UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Mustafa Ogün Öztürk

Evaluating Maintainability of Android Applications: Mooncascade Case Study

Master's Thesis (30 ECTS)

Supervisor: Jakob Mass, MSc

Tartu 2021

Evaluating Maintainability of Android Applications: Mooncascade Case Study

Abstract:

Android became one of the most extensive mobile platforms and active software development branches in the last decade. Android applications are known for their frequent updates to meet changing user requirements. At his point, maintainability emerges as a key non-functional requirement for Android applications because it determines how understandable, modifiable Android applications are and how easy it is to maintain them. Thus, developing maintainable Android applications is vital, especially for software product development companies with fast development and client circulation. In this case study, the technologies and methods used when developing Android applications by Mooncascede, a software product development company, will be evaluated from the maintainability point of view. The primary goal of this study is to identify and present the impact of these technologies and the methods on the maintainability of Android applications.

Keywords: Android, Maintainability, Metrics, Software Engineering

CERCS: P170 Computer science, numerical analysis, systems, control

Tüübituletus neljandat järku loogikavalemitele

Lühikokkuvõte:

Androidist sai viimase kümnendi jooksul üks ulatuslikumaid mobiilplatvorme ja aktiivseid tarkvaraarendusharusid. Androidi rakendused on tuntud oma sagedaste värskenduste poolest, et need vastaksid muutuvatele kasutajate nõuetele. Tema sõnul kerkib hooldatavus Androidi rakenduste peamise mittefunktsionaalse nõudena, sest see määrab, kui arusaadavad, muudetavad Androidi rakendused on ja kui lihtne on neid hooldada. Seega on ülihooldatavate Android-rakenduste väljatöötamine ülitähtis, eriti kiire arenduse ja klientide ringlusega tarkvaratoodete arendamise ettevõtete jaoks. Selles juhtumiuuringus hinnatakse tarkvaratoodete arendusettevõtte Mooncascede Androidi rakenduste väljatöötamisel kasutatud tehnoloogiaid ja meetodeid hooldatavuse seisukohast. Selle uuringu esmane eesmärk on tuvastada ja tutvustada nende tehnoloogiate ja meetodite mõju Androidi rakenduste hooldatavusele.

Võtmesõnad: Android, hooldatavus, mõõdikud, tarkvaratehnika

CERCS: P170 Arvutiteadus, arvanalüüs, süsteemid, kontroll

Contents

After reading, it seems logical to me to create new chapter replacing current 3. and 4. in the following way:
(old chapter numbers) -- new chapter:
3.1.1 + 4.1 - New Chapter called Android Developer Survey
3.1.2 + 4.2 - New Chapter called Interviews at Mooncascade
3.2 + 4.3 - New Chapter "Object Oriented Metrics-based Evaluation" <-- this title may be updated to something better

Should rename

1 Introduction

In the last decade, the impact of smartphones on our lives has significantly increased, and smartphones and mobile applications became people's primary way of interacting with technology. This situation has made the applications that work on these smartphones a vital part of daily and business life, from ordinary people to large companies. Today, mobile applications have become one of the most critical parts of digitalisation. Notably, as a successful open-source mobile operating system, Android has been a core element of this change, and the demand for Android applications has increased. Android application development has become one of the most necessary parts of the business area with a significant market share. Today, there are more than 2.5 billion active Android devices in the world [1]. Therefore, it is difficult to ignore the importance of Android application development. However, the increasing importance of the mobile era also brought more challenges to mobile application development and Android application development.

Mooncascade is a product development company based in Estonia. The company provides different software development services to its clients, including Android development. With the increasing demand for mobile applications, the company has been facing various challenges when providing Android application development services to different customers from different fields during the development process of Android applications. These difficulties can be examined under four major topics. Android's platform-specific complexity, sophisticated business needs, the frequent update rate of Android applications, and, lastly, growing codebases and fast-changing development teams. To overcome these challenges, "maintainability" emerges as one of the most important non-functional requirements for this purpose. Developing maintainable Android applications is the way to survive in the competitive market time and cost-effectively.

What is maintainability? How is it usually measured? Is measuring it easy or is that a problem? You can mention the things you go into depth about later thesis, at this stage you just name them as important sub-parts and cite related materials.

This study aims to evaluate the effectiveness of the Android development methods
nd technologies used by Mooncascade in terms of maintainability. To that aim, both
ualitative analyses (via interviews and surveys) and quantitative analysis (via object-
riented software quality metrics) are conducted. The main motivation for this study is
o eliminate the problems arising from the lack of maintainability in Android applica-
ion development and, thus, improve developer/team efficiency and provide time and
ost-efficient solutions to the clients. The target audience of this study includes Android developers and researchers who are already experienced with Android application development basics. The study only focuses on native Android application development.

Here you describe the solution (what to do/ how to study) to the problem, but your problem statement has not yet beenexplained in detail.

First describe the problems, then talk about your solutions how you will tackle the problem.

The author contributed to understanding how Android application codebases can be measured in terms of maintainability and providing empirical data on the impact of the well known Android development methods and technologies on maintainability.

1.1 Problem Statement

Mooncascade is a product development company based in Estonia. The company provides software development services, including Android application development. Demand for mobile applications in the industry has been increasing in the last decade. The company has been having an increasing number of requests for Android applications and facing several different challenges during the development and maintenance phases of these Android applications. These challenges are detailed as follows:

Android platform-specific complexity: A typical Android app consists of components such as activities, fragments, services, broadcast receivers, and content providers provided by the Android SDK and controlled by the Android OS. Besides, there are other components used in an Android app that are not a part of the Android SDK, e.g. managing network operations and business rules. The necessity of working together in harmony for all these components make Android applications complex software systems.

Business-specific complexity: Android applications get more and more sophisticated to fulfil increasing user needs, and apps become more business-critical. Consequently, the complexity of Android apps increases in terms of software complexity [2]. This situation increases the complexity and affects the maintainability of the Android applications.

High update rate: Android applications have a high update rate because of bug fixes and the frequent addition of new features based on the changing business requirement and user needs. That makes the software development life cycles of the Android applications quite active [3]. For that reason, Android applications should be developed in a way that modifications for new features and bug fixing can be done smoothly.

Growing Codebases and Fast-changing Development Teams: It gets harder to maintain Android applications as the codebase, and the development team grows or changes. The codebase of an Android application must be in an orderly fashion that enables developers to read and understand the app's purpose quickly.

The team aims to develop maintainable Android applications to solve these problems and have internalized some methods and technologies for this purpose. Although these methods and technologies are widely known by the Android community and are believed to be useful for maintainability, their impact on maintainability is empirically unknown. Thus, to evaluate the impact of these tools, techniques, and technologies used by Mooncascade's Android in terms of maintainability, this study will answer the following research questions.

- **RQ1:** How can the maintainability of Android applications be measured?
- **RQ2:** How efficient are the methods and technologies used by Mooncascade when developing Android applications in terms of software maintainability?

Here, after raising RQ-s, you should now explain how this thesis is going to attempt to answer these questions.

End of introduction chapter should include outline of rest of the chapters (See other theses for examples)

2 Background

In this section, the background related to the study topic is shared. The section covers the software engineering and Android concepts that are important for this study. Also, it covers the literature review and the way of Android development at Mooncascade. Thus, to facilitate the understanding of maintainability, which is the focal point of this study.

2.1 Android Platform

2.1.1 Android OS

Android is an open-source operating system for mobile devices. The Android project was initially created by the Open Handset Alliance which includes organizations from various industries such as Google, Vodafone, T-Mobile, LG, Huawei, Asus, Acer, and eBay to give some examples[1]. The main goal of the Android project is to provide an open software platform accessible for a variety of stakeholders such as developers, engineers, carriers, and device manufacturers to turn their innovative and imaginative ideas into successful real-world products that improve the mobile experience for the end-users. Today, numerous organizations from Open Handset Alliance and also other organizations are supporting and investing in Android and the project is led by Google. Android is designed in a distributed way to avoid the issue of the central point of failure. In another means, different industry players confine or control the advancements of another. As a result, a production-quality consumer product comes along with open source code that is ready for customization.

2.1.2 Fundementals of Android Applications

Since the launch of the first mobile device that works with Android, the Android operating system has improved, and the way the Android applications are developed changed a lot, but some fundamental components for developing Android applications have more or less stayed the same. Each of these fundamental components was developed for a specific purpose by the creators of the Android Software Development Kit (Android SDK). Knowing these components and their responsibilities are necessary to understand the problem that this study is trying to solve.

Java and Kotlin programming languages can be used to develop native Android applications. There are also other ways of developing Android applications, but this study only focuses on native Android application development. Native Android development

[1] http://www.openhandsetalliance.com/oha_members.html

means creating Android applications that run on Android-powered devices by using the Android Software Development Kit. Native Android applications consist of XML resources, images, data files, classes and so on. To understand the problem that this study tries to resolve, it is essential to understand the fundamental Android components. It is the first step for solving the maintainability issues of Android applications. These fundamental components are Activities, Services, Broadcast receivers and Content providers. In addition to these main components, the Fragment component also has an important place among Android components. Knowing the details of all these components (e.g. implementation details, life-cycle) plays an important role in solving the complications caused by Android components while developing Android applications. Details regarding key Android components and their responsibilities are beyond the scope of this study. For this reason, the summary information in this section has been shared to address these components, albeit briefly, and to raise awareness about their effects on the maintainability of Android applications. Also, examining the official Android documentation in this area can facilitate understanding the study topic[2].

2.2 Key Software Engineering Concepts

2.2.1 Maintainability

"Good programmers write code that humans can understand." - Martin Fowler [4]

A few decades ago programmers were using low-level programming languages that are working close to computer CPUs. These programming languages were designed to be understood by computers, not humans because computers lacked proper hardware, resources, and speed. Priorities back then were different. The computer programs had to be fast and less memory consuming. However, this situation has changed. Today computers are much stronger and software systems are more complex. However this situation also brought some difficulties to software development. Especially when developing large-enterprise software products, not considering how to overcome these difficulties may cause significant failures. In this context, this new reality brought new standards to software development. New programming paradigms were born and the priorities have altered [5].

When the priorities of modern software development are analyzed today, the maintainability of software systems emerges as one of the most critical ones. According to the IEEE Standard Glossary of Software Engineering Terminology, the term "maintainability" is the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed [6]. In other words, it is how well a software system is understandable, repairable, and

[2] `https://developer.android.com/guide/components/fundamentals`

extendable. Software systems are born, they live, they change, and eventually, they die. During their lifetime, new features are added, some features are removed, bugs are fixed, and often their development team changes. Usually, there is always a time gap between these changes. Developers should be able to understand the systems easily even months after. Besides, changes to the code base should be able to be done with ease without breaking the other parts of the software system. Ignoring all these might cause companies a significant amount of time and money. Developing maintainable software systems is the way to tackle such issues. In his famous book "Clean Code", Robert C. Martin explains how a top-rated company in the late 80s was wiped out from the business due to the lack of maintainability and poorly managed code organization. When the release cycles of their product extended due to the unorganized code base of their product, they were not able to fix bugs, prevent crashes, and add new features. Eventually, they had to withdraw their product from the market and went out of business. Lousy code and, consequently, lack of maintainability was the reason for this company to go out of the business [7]. This real-life example clearly shows how vital maintainability is to software systems and what fatal consequences it can cause if ignored.

"Importance of maintainability" - in other words, motivation

sounds likesomething which better belongs in introduction..

The importance of maintainability for software systems can also easily be seen when looking at its effect on the software development lifecycle and software development costs. A study has shown that the relative expense for maintaining software and dealing with its development speaks to over 90% of its absolute expense [8]. The maintenance period for a software system starts as soon as the system is developed. Thus, maintainability becomes a vital aspect for applying new customer needs, adding/removing new features, adapting to the environmental changes [9]. The time that has to be spent on the maintenance of complex software products is comparatively more than the rest of the software development lifecycle processes. Reports indicate that the amount of effort spent on software maintenance is between 65% and 75% of the total amount of effort [10]. Also, another report points out that maintenance cost is 75% of the total project cost, and the cost for maintaining source code is ten times bigger than developing the source code [11].

2.2.2 SOLID Principles

The steps to be taken to increase the maintainability of the software systems can be taken at the design stage. The SOLID Principles are very efficient when it comes to solving such design issues and increasing maintainability of software systems. SOLID stands for five principles, namely Single responsibilities principle, Open close principle, Liskov substitution principle, Interface segregation principle, and Dependency inversion principle. Application of SOLID principles when developing software systems facilitates improving critical factors such as maintainability, extendability, readability, and reducing code complexity and tight coupling, increasing cohesion [12]. Not following SOLID principles may lead to serious maintainability problems in the software development lifecycle, such as tight coupling, code duplication, and bug fixing. Application of SOLID design principles when developing software systems helps to achieve essential quality factors such as understandability, flexibility, maintainability, and testability [12].

2.2.3 Separation of Concerns

Today's software systems have many complex concerns such as persistence, real-time constraints, concurrency, visualization, location control [13]. Software engineering, first of all, aims to increase software quality, lower the expenses of software production, and assist maintenance and development. In achieving these aims, software engineers are continuously on the lookout for developing technologies and approaches that increase maintainability, decrease software complexity, enhance understandability, support reuse, and boost development. That is where "Separation of Concerns" (SoC) emerges as a solution. In the context of software engineering, SoC is a software design principle for separating a software system into discrete modules that each module addresses a single concern. A good application of SoC to a software system provides benefits such as increasing maintainability, reducing complexity. The borders for different concerns might differ from a software system to another. Concerns depend on the requirements of a software system and the forms of decomposition and composition [14].

Maybe you can add an example relevant to your context (Android) - MVVM has emerged as a way to foster SoC through architectural pattern

2.2.4 Software Architecture

It is not good academic style to start subsection with a quote. Quotes should be used as part of sentence, not standalone like here

"Good architecture makes the system easy to understand, easy to develop, easy to maintain, and easy to deploy. The ultimate goal is to minimize the lifetime cost of the system and to maximize programmer productivity." - Robert C. Martin [15]

Software systems grow and change over the time. As a software system changes, the interactions between different system elements lead to complexity. To develop reliable software systems that can overcome this complexity, programmers must implement software systems in a generalized fashion. Software systems written in such a fashion can be considered reliably usable, maintainable, testable, and extendable [16]. This fashion is called software architecture. Martin Fowler defines the software architecture as "the shared understanding that the expert developers have of the system design" [17]. The formal definition of software architecture is "the set of significant decisions about the organization of a software system, the selection of structural elements and their interfaces by which the system is composed, together with their behaviour as specified in the collaborations among those elements, the composition of these elements into progressively larger subsystems, and the architectural style that guides this organization – these elements and their interfaces, their collaborations, and their composition" [18].

The role of software architecture in software development can be elaborated under the six major aspects. Those aspects can be listed as follows [19]:

- **Understanding:** Software architecture simplifies the understanding and improves the readability of software systems.
- **Reuse:** Software architecture facilitates the code reuse between the components.
- **Construction:** Software describes the components of the system and dependencies between those components.
- **Evolution:** Software architecture helps developers to understand the consequences of changes more accurately and describes the concerns of the software systems.
- **Analysis:** Software architecture enables analyzing the system consistency, compliance with restrictions imposed by the architecture, and with the quality attributes.
- **Management:** Assessment of software architecture facilitates understanding of risks, requirements, and implementation strategies.

When the list above is examined, the effects of these roles on the maintainability of software systems are obvious. This situation is also clearly demonstrated in a study on the architecture of Android applications. A related study that is conducted on the topic of Android application architecture had revealed that the top quality requirement for Android application architecture is maintainability [20]. Results can be seen in Fig. 1.

