1. <https://www.androidpolice.com/2019/05/07/there-are-now-more-than-2-5-billion-active-android-devices/> [Online]. [Accessed: 25.01.2020].
2. Wasserman 2010
3. <https://www.peerbits.com/blog/update-mobile-app.html> [Online]. [Accessed: 15.02.2020].
4. Jussi Koskinen. Software maintenance costs. Jyvaskyl ¨ a: University of Jyv ¨ askyl ¨ a¨, 2010.
5. <https://source.android.com/> [Online]. [Accessed: 05.04.2020].
6. <http://www.openhandsetalliance.com/oha_members.html> [Online]. [Accessed: 05.04.2020].
7. <https://developer.android.com/guide/platform>[Online]. [Accessed: 12.04.2020].
8. <https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/> [Online]. [Accessed: 13.04.2020].
9. <https://developer.android.com/guide/components/fundamentals> [Online]. [Accessed: 14.04.2020].
10. <https://developer.android.com/guide/components/activities/activity-lifecycle> [Online]. [Accessed: 14.04.2020].

# Clean Architecture: A Craftsman's Guide to Software Structure and Design, First Edition 2018 Pearson Education, Inc.

1. <https://developer.android.com/reference/android/app/Activity#onRestart()> [Online] [Accessed: 14.04.2020].
2. <https://developer.android.com/guide/components/fragments> [Online] [Accessed: 16.04.2020].
3. <https://developer.android.com/reference/android/app/Fragment#Lifecycle> [Online] [Accessed: 16.04.2020].
4. <https://developer.android.com/guide/components/services> [Online] [Accessed: 16.04.2020].
5. Robert C Martin. Clean code: a handbook of agile software craftsmanship. Pearson Education, 2009.
6. Igor Bugayenko. Elegant Objects. Createspace Independent Publishing Platform
7. J. R. Mckee, “Maintenance as a Function of Design”. Proceedings AFIPS, National Computer Conference, Las Vegas, pp 187-93.
8. Guidelines for Architecting Android Apps: A Mixed-Method Empirical Study
9. F.P. Brooks. The mythical man-month: essays on software engineering. Addison-Wesley Pub. Co., 1975.
10. A Tale of Two Development Approach: Empirical Study on The Maintainability and Modularity of Android Mobile Application with Anti-Pattern and Model-View-Presenter Design Pattern
11. <https://martinfowler.com/architecture/> [Online] [Accessed: 16.04.2020].
12. Philippe Kruchten, *The Rational Unified Process: An Introduction*, Third Edition. Addison-Wesley Professional 2003.
13. An Empirical Study on the Impact of Android Code Smells on Resource Usage
14. KÄLLSTRÖM, Hugo. Increasing Maintainability for Android Applications: Implementation and Evaluation of Three Software Architectures on the Android Framework. 2016R.
15. A Tale of Two Development Approach: Empirical Study on The Maintainability and Modularity of Android Mobile Application with Anti-Pattern and Model-View-Presenter Design Pattern
16. Developers. (2019). Android architecture components. Retrieved March 17, 2019, from https://developer.android.com/topic/libraries/architecture.
17. [IEEE Standard Glossary of Software Engineering Terminology](http://standards.ieee.org/reading/ieee/std_public/description/se/610.12-1990_desc.html)
18. [Refactoring: Improving the Design of Existing Code](https://www.amazon.co.uk/Refactoring-Improving-Design-Existing-Technology/dp/0201485672) 1999 Martin fowler
19. C. Jones, "The economics of software maintenance in the twenty first century," Unpublished manuscript. http://citeseerx. ist. psu. edu/viewdoc/summary, 2006
20. Application of Ensemble Techniques in Predicting Object-Oriented Software Maintainability Hadeel Alsolai
21. Foundations for the Study of Software Architecture
22. Software Architecture: a Roadmap David Garlan
23. Software Metrics Proposal for Conformity Checking of Class Diagram to SOLID Design Principles
24. Separation of Concerns Walter L. Hursch and Cristina Videira Lopes
25. Degrees of Separation: Multi-Dimensional Separation of Concerns
26. <https://developer.android.com/kotlin> [Online] [Accessed: 18.03.2021].
27. <https://developer.android.com/reference/android/os/AsyncTask> [Online] [Accessed: 18.03.2021].
28. <https://developer.android.com/topic/libraries/architecture/coroutines> [Online] [Accessed: 18.03.2021].
29. <https://developer.android.com/training/dependency-injection/hilt-android> [Online] [Accessed: 18.03.2021].
30. Review on ‘Maintainability’ Metrics in Open Source software Abubakar Diwani Bakar1
31. Evaluating Maintainability of Android Applications (saifan)
32. T. J. McCabe, “A complexity measure,” IEEE Transactions on software Engineering, no. 4, pp. 308–320, 1976.
33. C. Chidamber, Towards a Metrics Suit for Object Oriented Design, ACM, OOPSL’91, pp. 197-211, 1991.
34. <https://www.codemr.co.uk> [Online] [Accessed: 21.03.2021].
35. Experiences on Managing Technical Debt with Code Smells and AntiPatterns ramirez
36. Reversing Entropy in a Software Development Project: Technical Debt and Anti Patterns

When the sources from the industry are examined, different sources answer this question in different ways as there is no certain way of answering such a question. In addition, when academic resources are examined, the inadequacy and outdatedness of these resources immediately become apparent.

