metrics from decompiled .apk files of large number of Android applications. Lastly, the third [22] explored the possibility of locating vulnerable classes using metrics.

Jost et al. [50] have successfully applied 8 software metrics to Android, Windows and iOS applications. The metrics included, DIT, NOC, LCOM, WMC, NOM, LOC and CC. They were applied separately to the three applications that the researchers have developed. The applications had the same logic and were performing the same tasks, but they were written in three different programming languages, according to the targeted platform. The goal was to test if traditional software metrics can be applied for mobile application development, hence the use of tools to calculate the metrics were chosen depending on the IDE. They concluded that the metrics can successfully be applied to the applications they have developed, keeping in mind that they are prototypes only and do not represent the variety of mobile applications that exist.

Giovanni et al. [51] created a dataset that among other metrics contains values for Code Quality Metrics[4] such as DIT, NOM and CBO. The metrics values were obtained from decompiling the .apk files. The reasoning behind their approach was to capture possible code optimizations, eventually applied by the compiler.

What is missing in their dataset[5] is module level values for the metrics. Providing module level instead of system level can help developers see which modules are prone to errors and possibly fix them. Having only a system level value does not give us a detailed overview of the modules inside and it becomes harder to locate the error prone modules. In addition, system level CBO does not provide the information that is needed. We are not sure if CBO for a particular application is the lowest/highest/average/sum of the CBO of the modules. A more appropriate system level metrics would be CF.

Scandariato et al. [22] investigated if metrics can potentially predict faulty

[4] https://github.com/sealuzh/user_quality/wiki/Code-Quality-Metrics
[5] https://github.com/sealuzh/user_quality/blob/master/csv_files/code_metrics.csv

class. Some of the metrics included in their study are LCOM, CBO, RFC and LOC. They conclude that a model that is based on metrics can identify faulty class, however, the size of the data set used in the study is on the small side. This study need to be replicated on a larger pool of applications and a bigger number of versions needs to be performed in order to verify the results obtained. The aim is to investigate if same/similar results can be found if multiple applications from different categories are used.

Taking into consideration the limitations and conclusions of these three papers, there are indications that the traditional software quality metrics are applicable to Android application, both on the source code and the compiled version of the application. In addition, there are indicators that metrics can reveal possible fault-prone modules. In this thesis, we chose to compute the metrics on source code. We argue that it will aid developers in their day-to-day coding. Developers would not need to wait to have a finished version and then to compute the metrics, but instead continuously monitor the values of the metrics as they progress with the development.

2.6 Metrics Investigated in this Thesis

The metrics investigated by this thesis are proposed by Chidamber and Kemerer [9, 8] and Henderson-Sellers [7]. They have been extensively validated and used in previous research, also they capture three important aspects of software quality, including, complexity, inheritance and coupling. These aspects are directly correlated to the factors depicted by McCalls's quality model depicted in figure 1.

In addition, they can help developers monitor the applications on a day-to-day basis possibly with every automated build of the application. Figure 5 illustrates the quality aspects that each metric is focusing on.

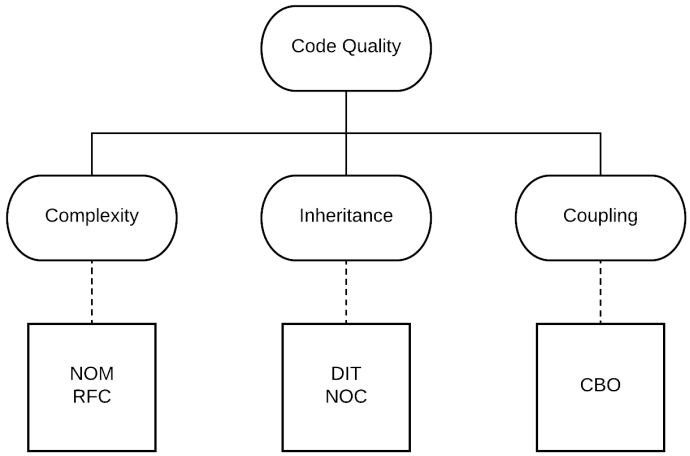

Figure 5: Quality aspect categorization of metrics

NOM [7, 52]

Number of methods represents the sum of the public and private methods in a class. The authors are also proposing that NOM can be divided into at least two metrics, number of internal or private (hidden) methods (NHM) and number of private or external methods (NEM).

$$\mathrm{NOM} = \mathrm{NHM} + \mathrm{NEM}$$

Viewpoints:

1. Increased number of methods decreases maintainability.
2. The size and the complexity of the class increases.
3. More testing efforts are needed with increase of NOM

RFC [8, 9, 34, 53]

RFC is defined as the number of methods that can potentially be called when the class receives a message. RFC can be defines as $|RS|$ where RS is the response set of the class.

RS can be expressed as: $\mathrm{RS} = \{\mathrm{M}\} \bigcup_{\mathrm{all\ i}} \{\mathrm{R_i}\}$

Where $\{\mathrm{R_i}\}$ is set of methods called by methods $_{\mathrm{i}}$ and {M} is set of all methods in the class.

Viewpoints:

1. Increase in the number of methods that can be called increases the complexity.
2. More testing resources should be spent as the RFC increases.

DIT [8, 9, 34]

DIT represents the length of the path from the node to the root of the tree. In other words, DIT is a measure of how many ancestor modules can potentially affect the particular module. In cases involving multiple inheritances the DIT will be the maximum length from the node to the root of the tree.

Viewpoints:

1. The number of methods inherited by a module is likely to increase as the DIT increases.
2. The design complexity increases as DIT increases.
3. The potential reuse of inherited methods is increasing by higher DIT.

NOC [8, 9, 34]

NOC represents the number of immediate subclasses of a class in the hierarchy. It measures how many subclasses are going to inherit the methods of the parent class. Viewpoints:

1. The greater the number of children, the greater the reuse.
2. If a class has large number of children, more testing efforts should be invested in testing the methods in that class.

CBO [9, 8, 34]

CBO is defined for a module as a count the number of other modules to which it is coupled. As defined by the creators of CBO [9] a coupling between classes happens when the methods or variables from one module are accessed by another one.

Viewpoints:

1. Enormous coupling prevents modular design.
2. Independent class is easier to be reused.
3. Coupling can be useful to understand how much resources should be invested into testing. Higher coupling leads to more rigorous testing.

Based on the work of Lincke et al [54], table 3 was generated and it defines the effect that metrics have on quality attributes.

Metric	Maintainability	Portability	Efficiency
NOM	Declines with increasing NOM	Declines with increasing NOM	Might decline with increasing NOM
RFC	Declines with increasing RFC	Declines with increasing RFC	Might decline with increasing RFC
DIT	Declines with increasing DIT	Declines with increasing DIT	Might decline with increasing DIT
NOC	Declines with increasing NOC	Declines with increasing NOC	Might decline with increasing NOC
CBO	Declines with increasing CBO	Declines with increasing CBO	Might decline with increasing CBO

Table 3: Effects of metrics on quality attributes

3 Methodology and Implementation

3.1 Methodology

As discussed in chapter 2 section 2.3, there are multiple methodologies to derive threshold values for software quality metrics. In this thesis we chose to define the thresholds by calculating the software quality metrics on a large number of applications. The calculations were done on the source code of the applications.

The reasoning behind favoring source code instead of decompiling the .apk file, is the purpose of these thresholds, which are designed to help developers in their day-to-day development. The idea is to continuously monitor the progress of the modules in the system, calculate the metric(s) on every build and seek possible improvements.

Figure 6 illustrates the overview of the methodology used to derive thresholds.

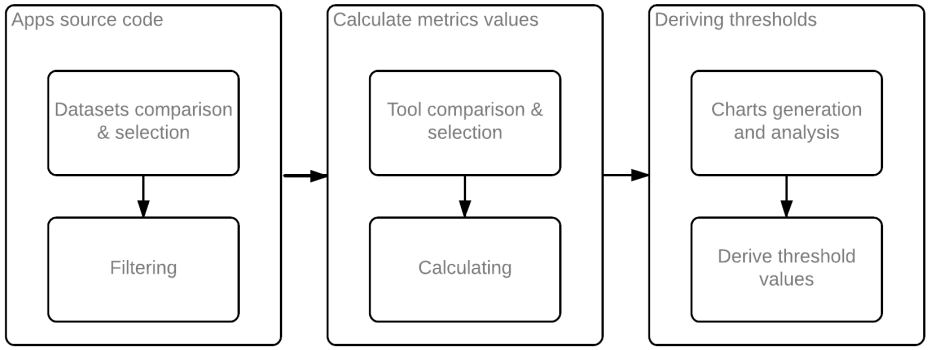

Figure 6: Methodology overview

3.1.1 Dataset Construction and Justification

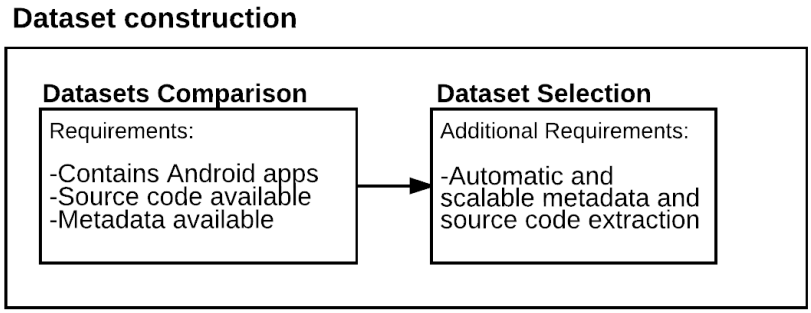

Figure 7: Dataset construction

Two main data sources were analyzed. Both contain Android applications and have been used in previous research.

The first data source evaluated is SourceForge[1]. It contains collection of open-source application for different operating systems: Windows, Linux and Mac. In addition, applications for Android platform are present. SourceForge has been extensively used in research that investigated Java systems [26, 27, 55, 56]. To the best of our knowledge, SourceForge does not provide scalable approach on how to extract Android applications only. The systems are categorized based on the programming language, i.e., Java can be specified as the target programming language. By doing that all system coded in Java would be extracted, hence, it is not possible to distinguish between applications written in pure Java and those written for Android. On the website, there is a possibility to manually specify characteristics of the applications that are important for this thesis. E.g. Android applications that have source code available, but not all characteristics. The website does not provide an option to sort them by last updated date. In addition, the biggest drawback is lack of automation. Manually downloading applications is not scalable.

Another data source is F-Droid[2], which is used by multiple studies related with Android application [51, 57, 58, 59, 60]. Once compared with SourceForge we observed that it is more suitable to be the data source for this thesis. F-Droid contains free and open source applications for Android devices. It specializes in one platform only, making it more tailored towards Android applications and all the meta data that they have. A XML file can be downloaded that contains information about all the application that are available on F-Droid. The latest file can always be accessed

[1] https://sourceforge.net
[2] https://f-droid.org

at the website[3]. Being a XML file, it can be parsed and the needed information can be extracted.

Figure 8 gives an example of how the information is stored for one application. Data about application ID, date when the application is added, date when the application is last updated, name and category are provided for each application. The package represents the versions of the application. In figure 8, there is only one version, if the application has multiple versions, multiple packages would be present.

Filtering

The focus of this thesis are applications that were updated in the last three years, accordingly, last updated date has to be after 31.12.2014.

The reasoning behind the specified time frame is to exclude applications that are not maintained and developed further. In addition, by excluding the applications, the risk of including applications that contain external libraries in their repositories is decreased. Newer applications generally use build tools, for instance Gradle[4], to manage external libraries and dependencies. This however does not eliminate the projects that keep their external libraries in their repositories.