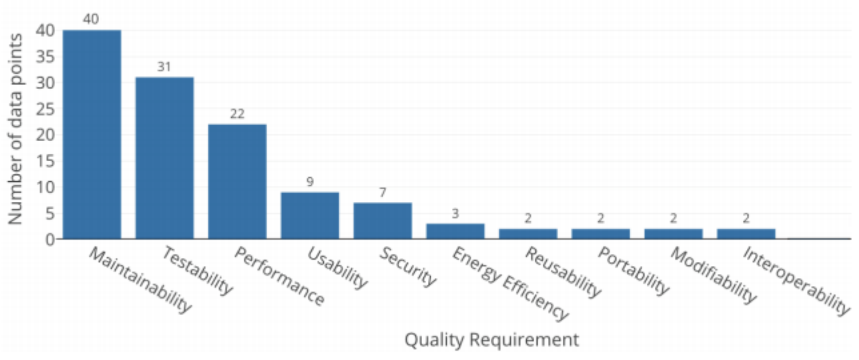


Figure 1. Quality requirement rankings for Android app architecture [20]

Also, other quality requirements presented in Fig. 1, are in correlation with maintainability. As a consequence, the impact of the architecture on the maintainability of software systems is one of the top aspects that should be considered before starting the development. As Brian Foote quotes, "if you think good architecture is expensive, try bad architecture"[15].

2.3 Android Development at Mooncascade

Mooncasade's Android team has made important strides in standardizing the methods and technologies they recently use. Therefore, up-to-date methods and technologies that can be used effectively by all team members have been determined, and different developers have applied these methods and technologies in different projects. The biggest reason for this change is undoubtedly the time and cost-effective development of Android applications and the increase in the development teams' effectiveness. In fact, looking at the general situation, it can be said that the aim is to develop maintainable, high-quality Android applications. In this section, brief information about the methods and technologies used by Mooncascade's Android team while developing Android applications are presented. Sharing information about these methods and technologies at the baseline level is important for this study because qualitative and quantitative evaluations of their impact on maintainability and obtaining results on these effects are the primary purpose of this study.

First of all, the team uses Kotlin programming language for Android development. It is believed that the use of Kotlin can improve code quality, readability, and productivity [21]. It is also the supported programming language by Google [22]. For the same reasons the team prefers Kotlin over Java. Also, the team tries to adapt the Clean Code and SOLID principles to their daily programming tasks. Clean Code principles are believed to be to improve the readability and understandability of the code [23]. Also, the application of SOLID principles facilitates the separation of concerns and improves code quality [12]. The team applies these principles for these reasons and tries to maximize their application by code reviews.

When it comes to the architecture of Android application, the team prefers Clean Architecture. The Clean Architecture facilitates changing third-party dependencies and implementation details of the software systems that it is used on. Also, it helps to modify the different layers of the application without affecting the business logic. Thus, it makes Android applications easy to understand, modify and test [24]. The team also uses MVVM design pattern to present data and applies this design pattern with the help of tools provided by the Android Architecture Components framework published by Google's Android team[3]. Displaying data on the UI according to the Android applications' business requirements properly and efficiently is a difficult problem to tackle when projects get more complicated. For this reason, the usage of a proper design pattern for the presentation of the data is essential. An important for an efficient design pattern is eliminating the bidirectional dependency of views and view models. Thus decoupling of data and view becomes possible, and an important software design objective of high cohesion and low cupping is achieved. The MVVM design pattern

[3]https://developer.android.com/topic/libraries/architecture

provides these requirements for Android applications [25].

The team also uses some third-party Android and Java/Kotlin libraries to develop Android applications. Although such libraries are not a must when developing Android applications, the use of some of them saves Android developers a lot of time and effort. However, the team tends to reduce the use of third-party libraries as much as possible and prefers to use raw solutions whenever possible. Community support, up-to-dateness, reliability and long-term maintainability criteria are considered when selecting the third-party libraries used. RxJava, Dagger 2, Retrofit, Apollo and Android Architecture Components are the most prominent libraries/frameworks and have the most impact on concepts such as Android application architecture and maintainability. RxJava is a library for designing asynchronous and event-based software applications by using observable sequences[4]. Dagger is a static, compile-time framework used for dependency injection and it can be used by Java, Kotlin, and Android based applications[5]. Retrofit is used to integrate REST-based back-end systems to Android applications[6], while Apollo is used for the integration of GraphQL based back-end systems[7]. Lastly, Android architecture components are a set of libraries that facilitate designing robust, testable, and maintainable apps[3].

I like this part of the writing: it has lots of information and direct details, explanations supported by references,

very little "bla-bla" ;)

As previously mentioned, evaluating the impact of the methods and technologies shared in this section on the maintainability of the Android applications constitutes the backbone of this study. The way of evaluation is presented in the third section.

2.4 Literature Review

In this section, the outcomes obtained from literature review conducted on the topic of Android application development are presented. Considering the tight relationship between the topic and industry, a Systematic Literature Review (SLR) was not adequate for finding relevant resources of data for the study. Hence, a Multivocal Literature Review (MLR) was conducted. As a type of SLR, MLR is collecting grey literature as well alongside formal literature [26]. MLR considers resources like blogs, white papers, articles, academic literature and allows gathering information from academics, developers, practitioners, and independent researchers [27]. Regardless of the direct relevance of the studies to maintainability, the research query presented in Fig. 2 has been used to determine all the studies related to Android application development. For the purpose of obtaining more successful findings, the search query was separated into six parts, each of them individually focusing on a single topic.

[4] https://github.com/ReactiveX/RxJava
[5] https://dagger.dev/
[6] https://github.com/square/retrofit
[7] https://www.apollographql.com/docs/android/

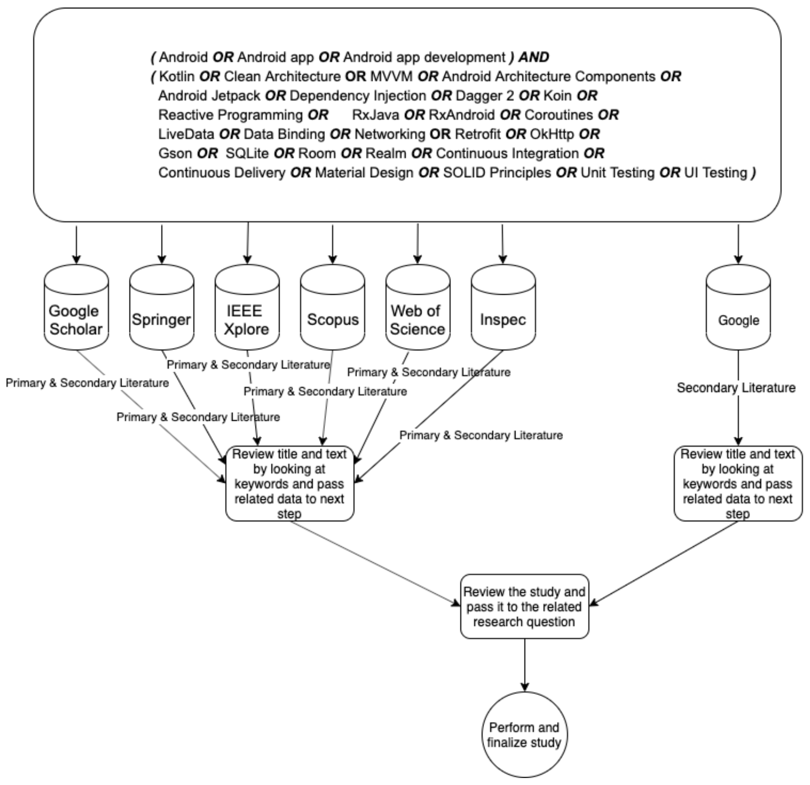
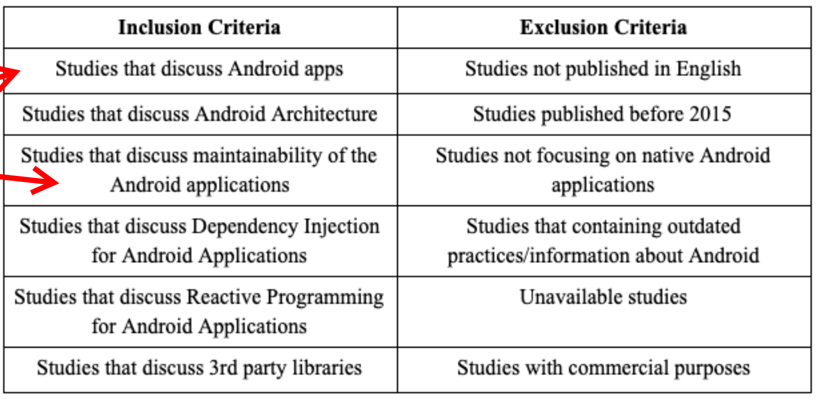

(Android OR Android app OR Android app development) AND
(Kotlin OR Clean Architecture OR MVVM OR Android Architecture Components OR
Android Jetpack OR Dependency Injection OR Dagger 2 OR Koin OR
Reactive Programming OR RxJava OR RxAndroid OR Coroutines OR
LiveData OR Data Binding OR Networking OR Retrofit OR OkHttp OR
Gson OR SQLite OR Room OR Realm OR Continuous Integration OR
Continuous Delivery OR Material Design OR SOLID Principles OR Unit Testing OR UI Testing)
Google Scholar
Springer
IEEE Xplore
Scopus
Web of Science
Inspec
Google
Primary & Secondary Literature
Secondary Literature
Review title and text by looking at keywords and pass related data to next step
Review the study and pass it to the related research question
Perform and finalize study

Figure 2. Search process visualization

After executing the query above, inclusion and exclusion criteria were applied to filter irrelevant studies. The inclusion and exclusion criteria are shown in Fig. 3.



Inclusion Criteria	Exclusion Criteria
Studies that discuss Android apps	Studies not published in English
Studies that discuss Android Architecture	Studies published before 2015
Studies that discuss maintainability of the Android applications	Studies not focusing on native Android applications
Studies that discuss Dependency Injection for Android Applications	Studies that containing outdated practices/information about Android
Studies that discuss Reactive Programming for Android Applications	Unavailable studies
Studies that discuss 3rd party libraries	Studies with commercial purposes

isnt it redundant to have both? Maybe 1st one is unnecessary, if all others are for android anyways

Figure 3. Inclusion and exclusion criteria

Finally, forty nine papers were found and reviewed. Grey literature results are not included in these numbers. However, the reviewed grey literature was also cited in this study. The literature found as a result of this query was read thoroughly. Finally, inclusion and exclusion criteria were applied to the primary studies to have suitable literature only. As soon as the early research outcomes were collected, chasing the inclusion and exclusion criteria shown in Fig. 3, the unrelated material was filtered out. Also, the studies conducted before 2015 are excluded from the query results to prevent any possible up-to-dateness issues. Among the determined studies, those related to maintainability have been examined in more detail. These reviewed academic studies examined on Android application development can be grouped according to their topic titles as follows.

- 4 studies bachelor theses regarding Android application development.
- 5 papers were related to 3rd party Android libraries.
- 6 papers were related to Kotlin programming language usage for Android.
- 6 papers were related to maintainability of the Android applications.
- 12 papers were related to Android app architecture.
- 16 papers were related to Android application development but not directly related to the topic that this study covers.

In addition to the papers mentioned above, 17 other papers are reviewed to get a better understanding of the studies conducted regarding software maintainability, metrics that can be used to measure software maintainability, general software engineering principles such as separation of concerns, and SOLID. These articles and books have been cited in various parts of this study.

Among the reviewed academic studies reviewed, the studies on Android application architecture and maintainability draw attention. Studies conducted on these two subjects account for more than one-third of the total studies examined. Based on these numbers, it can be said that the Android application architecture and the importance of maintainability are recognized by the researchers working in the Android field. In most of these studies, it is noteworthy that comparisons of Android application architectures in terms of performance, maintainability, and testability are common. As mentioned in previous chapters, considering the impact of software architecture on maintainability and the importance of maintainability in software development processes, it is not surprising that many academic studies have focused on these topics. Various studies on the maintainability of Android applications draw attention. For example, Hugo Källstrom (2016) conducted a study ~~on a similar topic~~ in which he implemented three different software architectures and evaluated the maintainability based on these architectures [28]. Also, Prabowo et al. (2018) have a research on the maintainability of Android applications

The level of detail is a bit problematic. Mostly, you are just mentioning that the papers exist, there is almost no analysis of their contents. What are the key methods arising? What types of conclusions are these papers making? What kind of analysis are they doing? They also usually identify research challenges?

as a reader i dont get any overview into the topic, I mainly learn that these papers exists which they compared MVP and anti-pattern approaches [29]. The research of Verdecchia et al. (2019) on architectural choices in Android applications is worth reading for both architecture and maintainability topics [20]. Lastly, while the research of Saifan and Al-Rabad (2017) presents a different perspective on measuring the maintainability of Android applications [30], the study of Andrä et al. (2020) draws attention as the most recent research on this subject [22]. On the other hand, most studies approach the maintainability of Android applications from an architectural perspective only. It is possible that other techniques and technologies can be used to increase the maintainability of Android applications besides architectural patterns. The fact that the studies do not focus on these techniques and technologies can be considered a limitation of the academic literature regarding this topic.

2.5 Summary

In section two, a set of essential information regarding software engineering and Android platform information was shared. The purpose of sharing fundamental software engineering principles such as maintainability and SOLID principles is to provide background information for this study. Again, with the same purpose, brief information about the Android platform was also shared. Also, information regarding Android development at Mooncascade was presented. Lastly, the results of conducted multivocal systematic review were shared.

this is poor style. you just mentioned what concepts you mentioned, but you should highlight the most important new details the reader now has seen after reading the chapter.

As it is right now, this part should be removed fully, it does not add much information, the same information has been give in chapter 2 beginning

3 Research Methodology

As stated before, this study focuses on evaluating the techniques and technologies used by Mooncascade's Android team from the maintainability point of view. In this section, the methods to evaluate the effectiveness of these techniques and technologies are explained. These evaluation methods are both quantitative and qualitative. In this way, it is aimed to evaluate with maximum efficiency by going beyond the traditional quantitative measurement techniques. Also, why and how these methods are chosen are explained in this section. Thus, the first research question is answered.

3.1 Qualitative Method

When other academic studies dealing with the measurement of maintainability of software systems were reviewed, it was seen that many studies use quantitative methods. For example, Verdecchia et al. researched this topic [20]. Although it cannot be claimed that such quantitative measurements are inaccurate, it would not be wrong to say that these measures are inadequate at times. It is essential to make qualitative evaluations and quantitative evaluations to increase the efficiency of the evaluations. In this way, it may be possible to measure developers' experiences and their effects on maintainability. For these reasons, an Android developer survey and interviews were conducted within this study's scope. This section covers these qualitative methods.

3.1.1 Android Developer Survey

The first step of qualitative evaluations in this study is the Android developer survey. The consistency and stability of principles and third-party libraries used in Android applications indirectly affect the maintainability of applications. Therefore, although it does not directly contribute to measuring maintainability, this survey was conducted to identify current Android trends and provide support for this study's evaluation. While determining this survey's questions, priority was given to principles and technologies that directly and indirectly affect maintainability. Although the questions are generally prepared to cover the methods used by the Mooncascade Android's team, it can be said that the questions reflect the Android technology stack in general. The content of the survey can be accessed publicly[8]. The questions are organised with the help of the Forms application provided by Google. The Android developer survey has been delivered to Android developers from different companies in different countries through accounts or groups of Android developer communities on social media platforms such as LinkedIn,

[8] `https://forms.gle/MoiTGMV874yzJwuv8`

Discord, and Twitter. The author of the study has also shared the survey with many of his colleagues/ex-colleagues working in different companies/countries.