The main purpose of these architectures is to overcome the issues and challenges we have mentioned above. Bob Martin has described a set of rules and principles in his book “Clean Architecture” to improve the separation of concerns and increase the maintainability of the software systems (11). Clean Architecture, based on SOLID principles, is a high-level guideline for creating software systems with a layered architecture. The main purpose of the Clean Architecture is to make a software system more understandable and maintainable. The same set of rules can be applied to Android application development to resolve the previously mentioned issues.

All of these methods focus on developing high-maintainability and high-quality Android applications, thus aiming to overcome the difficulties mentioned above.

It is also studied why Clean Architecture is the solution and what impact it has when solving the maintainability problems in Android application development, what are the pros and cons of applying Clean Architecture to the development of an Android application. In addition to the theoretical information regarding the application of Clean Architecture principles to Android, this study shares the best practices from the industry with elaborate examples.

In addition, insights from the most popular Android libraries and how these libraries can be adapted to the Clean Architecture in Android are given.

This study aims to provide detailed information regarding the development of Android applications with Clean Architecture to overcome the maintainability and complexity issues for large and enterprise Android applications.

Therefore it is a must to apply some kind of software engineering processes and techniques in order to build secure and high-quality mobile applications. In that sense, many classic software engineering techniques fit quite well with the mobile application development domain(2)

Over the last decade, a couple of different ideas were in point to resolve these issues in the context of Android application development. Some of the remarkable ideas amongst these ideas can be counted as Model-View-Controller, Model-View-Presenter, Model-View-View-Model, and VIPER. These are the well-known design patterns in the industry for Android application development and they are widely used(19).

Even if these methods are adopted to Android application development processes, they might be insufficient when it comes to real-world Android application development. Although many of these methods work well for separating the business and presentation logic from the view, they are not enough when an Android app gets bigger and the codebase becomes more complex. When the codebase becomes huge, the presentation related classes become bloated, and applying separation of concerns becomes very hard. Therefore, all the possible challenges mentioned above would be very hard to overcome. Consequently, these solutions are not sufficient to overcome the difficulties mentioned above, especially when it comes to the development of large and enterprise Android applications. As this information shows, it is clear that in the process of developing complex enterprise Android applications, a more advanced solution is needed to resolve the mentioned concerns. In order to survive the competitive Android market, develop a high-quality application with high maintainability, increase team efficiency, and faster delivery, modern software development companies must have a set of rules and software development techniques to scale their Android application development processes.

The high demand for Android applications and the high frequency of updates are other challenges when developing Android applications. These challenges make the development of high-quality and maintainable applications essential for Android application developers(19). Because users demand error-free, high-performance, easy-to-use, and low-energy applications. Developers, on the other hand, aim to develop maintainable, expandable, scalable and easily testable applications in the shortest time due to the rapidly changing and evolving user requests, and to be able to update the application as soon as possible to future user requests(21). Also, it gets harder to maintain the codebase as the codebase and the development team grows or changes. Any time a new developer joins the team, the time required to onboard the new developer to the codebase is directly related to the level of readability and maintainability of the codebase. Therefore, developing high-quality Android applications to meet all these expectations, overcome the mentioned challenges, and delivering the application rapidly is essential. Thus, processes such as updating Android applications, adding new features, fixing errors will be more time and cost-efficient in terms of software engineering principles and software quality standards.

Moreover, well-known design patterns and architectural approaches to overcome the maintainability issues in Android application development will be introduced with their pros and cons. Lastly, the reason for the need for a higher level architectural approach when developing complex, enterprise Android applications will be explained in the summary of the section. In this section, previous studies similar to this study, and the results of these studies will be covered as well.

For that reason, maintainability emerges as one of the most crucial quality requirements and maybe even the most important one in software development processes and also particularly in the field of Android application development(19).

, maintainability became one of the most important aspects that should be taken into consideration when developing complex software products with teams.

In the previous section, it was clearly stated the importance of maintainability and consequently the significance of architectural patterns. Moreover, it was indicated why the maintainability and architectural pattern selection is even more important for Android application development.