3.1.2 Tool Selection

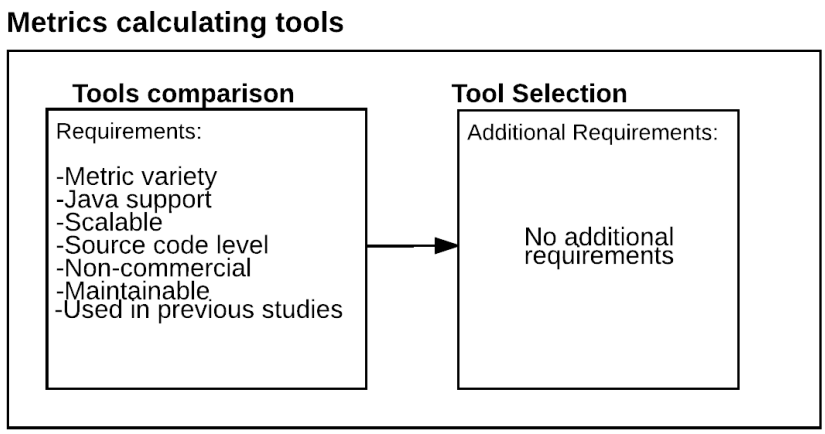

Figure 9: Tool selection

The candidate tools to be used for calculating the metrics were selected based on the following characteristics:

[3] https://f-droid.org/repo/index.xml
[4] https://gradle.org

```
<application id="subreddit.android.appstore">
    <id>subreddit.android.appstore</id>
    <added>2016-09-27</added>
    <lastupdated>2016-10-06</lastupdated>
    <name>/r/Android App store</name>
    <summary>Download apps curated by /r/android</
       summary>
    <icon>subreddit.android.appstore.6000.png</icon>
    <desc>&lt;p&gt;App inspired by this &lt;a </desc>
    <license>Apache2</license>
    <categories>System</categories>
    <category>System</category>
    <web></web>
    <source>https://github.com/d4rken/reddit-android-
       appstore</source>
    <tracker>https://github.com/d4rken/reddit-android-
       appstore/issues</tracker>
    <marketversion>0.6.0</marketversion>
    <marketvercode>6000</marketvercode>
    <antifeatures>NonFreeAdd</antifeatures>
    <package>
        <version>0.6.0</version>
        <versioncode>6000</versioncode>
        <apkname>subreddit.android.appstore_6000.apk</
           apname>
        <srcname>subreddit.android.appstore_6000_src.tar
           .gz</srcname>
        <hash type="sha256"> 3
           af6218c89697577fd15e9582caad6649bc62483c69dbe
            </hash>
        <size>7077312</size>
        <sdkver>16</sdkver>
        <targetSdkVersion>24</targetSdkVersion>
        <added>2016-10-06</added>
        <sig>ec5d720bd93bda25b1e68d6a449b7254</sig>
        <permissions>INTERNET</permissions>
        <nativecode>arm64-v8a,armeabi,armeabi-v7a,mips,
           x86,x86_64</nativecode>
    </package>
</application>
```

Figure 8: Metadata for app in F-Droid

1. **Metrics variety** - the tool must be able to calculate the metrics that are the focus of this thesis.
2. **Java support** - since Android applications are written in Java, the tool must support Java as a programming language for calculating the metrics.
3. **Scalable and automated** - the tool must be able to process large number of applications and must support the option to be automated.
4. **Source code level** - the tool must compute the metrics on the source code, and the tool can not rely on the application to be compiled.
5. **Non-commercial** - the tool can not be commercial, it must be available to be used without restrictions, to allow the replicability of this work and additionally, to allow developers to use it in order to compute metrics for their own projects.
6. **Maintainable** - the tool must be operable and is being maintained by developers/researchers. This includes having active developers/researches as a contact person(s), who know the tool architecture and can provide further assistance if necessary.
7. **Being used in previous research** - the tool must have been used in previous studies, not necessary for deriving thresholds but for calculating metrics values in general.

When doing the background and related work, the tools used by previous studies were evaluated based on the criteria explained above. In addition to the tools provided by previous research, searching for tools was performed using keywords, e.g., "quality metrics tool calculator", "code metrics calculator source code", "Chidamber and Kemerer metrics calculator". Following blog posts, Quora[5] answers and Stack Overflow[6] helped us explore the different tools available.

The tool selection was performed by first reading the information on the tool's website and in the previous research (if the tool was used in previous research),and if the tool was available for download, we proceeded to installing and testing it.

The tools that were evaluated are described below.

(PRO Metrics)

PROM framework was the first tool that was considered. Unfortunately. after tracking back the references [61, 62, 63, 34], it was impossible to find the tool, the papers only explained how it works and its specifications. After contacting the creators of the tool, we were informed that the tool has been discontinued and no longer available for usage.

CCCC

[5] www.quora.com
[6] www.stackoverflow.com

The next tool was C and C++ Code Counter (CCCC)[7]. Even though the name does not suggest, it supports Java source code. The main disadvantage is that the tool is not extendable and automated.

Source Monitor

Source Monitor[8] allows for inspection of the complexity of the project both on module level and as a whole. As with CCCC, the main disadvantage of this tool is the lack of automation.

Plugins

It is worth mentioning that multiple plugins for IDE[9,10,11] were found. They provide an integrated approach to metrics calculation in parallel with coding. Developers can install the plugins on their IDE and obtain metric values as the system progresses. The reason why this type of tools are not suitable for this thesis is because they are not extendable and can not process large number of application automatically.

Based on the seven characteristics presented earlier and the comparison of the previously presented tools, **Analizo**[12] [64] was chosen as the tool that we used to perform the calculations of the metrics. More specifically, Analizo version 1.20.0-rc1. It was chosen as the best suitable tool for this thesis. It calculates the metrics that are focus of this work, supports Java, it is scalable and provides an automated process of calculating the metrics on source code level. Additionally, it is free and maintained by researchers at University of Sao Paulo and University of Brasilia. When we contacted the team behind it, we were assured to have a point of contact if we need clarifications. Also, when we got in touch with them, they informed us that they are also using Analizo, parallel with us, performing a study to calculate software quality metrics on Android applications using Analizo.

Analizo's architecture

Analizo architecture is presented in Figure 10, each layer in the diagram uses only the services provided by the layers directly below it.

The **Core** layer is independent from any other layer. The layer implements most of Analizo logic. It stores information regarding the source code of the application that is being analyzed. Information include list of all modules, attributes, methods, dependency information. [64]

[7] https://sourceforge.net/projects/cccc/
[8] http://www.campwoodsw.com/sourcemonitor.html
[9] http://metrics2.sourceforge.net
[10] http://www.arisa.se/products.php?LOG=1&item=9
[11] http://eclipse-metrics.sourceforge.net
[12] http://www.analizo.org

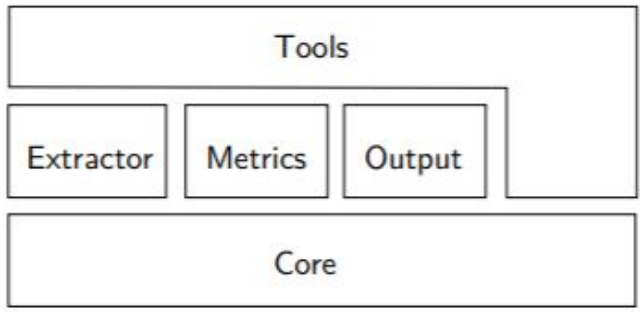

Figure 10: Analizo architecture [64]

The **Extractors** layer contains the different source code information extraction strategies built in Analizo. Extractors get information from source code and store them in the Core layer. Currently, the extractor that is used is Doxyparse[13]. Doxyparse is a source code parser for C, C++ and Java. Doxyparse is based on Doxygen[14], a multi-language source code documentation system that contains a robust parser. [64]

The **Metrics** layer processes the information extracted from the extractor layer in order to calculate metrics. Detailed information about supported metrics can be found on the website[15] or by using the command *analizo metrics –list*.

The **Output** layer is responsible for handling different file formats. The output format used is YAML[16] [64].

The **Tools** layer includes a set of command-line tools[17] that are part of Analizo interface. These tools use services provided by the other layers: they instantiate the core data structures, the extractor, optionally the metrics processors, an output format module, and orchestrate them in order to provide the desired result. [64]

> "Those tools are designed to adhere to the UNIX philosophy: they accomplish specialized tasks and generate output that is suitable to be fed as input to other tools, either from Analizo itself or other external tools" [64].

Features provided by Analizo

Analizo supports multiple features, the ones that are relevant for this thesis are explained below.

Analizo supports multiple programming languages, C, C++ and Java making it suitable to use it on Android applications. The analyzing of the source code is done

[13] https://github.com/analizo/doxyparse
[14] www.doxygen.org
[15] http://www.analizo.org/faq.html
[16] http://www.yaml.org/
[17] http://www.analizo.org/man/analizo-metrics.html

by Doxygen.

Analizo can calculate both project level metrics, which are calculated for the entire project in addition to module level metrics, which are calculated individually for each module. In the case of Java, the module level calculation of the metrics is done for every class, interface and enum. For the Android application, the following module level metrics are calculated: NOM, DIT, LOC, NOA, NOC, NOM, RFC, SC.

There are two ways to calculate the metrics in Analizo. The first approach, called "batch" processing, meaning the tool can calculate the metrics for more than one project simultaneously. The feature that is missing when doing "batch" processing is specifying the language that the source codes are written in. The second approach is calculating the metrics for each project separately. In order to automate this approach, a script was created that will navigate to the projects and treat them as single projects. The script can be found in Appendix A.1.1. This approach is more flexible, it allows to specify the targeted language for which the metrics should be calculated. Specifying the language ensures that possible application which are not written in the targeted language are excluded from the calculations.

After doing test cloning, some applications were included whose source codes was in languages different than Java. Accordingly, the second approach for calculating the metrics values had to be used where the language (Java) was specified. The tool automatically excluded the source projects written in languages different than Java. In addition, it keeps track of all excluded projects.

Calculations and storing metrics values

In order to calculate the metrics values on the source code of the application, we first need to obtain a copy of the source codes locally. In order to achieve that, each application is cloned to the appropriate folder based on the categorization of F-Droid.

Once the metrics are calculated they are stored in local database.

3.1.3 Analysis and Deriving Thresholds

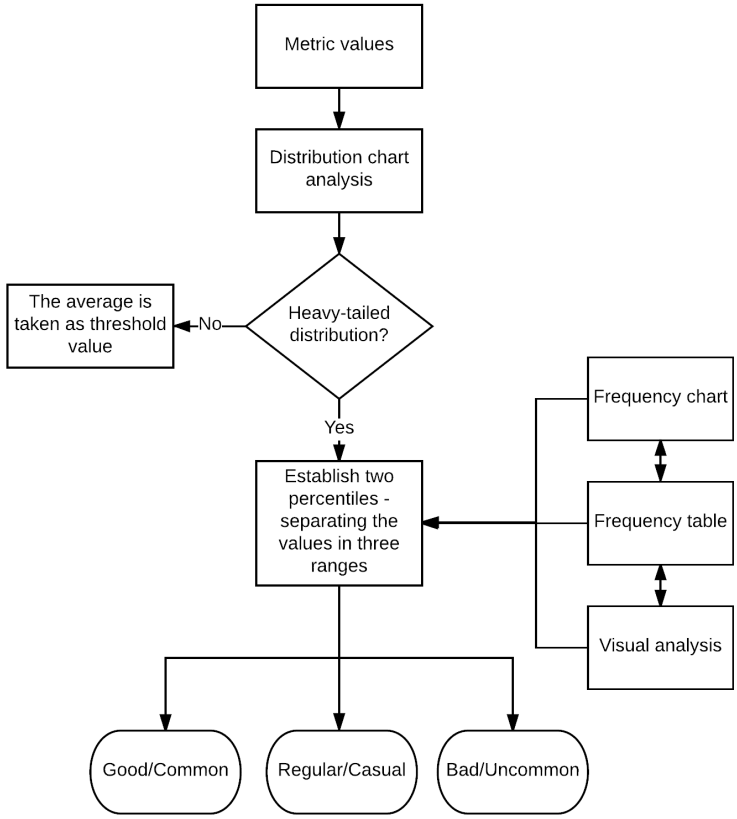

Figure 11: Thresholds identification

Considering every metrics is independent, the analysis of metric values is performed separately for each metric in every category of applications. Using EasyFit[18] the histograms of the values obtained for each metric were fitted into the family of two-parametric Weibull distributions as proposed by the preceding researches [26, 27]. The probability density of two-parameter Weibull distribution is given by:

$$f(x) = \frac{\alpha}{\beta}\left(\frac{x}{\beta}\right)^{\alpha-1} e^{-(x/\beta)^{\alpha}} \qquad x, \alpha, \beta > 0 \tag{3.1}$$

Parameter β is called a scale parameter and its role is just 'rescaling' the density

[18] www.mathwave.com/easyfit-distribution-fitting.html

graph, while α is a shape parameter and, as such, it describes the general appearance of the density graph and, correspondingly, the properties of the distribution.

It has to be pointed out that the distribution is unimodal when the value for $\alpha > 1$, while for $0 < \alpha \leq 1$ the density function 3.1 is monotone decreasing for all $x > 0$. It can be noticed that for large values of α, the values of the mean and mode are rather close, and consequently, the mean value can be used as an approximation for the most typical value, although the expected value as well as the moments of all orders exist for all α, $\beta > 0$. However, the moment-generating function does not exist when $\alpha < 1$ as the distribution for these values of α is heavy-tailed. For Weibull family of distributions, where being heavy-tailed and unimodal coincide.

In addition, for every metric in each category a frequency chart and frequency table is generated using SPSS[19]. The chart and the table shows the percentage of the modules have particular value for specific metrics in a category.

Based on previous research, frequency chart, frequency tables and visual analysis, two percentiles are defined.