3.1.2 Interviews with Team Members

The second step of the qualitative evaluations carried out within this study's scope is the interviews conducted with Mooncascade's Android team members. Unlike the Android developer survey that was explained in the previous section, these interviews are designed and conducted to qualitatively evaluate the techniques and technologies used by Mooncascade's Android team in terms of maintainability. Also, the interviews aim to determine the importance of maintainability from the case company's point of view. Thus, proving or disproving the study's claim that maintainability is a key concept to overcome the issues mentioned in the problem statement section would be possible. Eight questions were determined for this purpose. The interview questions can be accessed publicly[9]. Three main criteria were taken into consideration while preparing these questions. First of all, questions were chosen to get to know about the participants' background and experience. Later, some questions were designed to learn participants' understanding of maintainability in software engineering. Lastly, questions were drafted to learn about participants' thoughts about the impact of technologies and principles used by Mooncascade on maintainability. The interview was conducted privately with each team member. It is aimed that the data gathered through these interviews will support the accuracy and validity of evaluations.

I think this section better fits into backgorund

3.2 Maintainability Evaluation with Object-Oriented Metrics

As a part of this study, many other academic works on the measurement of maintainability in Android and software engineering were examined to find the appropriate maintainability evaluation methods. The examination has shown that different methods can be used to evaluate the maintainability of software systems [31, 32]. Particularly for evaluating the maintainability of Android applications, different studies have been conducted by using different methods [22, 30]. Various metrics can be used to evaluate object-oriented software systems in terms of maintainability. These metrics are detailed in several studies [33, 34]. As a result of this examination, it was seen that the concepts of complexity, cohesion and coupling were emphasised in many studies, and it was concluded that the measurements made based on these concepts would be more efficient when measuring maintainability. Studies have shown that results retrieved from evaluating these concepts proved to define the level of maintainability [33]. Therefore, research was done on measuring the concepts of complexity, cohesion and coupling effectively, and five metrics

[9]https://forms.gle/paMSj5e8Up5ZW6raA

were selected for this purpose. While selecting these metrics, priority has been given to metrics that can handle a software system as a whole in the areas of complexity, cohesion and coupling to make more efficient evaluations. These metrics are Weighted Method Count **(WMC)**, Depth of Inheritance Tree **(DIT)**, Number of Children **(NOC)**, Coupling Between Object Classes **(CBO)**, and Lack of Cohesion of Methods **(LCOM)**. Detailed information regarding these metrics can be found in various studies [33, 30, 35, 34].

After determining the metrics to be used, the options were to apply the metrics manually or with a tool. Since manual implementation is prone to error, it has been decided to apply metrics with a reliable tool. After research, the CodeMR static code analysis tool drew attention. CodeMR is a powerful software quality tool that is integrated with IDEs and supports multiple programming languages. The tool provides a set of metrics to measure coupling, complexity, cohesion and size. These metrics are often affected by various code characteristics, making them promising for evaluating software maintainability. Besides, the tool provides a visualisation centric approach and generates detailed reports supported by different visualisation options. In this way, it facilitates understanding the results and allows understanding the bigger picture of the projects from the complexity, coupling, and cohesion point of view. Detailed information on these metrics and assessment methods can be found on the tool's website[10]. The tool can also be installed as a plugin in Android Studio and is very easy to use. Also, two studies conducted using this tool were examined to gather more information regarding the tool [36, 37]. Lastly, the CodeMR static code analysis tool has been chosen for this study due to reasons mentioned above. After this decision, communication was established with the CodeMR team, and a free license was obtained to be used in academic studies.

To evaluate the impact of the methods and technologies mentioned in section 2.3 on maintainability, the metrics were applied on two different code bases of the same project. Thus, the impact of the team's current methods on maintainability could be measured by comparing the results from both versions of the same project. Although the project's full content cannot be published due to privacy principles and confidentiality reasons, more information regarding these codebases is shared in the next section.

3.3 Summary

In this section, detailed information about the qualitative and quantitative evaluation methods performed within this study's scope was shared. The knowledge regarding these methods' contents, why they were preferred, and how they were applied were presented. Thus, the first research question was answered.

[10]`https://www.codemr.co.uk`

4 Evaluation

In this section, the results obtained from applying quantitative and qualitative assessment methods, whose details were shared in section 3, will be presented. As explained in the 3rd section before, the Android developer survey findings and the results obtained from the interviews made with the members of Mooncascade's Android team, which were applied within the qualitative evaluation scope, will be shared in this section. Results obtained from the evaluation with object-oriented metrics, which were detailed in the third section, will also be presented in this section. Thus, through the results obtained from the qualitative and quantitative evaluations, the second research question will also be answered in this section.

4.1 Android Developer Survey

The Android developer survey results, which is the first step of qualitative evaluations, are shared in this section. Android developer trends have been identified regarding technologies and principles that are likely to directly or indirectly impact the maintainability.

4.1.1 Conduction Period of the Survey

The survey accepted answers between April 2020 and March 2021 and reached 164 participants in total.

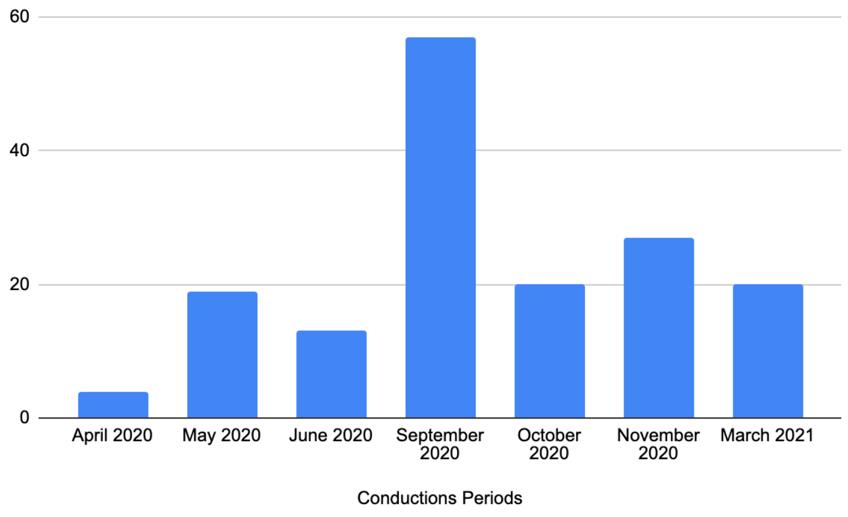


Figure 4. Conduction period results

As shown in Fig. 4, participation in the survey took place only during specific periods of this one-year term. Although the questionnaire accepted answers for a year-long, it was boosted in different periods, as shown in Fig. 4. It would be logical to consider this situation while examining the response numbers.

4.1.2 Participant Background

In this kind of survey, it is essential to ensure the participants' diversity to get sufficiently accurate and generally reflective results. The Android developer survey found participants in 5 different countries (Germany, Turkey, Portugal, Estonia, and Russia) through the author's current and former colleagues. The survey was answered by Android developers working in 7 different well-known companies in these countries. Also, as previously stated in the 3rd part, the survey reached answers from random Android developers through various social media platforms. In this way, the participants' diversity was increased, and getting better results was ensured.

~~Also, the~~ first question was added to the survey to learn about the competence of participants. As can be guessed, the probability of getting more accurate results from a survey formed to determine the preferred technology and methods for Android application development is directly proportional to the participants' experience. Results can be seen in Fig. 5 below.

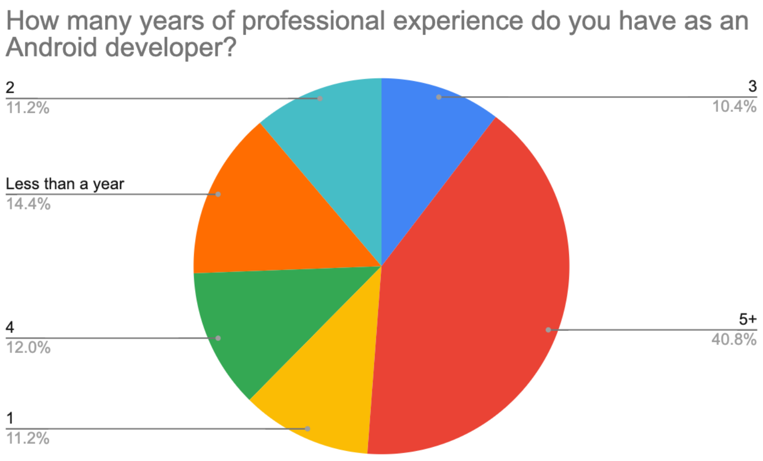


Figure 5. Participant background results

As stated before, the participant diversity was provided to improve the accuracy of the survey. When the chart above is examined, 63% of the participants have more than 3 years of experience while 40% of them has 5+ years of experience. People with 0-2 years of experience are around 37% of the all participants.

4.1.3 Issues

The most severe limitation of the survey was undoubtedly in finding respondents. Even though the survey was shared on many different social media platforms and developer community pages, it was able to find very few participants compared to the number of people on these platforms and communities. The initial target for the number of participants was 150-200, and a serious effort was made to reach this number.

Besides, some corrections and filtering were made for the 164 responses collected from these Android developers to avoid off-topic responses and collect non-standard answers under a single topic, thus presenting more consistent results both visually and numerically. For example, for some of the questions, there is the option "other" among the answer options offered, and users can choose this option and enter their answers. In this case, when the data is plotted, results such as the following may occur.

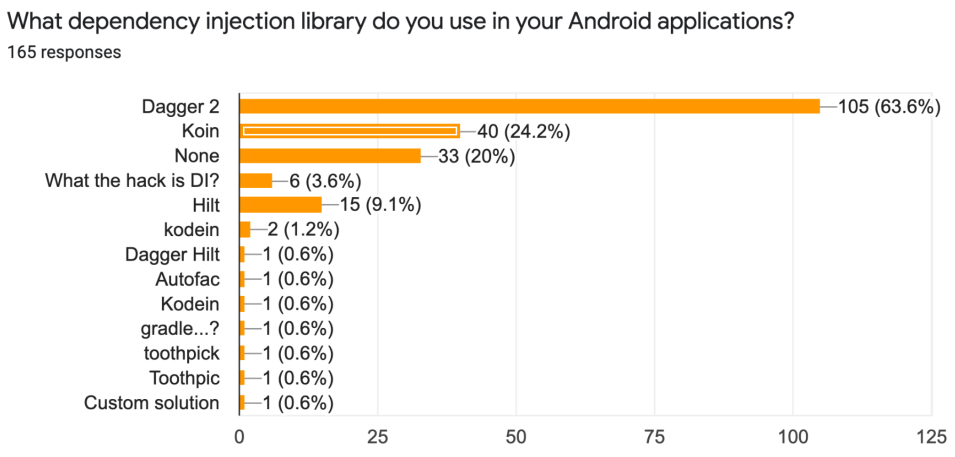


Figure 6. Example chart plotted with non-standardized answers

As shown in the chart, "Toothpick" or “Hilt” libraries do not have a ready-made response option. Android developers using this library have given their answers in different forms. Thus a chart such as this has emerged. Therefore, it was deemed necessary to arrange the inputs in different forms, which correspond to the same answer.

As an example of the need to filter some answers, a situation in the chart below can be shown. As can be seen in the chart above, which was obtained from the unfiltered survey results, it is seen that developers who develop Android in other forms also participated in the survey. Although it was clearly stated to the participants that the survey included only "Native" Android developers before they filled in the questionnaire, a few such cases were unfortunately not avoided due to the human factor. These kinds of responses have been filtered and edited as they will not contribute to this survey's purpose and reduce the survey's accuracy.

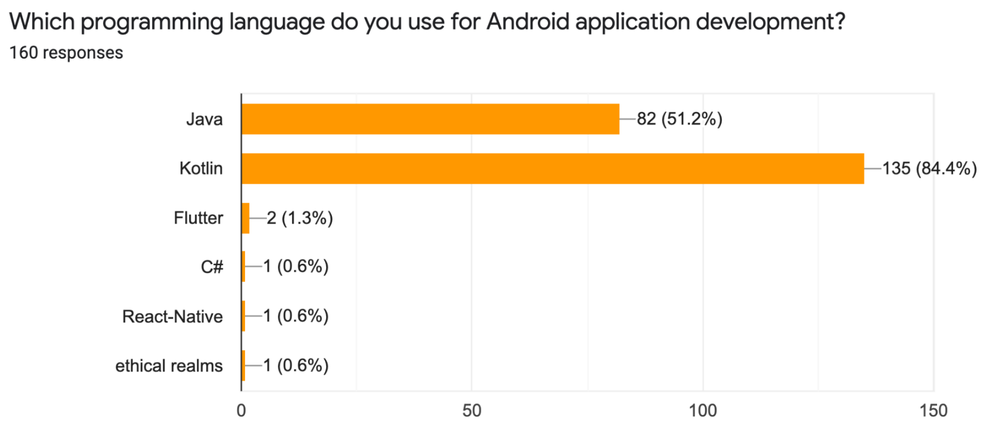


Figure 7. Example chart plotted with corrupted data

The above and other similar situations have been filtered and edited as they will not contribute to the survey's purpose and they reduce the survey's accuracy. Firstly, the survey results were extracted as a Google Sheets file to do this filtering and editing. Then inappropriate data in this file was corrected or filtered, and finally, the charts and numbers were obtained from the accurate data of the file. While applying corrections and filtering, no changes or manipulations were made on the relevant data. Around 20 of the 164 responses were edited to arrange the inputs in different forms, which correspond to the same answer. Besides, five corrupted responses were removed. After the filtering process, charts were created from the remaining 159 responses to obtain more accurate data. The interpretation of the responses is based on this filtered and corrected final version. The charts obtained through filtered and corrected responses and the data obtained from these charts' interpretation are presented below, respectively. Since it is possible to choose more than one answer for some questions, it should be taken into account that the total number of answers for each question may exceed the total number of participants. Besides, since the first question was added to the survey a bit later than the answers started being accepted, the answers to these questions are less than the rest. While examining the answers, it will be helpful to consider these two situations to prevent confusion.

4.1.4 Programming Language

The second question of the survey asks Android developers the programming language or programming languages they use to develop Android applications. As mentioned in section 2, different programming languages can be used while developing Android applications. Nevertheless, this study focuses solely on "Native" Android development, as mentioned many times before. Therefore, only Java and Kotlin programming languages are among the options offered to answer the second question.

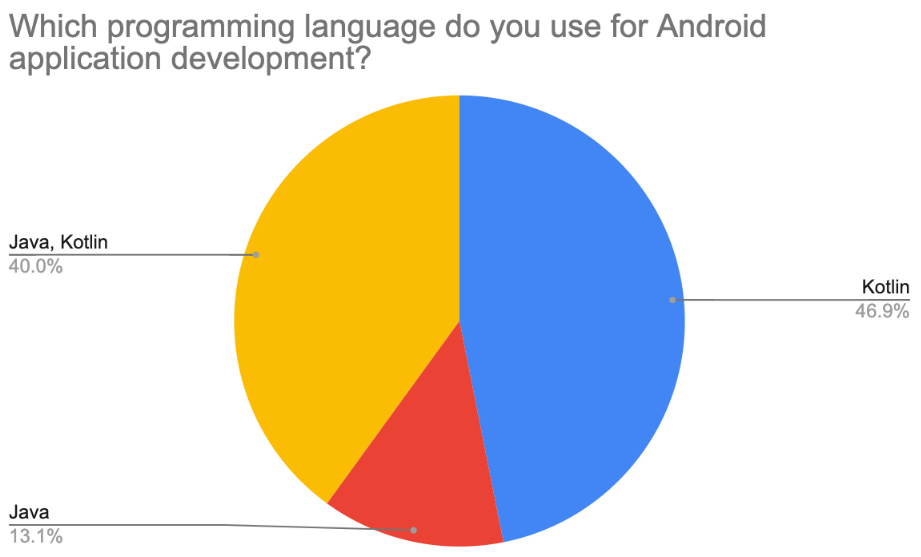


Figure 8. Programming languages results

In Fig. 8 above, Android developers' trends regarding the programming language they use while developing "Native" Android applications can be seen. Around 47% of the Android developers surveyed seem to use the Kotlin programming language. 40% of the participants use Java and Kotlin together, while only 13.1% use the Java programming language.

4.1.5 Architecture

The third question of the survey asks participants about their choice of presentational design patterns. In this question, MVVM, MVP, MVI, etc. design patterns were defined as presentational design patterns. When the results are examined, a diverse set of answers is seen. Below, in Fig. 9 the participants' preferences for presentational design patterns are displayed.

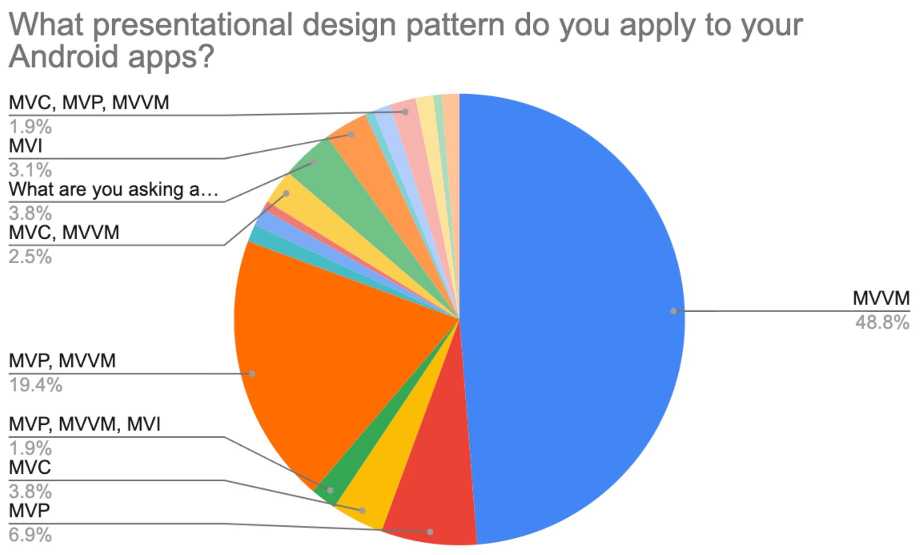


Figure 9. Presentational design patterns results

The first notable conclusion is that almost half of the participants use the MVVM presentational design pattern. Besides, it is seen that another 25% of the participants stated that they used this design pattern and different design patterns. In other words, a total of 3 quarters of the participants stated that they used the MVVM design pattern in some way or another. On the other hand, the chart presents us that design patterns such as MVP, MVC, MVI are also frequently used. When the participants' responses are sifted through, it is seen that design patterns such as MVP, MVC and MVI are generally preferred by some developers alongside the MVVM design pattern. These developers have more experience than those who have a single choice of design pattern. Also, it was seen that developers with 0-3 years of experience have answered this question by selecting the MVVM option. Another important detail is that 5 out of 6 participants that answered this question as "What are you asking about" had one year or less experience. Lastly, it was seen that the MVI design pattern was selected nine times in the last six months (out of 120 answers with a ratio of 0.075). However, it was selected only once (out of 40 answers with a ratio of 0.025) by the participants in the first six months in which the survey accepted answers.

In the following question, users were asked whether they use the "Clean Architecture". It is essential to mention why Clean Architecture was asked to the participants differently from the presentational design patterns. Clean Architecture allows the arrangement of an entire application in terms of architecture, unlike the presentational design patterns. The graphical breakdown of participant answers to this question is presented below in Fig 10.

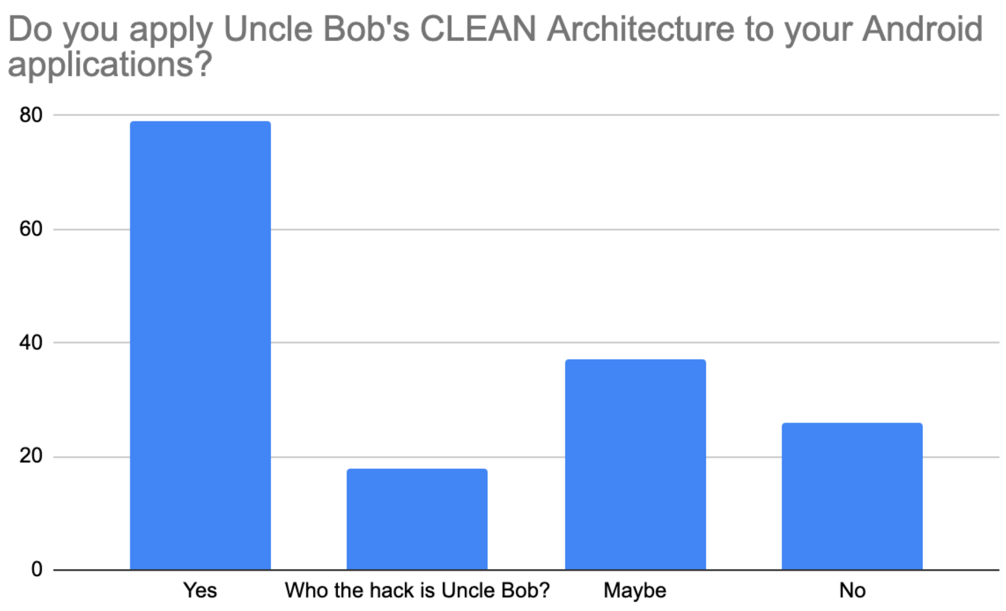


Figure 10. Clean Architecture usage results

When examining the participatory tendencies to use Clean Architecture, it is seen that the majority of the participants adopt this architectural approach. The number of respondents who declared their use of this architectural pattern is almost more than the total of those who declared that they did not or could use it. Besides, 38 of 51 Android developers with five or more years of experience who participated in the survey declared that they use or can use this architectural pattern.

4.1.6 Principles

The fifth question of the survey asked the participants whether they follow SOLID principles while developing Android applications. Results have shown that 66.3% of the participants declared that they follow SOLID principles and 20.6% of them declared that they might follow these principles. 6.3% of the participants stated that they do not apply the SOLID principles, and 6.9% stated that they are not aware of these principles. The figure below contains the graphical breakdown of this data.

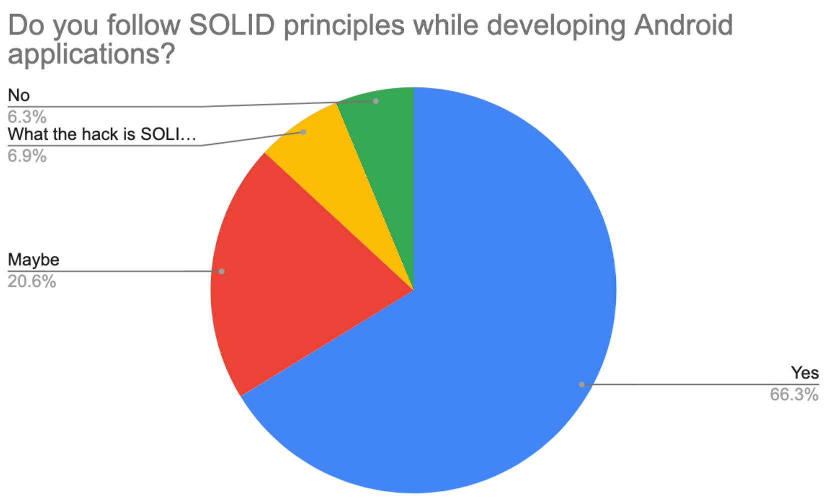


Figure 11. SOLID principles usage results

The 6th question of the survey asks the participants whether they apply the "Clean Code" principles while developing Android applications. The results in the form of a pie chart can be seen below in Fig. 12.

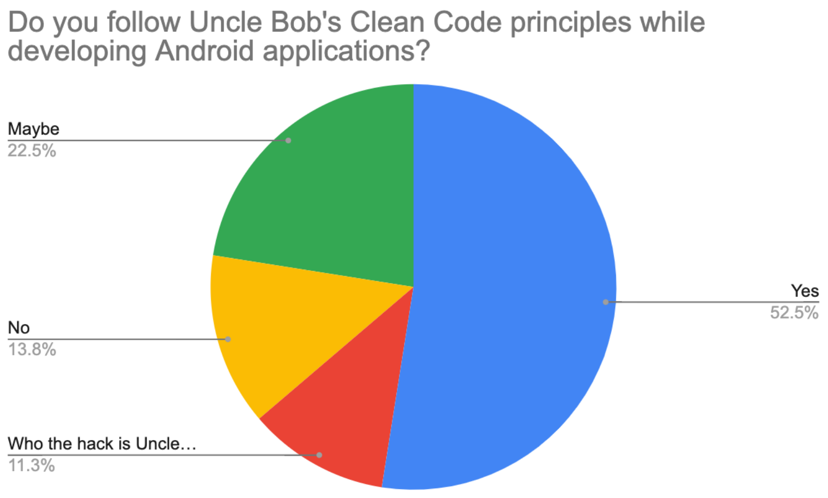


Figure 12. Clean Code principles usage results.

According to the results, 75% of the participants stated that they either used or could use these principles. While 13.8% of the participants stated that they do not use these principles, it was observed that 11.3% of the participants were not even aware of these principles.

4.1.7 Libraries

The next question is designed to ask the participants which networking library they use. The graphical breakdown of responses is presented below in Fig. 13.

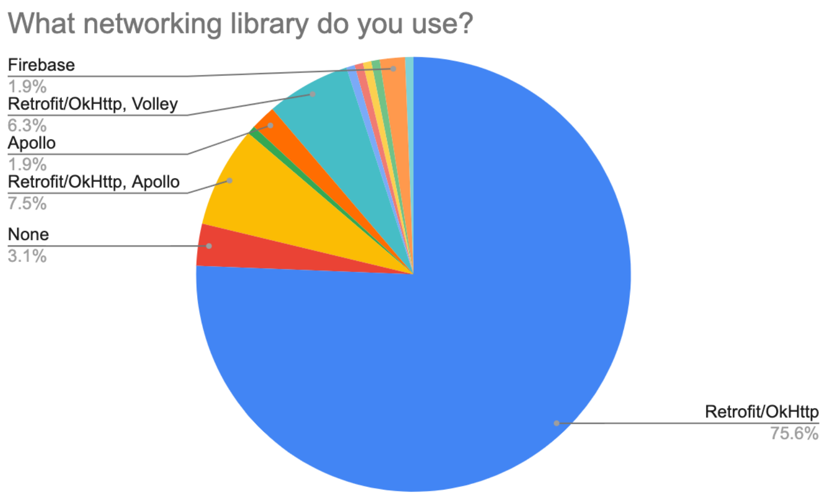


Figure 13. Networking library preferences results

When looking at the results, it is seen that the Retrofit / OkHttp library is dominating. It is observed that 75.6% of the participants use only this library and approximately 13.8% prefer Retrofit/OkHttp libraries and other libraries. Besides, it is seen that 9.4% (the second-highest rate) of the participants stated that they also used the Apollo library. Apollo, which is the most used library used in the integration of GraphQL based back-end systems to Android applications, has the second-highest rate among the answers.

The 8th question of the survey asks the question of what libraries they use to manage asynchronous processes while developing Android applications to the participants. When the results are examined, it is seen that Android developers mostly prefer the Kotlin Coroutines, RxJava and AsyncTask solutions. It is also seen that some of the participants declared that they used more than one solution. Recently, the AsyncTask has been deprecated by the Android team. However, it seems that some of the developers continued to use this solution. Besides, it is seen that the Kotlin coroutines, which Android officially recommends [11], has the highest percentage in the survey (32.5%), which is followed by RxJava (28.1%). Details of the results can be seen in the below in Fig.14.

[11]https://developer.android.com/topic/libraries/architecture/coroutines

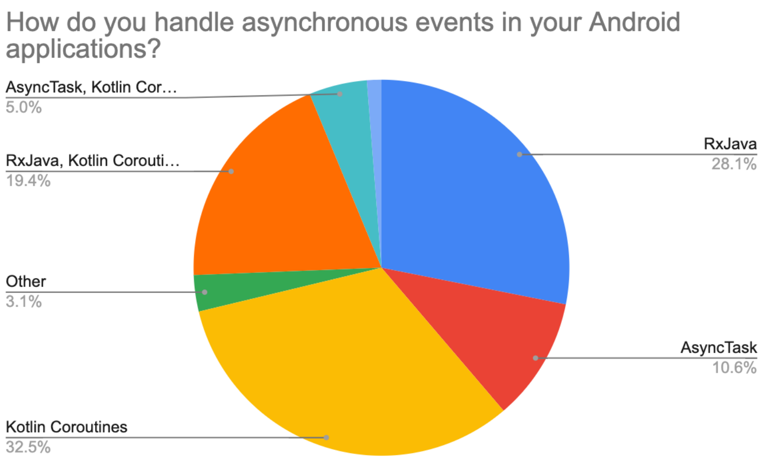


Figure 14. Threading management library preferences results

The 9th question of the survey asks the participants which solutions are preferred to apply dependency injection (DI) principles. As shown in Fig. 15, Dagger 2 is the most commonly used DI framework amongst the participant Android developers.

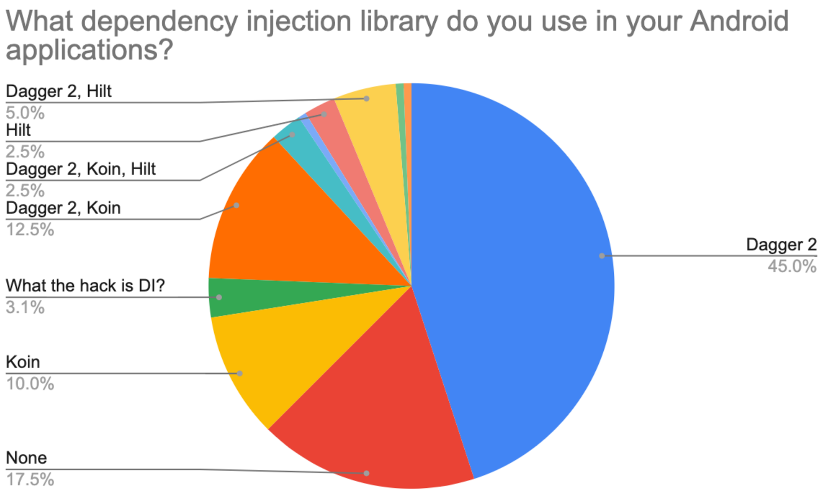


Figure 15. DI Library preferences results

Approximately 67.5% of users declared that they used this solution in some way. Besides, Hilt, another DI framework developed based on Dagger 2 by Google's Android team, a relatively new technology, was able to find a place in the survey. Apart from these solutions, the Koin DI framework also stands out among the results. Lastly, it is seen that 3.1% of the participants are not aware of the concept of DI and 17.5% of them declared that they do not use any framework for DI. Detailed results can be seen in Fig. 15 below.