If the defined percentiles are changed/adjusted for a particular metrics in a category, this information will be specified and explained in chapter 4, results and discussion.

One frequency chart is also created on the dataset as whole, this will give us the opportunity to observe the change (if any) between the categories and the dataset as a whole. Accordingly, we will conclude if we can derive general thresholds for Android applications.

Not all application belonging to one category can be treated the same. In each category there are applications that are simple and do few tasks and applications that are larger, i.e. contain more modules. Correspondingly, in addition to the thresholds for each category of applications and on the dataset as a whole, subcategories are defined based on the number of modules. The aim with this approach is to help developers locate their application easily. We make an assumption that metrics values differ based on the size of the application. If the application has only few modules, the values of the metrics will differ compared to a large application.

[19] https://www.ibm.com/products/spss-statistics

3.2 Implementation

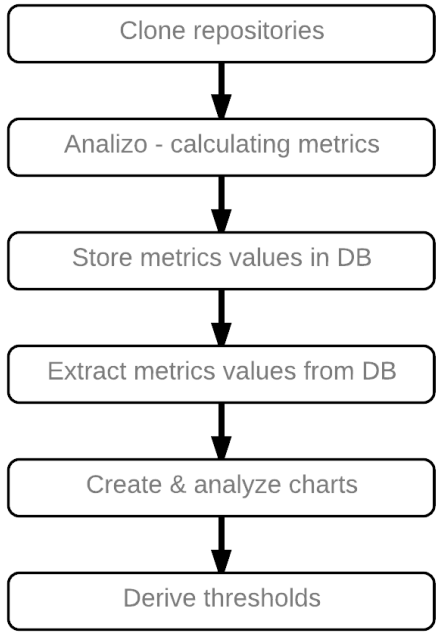

Figure 12: Implementation process

The dataset construction and metrics calculations were done on Debian 9 instance running on skyhigh.hig.no. SkyHigh[20] is a cloud service provided by NTNU in Gjøvik. Once ready for importing, the values were imported in local MySQL[21] database running locally on Windows 10. SPSS was used to create the graphs. The implementation process is summarized in figure 12.

A GitHub repo[22] contains the metrics values for all categories and dataset as whole, list of applications and their meta data extracted from F-Droid and an example how metrics values were stored in the database.

3.2.1 Data Collection

F-Droid provides very efficient way on how to extract the metadata for all applications that can be found on the website. By using single command *wget https://f-droid.org/repo/index.xml*, the latest *index.xml* file can be downloaded. The file contains all the necessary information for every application. An example is shown

[20] https://www.ntnu.no/wiki/display/skyhigh/Openstack+documentation
[21] https://www.mysql.com
[22] https://github.com/milestojkovski/MACS490

in Figure 8. Parsing the XML document to extract the necessary information was done with a Bash Shell[23], a script A.1.2 was created to extract the needed metadata: package (id), the date when the application was added in F-Droid (added), the date when the application was updated (lastupdated), the name of the application (name), what category does it belong to (category), the link to the repository (source) and the version (marketversion). From the package, only the most recent version was extracted (the first package tag), size, and the date when it was added. The market version must be the same as the version.

Applications that have their source code on GitHub[24] or GitLab[25] were extracted. These two platforms are Git[26] version control repositories, making the cloning of the source code automated.

When parsing the XML, the data was tailored so it can be imported in the local database. *LOAD DATA LOCAL INFILE* was used to import the data from the file created by the script.

Extracting the source code URL for the application that fulfill the predefined time period from the database was trivial.

3.2.2 Calculating and Storing Metrics Values

Cloning the source code from the applications is the first step towards calculating the metric values. A script for cloning the applications was created by adding *git clone* in front of the source code URL extracted from the database. In addition, the script also created the folders with the category name and cloned the appropriate applications in the appropriate folder (category).

The data set was split into 17 unique categories: System, Games, Reading, Time, Security, Multimedia, Navigation, Money, Graphics, Internet, Development, Theming, Connectivity, Science and Education, Writing, Sports and Health, Phone and SMS according to F-Droid categorization.

As shown in the figure 13 the projects are placed in the category that they belong to in accordance to F-Droid categorization.

[23] https://www.gnu.org/software/bash/
[24] https://github.com
[25] https://about.gitlab.com
[26] https://git-scm.com

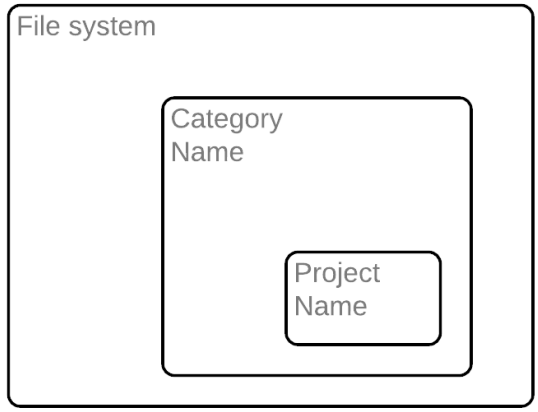

Figure 13: Folder Structure

After cloning, the script A.1.1 calculated the values for the each metric.

Once started in every category folder it calculates the metrics for each project independently. The core command is, *analizo metrics -o output –language java -x test:androidTest*. It specifies that the target language is Java. This way all other projects (projects written in different language) are automatically excluded. Thus, a list of excluded projects is generated. It also excludes test files that are located in folders with named *test* and *androidTest*. However, this does not guarantee that all test files will be excluded. If the test files are located in the same folder as the source files they will be taken into consideration in the calculation.

The script extracts the module level (class, interfaces and enums are all considered modules) metrics from each project and prepared them for importing into the database. After successfully computing the metrics, there are 17 files to be loaded in the database (one file for each category). The file contained all module level metrics for all modules of the project. In order to load the data into the database, 17 tables were created, a table for each category. Each table contains the module level metrics values for each project in that particular category. The last column of the tables is the name of the project, which helps to locate where does the values come from. If there are outliers in the data set, it would be valuable to manually inspect it.

After successful filtering, cloning and calculating the metrics we ended up with a data set of 865 applications that the metrics were calculated for. Table 4 represents the detailed numbers.

For each category the following information are presented:

- **Category name** - represents the 17 categories that the applications are separated in. The categorization is done in correspondence with F-Droid categories.
- **Apps in F-Droid** - the number represents the total number of applications that were extracted from F-Droid. The condition that the application had to be updated in the last 3 years must be met.
- **Apps cloned** - the number represents the total number of applications source codes that were cloned from the remote repository on GitHub and GitLab.
- **Percentage cloned**- represents the successfully cloned applications.
- **Apps metrics computed** - represents the number of applications that Analizo was able to parse and compute the metrics for.
- **Percentage computed** - represents the percentage of total applications that Analizo was able to parse and compute metrics for.
- **Number of modules** - represents the total number of classes, interfaces, enums in the category. The value was calculated for each project separately. The sum of all project in each category is giving us this number.
- **Modules metrics computed** - represents the total number of modules that Analizo could parse and calculate the metrics for.
- **Percentage computed** - represents the percentage of total modules that Analizo could parse and calculate metrics for.

By looking at the columns with percentages that are representing the degree of successfulness, we can observe that most of them are not 100% successful.

The reasons why some applications failed to get cloned are: wrong URL to the source code, empty repositories and migrated repositories.

The reasons why Analizo could not compute metrics for all cloned applications are: applications written in language different than Java. This can be applications that are using cross platform languages that allow developers to write in one language but deploy the app on multiple platforms and applications that Analizo could not parse.

The occurrences of empty modules and modules with methods and attributes being commented out is the cause why a small percentage (1.87%) of the total number of modules that metrics were computed for had value 0 for all metrics. Accordingly, these metrics values were deleted from the dataset.

3.2.3 Analysis and Deriving Thresholds

Extracting the values from the database was performed with queries, examples can be found in appendix A.1.3. To extract the metrics values for each category is

Category name	Apps in F-Droid	Apps cloned	Percentage cloned	Apps metrics computed	Percentage computed	Number of modules	Modules metrics computed	Percentage computed
System	145	141	97.24%	124	87.94%	4777	4671	97.78%
Games	108	107	99.07%	95	88.79%	5955	5841	98.09%
Reading	48	47	97.92%	40	85.11%	4432	4370	98.60%
Time	62	59	95.16%	52	88.14%	2862	2790	97.48%
Security	36	35	97.22%	33	94.29%	2517	2480	98.53%
Multimedia	131	129	98.47%	112	86.82%	8149	7971	97.82%
Navigation	69	61	88.41%	52	85.25%	4650	4556	97.98%
Money	37	32	86.49%	21	65.63%	1214	1196	98.52%
Graphics	11	11	100.00%	7	63.64%	135	134	99.26%
Internet	154	147	95.45%	121	82.31%	14706	14423	98.08%
Development	35	34	97.14%	29	85.29%	1951	1912	98.00%
Theming	37	33	89.19%	25	75.76%	483	468	96.89%
Connectivity	49	48	97.96%	37	77.08%	2903	2877	99.10%
Science and Education	52	50	96.15%	44	88.00%	2594	2566	98.92%
Writing	62	42	67.74%	34	80.95%	2269	2212	97.49%
Sports and Health	25	23	92.00%	20	86.96%	1419	1397	98.45%
Phone and SMS	20	19	95.00%	18	94.74%	1649	1628	98.73%
TOTAL:	**1081**	**1019**	**94.26%**	**865**	**84.89%**	**62665**	**61492**	**98.13%**

Table 4: Dataset

trivial since all metrics values were stored in their appropriate table (a table for each category).

Using EasyFit the Weibull distribution fitting was generated and at the same time the frequency charts and tables were generated with SPSS.

From here on, we proceed in the following manner, when the fitted two parameter Weibull distribution - see section 3.1.3 and formula 3.1 - is unimodal, that is, the shape parameter $\alpha > 1$. We use the the mean value of the obtained distribution as the threshold. Otherwise, the threshold values are obtained using percentile from the respective histograms.

Based on visual analysis of the frequency chart, frequency tables and previous research 70^{th} and 90^{th} percentiles are used to derive thresholds. For every metrics percentiles might be adjusted in cases where distributions are taller e.g. when more than 90% of the values are 0 and or 1. The percentiles are dividing the dataset into three parts, from the lowest value until the 70^{th} percentile are values that are common, from 70^{th} until 90^{th} are values casual and every value that is greater than 90^{th} percentile is uncommon.

For NOC 70^{th} and 90^{th} percentile had to be adjusted since across each category more than 95% of the values for NOC are 0 and/or 1. The 85^{th} and 95^{th} percentiles were used instead. By increasing the percentages this high, we can observe that the frequency of values 0 and 1 is extremely high.

For the subcategories, a scatter graph is created which plots the number of modules on the X axis and the value of the metrics on the Y axis. By observing the chart, clusters of applications were identified based on number of modules.

4 Results and Discussion

4.1 Results

The threshold values are presented for each category. Additionally, sub-categorization was performed based on the number of modules and thresholds are derived for each subcategory.

Category level

For every metric in every category a frequency chart, frequency table and a fitting to Weibull distribution was generated. An overwhelming number of charts and tables have been produced and it is not practical to place them in this report.

Instead, table 5 summarizes the α value for Weibull distribution for all categories.

We provide two Weibull histograms: figure 14 shows heavy-tailed distribution whereas figure 15 does not indicate heavy-tailed distribution.

Additionally, we provide three frequency charts:

- Figure 16 that shows the 70^{th} and 90^{th} percentile is applicable
- Figure 17 that shows the need for adjusted percentiles when deriving the thresholds
- Figure 18 shows the frequency of the metric for which we take the average as threshold value

Based on the Weibull shape parameter α, presented in table 5 we can observe that the distributions for all metrics except DIT are heavily-tailed across all categories. Accordingly, the average value can be defined as threshold for DIT only. The average was rounded so we get whole number, even though the α parameter changes across the categories.

The rest of the metrics follow heavily-tailed distributions and we need to derive thresholds using percentiles.

Weibull distribution for RFC in system category is shown in figure 14, α=0.77365 and β=15.403.

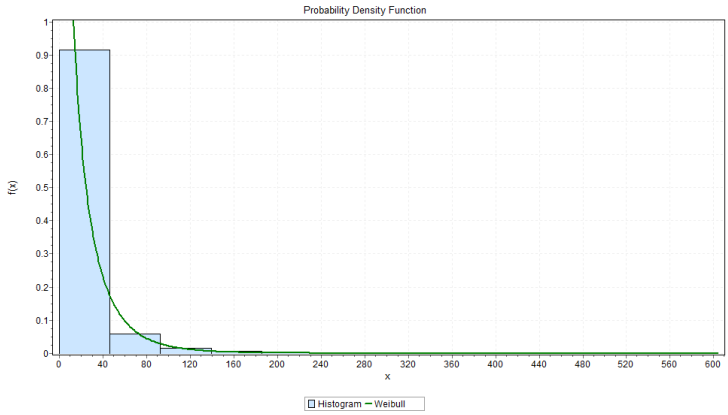

Figure 14: Weibull distribution for RFC - system category

Weibull distribution for DIT in system category is shown in figure 15 with parameters α=1.3969 and β=1.0044.