The final question of the survey asks respondents whether they are using the Android Architecture Components framework. When the results are examined, it is seen that more than 92% of the participants stated that they use or can use this framework while only 7.5% of the applicants declared that they do not use it. Fig. 16 presents the participant responses to this question in the form of a column chart.

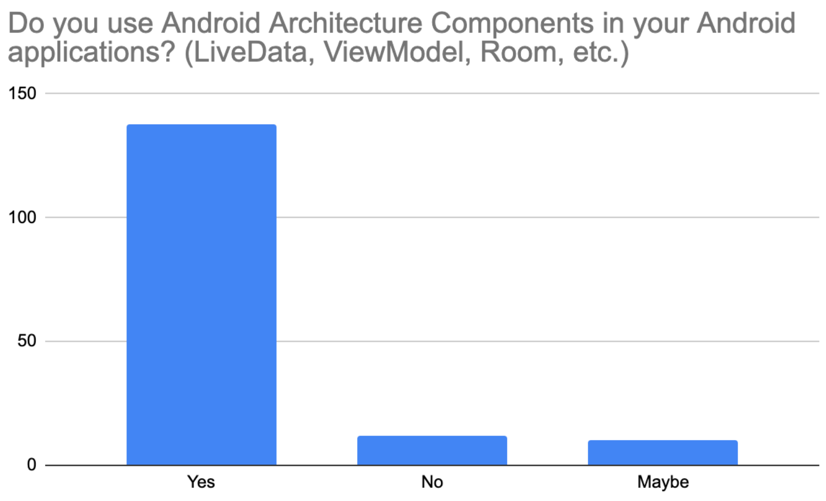


Figure 16. Android Architecture Components usage results

4.2 Interviews with Team Members

In this section, the results obtained from the interviews made with the members of the Mooncascde Android team within the scope of qualitative measurements are shared.

4.2.1 Conduction

The questions mentioned in the interview, the details of which are shared in section 3.1.2, were conducted with the members of Mooncascade's Android team within the scope of qualitative evaluations. Interviews were conducted online, privately with each member of the team. The team was eight in total, and every team member, except the author of this study, participated in these interviews. The interviews took place in April 2021, and all the responses of the team members were kept in written and video record format.

Subsequently, the presentation of the results was realized as a result of the rewording of these records.

4.2.2 Participant Background

The first three questions of the interview were designed to understand the competencies of the participants in the field of Android application development and to see how well their knowledge are regarding the concept of maintainability in software engineering. ~~When the answers given to these questions were examined, it was seen that~~ the participants had an Android application development experience ranging from 3 to 10 years. In addition, the participants stated that they accomplished many Android projects in different fields, and the Android applications implemented through these projects are the applications that are actively used today. Considering the knowledge levels of the participants about maintainability in software systems, it was seen that the participants emphasized the areas of readability, understandability, modifiability and up-to-dateness. Without exception, all the participants highlighted these concepts in their answers to the questions related to the understanding of software maintainability.

4.2.3 Importance of the Maintainability

The fourth, fifth and sixth questions of the interviews are designed to determine the importance of maintainability for the case company.

The fourth question of the interview asked the participants the importance of maintainability for the case company. Five of the participants stated that maintainability is an essential requirement for the company due to how the company works. Since the company works for different customers from different fields and the developer circulation in these projects is high, these participants stated that the maintainability of the projects is critical in terms of cost, time and effort. These participants emphasized that it is vital for the company to develop Android applications with high understandability, high modifiability, and high maintainability due to facilitating the development, hand over of the projects to the customer and maintenance operations required in the long term. Also, two participants stated that maintainability is already critical for software systems and that there is no extra situation that makes it more important for the company.

The fifth question of the interview asked the participants the importance of maintainability for the Android applications. Four of the participants stated that maintainability is important in solving the complexity issues in Android applications. Also, the participants stated that the fact that the Android platform and third-party Android libraries are evolving rapidly makes the maintainability of the Android application crucial. Participants stated that if unstable libraries are preferred for the sake of the industry trends the fast evolution of the platform may cause problems in terms of maintainability. Also, two participants stated that maintainability is already important for software systems and that there is no extra situation that makes it more important for the Android applications.

The sixth question of the interview asked the participants for the most important matter for Android applications when it comes to maintainability. In general, it was seen that the participants provided 2 different answers to this question rather than stating a single matter. When the responses of the participants are examined, it is seen that the application architecture is emphasized four times, the third-party libraries emphasized 4 times, and the documentation once. The reason why third-party libraries affect maintainability was explained in the previous paragraph. It was seen that the developers put the same emphasis when answering this question as well. In addition, it was emphasized by the participants that the choice of architecture increases the maintainability of the applications as it makes the codebases of the applications better structured, standardized and consistent. Lastly, one participant stated that the documenting of the Android codebases improves understandability.

4.2.4 Qualitative Evaluations

The last two interview questions were directed to the participants to qualitatively evaluate the methods and technologies used by the company while developing Android applications (see section 2.3) from the maintainability point of view.

When the responses of the participants are examined, positive comments about Kotlin programming language draw attention. Kotlin was found to be more concise and more readable than Java by the participants, and the participants stated that this situation has a positive effect on maintainability. Also, one participant stated that programming language did not have any positive or negative effect on maintainability.

When looking at the answers about the SOLID and Clean Code principles, again, mostly positive reviews are seen. Participants think that the impact of these principles are positive on maintainability in terms of regulating the coding conventions of their codebases, increasing readability, facilitating separation of concerns and increasing consistency.

On the subjects of architecture and design patterns, the participants stated that they found the MVVM design pattern positive for maintainability for it decreasing the coupling level between the classes responsible for view and presentation. Besides, the support of the Android Architecture Components framework offered by Google's Android team for MVVM and the fact that this framework was seen as a stable framework by the participants were considered positively by the participants in terms of maintainability. In the case of Clean Architecture, most participants stated that this architecture successfully managed to separate concerns, thus reducing the complexity and coupling levels of Android codebases, therefore increase the maintainability of the Android applications. However, these participants also stated that this architecture increases the complexity of small and medium-sized projects and decreases the readability of the codebase thus negatively impact the maintainability.

It was observed that the participants made different comments about the effect of the third-party libraries used by the team on the maintainability of the applications. For example, the RxJava library was criticized in different ways. Participants stated that the steep learning curve of the RxJava library decreases the understandability of Android applications. Thus maintainability was negatively affected. In addition, the risk of RxJava becoming obsolete in the near future was also stated by some participants. It was mentioned that this situation had a negative effect on maintainability. On the other hand, some participants stated that this library has positive effects on maintainability due to its strong community, documentation and advanced features. Participants stated that Dagger 2 library has a positive impact on maintainability since it is a reliable and well-documented library supported by the Google Android team as a standard way of dependency injection. On the other hand, it was emphasized by the two participants that the complex structure of this library and its steep learning curve may cause problems in readability and understandability and that maintainability may be negatively affected. Regarding the Retrofit and Apollo libraries, all of the participants stated that these libraries have positive effects on maintainability. Reasons for that mentioned by the participants as strong community support and documentation, stability and reliability. In addition, the participants stated that the Retrofit library had become the industry standard, and it has a positive effect in terms of maintainability as it is easier to use than other alternatives. Architecture Components framework offered by Google's Android team has also found handy by the participants when it comes to the maintainability of Android applications. And as stated before, the reason for that is that the framework was seen as a stable framework by the participants and considered positively by the participants in terms of maintainability. However, one participant has stated that due to some internal issues that this framework has, the framework might badly affect maintainability.

Ultimately, participants also stated that improvements can be made in the currently used methods and technologies. It is seen that the participants mentioned that the documentation of the Android applications could be improved. Also, they also mentioned that libraries can be reconsidered based on the current trends and up to date technologies.

4.3 Evaluation with Object-Oriented Metrics

In this section, more information regarding the application of object-oriented metrics and the results obtained from this application are presented.

it would be good to already give the high-level structure of the CodeMR analysis: first you compare 2 codebases at project-level, then you give a more detailed comparison at certain feature level. then reader knows what to expects

4.3.1 Sample Codebases

In this section, detailed information about the codebases which the metrics mention in section 3.2 are shared. As mentioned in section 3.2, two codebases belonging to the same project have been selected to apply the metrics. To facilitate the explanation, these codebases will be mentioned in the form of ***cb-1*** and ***cb-2*** in the remainder of the study.

When cb-1 is examined in detail, a situation like the following is encountered. First of all, it the Android application's actively used version. The project was started to be developed using the Java programming language. Later, some features were developed using Kotlin programming language. There is no consistent choice of software architecture throughout the application. Although the application consists of 5 different modules, the modules' boundaries are not determined according to a certain standard. Some modules are feature-based, while some are layer-based. A similar situation is observed in packaging. The packaging organization of the application is inferior. While some features of the application have been developed with the MVP design pattern, some features have been developed with MVVM. It is controversial to what extent these design patterns are applied correctly. Static classes and singleton solutions have been used dangerously in practice. Besides, the SOLID principles have been ignored and no dependency injection is applied. It is also worth noting the use of some outdated libraries. Also, when looking at the Git history of the application, the commits of 11 different developers are seen. This situation is critical in describing the developer circulation in the application and explaining its serious organisation problems. The relationship between the developer circulation and the maintainability of software systems was previously mentioned in section 1.1.

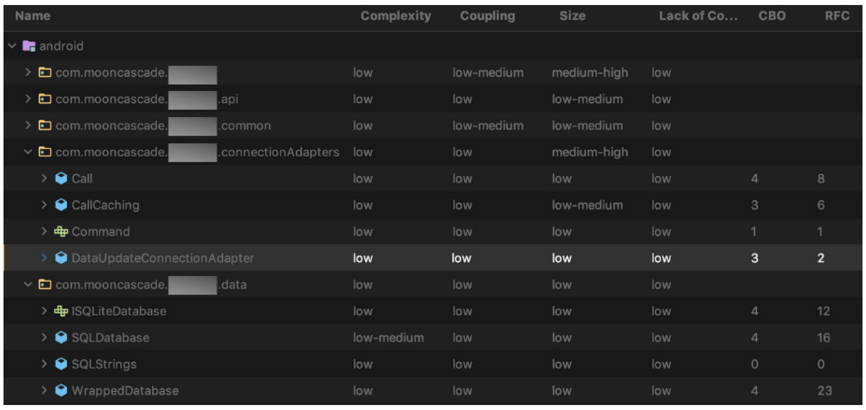
When cb-2 is examined, it is seen that this version of the application is being developed with the methods detailed in section 2.3. It has started to be developed using Kotlin programming language, and the entire application is in Kotlin programming language. Clean architecture and clean code, and SOLID principles are followed during the development. The application modules are separated based on the layers, and the packaging is feature-based. While developing the application, maintainable and reliable Android libraries have been used. The organisation level of the application's this version is very high, and it has been arranged to be a standard throughout the whole application. On the other hand, development for this version of the application's is still ongoing, and there are only 4 main features that have already been developed. The features currently developed for cb-2 are splash, login, register and main screen features. In this respect, cb-2 falls short in terms of fully developed features compared to cb-1.

~~As stated in the previous paragraph, the development of only four features of the cb-2 has been completed. Therefore,~~ the evaluation was only be made over the features that have been developed in both cb-1 and cb-2, namely, splash, login, register and homepage features. So, the other features of cb-1 were ignored when making the quantitative evaluation and then the comparison.

4.3.2 CodeMR

In this section, the information that would facilitate the interpretation of the charts and reports produced by CodeMR is shared.

The CodeMR tool uses different visualization methods in the reports generated by the application of metrics. The charts are created with the help of these different visualization techniques based on the metric values obtained from the analysis conducted by the tool. Within this study's scope, the visualization methods known as "Metric Distribution" and "Package Structure" in CodeMR literature were preferred to present the results of the evaluations. The main reason for choosing this methods is that it makes the visual understanding of the results easier than other methods and also these methods provide a better situational perceptibility for projects. These visualization techniques were used to visualize the data obtained as a result of applying the metrics specified in section 3.2 and the analysis presented by CodeMR on complexity, coupling and cohesion concepts. Other visualization methods and charts are very detailed, and sharing such detailed results is beyond this study's scope. The tool also presents the values for each metric over the provided codebase and presents these values. Values for each metric are presented at project, module, package and class scopes. However, in the evaluations within the scope of this study, only the results of the metrics previously determined and explained in section 3.2 were used. Fig 17 presents the metric values calculated with the help of the CodeMR Intellij IDEA plugin.



Name	Complexity	Coupling	Size	Lack of Co...	CBO	RFC
android						
com.mooncascade.[redacted]	low	low-medium	medium-high	low		
com.mooncascade.[redacted].api	low	low	low-medium	low		
com.mooncascade.[redacted].common	low	low-medium	low-medium	low		
com.mooncascade.[redacted].connectionAdapters	low	low	medium-high	low		
Call	low	low	low	low	4	8
CallCaching	low	low	low-medium	low	3	6
Command	low	low	low	low	1	1
DataUpdateConnectionAdapter	low	low	low	low	3	2
com.mooncascade.[redacted].data	low	low	low	low		
ISQLiteDatabase	low	low	low	low	4	12
SQLDatabase	low-medium	low	low	low	4	16
SQLStrings	low	low	low	low	0	0
WrappedDatabase	low	low	low	low	4	23

Figure 17. CodeMR Metric Value Presentation

There are two important points to be aware of when interpreting the charts created by these methods using CodeMR. The first of these points is legends used to indicate metric levels. As can be seen in Fig. 18, each colour corresponds to a metric level that is represented by a certain metric value threshold.

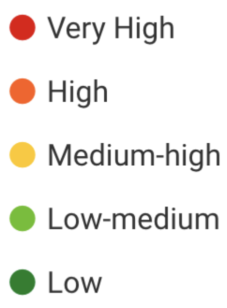


Figure 18. CodeMR Metric Level Indicators

The second important point is how the percentages on these charts are calculated while realizing the charts created by CodeMR. There might be different metric levels in different percentages in the charts. These percentages of metric levels for the selected metrics are illustrated in charts proportionally to the code size of classes in this level. Detailed information about the formulas used in calculating metric values and how the values are interpreted amd visualized can be accessed through the CodeMR documentation[12].

Presentation of the results obtained via CoreMR is done as explained below. The charts and the metrics values obtained from the evaluation have been shared for the project level. Although these techniques can be applied at the class and package level, it was preferred to present the project level results. It would not be very effective and useful to present these results for each class and package since there are many classes and packages for both codebases. On the other hand, at the class level, for making some evaluation and comparison, a few sample classes with high functionality were selected from both codebases, and they were evaluated and compared. Also, in accordance with the confidentiality requirements, the package and class names that will call the application name were hidden in the shared analysis results and figures. Then the results were presented via visualisation methods and numeric values. In the following sections, the results of this evaluation and the findings for comparisons are shared.

I think you should structure it as a comparison feature by feature / metric by metric, not looking at the two codebases fully speratafly.

4.3.3 CB-1 Results

When the numerical analysis results performed on the cb-1 via the CodeMR tool are examined, it is seen that the tool analyzes 2079 lines of code belonging to this codebase. These lines of code belong to 118 classes in 12 different packages of 4 different features

[12] https://www.codemr.co.uk/documents/

which were mentioned in section 4.3.1. When the metric values of the analysis result performed by the CodeMR tool of the cb-1 are reviewed, some problems draw attention, although the general situation is not extremely problematic. In the figure below, a CodeMR table that reflects the general situation of cb-1 is shared. In this table, a grouped overview of the complexity, coupling and cohesion levels determined through the metric values obtained from the analysis of the codebase is shown.

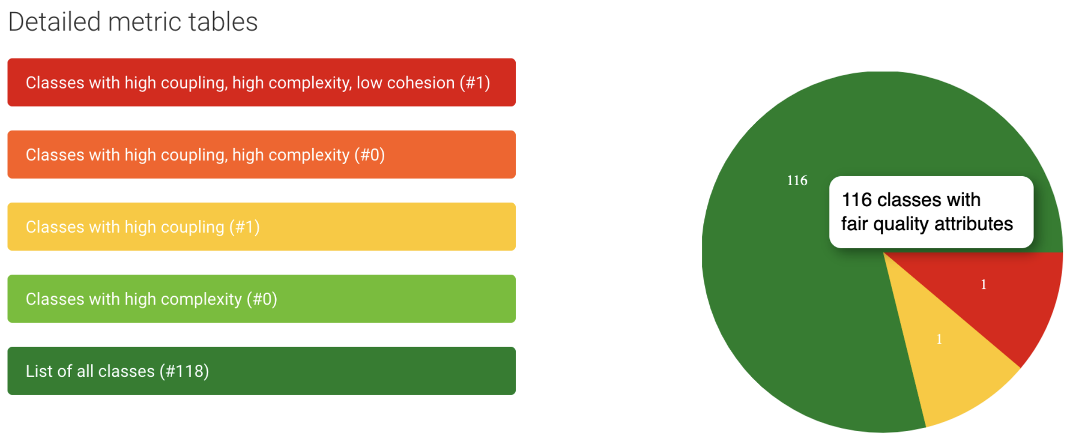

Detailed metric tables
Classes with high coupling, high complexity, low cohesion (#1)
Classes with high coupling, high complexity (#0)
Classes with high coupling (#1)
Classes with high complexity (#0)
List of all classes (#118)
116
116 classes with fair quality attributes
1
1

Figure 19. CodeMR Metrics Overview of CB-1

~~The detailed evaluation of the results on the basis of metric is as follows.~~ Among the analyzed classes, it is seen that two classes corresponding to 23% of the total code size have medium-high WMC values. It is seen that these classes, which are classified as A and B classes, have WMC values of 61 and 58, respectively. There are also two other classes corresponding to 14% code size have low-medium WMC values. Rest of the classes appear to have low level of WMC values. When the values of the DIT metric of the classes are examined, it is seen that 14 classes, corresponding to 22.7% of the total code size, have middle-upper level DIT values and the other 14 classes corresponding to 26.4% of the total code size have low-medium level DIT values. When the NOC metric values of the classes are examined, it is seen that 7 classes corresponding to only 3.4% of the whole codebase have low-medium NOC values and all other classes have low NOC values. According to the results, it is seen that a class, which corresponds to 12.5% of the code volume of the application, has a very high COB value. It also appears that a class that is corresponding to 11.2% of the application's code volume has a high COB value, and another class that is corresponding to 4.4% of the code volume has a medium-high COB value. When the values of LCOM, the last of the selected metrics, are examined on a class basis, it is seen that 6 classes corresponding to 40.8% of the total code size have high LCOM values. Besides, it is seen that two classes have middle-upper level LCOM values, which corresponds to approximately 4% of the total code size. In the figure below, the results of the class-based metric values, which is detailed and interpreted above, visualized by the "Metric Distribution" method offered by the CodeMR tool are presented. Charts are in doughnut form, and information regarding the methods of creating them has been shared in the previous sections. As previously mentioned, the size/percentage of each part in the doughnut is proportional to the size of the class/interface it represents.

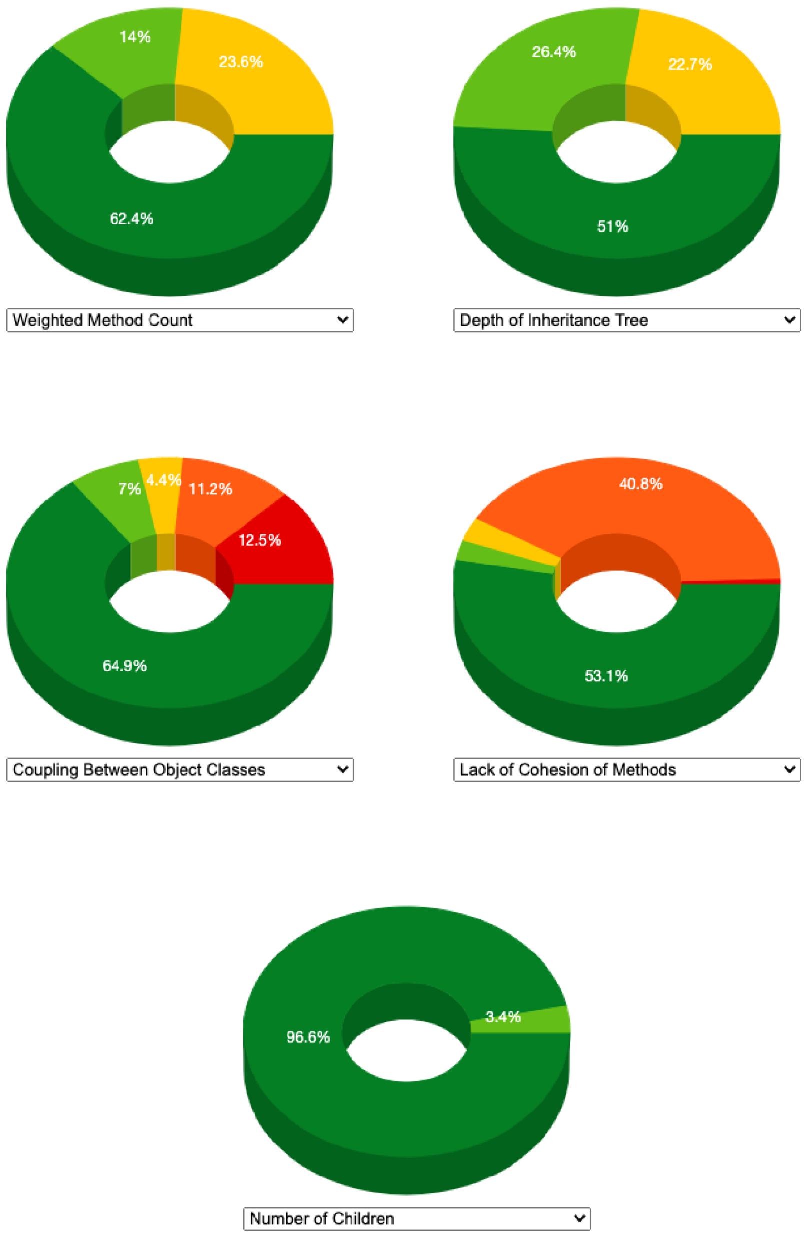


Figure 20. CodeMR Metric Distribution for CB-1

In the figure below, a more general view of cb-1 in terms of complexity, coupling and cohesion is presented. In Fig. 21, the largest circle represents the project, while the inner circles represent the packages and the small circles inside the inner circles represent the classes. The size of each circle is directly proportional to the size of the structure it represents.

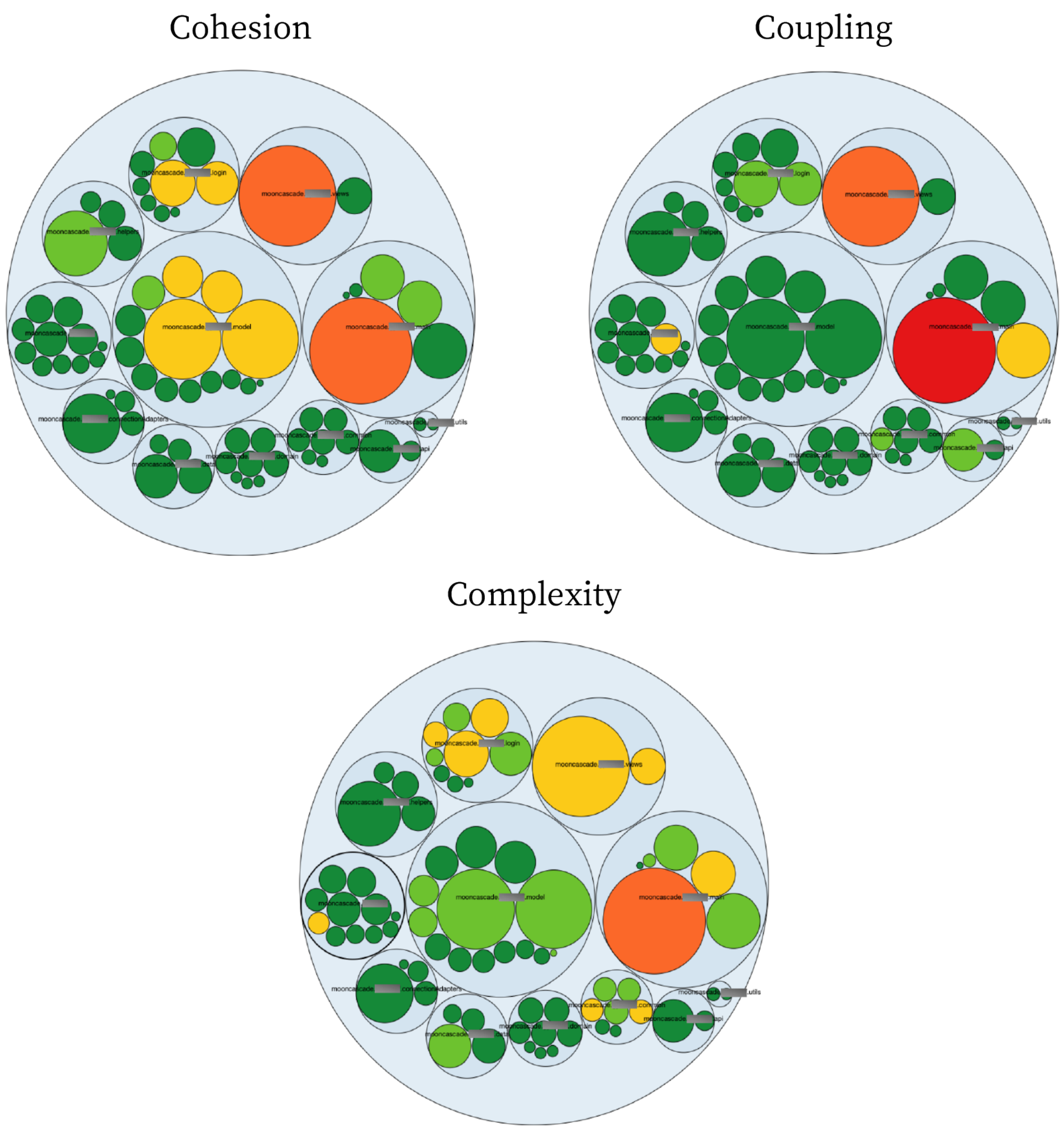
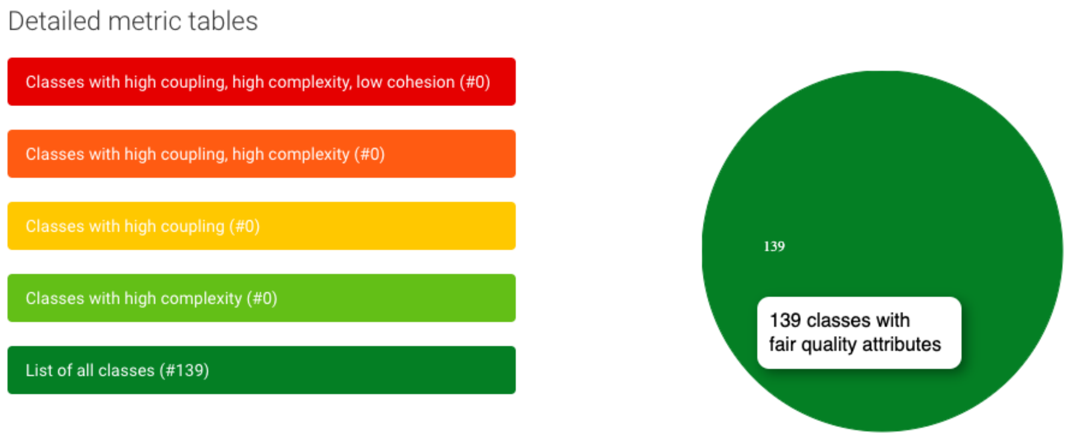


Figure 21. CodeMR Metric Values by Packages for CB-1

4.3.4 CB-2 Results

When the numerical analysis results produced on the cb-2 via the CodeMR tool are reviewed, it is seen that the tool analyzes 1160 lines of code belonging to this codebase. These lines of code belong to 139 classes in 59 different packages of 4 different features which were mentioned in section 4.3.1. When the CodeMR analysis results of this codebase are examined, a very positive situation is encountered. In the figure below, a CodeMR table that reflects the general situation of cb-2 is shared.


Detailed metric tables

Classes with high coupling, high complexity, low cohesion (#0)

Classes with high coupling, high complexity (#0)

Classes with high coupling (#0)

Classes with high complexity (#0)

List of all classes (#139)

Figure 22. CodeMR Metrics Overview of CB-2

Between the analyzed classes, it is seen that two classes corresponding to 9% of the total code size have low-medium WMC values, and the rest of the classes appear to have a low level of WMC values. It appears that the majority of the classes belong to cb-2 have low complexity levels. When the values of the DIT metric of the classes are examined, it is seen that 10 classes, corresponding to 28.9% of the total code size, have middle-upper level DIT values and the other 27 classes corresponding to 14.1% of the total code size have low-medium level DIT values. The rest of the classes have a low level of DIT values. When DIT values of these classes are examined, it is seen that some classes have a DIT value of 2 and classes with more than a DIT value of 2 values are quite low (DIT values between 1-3 considered as low-medium by CodeMR). When the NOC metric values of the classes are examined, it is seen that 7 classes corresponding to only 6.6% of the whole codebase have low-medium NOC values, and 1 very concise class has medium-high NOC value. The rest of the classes have low NOC values. According to the results, it is seen that ten classes are corresponding to 26.6% of the application's code volume with a low-medium COB value and the rest of the classes have a low COB value. According to the results of the CodeMR analysis, all classes within the cb-2 have low LCOM values. According to the results of the CodeMR analysis, all classes within the cb-2 have low LCOM values. In the figure below, the results of the class-based metric values, which is detailed and interpreted above, visualized by the "Metric Distribution" method offered by the CodeMR tool are presented.