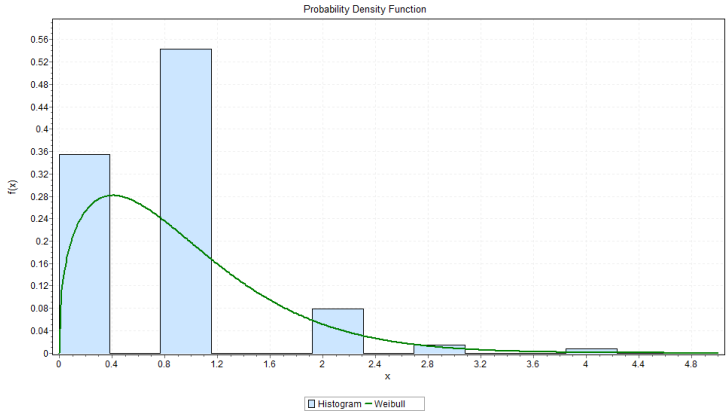

Figure 15: Weibull distribution for DIT - system category

Category	NOM	RFC	DIT	NOC	CBO
	α	α	α	α	α
System	0.82434	0.77365	1.3969	0.94989	0.68671
Games*	0.97161	0.76624	1.6239	0.75802	0.48517
Reading	0.73603	0.5245	1.1156	0.9593	0.57428
Time	0.86175	0.6091	1.5371	0.95428	0.66615
Security	0.79004	0.55464	1.3712	0.8989	0.56966
Multimedia*	0.95229	0.73314	1.4214	0.97171	0.62977
Navigation	0.87116	0.8065	1.0791	0.95177	0.59852
Money	0.88494	0.79416	1.547	0.99543	0.60288
Graphics	0.8647	0.7027	2.11	/	0.78354
Internet*	0.91783	0.69756	1.4196	0.98344	0.53426
Development	0.95366	0.71281	1.2678	0.79183	0.63009
Theming	0.99945	0.74032	1.5661	0.94318	0.88337
Connectivity	0.81512	0.77486	1.3049	0.9812	0.60533
Science and Education	0.6752	0.71409	1.1403	0.88946	0.54858
Writing	0.98843	0.73478	1.2824	0.96788	0.61633
Sports and Health	0.9117	0.8579	1.2591	0.80148	0.64828
Phone and SMS	0.97383	0.75429	1.284	0.9788	0.58726

Table 5: Weibull shape parameter α

**Because of licensing limitations, the evaluation trial period of EasyFit supports up to 5000 records. The results in the categories exceeding 5000 modules are obtained from 85.6% of the modules in games, 62.7% of the modules in multimedia and 34.6% of the modules in internet.*

Figure 16 shows the frequency chart for NOM in the system category, we can see that the data is more evenly distributed compared to NOC in figure 17 hence the 70^{th} and 90^{th} percentiles can be used.

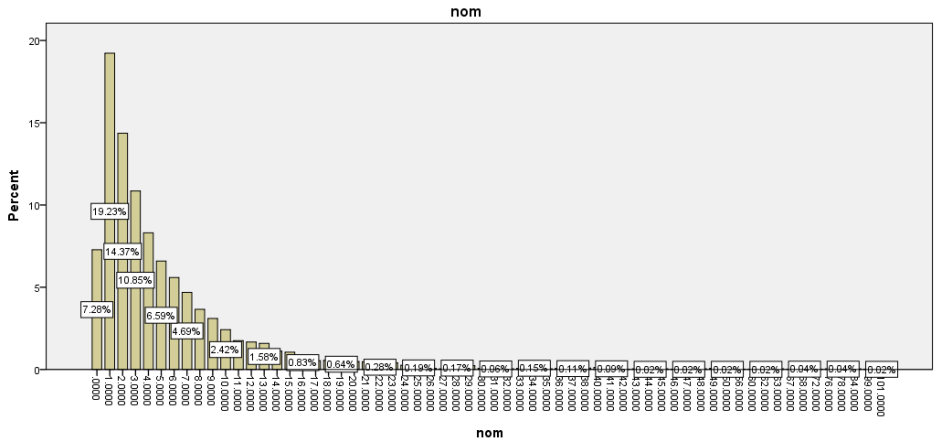

Figure 16: NOM System

Figure 17 shows the frequency chart for NOC in the system category. Since 92.21% of the modules in this category have NOC 0, the 70th and 90th percentiles are 0. Accordingly, the thresholds were derived using the 85th and 95th percentiles.

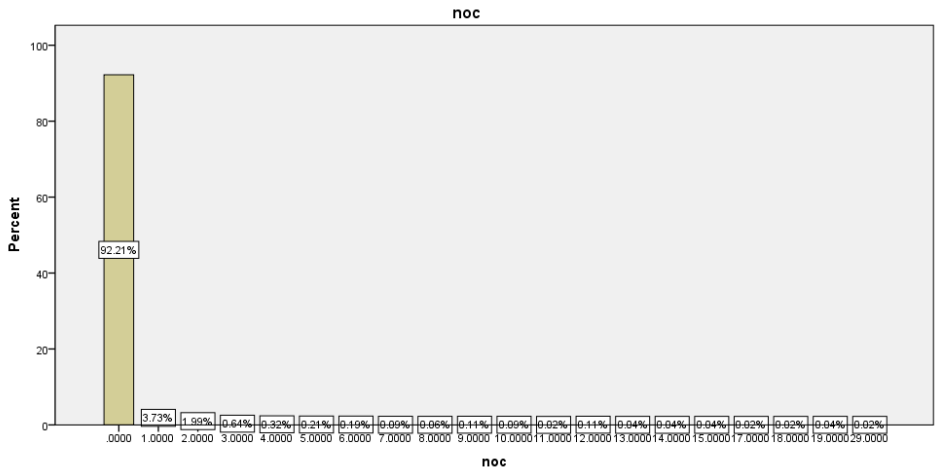

Figure 17: NOC System

Figure 18 shows the frequency chart for DIT in the system category, we can observe the difference between figure 16 and 17, the data does not follow heavy-tailed distribution.

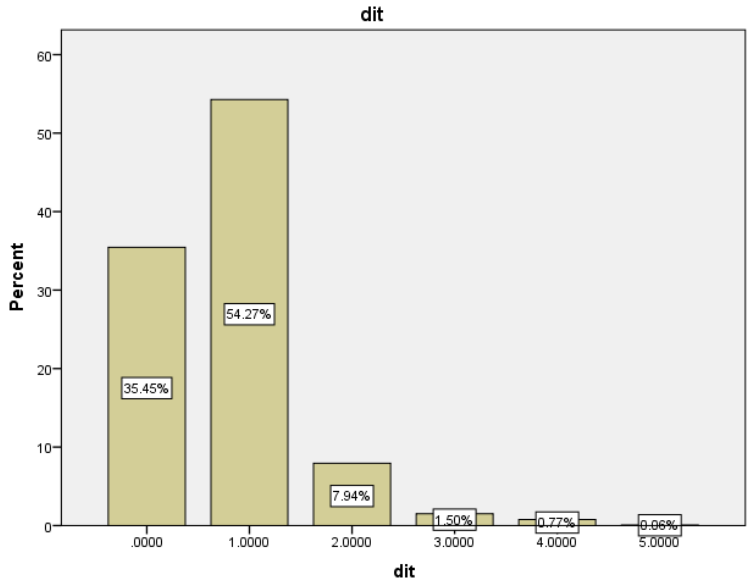

Figure 18: DIT System

Following the same approach as presented for category system, the thresholds for the rest of the categories were derived. The percentiles for NOM, RFC and CBO remained the same for the rest of the categories, 70^{th} and 90^{th}. DIT is not following a heavy-tailed distribution so the average was taken as threshold value for the rest of the categories. The percentiles were adjusted for NOC accordingly, 85^{th} and 95^{th} percentile for the rest of the categories.

Table 6 shows an overview of the results for each category. The categories name are in the first column and for each metrics the threshold values are separated into common/casual/uncommon.

Category	NOM Common/Casual/Uncommon	RFC Common/Casual/Uncommon	DIT Average	NOC Common/Casual/Uncommon	CBO Common/Casual/Uncommon
System	0-6 / 7-13 / > 13	0-15 / 16-41 / > 41	1	0 / 1 / > 1	0-1 / 2-3 / > 3
Games	0-6 / 7-14 / > 14	0-20 / 21-54 / > 54	1	0-1 / 2 / > 2	0-2 / 3-7 / > 7
Reading	0-6 / 7-14 / > 14	0-17 / 18-45 / > 45	1	0-1 / 2 / > 2	0-1 / 2-5 / > 5
Time	0-6 / 7-14 / > 14	0-18 / 19-49 / > 49	1	0 / 1 / > 1	0-1 / 2-4 / > 4
Security	0-6 / 7-13 / > 13	0-17 / 18-46 / > 46	1	0 / 1 / > 1	0-1 / 2-5 / > 5
Multimedia	0-6 / 7-15 / > 15	0-18 / 19-49 / > 49	1	0-1 / 2 / > 2	0-1 / 2-4 / > 4
Navigation	0-7 / 8-14 / > 14	0-19 / 20-47 / > 47	1	0-1 / 2 / > 2	0-2 / 3-4 / > 4
Money	0-7 / 8-13 / > 14	0-18 / 19-44 / > 44	1	0 / 1 / > 1	0-1 / 2-4 / > 4
Graphics	0-6 / 7-14 / > 14	0-11 / 12-31 / > 31	1	0 / / > 0	0-1 / 2-3 / > 3
Internet	0-6 / 7-15 / > 15	0-18 / 19-56 / > 56	1	0 / 1 / > 1	0-1 / 2-5 / > 5
Development	0-6 / 7-14 / > 14	0-18 / 19-42 / > 42	1	0 / 1 / > 1	0-2 / 3-4 / > 4
Theming	0-6 / 7-12 / > 12	0-13 / 13-34 / > 34	1	0 / / > 0	0-1 / 2 / > 2
Connectivity	0-6 / 7-14 / > 14	0-18 / 19-49 / > 49	1	0-1 / 2 / > 2	0-2 / 3-4 / > 4
Science and Education	0-7 / 8-17 / > 17	0-21 / 22-65 / > 65	1	0 / 1 / > 1	0-2 / 3-5 / > 5
Writing	0-7 / 8-14 / > 14	0-17 / 18-47 / > 48	1	0 / 1 / > 1	0-1 / 2-4 / > 4
Sports and Health	0-7 / 8-14 / > 14	0-21 / 22-53 / > 53	1	0 / 1 / > 1	0-2 / 3-4 / > 4
Phone and SMS	0-7 / 8-15 / > 15	0-18 / 19-49 / > 49	1	0-1 / 2 / > 2	0-2 / 3-4 / > 4
Dataset as whole	0-6 / 7-14 / > 14	0-18 / 19-50 / > 50	1	0 / 1 / > 1	0-2 / 3-4 / > 4

Table 6: Threshold values per category

Subcategory level

By looking at the frequency chart for modules in the dataset as a whole in figure 19, we can observe that the applications tend to lean towards being relatively small to medium.

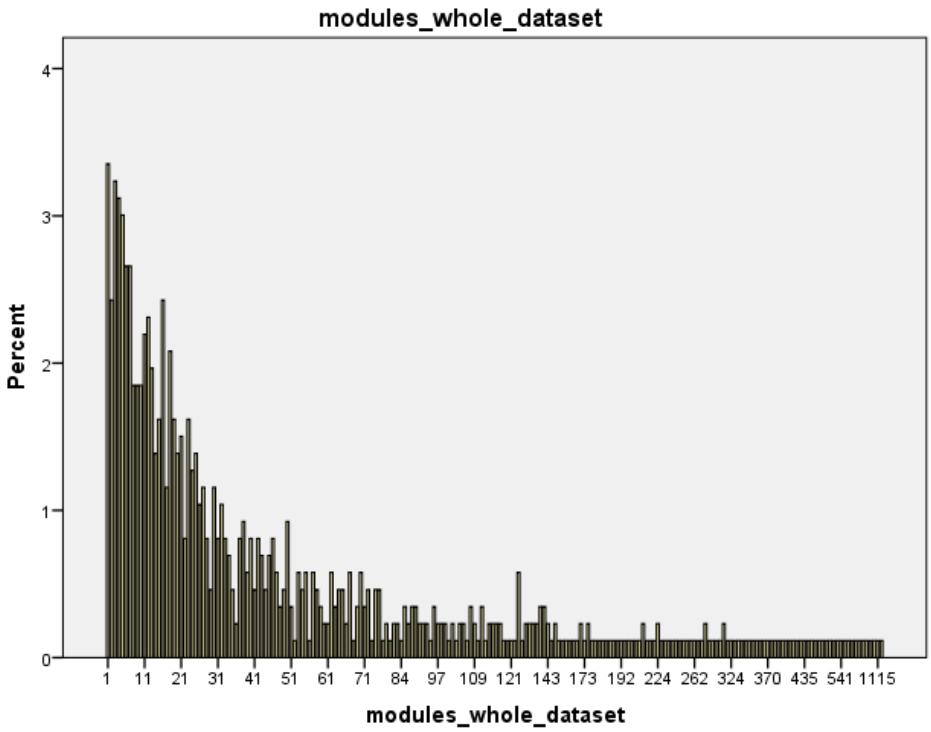

Figure 19: Frequency of modules on the dataset

More clearly, the histogram in figure 20 shows that 686 out of 865 applications have between 1 and 100 modules.