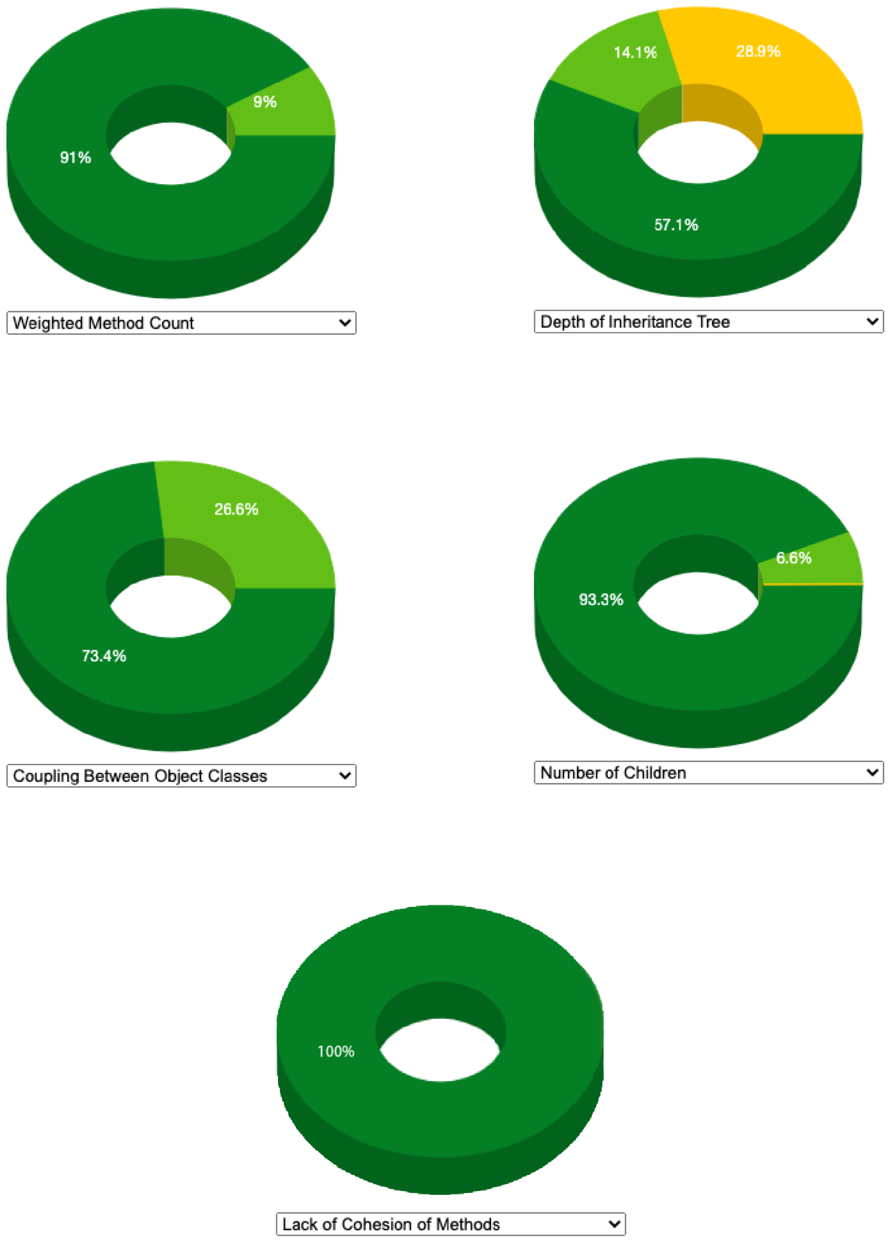


Figure 23. CodeMR Metric Distribution for CB-2

In the figure below, a general view of cb-2 in terms of complexity, coupling and cohesion is shown. As explained in the previous section, the largest circle represents the project, inner circles represent the packages and small circles inside the inner circles represent the classes. Sizes of the circles proportional to the size of the represented entity.

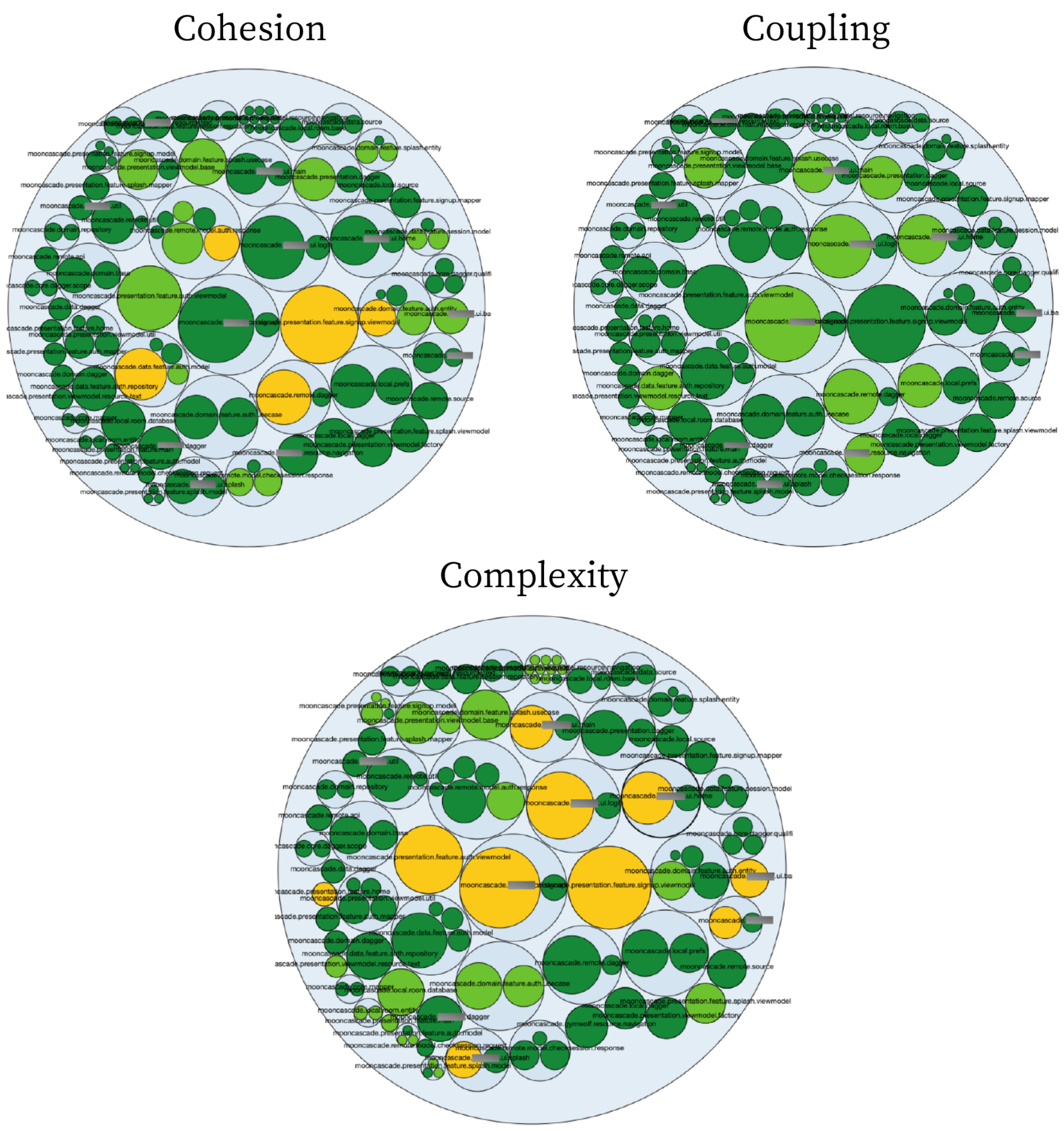
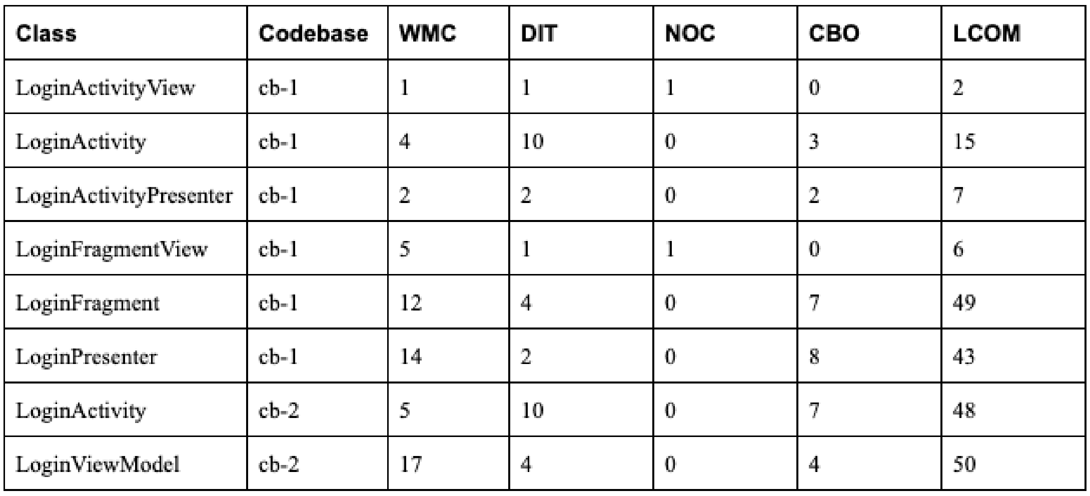


Figure 24. CodeMR Metric Distribution for CB-2

4.3.5 Feature-Based Comparison Results

In this section, comparison of the cb-1 and cb-2 codebases' results are shared. To make a more effective and accurate evaluation, the numerical evaluation results obtained from a few sample classes were collected, and the results were compared.

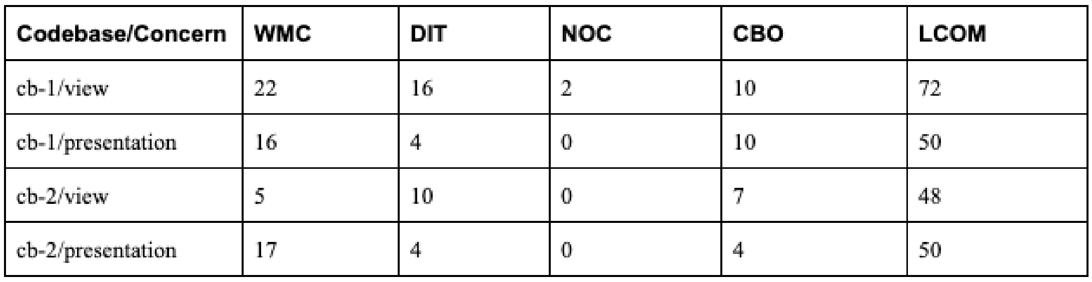
The login feature has been selected among the available features to compare the maintainability differences between the two codebases. The reason for choosing it is that it has a more complex logic (e.g. validation, different error types, etc.) than other existing features mentioned in the previous sections. Classes containing view and view logic parts for the login feature from both codebases have been selected for comparison. While making the evaluation, other lower-level dependencies were excluded. When the cb-1 using the MVP design pattern is examined, it is seen that 6 classes and interfaces are used related to the presentation and logic of data related to login. These classes and interfaces are: LoginActivityView (interface), LoginActivity (class), LoginActivityPresenter (class), LoginFragmentView (interface), LoginFragment (class), LoginFragmentPresenter (class). When the cb-2, which uses the MVVM design pattern, is examined, it is seen that only 2 classes are used and they are LoginActivity and LoginViewModel. The difference in the number of classes and interfaces used for the login feature in the codebases is due to the differences in the design patterns used and the fact that the cb-1 also uses the Fragment class of Android. In Fig. 25, the metric values of each class listed above, obtained from the quantitative evaluation results, can be seen in the form of a table. The metric values shared in the tables were obtained as a result of the analysis performed using the Android Studio CodeMR plugin.



Class	Codebase	WMC	DIT	NOC	CBO	LCOM
LoginActivityView	cb-1	1	1	1	0	2
LoginActivity	cb-1	4	10	0	3	15
LoginActivityPresenter	cb-1	2	2	0	2	7
LoginFragmentView	cb-1	5	1	1	0	6
LoginFragment	cb-1	12	4	0	7	49
LoginPresenter	cb-1	14	2	0	8	43
LoginActivity	cb-2	5	10	0	7	48
LoginViewModel	cb-2	17	4	0	4	50

Figure 25. CodeMR Metric Values for Login Feature

In order to examine the results better, groupings can be made between these classes, and the metric values of these groups can be compared. This grouping can be made based on the responsibilities of the classes. The responsibilities to be taken as a basis while groups are determined as view and view logic. More detailed information on these responsibilities was shared in previous sections. These classes and their interfaces can be grouped for each codebase as follows. For cb-1, LoginActivityView, LoginActivity, LoginFragmentView and LoginFragment are only responsible for the presentation of the data. LoginActivityPresenter and LoginFragmentPresenter are the classes responsible for how and when the data is presented. For cb-2, LoginActivity is responsible for presenting data, and LoginViewModel is responsible for how and when data is presented. This grouping will be useful for comparing the metric values shared in Fig. 17. In the table below, the metric values for each group of both codebases that are obtained from the analysis made within the scope of this study are shared.



Codebase/Concern	WMC	DIT	NOC	CBO	LCOM
cb-1/view	22	16	2	10	72
cb-1/presentation	16	4	0	10	50
cb-2/view	5	10	0	7	48
cb-2/presentation	17	4	0	4	50

Figure 26. CodeMR Metric Values for Login Feature

4.4 Summary

In this section, details of the qualitative and quantitative evaluation methods performed within the scope of this study and the results of these evaluations are shared. The qualitative methods are the Android developer questionnaire and interviews with the Mooncascade Android team. In contrast, the quantitative method consists of the measurements of five metrics applied with the CodeMR static code analysis tool. The interpretation of these evaluations, the results of which are shared in this section, will be carried out in the next section.

The style of writing is far too verbose. You repeat a lot of things at length which was presented in previous chapters. The discussion is meant to highlight the more interesting findings, connect the dots, think about what the findings imply, it is not about fully repeating the findings.

I think easily this content can be reduced by 50% without losing any important message.

5 Discussion

The interpretation of the data obtained as a result of answering the research questions within the scope of this study were presented in this section. Also, the effectiveness of the research method, the comments on the methods used by the case company that is mentioned in this study were shared. Besides limitations of the study is also discussed in this section.

5.1 RQ1 Discussion

First of all, the information obtained while answering the first research question guided the whole of this study. Studies conducted to answer this research question have shown that qualitative measurements are necessary as well as quantitative measurements. It has been seen that qualitative measurements can make important contributions to the results of the evaluation with the information obtained directly from the developers and support the quantitative measurement results. Therefore, qualitative assessment methods were determined and decided to be applied in the scope of this study. The fact that experienced Android developers make evaluations about the methods and technologies they use every day from the maintainability point of view and the results obtained from these evaluations can be added to this study increases the study's accuracy. From this point of view, it would not be wrong to say that the addition of qualitative methods as well as quantitative methods to studies focusing on the measurement of software development concepts such as maintainability would increase the qualification of the study.

The research conducted to answer the first research question has also shown that the maintainability of object-oriented software systems can be evaluated quantitatively by using many different metrics. In this study, an evaluation method based on the concepts of complexity, coupling and cohesion was chosen in order to measure the maintainability of Android applications and 5 metrics that can measure these concepts were determined. However, it cannot be said that these methods and metrics used in this study ~~to measure the maintainability of software systems~~ are the best solutions that can be used for this purpose. ~~It was explained in the previous sections why these methods and metrics are chosen.~~ Although these methods and metrics are sufficient for this study, it would not be wrong to say that there may be more effective quantitative measurement methods. Several different studies cover different maintainability evaluation methods for Android applications [22, 30]. However. the best quantitative measurement method for maintainability evaluation is controversial, and it would be appropriate to conduct comparative studies ~~using different metrics and methods~~ to determine the most effective solution. ~~Nevertheless, the used methods are considered sufficient for this study, and the existence of more effective methods is beyond the scope of this research~~.

5.2 RQ2 Discussion

First of all, when the results obtained from the Android developer survey are compared with the methods used by the Mooncasacade Android team, it is seen that the methods used largely overlap with the community choices. Although there are slight differences between the methods used by the team and the general Android developer choices, it can be said that these differences can be met normally. Although all the participants are Android developers, the users participating in the survey work in different companies and different domain, and it would not be wrong to say that the needs of these participants may vary according to the company and field they work in and that the methods and technologies they use are shaped according to their needs. On the other hand, the case of different user tendencies seen in the answers of the survey questions on the architectural pattern choice and the asynchronous event management libraries might be interpreted as the case company needs to re-evaluate the solutions used for these areas.Lastly, in the participants' responses to some questions, it is noteworthy that some participants choose more according to the industry trends, and this situation is observed especially in the answers given by relatively less experienced developers. As stated in the previous chapters, the Android platform is a rapidly developing platform. Therefore, it is worth noting that while choosing the methods and technologies used for developing Android applications, decisions should be made based on long-term stability and maintainability rather than industrial trends.