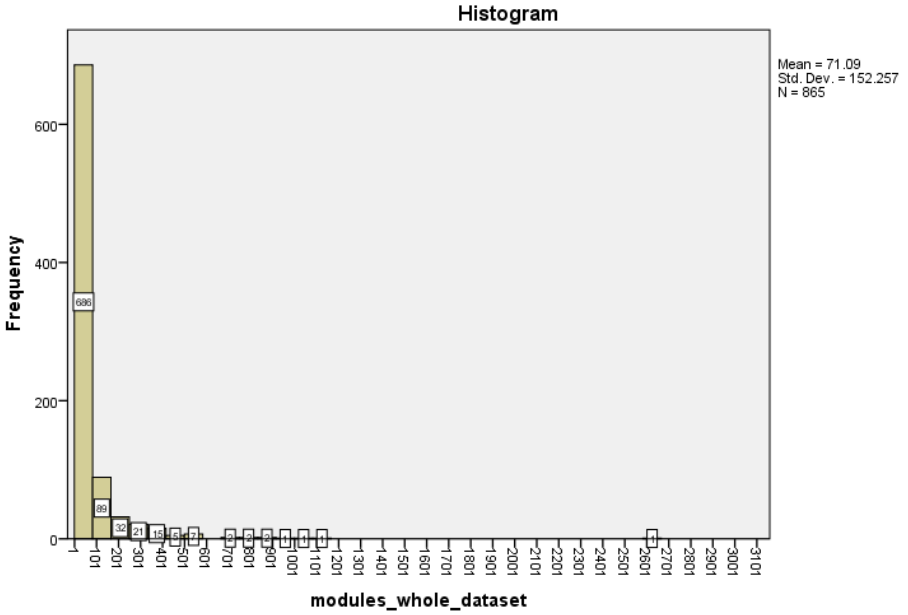

Figure 20: Histogram of modules seen on the dataset

By plotting the number of modules on the X axis and any metrics on the Y axis of a scatter plot, we can observe for possible clusters as in figure 21. For each category we have derived approximate boundaries for small, medium and large applications, keeping in mind the observation from figure 19 and figure 20 showing the tendency of our applications.

By observing figure 21, we can define two clusters, one for applications that have from 1 to 50 modules, medium size application that have from 55 to 110 modules and for large applications we can observe that there are only few and can not derive thresholds for.

We should note that the number of applications in the first two clusters is higher compared to the third cluster, hence we can suspect outliers and will not define a range for large applications.

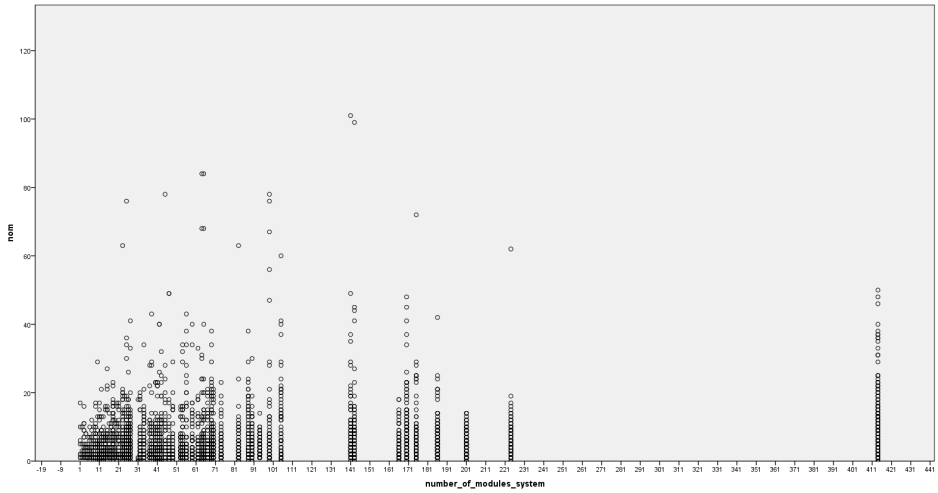

Figure 21: Number of modules in System category

The same approach was applied to the rest of the categories. Table 7 defines the subcategories for every category. When we observe the table 7 and the values for small, medium and large applications, we can notice gaps between the small and the medium for instance. The reason behind the gap is the lack of applications with modules that equal a value from the gap. For instance on the category system there is a gap between 50 (the upper boundary to small applications) and 55(the lower boundary for medium app). This is caused by the fact that none of the applications from the dataset in the category system have modules between 51 and 54. This is a limitation of our dataset and might improved by including even more applications in the dataset.

Category	Subcategories		
	small	**medium**	**large**
System	1-50	55-110	/
Games	1-75	89-120	/
Reading	2-33	55-105	/
Time	1-50	60-110	>124
Security	1-60	/	/
Multimedia	1-104	115-210	/
Navigation	1-50	60-120	/
Money	1-60	/	/
Internet	1-109	140-284	>300
Development	2-15	30-70	/
Theming	1-11	16-23	/
Connectivity	2-50	65-140	/
Science and Education	2-23	33-65	90-130
Writing	1-32	40-58	70-128
Sports and Health	2-43	90-139	/
Phone and SMS	1-13	30-75	/
Dataset as whole	1-27	30-100	>100

Table 7: Subcategories

In order to derive threshold values for each metrics in the subcategories, we use the same approach as for deriving thresholds for the categories. The category graphics was excluded because there are only eleven applications withing the category.

Tables 8, 9, 10, 11 and 12 show the shape parameter α of the Weibull distribution for each metrics in the subcategories.

By observing the Weibull shape parameter α for NOM given in table 8, we can see that for small applications (1-50 modules) and medium (55-110 modules) in the category system defined in table 7 the shape parameter is less than 0. The value remains less than 0 for the rest of the subcategories, meaning the distributions for NOM are heavy-tailed.

Category	α for NOM		
	small	medium	large
System	0.80724	0.94052	/
Games	0.72019	0.90606	/
Reading	0.90272	0.76613	/
Time	0.79598	0.99343	0.9856
Security	0.85508	/	/
Multimedia	0.98736	0.95379	/
Navigation	0.86936	0.81969	/
Money	0.95618	/	/
Internet	0.97754	0.91371	0.90626**
Development	0.45558	0.96344	/
Theming	0.99245	0.95341	/
Connectivity	0.8517	0.86089	/
Science and Education	0.87806	0.98304	0.96767
Writing	0.99855	0.8604	0.9745
Sports and Health	0.89456	0.84867	/
Phone and SMS	0.91345	0.83926	/

Table 8: α parameter for NOM in subcategories

The values of α for RFC is given in table 9. All subcategories (small, medium, large) have α lower than 1 indicating that the distribution of RFC is heavy-tailed.

Category	α for RFC		
	small	medium	large
System	0.74549	0.73976	/
Games	0.72169	0.69776	/
Reading	0.64125	0.75904	/
Time	0.78158	0.75611	0.71483
Security	0.76723	/	/
Multimedia	0.73463	0.7518	/
Navigation	0.76279	0.71585	/
Money	0.80514	/	/
Internet	0.75741	0.70194	0.68177**
Development	0.79917	0.77481	/
Theming	0.80907	0.82663	/
Connectivity	0.85444	0.75444	/
Science and Education	0.81938	0.69231	0.56127
Writing	0.73585	0.71278	0.71652
Sports and Health	0.94874	0.85685	/
Phone	0.88589	0.79562	/

Table 9: α parameter for RFC in subcategories

The shape parameter could not be calculated for small applications in the theming category as presented in table 10. However, we assume since all values for the CBO in the subcategories are less than 0, small applications from the theming category are following heavy-tailed distribution too.

Category	**α for CBO**		
	small	**medium**	**large**
System	0.86655	0.67049	/
Games	0.70199	0.60594	/
Reading	0.87076	0.67003	/
Time	0.75704	0.71024	0.60626
Security	0.77026	/	/
Multimedia	0.70666	0.63098	/
Navigation	0.75949	0.66754	/
Money	0.71554	/	/
Internet	0.69171	0.5765	0.49146**
Development	0.89022	0.65656	/
Theming	no fit	0.8758	/
Connectivity	0.79032	0.62516	/
Science and Education	0.88437	0.62705	0.63029
Writing	0.70415	0.62734	0.67794
Sports and Health	0.99807	0.70386	/
Phone and SMS	0.9978	0.69301	/

Table 10: α parameter for CBO in subcategories

By observing the values of α in table 11, we can observe that for all subcategories (small, medium, large) the value is bigger than 1, meaning the distributions for DIT are not heavy-tailed.

Category	α for DIT		
	small	medium	large
System	2.1284	1.4574	/
Games	1.4945	1.4846	/
Reading	2.0472	1.6658	/
Time	2.1976	1.3762	1.5432
Security	1.7626	/	/
Multimedia	1.5222	1.4742	/
Navigation	1.7675	1.9262	/
Money	1.6992	/	/
Internet	1.7948	1.485	1.1309**
Development	1.8901	1.2864	/
Theming	no fit	1.4262	/
Connectivity	1.9637	1.3176	/
Science and Education	1.8548	1.471	1.3166
Writing	no fit	1.4955	1.8031
Sports and Health	1.8944	1.2169	/
Phone and SMS	no fit	1.6309	/

Table 11: α parameter for DIT in subcategories

In table 12, we can observe that α is not calculated for small applications in theming, writing and phone and SMS categories. The reason being that all values for NOC were 0. The rest of the values for the shape parameter are bigger than 1, meaning the distribution is not heavy-tailed.

Category	α for NOC		
	small	**medium**	**large**
System	1.4507	1.3047	/
Games	1.1816	1.0155	/
Reading	1.5743	1.2787	/
Time	2.0562	1.1504	1.1749
Security	1.5243	/	/
Multimedia	1.2064	1.1345	/
Navigation	1.6319	1.1455	/
Money	1.3054	/	/
Internet	1.1841	1.2114	1.0542**
Development	no fit	1.4119	/
Theming	all values are 0	1.5866	/
Connectivity	1.2669	1.1968	/
Science and Education	2.1144	1.3146	1.1035
Writing	all values are 0	1.4486	1.5023
Sports and Health	1.5189	1.1467	/
Phone and SMS	all values are 0	1.2102	/

Table 12: α parameter for NOC in subcategories

***Because of licensing limitations, the evaluation trial period of EasyFit supports up to 5000 records. Therefore the α for the large applications in internet category is obtained by fitting the first 5000 values, that is 57.2% of the total values.*

Accordingly, we proceed with deriving thresholds using percentiles for NOM, RFC and CBO while average is taken as threshold value for DIT and NOC.

Tables 13, 14, 15 are summarizing the threshold values NOM, RFC and CBO respectively.

The threshold values for DIT and NOC remained constant throughout the subcategories. All subcategories had DIT value of 1 and NOC value of 0, therefore it is not useful to create subcategories in these cases and we excluded the two tables.

Category	NOM Common/Casual/Uncommon small apps	NOM Common/Casual/Uncommon medium apps	NOM Common/Casual/Uncommon large apps
System	0-6 / 7-12 / $>$ 12	0-6 / 7-14 / $>$ 14	/
Games	0-6 / 7-14 / $>$ 14	0-7 / 8-16 / $>$ 16	/
Reading	0-6 / 7-12 / $>$ 12	0-5 / 6-12 / $>$ 12	/
Time	0-6 / 7-13 / $>$ 13	0-6 / 7-13 / $>$ 13	0-7 / 8-14 / $>$ 14
Security	0-6 / 7-13 / $>$ 13	/	/
Multimedia	0-6 / 7-14 / $>$ 14	0-6 / 7-14 / $>$ 14	/
Navigation	0-7 / 8-16 / $>$ 16	0-5 / 6-13 / $>$ 13	/
Money	0-6 / 7-13 / $>$ 13	/	/
Internet	0-6 / 7-13 / $>$ 13	0-6 / 7-16 / $>$ 16	0-7 / 8-16 / $>$ 16
Development	1-8 / 9-11 / $>$ 11	0-6 / 7-15 / $>$ 15	/
Theming	0-5 / 6-8 / $>$ 8	0-7 / 8-13 / $>$ 13	/
Connectivity	0-6 / 7-12 / $>$ 12	0-7 / 8-14 / $>$ 14	/
Science and Education	0-5 / 6-12 / $>$ 12	0-8 / 9-16 / $>$ 16	0-7 / 8-15 / $>$ 15
Writing	0-7 / 8-15 / $>$ 15	0-6 / 7-14 / $>$ 14	0-8 / 9-16 / $>$ 16
Sports and Health	0-5 / 6-11 / $>$ 11	0-8 / 9-15 / $>$ 15	/
Phone and SMS	1-5 / 6-12 / $>$ 12	0-5 / 6-10 / $>$ 10	/
Data set as whole	0-6 / 7-12 / $>$ 12	0-6 / 7-14 / $>$ 14	0-6 / 7-15 / $>$ 15

Table 13: Threshold values for NOM in the subcategories

Category	RFC Common/Casual/Uncommon small apps	RFC Common/Casual/Uncommon medium apps	RFC Common/Casual/Uncommon large apps
System	0-14 / 14-39 / $>$ 39	0-15 / 16-45 / $>$ 45	/
Games	0-18 / 19-51 / $>$ 51	0-20 / 21-57 / $>$ 57	/
Reading	0-16 / 17-39 / $>$ 39	0-15 / 16-42 / $>$ 42	/
Time	0-19 / 20-44 / $>$ 44	0-16 / 17-49 / $>$ 49	0-19 / 20-47 / $>$ 47
Security	0-17 / 18-45 / $>$ 45	/	/
Multimedia	0-14 / 15-47 / $>$ 47	0-17 / 18-46 / $>$ 46	/
Navigation	0-18 / 19-51 / $>$ 51	0-17 / 18-65 / $>$ 65	/
Money	0-14 / 14-38/ $>$ 38	/	/
Internet	0-16 / 17-42 / $>$ 42	0-17 / 18-55 / $>$ 55	0-19 / 20-62 / $>$ 62
Development	1-24 / 25-41 / $>$ 41	0-19 / 20-49 / $>$ 49	/
Theming	0-8 / 9-20 / $>$ 20	0-18 / 19-51 / $>$ 51	/
Connectivity	0-14 / 15-38 / $>$ 38	0-20 / 21-52 / $>$ 51	/
Science and Education	0-13 / 14-37 / $>$ 38	0-23 / 24-76 / $>$ 76	0-17 / 18-55 / $>$ 55
Writing	0-21 / 22-71 / $>$ 72	0-19 / 20-45 / $>$ 45	0-17 / 18-51 / $>$ 51
Sports and Health	0-17 / 18-40 / $>$ 40	0-23 / 24-57 / $>$ 57	/
Phone and SMS	1-12 / 13-25 / $>$ 25	0-11 / 13-37 / $>$ 37	/
Data set as whole	0-14 / 15-39 / $>$ 39	0-17 / 18-47 / $>$ 47	0-18 / 19-52 / $>$ 52

Table 14: Threshold values for RFC in the subcategories

Category	CBO Common/Casual/Uncommon small apps	CBO Common/Casual/Uncommon medium apps	CBO Common/Casual/Uncommon large apps
System	0-1 / 2 / > 2	0-1 / 2-3 / > 3	/
Games	0-1 / 2-3 / > 3	0-1 / 2-4 / > 4	/
Reading	0-1 / 2 / > 2	0-1 / 2-4 / > 4	/
Time	0-1 / 2-3 / > 3	0-1 / 2-3 / > 3	0-1 / 2-4 / > 4
Security	0-1 / 2-3 / > 3	/	/
Multimedia	0-1 / 2-3 / > 3	0-2 / 3-4 / > 4	/
Navigation	0-1 / 2-3 / > 3	0-1 / 2-3 / > 3	/
Money	0-1 / 2-3 / > 3	/	/
Internet	0-1 / 2-3 / > 3	0-1 / 2-5 / > 5	0-2 / 3-6 / > 6
Development	0-1 / 2 / > 2	0-1 / 2-4 / > 4	/
Theming	0 / 1 / > 1	0-1 / 2-3 / > 3	/
Connectivity	0-1 / 2-3 / > 3	0-2 / 3-4 / > 4	/
Science and Education	0-1 / 2 / > 2	0-1 / 2-4 / > 4	0-1 / 2-4 / > 4
Writing	0-1 / 2-3 / > 3	0-1 / 2-4 / > 4	0-1 / 2-4 / > 4
Sports and Health	0-1 / 2 / > 2	0-2 / 3-4 / > 4	/
Phone and SMS	0 / 1 / > 1	0-1 / 2-3 / > 3	/
Data set as whole	0-1 / 1-2 / > 2	0-1 / 2-3 / > 3	0-2 / 3-5 / > 5

Table 15: Threshold values for CBO in the subcategories

4.2 Discussion

This discussion is divided into three parts.

The first part consists of discussion on the results and the interpretations of the same. The second section is be a discussion regarding the validity of the chosen method for deriving thresholds. In the third section, the limitation of this work will be outlined.

4.2.1 Interpretation of the Results

We should note that when interpreting the results we also need to refer to table 4 which contains the number of applications and modules for each category. For instance, it is important to stress out that some categories have substantial differences in number of applications. However, graphics category being one that has the least applications.

NOM represents the sum of methods in a module, we can observe that the most common values have range between 0-6, casual occurrences are from 7-14 and values bigger than 14 are uncommon. When comparing every category to the dataset as whole we can observe that the maximum change of values is +/- 3, (science and education). Accordingly, we can propose that the general threshold for NOM can be derived from the dataset as whole. However, if the applications that is being developed can be classified to one of the categories provided, the appropriate thresholds are more fitting.

By observing the common threshold values for NOM we notice they start from 0, i.e., there are modules that have 0 methods. The reason behind this is the fact that enums usually do not have any methods only attributes. Additionally, if a class is used to declare constants only, it will have no methods.

By observing the thresholds for NOM in the subcategories, presented earlier in table 13, we can see NOM increases with the size of the applications i.e., bigger applications have larger NOM. Hence, it is important for developers to take size of their application into consideration when referring to thresholds for NOM.

RFC counts all methods that can be invoked in response to a message. This includes all methods accessible withing the hierarchy. The greater the RFC, developers and testers should spend more testing resources. Additionally, testing and debugging of the class becomes difficult since it requires greater level of understanding.

By comparing the thresholds for RFC between categories and the dataset as whole presented earlier in table 6, we can observe greater change among the categories and also when each category is compared to the dataset as whole, hence we propose the usage of the thresholds for each category separately.

The threshold values for RFC obtained from the dataset as a whole shall be

used only when the application can not be categorized according to the categories provided. By applying the general thresholds we believe that the minimal quality criteria will be met.

By observing the thresholds for RFC for the subcategories presented in table 14, we can see overall increase in the threshold values as the application size increases. This should be taken into account by developers when they refer to threshold values for RFC.

We have noticed that DIT is taken as the longest path from the measured module to the root module. However, the tool does not distinguish between when a class is extending another class and implementing interface. We believe that there is a need for two separate metrics, one that will calculate the DIT of a class that is inheriting other class and a new metric, that will measure the depth of inheritance tree for a class that is implementing interface(s).

In addition, the calculates DIT only based on the modules that are written by the developers and are present in the source code. This leads to rather low DIT if we have in mind that the system classes in Android have high DIT themselves.

If we take an example of a class that is extending activity, that class in our case has DIT 1 because activity is a system class and is not present in the source code of the application.

When we look at the DIT of activity[1] we can observe the following hierarchy:

A class < Activity < ContextThemeWrapper < ContextWrapper < Context < Object

So a class that extends activity has DIT of 5 if system classes are taken into consideration.

Taking into consideration that system classes can not be directly manipulated by developers and they are heavily tested, we propose that developers look at the classes written directly by them when comparing their DIT to the ones proposed in this thesis.

Both on category and subcategory level the average value of DIT is 1.

The thresholds for NOC presented earlier in table 6 are on the low side, this means that very few (up to two) sub-classes inherit the methods of the parent class. This indicates also the low value for DIT. We should note that if developers wish to create classes that provide methods to more than two sub-classes i.e. super/utility classes, more testing resources and time need to be dedicated to them.

When the values for NOC were split across small/medium/large applications we observed change in the Weibull distribution density function. As shown earlier in table 12, the α parameter is > 1 meaning average can be taken as the threshold value. As presented in the results above, across all subcategories the value for NOC

[1] https://developer.android.com/reference/android/app/Activity.html

is 0. Both subcategories and categories of application have low value for NOC, the latter one being more flexible (allowing value for NOC up to two). Accordingly, we propose category the thresholds presented in table 6 to be used for NOC.

The CBO threshold values presented in table 6 are on the low side. This is good indication that the applications are loosely coupled, this fact makes it easier to make changes in the modules without many side effects. We can assume that open source applications prefer loosely coupling because it makes it easier for the community to contribute to the projects. The higher the coupling is, the harder it becomes to understand and change the modules.

If a change needs to be made in a class (X) that depends on a class (Y) any change on a public member of Y that is used by X needs to be changed in X as well. As pointed in the viewpoints for CBO, low coupling encouraged modular design and reuse.

We can propose the general threshold values derived from the dataset as whole for types of applications that are not covered by our categorization. However, if the application belongs to a category that has specific threshold, we propose that specific values to be used.

By looking at the thresholds for the subcategories presented in table 15, we can observe that CBO thresholds increase as the applications size increase. This needs to be considered by developers, the larger the application the higher the upper limit of CBO is.

By summarizing both category level and subcategory level metrics threshold values for the five metrics investigated in this thesis we can conclude that maintainability efforts increase in applications that have the highest values for all five metrics. Portability and efficiency decrease with increase of the values of every metric.

Keeping in mind that the dataset contains open source applications only, we need to point out the applicability of these thresholds to a commercial mobile applications development might be arguable. There might be substantial differences in how commercial applications are build compared to open source. These difference might arise from the specific type of application build, the specific culture of the enterprise and the development team, resources available or the customers' requirements. On the other side, contributors to the open source projects are often experienced developers that have the experience and knowledge to contribute to project outside their professional work which may provide support to the thresholds. Accordingly, we argue that the thresholds are acceptable to professional software developers developing commercial applications.

4.2.2 Validity of the Metrics Thresholds

By gathering large number of applications, utilizing the tool to its best abilities and by elaborating on the limitations we are confident that the thresholds can be used in developers day-to-day coding as indicators of possible issues with their modules.

The categorization of applications is done according to the F-Droid categories. A possible misplacement of application in a category where they do not belong could affect the threshold. However, taking into consideration that F-Droid is maintained actively by the community, and it is used for research and the number of applications in 16 out of 17 categories is high, it is reasonable to assume that a possible mismatch would not impact the thresholds.

As the sub-categorization is based on a particular dataset and the clusters of applications, different datasets might have yield different subcategories. However, we believe that this sub-categorization will help developers locate their application more easily in terms of size. Furthermore, taking the size into consideration is important for metrics that thresholds change according to the size of the application. Additionally, different subcategories can be defined, for instance based on lines of code.

In order to use the full potential provided by Analizo we explored two ways of computing the metrics: the first one is used in this thesis[2] where each project is treated separately and the metrics values need to be processed by us in order to make them suitable to be imported in the database.

The second approach is so called "batch"[3]. On the first glance it seems that batch process is better suited for this thesis since it handles the metrics values and they are ready to be imported in a database right away. In addition, there is no need to process the metric values, the output of the batch process is a .csv file that contains all metrics values for all applications in the folder where the batch process is executed. In our case we have executed the batch process in every category folder. The final output would be the same and the only difference is that by using batch, the final output is created by the process itself.

However, after performing the initial cloning of the applications for testing the tool, we observed source codes written in languages different than Java. Some examples included C# possibly developed as cross platform tools like Xamarin[4]. Additionally, examples of React Native[5] were observed.

Unfortunately, the batch option is not flexible and does not accept an argument for the target language nor excludes specified directories. Compared to the approach used by us, it allows us to specify the target language, names of folders that

[2] http://www.analizo.org/man/analizo-metrics.html
[3] http://www.analizo.org/man/analizo-metrics-batch.html
[4] www.xamarin.com
[5] https://facebook.github.io/react-native/

should be ignored and we were also able to have a list of excluded applications for each category.

Despite the fact that the threshold values are derived based on the frequency of the values, and not on the overall quality of the applications in the dataset, they present a pattern of most applications in a particular category. We do not claim that the values taken as most common and recommended by us are derived based on applications that follow best practices in software engineering. However, the values provide a consistent behaviours across categories.