When the results of the interviews with the case company's Android team are examined, it is seen that the team consists of highly experienced Android developers, and they are aware of the importance of maintainability in software engineering and Android applications. Therefore, it would not be wrong to mention that the quality of the answers given by the team members to the questions in the interviews is high. First of all, five out of seven participants stated that maintainability is vital for the company and Android applications due to the way the company operates and the nature of Android applications. The other two participants stated that maintainability is already essential for software systems. These results support the theory claimed in the problem statement section of this study on maintainability that it is a critical non-functional requirement for software systems, Android applications and the case company. Besides, while the participants' evaluations about the methods and technologies used by the company in developing Android applications are generally positive, some constructive criticism also draws attention. The necessity of reconsidering the choice of architectural pattern and the asynchronous event management libraries, which was also mentioned in the Android developer survey interpretations above, was also emphasized by some case company members in the interviews. This situation stands out as a topic that the case company should focus on. The difficulties of using the RxJava library and its beginning to become out of date and the usage of Clean Architecture causing too much complexity for small and medium-sized projects are indicated as the reasons for this situation by the participants. In addition, the participants expressed their views that making the methods used into more standard coding conventions and arranging these conventions as if they were a familiar language among all members of the team would have a positive effect in terms of maintainability. Lastly, the views and awareness of most of the participants about choosing the libraries used in Android projects among stable and long-lasting libraries prove the effect of third party library use on the maintainability of Android applications.

Considering the quantitative evaluation results, some issues from CB-1 stand out. The first thing that catches the eye is a complex and unorganized packaging structure. Layer and feature-based packaging methods are internal in the project, which cause serious maintainability, understandability and organization problems. It is also noteworthy that some classes are large in size, which can be interpreted as lack of proper separation of concerns. The codebase appears to have complexity, coupling and cohesion problems as well. In a few classes, these problems are at a very high level, and maintainability and organization problems at different levels in the project draw attention. Especially the coupling problem stands out for this code base. Considering that there are no healthy abstraction and dependency injection application in the project, this result is not very surprising. When all these analysis results are taken into account, it is clear that the cb-1 codebase shows a low maintainability character. Nevertheless, cb-1's situation offered an opportunity for this study to evaluate maintainability. On the other hand, results of CB-2 has shown that there are visible improvements in complexity, coupling and cohesion. In addition, significantly reduced entity sizes and increased class and package numbers were noticed. Concise classes and the increase in the number of packages point to a more organized code base and a better separation of concerns, making the understandability of the codebase high. Besides, the levels of the classes in complexity, coupling are quite low, and cohesion is high. This situation can be shown as proof that the SOLID and SoC principles are applied correctly, and it can be said that there is a very positive effect on maintainability. However, a few classes with moderate complexity and cohesion issues stand out. When these classes were investigated, it was seen that they were the classes called "View Model" in the MVVM design pattern that are responsible for how and when the data will be displayed. According to the principles of the MVVM design pattern, each view should have only one view model. In this case, the view models belonging to the views with more than one responsibility also have the logic of belonging to more than one responsibility. Therefore, these classes become more complex, and the cohesion of the classes decreases due to the methods and dependencies they have for different responsibilities. As long as the principles of the MVVM design pattern are followed, it would not be appropriate to divide these responsibilities between different classes. Nevertheless, the effects of the technology and principles used in the development of cb-2 on maintainability are obvious.

When the feature-based comparison results between the two codebases are examined, it is seen that the cb-1 codebase uses 6 classes and interfaces for the visual part of the login feature, while cb-2 uses only 2 classes for this feature. From this point of view, the cb-2 code is better in terms of organization and understandability. Also, the metric values presented in Fig. 25 have shown that the results of cb-2 are better than the results of cb-1. In addition, the metric values presented in Fig. 26 has shown the difference between the view responsibilities of the two projects on complexity. When the values of WMC, DIT and NOC metrics used in complexity measurement are compared, it is seen that the complexity level of cb-2 for the view responsibility is lower. On the other hand, when the results of the complexity metric values of view logic related responsibilities are examined, it is observed that there is not much difference between the two code bases. As explained in previous sections, these results should be considered normal given the complexity of functionality of the classes involved in this responsibility. When the CBO metric related to measuring the coupling level is examined, it is seen that the cb-2 codebase gives better results. Since the SOLID and DI principles are used much more effectively in the cb-2 codebase, these results are not different from what is expected. Also, cb-1 uses the MVP design pattern. Coupling is increasing due to the bi-directional dependency between view and presentation layers in the MVP design pattern. It would not be wrong to say that this situation increases the coupling level of the cb-1. However, this situation is not the case for cb-2, which uses the MVVM design pattern. In the MVVM design pattern, only the view layer has a dependency to the presentation layer. When the results of the LCOM metric related to cohesion are examined, it is seen that the results are similar to the results of the complexity metrics. While the results of cb-2 are much better than cb-1 for the responsibility of view, there is not much difference between the results for the responsibility of presentation. As explained in the previous section, there is only one view model per view principle in the MVVM design pattern. Therefore, one view model might have different responsibilities, especially those that belong to the complex views. The same is true for the MVP design pattern as well. There can be only one presenter for each view. Naturally, cb-1, which uses the MVP design pattern, seems to have low cohesion values for presentation responsibility. Ultimately, results obtained from the qualitative and quantitative evaluations have shown that the effects of the methods and technologies used by the case company on the maintainability of Android applications are seen as positive, yet some points need improvement. The results also showed the importance of maintainability for the software systems, Android applications and particularly for the case company.

5.3 Limitations

Within the scope of this study, qualitative and quantitative methods to evaluate the methods and technologies used by Mooncascade's Android team were conducted. There were some limitations during the process these evaluations. First of all, it should be stated that the features used in quantitative evaluations are relatively less complex features of the Android application used for the evaluation. Therefore the efficiency of evaluation and comparison was affected by this situation. Using more complex feature could provide better insight into the impact of the methods and technologies used by the case company. Unfortunately, this was not possible within the scope of this study due to a lack of features and time to develop more complex features. However, it is true that using relatively less complicated features while making the assessment slightly reduces the results' effectiveness. Still, it does not mean that these results are false or unrealistic. Also, considering that there may be deficiencies in the methods used in the evaluations, the necessity to carry out more detailed studies on how to evaluate maintainability better and how to develop Android applications in terms of maintainability is obvious. For example, there are different methods to evaluate the software system's maintainability, and different metrics can be used. Therefore choosing the most efficient metrics to measure the maintainability of software systems and Android applications is controversial. Further research should be conducted to find the most efficient quantitative evaluation method. In addition, this study provides an overview rather than measuring the impact of each technology and method used by the case company to maintainability individually. The study does not provide detailed information on how much effect which matter has on maintainability. From this point of view, this can be seen as a limitation for this study. Lastly, although very important information was collected within the scope of qualitative evaluations, the number of participants is relatively low. More accurate and more reflective results can be obtained with interviews and surveys with a higher number of participants.

6 Conclusion

This study covers the subject of evaluating the impact of the methods and technologies used by the software product development company Mooncascade while developing Android applications on the maintainability of these applications. For this purpose, quantitative and qualitative evaluation methods were determined to measure the impact of the methods and technologies used by the Android team of the case company on maintainability and the evaluations were carried out using these methods.

To make a qualitative evaluation, a public Android survey was carried out public with random Android developers and interviews were conducted with each member of the Android team of the company. ~~Qualitative evaluation~~ results indicate that the methods and technologies used by the company while developing Android applications have a positive impact on the maintainability of these applications while also revealing that there are some shortcomings and areas open to improvement. Qualitative evaluations also led to findings that prove the importance of maintainability for software systems, Android applications and case company.

To make the quantitative evaluation, a set of object-oriented software metrics were used. These metrics were applied to common features of the different codebases belonging to the same project through the CodeMR static code analysis tool. While determining these metrics, metrics that can best evaluate the entire software system in terms of maintainability were preferred. For this purpose, priority was given to metrics related to concepts such as complexity, coupling and cohesion, whose relationship with maintainability has been proven. ~~After obtaining the results, interpretations and comparisons were made for each codebase. As a result of these evaluations,~~ it has been observed that the methods and technologies used by the case company while developing Android applications have a positive effect on maintainability. The comparison between the two codebases, one developed by using the case company's methods and technologies, the other developed without a specific order and standard, showed that even for the relatively simple application features, maintainability had been increased.

As a result, the importance of maintainability for the case company's Android applications and software systems has been emphasized. In addition, the primary goal of this study, which is to measure the impact of the methods and technologies used by the case company while developing Android applications on maintainability, was reached. Although the results allow having information about the subject at the intended level, it should not be forgotten that the limitations stated in the previous section affect the study results. From this point of view, it can be said that this study is like preliminary research that led to new studies that could overcome the limits stated in the previous section. Thus more effective and more efficient results could be obtained with the help of new studies where these limitations can be resolved.

References

[1] Taylor Kerns. *There are now more than 2.5 billion active Android devices*. 2019. URL: `https://www.androidpolice.com/2019/05/07/there-are-now-more-than-2-5-billion-active-android-device`. [Online; Accessed: 25.01.2020].

[2] Anthony I. Wasserman. "Software Engineering Issues for Mobile Application Development". In: *Conference: Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, at the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010* (2010).

[3] Ubaid Pisuwala. *How often should you update your mobile app?* URL: `https://www.peerbits.com/blog/update-mobile-app.html`. [Online; Accessed: 15.02.2020].

[4] Martin Fowler and Kent Beck. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professionaly, 1999.

[5] Yegor Bugayenko. *Elegant Objects*. Createspace Independent Publishing Platform, 2017.

[6] IEEE. "IEEE Standard Glossary of Software Engineering Terminology". In: *Institute of Electrical and Electronics Engineers, New York, 1990* (1990).

[7] Robert Cecil Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, Inc., 2009.

[8] Jussi Koskinen. "Software Maintenance Costs". In: *Information Technology Research Institute, ELTIS-Project University of Jyväskylä* (2010).

[9] Hadeel Alsolai and Marc Roper. "Application of Ensemble Techniques in Predicting Object-Oriented Software Maintainability". In: *EASE '19: Proceedings of the Evaluation and Assessment on Software Engineering* (2019).

[10] J. R. Mckee. "Maintenance as a Function of Design". In: *Proceedings AFIPS, National Computer Conference, Las Vegas* (1984).

[11] Capers Jones. "The Economics of Software Maintenance in the Twenty First Century". In: *Unpublished manuscript* (2006). URL: `http://citeseerx.%20ist.%20psu.%20edu/viewdoc/summary`.

[12] Intan Oktafiani and Bayu Hendradjaya. "Software Metrics Proposal for Conformity Checking of Class Diagram to SOLID Design Principles". In: *2018 5th International Conference on Data and Software Engineering (ICoDSE)* (2018).

[13] W. L. Hursch and Cristina Videira Lopes. "Separation of Concerns". In: *Technical report by the College of Computer Science, Northeastern University* (1995).

[14] P. Tarr et al. “N degrees of separation: multi-dimensional separation of concerns”. In: *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)* (1999).

[15] Robert Cecil Martin. *Clean Architecture: A Craftsman’s Guide to Software Structure and Design*. Pearson Education, Inc., 2018.

[16] Jr Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Professional, 1975.

[17] Martin Fowler. *Software Architecture Guide*. 2019. URL: `https://martinfowler.com/architecture/`. [Online; Accessed: 16.04.2020].

[18] Philippe Kruchten. *Philippe Kruchten, The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 2004.

[19] David Garlan. “Software Architecture: a Roadmap”. In: *ICSE ’00 2000* (2000).

[20] Roberto Verdecchia, Ivana Malavolta, and Patricia Lago. “Guidelines for Architecting Android Apps: A Mixed-Method Empirical Study”. In: *2019 IEEE International Conference on Software Architecture (ICSA)* (2019).

[21] Victor Oliveira, Leopoldo Teixeira, and Felipe Ebert. “On the Adoption of Kotlin on Android Development: A Triangulation Study”. In: *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2020).

[22] Lisa-Marie Andrä et al. “Maintainability Metrics for Android Applications in Kotlin: An Evaluation of Tools”. In: *ESSE 2020: Proceedings of the 2020 European Symposium on Software Engineering* (2020).

[23] Henning Grimeland Koller. “Effects of clean code on understandability: An experiment and analysis”. In: *Master’s thesis, University of Oslo* (2016).

[24] Tung Bui Duy. “Reactive Programming and Clean Architecture in Android Development”. In: *Bachelor thesis, Helsinki Metropolian University of Applied Sciences* (2017).

[25] Wei Sun, Haohui Chen, and Wen Yu. “The Exploration and Practice of MVVM Pattern on Android Platform”. In: *4th International Conference on Machinery, Materials and Information Technology Applications (ICMMITA 2016)* (2016).

[26] V. Garousi, M. Felderer, and M. V. Mäntylä. “The Need for Multivocal Literature Reviews in Software Engineering: Complementing Systematic Literature Reviews with Grey Literature”. In: *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, 26:1–26:6* (2016).

[27] R. Ogawa and B. Malen. “Towards Rigor in Reviews of Multivocal Literatures: Applying the Exploratory Case Study Method”. In: *Review of Educational Research 61(3):265-286* (1991).

[28] Hugo Kallström. “Increasing Maintainability for Android Applications”. In: *Unpublished master’s thesis, Umea University, Department of Computing Science* (2016).

[29] Ginanjar Prabowo, Hatma Suryotrisongko, and Aris Tjahyanto. “A Tale of Two Development Approach: Empirical Study on The Maintainability and Modularity of Android Mobile Application with Anti-Pattern and odel-View-Presenter Design Pattern”. In: *2018 International Conference on Electrical Engineering and Informatics (ICELTICs), Sept. 19-20, 2018* (2018).

[30] Ahmad A. Saifan and Areej Al-Rabadi. “Evaluating Maintainability of Android Applications”. In: *The 8th International Conference on Information Technology ICIT 2017At: Jordan Volume: 8* (2017).

[31] I. Heitlager, T. Kuipers, and J. Visser. “A practical model for measuring maintainability”. In: *In 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)* (2007).

[32] Rimmi Saini, Sanjay Kumar Dubey, and Ajay Rana. “Analytical Study of maintainability Models for Quality Evaluation”. In: *Indian Journal of Computer Science and Engineering (IJCSE)* (2011).

[33] A. Bakar et al. “Review on Maintainability Metrics in Open Source Software”. In: *International Review on Computers and Software* (2012).

[34] Shyam R. Chidamber and Chris F Kemerer. “Towards a Metrics Suite for Object Oriented Design”. In: *ACM SIGPLAN Notices* (1991).

[35] T. J. McCabe. “A complexity measure”. In: *IEEE Transactions on Software Engineering (Volume: SE-2, Issue: 4, Dec. 1976)* (1976).

[36] Jacinto Ramirez Lahti, Antti-Pekka Tuovinen, and Tommi Mikkonen. “Experiences on Managing Technical Debt with Code Smells and AntiPatterns”. In: *4th International Conference on Technical Debt (TechDebt 2021)* (2021).

[37] Jacinto Ramirez Lahti. “Reversing Entropy in a Software Development Project: Technical Debt and AntiPatterns”. In: *Master’s thesis, University of Helsinki, Faculty of Science* (2021).

Appendix

I. Licence

Non-exclusive licence to reproduce thesis and make thesis public