These metrics thresholds should not be taken as the absolute ranges for a particular metric but rather as a guide towards better quality. As with the definition of metrics given by Lorenz and Kidd [6], if a module exceeds the threshold value it does not automatically implies that it will cause fault or bugs. However, it is a strong pointer that for instance it needs to be divided into more modules or more testing efforts should be dedicated.

As stated in the methodology and implementation, chapter 3, to the best of our abilities and the abilities of the tool we excluded test files. We exclude folders with names: *test* and *androidTest*. However, we can not confirm that every test file in every application has been excluded, there is a possibility that the developers are keeping their test files in the same folder as their source files. If this is the case, we can not exclude the files form being treated as part of the source code of the application with 100% accuracy.

In addition, as stated previously, all classes, interfaces and enums[6,7] are treated as modules and metrics are calculated for them. There is a discussion in the Android community about usage of enums in Android apps, on the topic of "Manage Your App's Memory" [65] in "Be careful with code abstractions" section it is indicated that "Enums often require more than twice as much memory as static constants. You should strictly avoid using enums on Android."

On the other hand, on the "Performance tips" [66] the section about avoiding enums has been deleted [67].

We have observed applications that are using enums for declaring constants and also enums that inherit other classes and have logic. Hence, we argue that we need to include them in the calculations, ideally they would be treated separately and specific thresholds would be derived.

4.2.3 Limitations

The thresholds now are defined for a module in an application, by modules as defined previously we include classes, interfaces and enums. Despite the fact that they are part of a application, having a separate thresholds for each of them would

[6] https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html
[7] https://developer.android.com/reference/java/lang/Class.html

be more reliable. Enums usually contain only constants and are decreasing the value for NOM for instance. However, we have observed enums that contain logic, hence deriving thresholds separately would be beneficial.

Furthermore, by treating all classes as one, we do not provide a specific thresholds for classes that extend activities or services for instance. These special Android classes might have their own characteristics that are distinguishable when compared to each other.

We were not able to exclude test files that are placed together with the source code or in a folder named differently than *androidTest* or *test*, we could not estimate how much is their impact, if any. However, we believe test files that are places in the same folders as the source code, can have impact on the threshold ranges.

As they are now, threshold values treat all modules the same. However, in practice, there might be a need for a particular number of modules that exceeds the threshold values. This this is inevitable and the developers have a solid reason why a module with extensive number of methods, long inheritance tree, high coupling is needed they should be free to have it. However, even if the need for this type of module(s) is justified, the fact that more testing maintenance efforts will be spent on the modules and the application itself remains. Additionally, a different type thresholds for instance relative thresholds [68], that take into consideration the need for modules that exceed the threshold can be proposed.

Different tools might interpret and calculate metrics differently. The applicability of the results is dependent on the calculation method. To avoid this problem the tool used to calculate the metrics must follow the metrics descriptions presented in this thesis.

5 Conclusion

Software quality metrics can reveal internal characteristics of a software product. By calculating quality metrics, the system being developed can be monitored for faults and potential issues. The software quality metrics are expressed by a number only. Despite the fact that the metrics have a viewpoints on what does the value represents and what is the ultimate goal in terms of desired value, we can not be sure what does a particular number represents without having a reference value to compare it to. For instance, we know that increase of DIT leads to raise of the complexity in the system increases but we do not know what is the upper limit for DIT. More importantly, we do not know if there is one upper limit for all applications or it changes depending on the functionality of the application or perhaps the size.

This thesis proposes threshold values that indicates the lower/upper limit for five metrics.

By successfully calculating values for NOM, RFC, DIT, NOC and CBO using Analizo[1] [64], we have provided a method on how to pragmatically calculate the metrics on source code level of large number of applications. Our dataset includes 865 Android applications across 17 categories, extracted and categorized from F-Droid[2] The thresholds are given for each metric in each category, where categories are based on F-Droid categorization that groups applications together based on their functionality.

Furthermore, we have sub-categorized these applications in each category based on their size with the aim to help developers locate the thresholds based on the number of modules of their application.

Below, our main findings are presented with answering the research questions.

RQ1: *To what extent can the traditional object-oriented software quality metrics be applied to Android applications?*

By successfully computing traditional object-oriented software quality metrics, on large number of Android applications, our work can serve as a support to the previous research for metrics calculation and thresholds identification [26, 27, 24] and the area of computing quality metrics in Android applications [51]. In addition, we propose possible improvement of DIT by introducing a new DIT-like metric specially designed to capture the DIT of a class when it implements interface(s).

[1] www.analizo.org
[2] www.f-droid.org

Furthermore, new metrics, specifically for Android applications including: Number of Activities, Number of Services, Number of Broadcast Receivers and Number of Content Providers were identified as potential improvements and are elaborated in future work, 5.1.

RQ2: *What are the threshold values for different categories of application and the dataset as whole?*

Our analysis yield thresholds for each of the five metrics investigated.

For NOM and CBO were able to define general thresholds taken from the dataset as a whole. However, we suggest taking the general thresholds values only in the case when an application can not be categorized according to our categorization. We believe that category level thresholds are more appropriate and should be used whenever possible.

For RFC we propose the category specific thresholds. However, in cases when the application can not be categorized in any of the categories provided by us nor to similar category, we propose the usage of the general threshold for RFC derived on the dataset as a whole.

Both inheritance metrics, DIT and NOC reveal low inheritance across the categories. For DIT we propose that modules extend one only another module and for NOC we suggest value up to two. Keeping in mind that DIT is calculated only by taking into consideration modules written by developers and present in the source code. System classes are excluded from the calculations.

RQ3: *What are the threshold values for each of the subcategories of applications and the dataset as whole?*

For NOM, RFC and CBO, the threshold values increases as the size of the application increases. This indicates that developers need to take the size of the application into account when using the thresholds for these three metrics. We believe that subcategory metrics for NOM, RFC and CBO can aid developers more compared to the category level thresholds since these thresholds change with the size of the application.

For NOM, RFC and CBO we propose the usage of general thresholds based on the dataset as a whole only in cases when the application developed can not be categorized according to our categorization.

On the contrary, for DIT we did not observe any changes across subcategories. The thresholds stay low in all subcategories and the value is one, irrespective of the size.

Lastly for NOC we propose the usage of the category level instead of subcategory. We believe that category level depicts the characteristics of the metrics better

and gives a more suitable thresholds.

RQ4: *What are the quality implications of the thresholds values?*

Threshold values should be be seen as pointers to a possible problem, as defined by Lorenz and Kidd [6] the thresholds are identifying anomalies which may not necessarily cause problems. However, modules that exceed the threshold values should be examined and tested rigorously.

Summarizing the viewpoints for the metrics and their impact investigated by Lincke et al [54] we conclude the following:

Maintainability efforts rise with increase of all five metrics (NOM, RFC, DIT, NOC, CBO). I.e., it becomes harder to make changes in the system, write tests, maintain existing and implement new features.

Efficiency might decline as the metrics values increase. I.e., for developers it means they might spend more time understanding the code and the functionality. From users point of view, more computational resources will be needed for the application to provide the expected performance.

Portability decreases as the values for all metrics increase. I.e., it becomes harder to adapt/move classes in order to introduce new features or to support new devices.

5.1 Future Work

The thresholds indeed point out characteristics of a module, for instance by having large value for NOM, it means the module is responsible for multiple jobs and is a candidate for splitting into more than one module. However, further studies of how these thresholds affect applications are needed in order to fully understand the benefits they offer.

It is important to explore the implications that these thresholds might have on the application, both from developers point of view and from users point of view. One approach would be to investigate the success of an application that obeys threshold values while being developed. A success can be examined from different views, one being the user ratings and reviews on Google Play.

Comparing versions of the same application, where some versions consider thresholds and other versions not, against the user ratings and user reviews extracted from Google Play would give a comparable and insight data on how the application is viewed by the customers.

From the developers point of view, error data and their personal opinions can give us indications on how these thresholds performed. In addition, the most important implication is how these metrics correlate with the Android specific features e.g. permissions, activities, services, broadcast receivers and content providers.

A possible path for a new research would be deriving thresholds by splitting the

classes into classes that extend for instance, activities, services, broadcast receivers, content providers etc. By splitting the classes further than just between class, interface and enums, further research can derive more specific thresholds for each type of class.

Possible names of the metrics are:

- Number of Activities - count of the total classes that extend Activities.
- Number of Services - count of all classes that extend Services.
- Number of Broadcast Receivers - count of all classes that extend BroadcastReceiver.
- Number of Content Providers - count of all classes that extend ContentProviders .

Possible research areas include:

1. Identifying Android specific metrics.
2. Deriving thresholds for Android specific metrics.

A continuity of this thesis can be created on how the application performed when using thresholds. Goals of the research can include:

- How do applications that belong to "common" threshold values performed compared to "casual" and "uncommon" in terms on user ratings and reviews.
- How does application perform when using threshold values in terms of user satisfaction.
- How does application perform when using threshold values from developers perspective.

A study on relative thresholds for Android applications can potentially define the number of modules inside the applications that can have values for the metrics outside the defined metric thresholds. An approach is presented by Paloma et al. [68]. The concept of flexible thresholds captures the percentage of modules that exceed the threshold limits. The assumption here is that applications can have modules that exceed the metrics thresholds values. The cause can be complex requirements, performance optimization and machine generated code.

Bibliography

[1] Martin, W., Sarro, F., Jia, Y., Zhang, Y., & Harman, M. 2017. A survey of app store analysis for software engineering. *IEEE transactions on software engineering*, 43(9), 817–847.

[2] Zahra, S., Khalid, A., & Javed, A. 2013. An efficient and effective new generation objective quality model for mobile applications. *International Journal of Modern Education and Computer Science*, 5(4), 36.

[3] Franke, D., Kowalewski, S., & Weise, C. 2012. A mobile software quality model. In *Quality Software (QSIC), 2012 12th International Conference on*, 154–157. IEEE.

[4] Sommerville, I. 2011. *Software engineering*. Pearson.

[5] Li, W. 1998. Another metric suite for object-oriented programming. *Journal of Systems and Software*, 44(2), 155–162.

[6] Lorenz, M. & Kidd, J. 1994. *Object-oriented software metrics: a practical guide*. Prentice-Hall, Inc.

[7] Henderson-Sellers, B. 1995. *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc.

[8] Chidamber, S. R. & Kemerer, C. F. 1994. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6), 476–493.

[9] Chidamber, S. R. & Kemerer, C. F. 1991. *Towards a metrics suite for object oriented design*, volume 26. ACM.

[10] Jabangwe, R., Börstler, J., Šmite, D., & Wohlin, C. 2015. Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. *Empirical Software Engineering*, 20(3), 640–693.

[11] e Abreu, F. B. & Carapuça, R. 1994. Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems and Software*, 26(1), 87–96.

[12] Briand, L., Daly, J., Porter, V., & Wüst, J. 1998. A comprehensive empirical validation of product measures for object-oriented systems. *Journal of Systems and Software, to appear. Also available as Technical Report ISERN-98-07*.

[13] Briand, L. C., Wüst, J., Daly, J. W., & Porter, D. V. 2000. Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of systems and software*, 51(3), 245–273.

[14] Li, W. & Henry, S. 1993. Object-oriented metrics that predict maintainability. *Journal of systems and software*, 23(2), 111–122.

[15] Zhou, Y. & Leung, H. 2007. Predicting object-oriented software maintainability using multivariate adaptive regression splines. *Journal of Systems and Software*, 80(8), 1349–1361.

[16] Aggarwal, K., Singh, Y., Kaur, A., & Malhotra, R. 2006. Application of artificial neural network for predicting maintainability using object-oriented metrics. *Transactions on Engineering, Computing and Technology*, 15, 285–289.

[17] Wang, L.-j., Hu, X.-x., Ning, Z.-y., & Ke, W.-h. 2009. Predicting object-oriented software maintainability using projection pursuit regression. In *Information Science and Engineering (ICISE), 2009 1st International Conference on*, 3827–3830. IEEE.

[18] e Abreu, F. B. & Melo, W. 1996. Evaluating the impact of object-oriented design on software quality. In *Software Metrics Symposium, 1996., Proceedings of the 3rd International*, 90–99. IEEE.

[19] Harrison, R., Counsell, S. J., & Nithi, R. V. 1998. An evaluation of the mood set of object-oriented software metrics. *IEEE Transactions on Software Engineering*, 24(6), 491–496.

[20] Al Dallal, J. 2013. Object-oriented class maintainability prediction using internal quality attributes. *Information and Software Technology*, 55(11), 2028–2048.

[21] Al Dallal, J. & Morasca, S. 2014. Predicting object-oriented class reuse-proneness using internal quality attributes. *Empirical Software Engineering*, 19(4), 775–821.

[22] Scandariato, R. & Walden, J. 2012. Predicting vulnerable classes in an android application. In *Proceedings of the 4th international workshop on Security measurements and metrics*, 11–16. ACM.

[23] Kitchenham, B. 2010. What's up with software metrics?–a preliminary mapping study. *Journal of systems and software*, 83(1), 37–51.

[24] Alves, T. L., Ypma, C., & Visser, J. 2010. Deriving metric thresholds from benchmark data. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, 1–10. IEEE.

[25] Lanza, M. & Marinescu, R. 2007. *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media.

[26] Ferreira, K. A., Bigonha, M. A., Bigonha, R. S., Mendes, L. F., & Almeida, H. C. 2012. Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, 85(2), 244–257.

[27] Filó, T. G., Bigonha, M., & Ferreira, K. 2015. A catalogue of thresholds for object-oriented software metrics. *Proc. of the 1st SOFTENG*, 48–55.

[28] Galin, D. 2004. *Software quality assurance: from theory to implementation*. Pearson Education India.

[29] Fenton, N. E. & Neil, M. 2000. Software metrics: roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, 357–370. ACM.

[30] Whitmire, S. A. 1997. *Object oriented design measurement*. John Wiley & Sons, Inc.

[31] Rahman, F. & Devanbu, P. 2013. How, and why, process metrics are better. In *Proceedings of the 2013 International Conference on Software Engineering*, 432–441. IEEE Press.

[32] Wahyudin, D., Schatten, A., Winkler, D., Tjoa, A. M., & Biffl, S. 2008. Defect prediction using combined product and project metrics-a case study from the open source" apache" myfaces project family. In *Software Engineering and Advanced Applications, 2008. SEAA'08. 34th Euromicro Conference*, 207–215. IEEE.

[33] Scharff, C. & Verma, R. 2010. Scrum to support mobile application development projects in a just-in-time learning context. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, 25–31. ACM.

[34] Scotto, M., Sillitti, A., Succi, G., & Vernazza, T. 2004. Dealing with software metrics collection and analysis: a relational approach. *Stud. Inform. Univ.*, 3(3), 343–366.

[35] Elish, M. O. & Elish, K. O. 2009. Application of treenet in predicting object-oriented software maintainability: A comparative study. In *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*, 69–78. IEEE.

[36] Basili, V. R., Briand, L. C., & Melo, W. L. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on software engineering*, 22(10), 751–761.

[37] Melo, W. L., Briand, L., & Basili, V. R. Measuring the impact of reuse on quality and productivity in object-oriented systems. Technical report, 1998.

[38] Erni, K. & Lewerentz, C. 1996. Applying design-metrics to object-oriented frameworks. In *Software Metrics Symposium, 1996., Proceedings of the 3rd International*, 64–74. IEEE.

[39] Shatnawi, R., Li, W., Swain, J., & Newman, T. 2010. Finding software metrics threshold values using roc curves. *Journal of Software: Evolution and Process*, 22(1), 1–16.

[40] Benlarbi, S., El Emam, K., Goel, N., & Rai, S. 2000. Thresholds for object-oriented measures. In *Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on*, 24–38. IEEE.

[41] El Emam, K., Benlarbi, S., Goel, N., Melo, W., Lounis, H., & Rai, S. N. 2002. The optimal class size for object-oriented software. *IEEE Transactions on software engineering*, 28(5), 494–509.

[42] Herbold, S., Grabowski, J., & Waack, S. 2011. Calculation and optimization of thresholds for sets of software metrics. *Empirical Software Engineering*, 16(6), 812–841.

[43] Oliveira, P., Borges, H., Valente, M. T., & Costa, H. A. X. 2013. Metrics-based detection of similar software (s). In *SEKE*, 447–450.

[44] Yoon, K.-A., Kwon, O.-S., & Bae, D.-H. 2007. An approach to outlier detection of software measurement data using the k-means clustering method. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 443–445. IEEE.

[45] McCabe, T. J. 1976. A complexity measure. *IEEE Transactions on software Engineering*, (4), 308–320.

[46] Nejmeh, B. A. 1988. Npath: a measure of execution path complexity and its applications. *Communications of the ACM*, 31(2), 188–200.

[47] Coleman, D., Lowther, B., & Oman, P. 1995. The application of software maintainability models in industrial software systems. *Journal of Systems and Software*, 29(1), 3–16.

[48] Franke, D. & Weise, C. 2011. Providing a software quality framework for testing of mobile applications. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, 431–434. IEEE.

[49] Ryan, C. & Rossi, P. 2005. Software, performance and resource utilisation metrics for context-aware mobile applications. In *Software Metrics, 2005. 11th IEEE International Symposium*, 10–pp. IEEE.

[50] Jošt, G., Huber, J., & Heričko, M. 2013. Using object oriented software metrics for mobile application development. In *2nd Workshop of Software Quality Analysis, Monitoring, Improvement, and Applications*, 17–27.

[51] Grano, G., Di Sorbo, A., Mercaldo, F., Visaggio, C. A., Canfora, G., & Panichella, S. 2017. Android apps and user feedback: a dataset for software evolution and quality improvement. In *Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics*, 8–11. ACM.

[52] Aggarwal, K., Singh, Y., Kaur, A., & Malhotra, R. 2006. Empirical study of object-oriented metrics. *Journal of Object Technology*, 5(8), 149–173.

[53] Harrison, R., Counsell, S. J., & Nithi, R. V. 1998. An investigation into the applicability and validity of object-oriented design metrics. *Empirical Software Engineering*, 3(3), 255–273.

[54] Lincke-Rudiger, R. & Löwe-Welf, W. 2007. Compendium of software quality standards and metrics–version 1.0.

[55] Barkmann, H., Lincke, R., & Löwe, W. 2009. Quantitative evaluation of software quality metrics in open-source projects. In *Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on*, 1067–1072. IEEE.

[56] Lincke, R., Lundberg, J., & Löwe, W. 2008. Comparing software metrics tools. In *Proceedings of the 2008 international symposium on Software testing and analysis*, 131–142. ACM.

[57] Krutz, D. E., Mirakhorli, M., Malachowsky, S. A., Ruiz, A., Peterson, J., Filipski, A., & Smith, J. 2015. A dataset of open-source android applications. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*, 522–525. IEEE.

[58] Minelli, R. & Lanza, M. 2013. Software analytics for mobile applications–insights & lessons learned. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, 144–153. IEEE.

[59] Freiling, F. C., Protsenko, M., & Zhuang, Y. 2014. An empirical evaluation of software obfuscation techniques applied to android apks. In *International Conference on Security and Privacy in Communication Systems*, 315–328. Springer.

[60] Lamba, Y., Khattar, M., & Sureka, A. 2015. Pravaaha: Mining android applications for discovering api call usage patterns and trends. In *Proceedings of the 8th India Software Engineering Conference*, 10–19. ACM.

[61] Scotto, M., Sillitti, A., Succi, G., & Vernazza, T. 2006. A non-invasive approach to product metrics collection. *Journal of Systems Architecture*, 52(11), 668–675.

[62] Sillitti, A., Janes, A., Succi, G., & Vernazza, T. 2003. Collecting, integrating and analyzing software metrics and personal software process data. In *null*, 336. IEEE.

[63] Sillitti, A., Janes, A., Succi, G., & Vernazza, T. 2004. Measures for mobile users: an architecture. *Journal of Systems Architecture*, 50(7), 393–405.

[64] Terceiro, A., Costa, J., Miranda, J., Meirelles, P., Rios, L. R., Almeida, L., Chavez, C., & Kon, F. 2010. Analizo: an extensible multi-language source code analysis and visualization toolkit. In *Brazilian conference on software: theory and practice (Tools Session)*.

[65] Manage your app's memory | android developers. https://developer.android.com/topic/performance/memory.html. (Accessed on 11/20/2017).

[66] Performance tips | android developers. https://developer.android.com/training/articles/perf-tips.html#avoid_enums. (Accessed on 11/20/2017).

[67] java - should i strictly avoid using enums on android? - stack overflow. https://stackoverflow.com/questions/29183904/should-i-strictly-avoid-using-enums-on-android. (Accessed on 11/20/2017).

[68] Oliveira, P., Valente, M. T., & Lima, F. P. 2014. Extracting relative thresholds for source code metrics. In *Software Maintenance, Reengineering and*

Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on, 254–263. IEEE.

A Appendix

A.1 Scripts

A.1.1 Calculating Metrics Script

```
#enters every sub folder and calculates the metrics.
    output is the name of the file that gets created
for D in ./*; do
if [ -d "$D" ]; then
cd "$D"
analizo metrics -o output --language java  -x test:
    androidTest
cd ..
fi
done
#go in every app folder from the root directory and
    extract the package data and put it in output file
for D in ./*; do
if [ -d "$D" ]; then
cd "$D"
echo "${PWD##*/}"  >> output
cd ..
fi
done
#if the files are greater  than XXX lines delete them.
for D in ./*; do
if [ -d "$D" ]; then
 cd "$D"
var=220
  lineCount=$(wc -l < output)
  if [ "$lineCount"  -lt  "$var" ]; then
  printf '%s\n' "${PWD##*/}"  > deleted.out
  rm output
  fi
cd ..
fi
 done
#concatinates all the deleted folders into one
find . -name 'deleted.out' | xargs cat >
    allExcludedProjects.txt
```

```
# start to create the final output file by getting the
    class values acc.dit,drn dnrn drn
for D in ./*; do
if [ -d "$D" ]; then
 cd "$D"
cat output | grep -w "acc\|accm\|amloc\|anpm\|cbo\|dit\|
    lcom4\|loc\|mmloc\|noa\|noc\|nom\|npa\|npm\|rfc\|sc"
    > temp_values
sed -e 's/$/ | /' -i temp_values #adds | after each end
    line
cut -d ":" -f2 temp_values > temp_values1
split  --additional-suffix=.tmpfv -l 16 temp_values1 #
    slpit temp_values into smaller files containing 15
    lines each
for f in *.tmpfv
do
        cat $f | tr -d '\n' > $f.vert #makes them
           vertial
        tail -1 output >> $f.vert #adds the last line of
            output file (folder name)
done
cat *.vert > final_vert.out
sed -i -e 's/~/\\N/g' final_vert.out #replaces ~ with \N
    for null in db
rm x*
rm *.tmpfv
cd ..
fi
done

find . -name 'final_vert.out' | xargs cat > sum_final.
    out
tr -d "␣\t" < sum_final.out > finalOutput.txt #deletes
    all spaces betteen the pipes
rm sum_final.out
sed -e 's/$/|/' -i finalOutput.txt #adds pipe for
    importing in DB
```

Listing A.1: Metrics calculating script

A.1.2 Script for Parsing XML

```
 #extractin the values in () from the index.xml file.
     they are saved in application_data.out
xmlstarlet sel -t -m "/fdroid/application" -v "concat(id
    ,␣'␣|␣'␣,added,␣'␣|␣'␣,lastupdated,␣'␣|␣'␣,name,␣'␣|␣
    '␣,category,␣'␣|␣'␣,source,␣'␣|␣'␣,marketversion)" -n
```

```
    index.xml > application_data.out

#extracts the values in () from fdroid/applciation/
   package for each applcation, it gets the first
   package node. they are saved in package_data.out
xmlstarlet sel -t -m "/fdroid/application/package[1]" -v
    "concat(version,␣'␣|␣',size,␣'␣|␣',sdkver,␣'␣|␣',
   targetSdkVersion,␣'␣|␣',added)" -n index.xml >
   package_data.out

#adding the data from the application application_data.
   out and package_data.out. not one under another but
   in one line
paste -d '|' application_data.out package_data.out >
   unfiltered_applications_data.out
cat unfiltered_applications_data.out | grep "https://
   github.com" > github.out
cat unfiltered_applications_data.out | grep "https://
   gitlab.com" > gitlab.out
#filtering, gets only the projecgs that are hosted on
   github and gitlab

#cocatinates the projects from githib and gitlab. one
   below antoher
cat github.out gitlab.out > lab_hub.out
rm github.out
rm gitlab.out
```

Listing A.2: XML parsin script

A.1.3 Queries

```
  select nom, rfc, dit, noc, cbo
from class_level_values_<category name>;
```

Listing A.3: Selecting metrics from category

```
select a.nummod as numberofmodules, b.<any metric>
from  class_level_values_<category name> b, temp_view_<
   category name>  a
where b.folder_name=a.folder_name;
```

Listing A.4: Defining subcategories

```
select b.nom, b.rfc, b.dit, b.noc b.cbo
from class_level_values_<category name> b
where b.folder_name IN (select a.folder_name from
```

```
class_level_values_<category name> a
   group by 1 having count(a.folder_name
   ) between X and Y);
```

Listing A.5: Selecting metrics values for subcategories