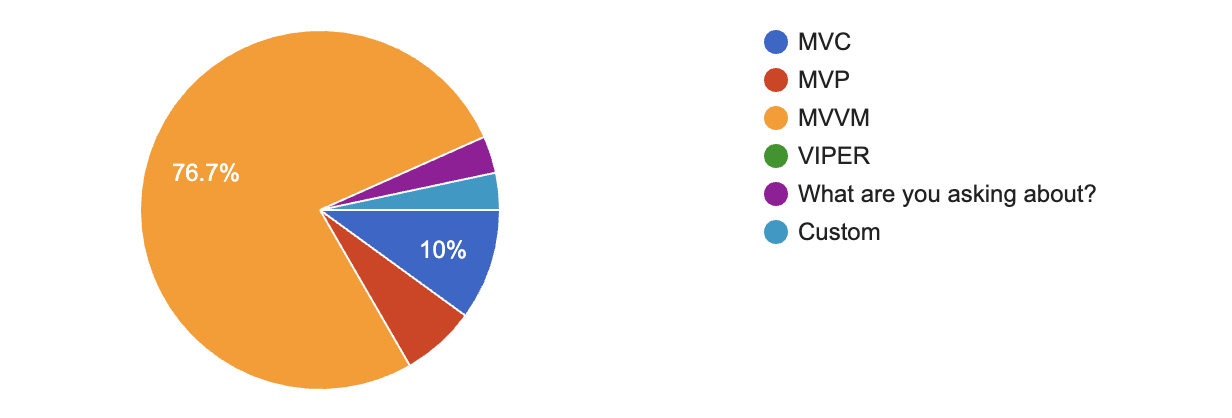
***“Of course bad code can be cleaned up. But it’s very expensive.” - Robert C. Martin (16)***

The reviewed literature and the information gathered from industry also shows that the debate on choosing the right architectural pattern for an Android application remains quite controversial. Moreover, as a part of this study, a survey conducted amongst the professional Android developers shows that tendencies of the Android community are mainly focused on three main presentational patterns. The main purpose of this Android developer survey was to gather the most up to date data regarding the tendencies of the Android community as the tendencies and technological hypes change quite fast in the Android environment. Since the study topic is tightly bonded to the industry, it was also important to gather information from the Android developers who work in the industry, in addition to the academic literature review, in order to assemble the most up to date and most reliable information, regarding the study topic and the existing solutions to the problem that the study is trying to solve. The interpretation of the literature review and the developer survey is below.

A systematic review through the related academic resources shows that MVP, MVVM, and Clean Architecture are the most attributed design patterns and architectural patterns. MVP and MVVM are 2 different derivatives of the MV+X model. They are widely used in GUI-heavy software applications. Although ranked second in the results, MVVM is the recommended architectural pattern by Google as Google’s Android team released Architecture Components as a part of Android Jetpack. Google encourages Android developers to use Architecture Components and MVVM as their architectural pattern and it provides lots of guides for this aim (27). The Clean architectural pattern shows up to be also often considered by developers with regards to architecting Android applications. The remaining architectural patterns can be ignored as they are quite outdated. Another study that conducted a survey between Android professionals and a systematic review made amongst academic literature makes these architectural patterns known. A list of these patterns is shown in the figure below(19).

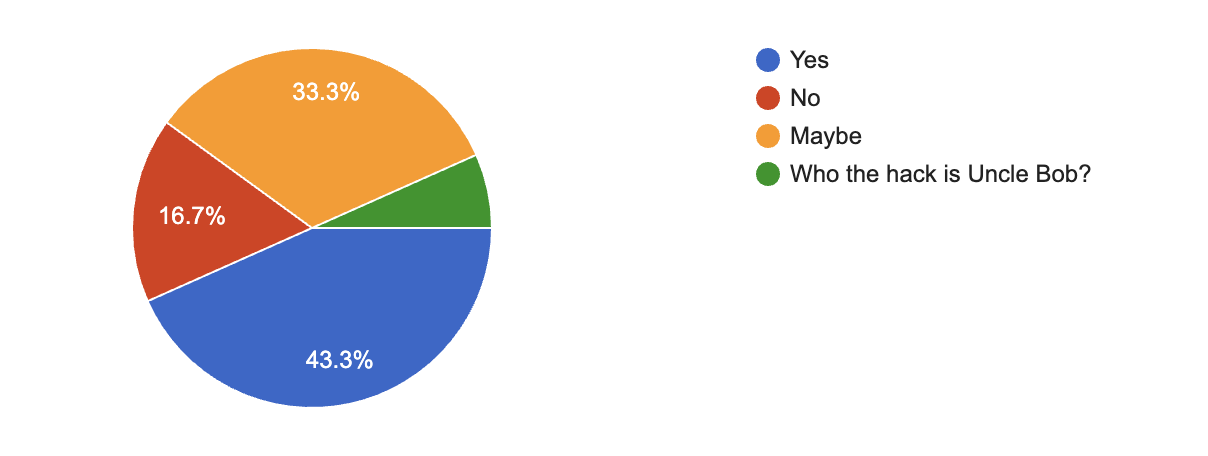
There are well-known design patterns amongst the Android community that targets to solve maintainability issues of Android. These well-known design patterns were already mentioned in the background section. As a reminder from the previous section, a couple of remarkable ones amongst these design patterns can be listed as Model-View-Controller(MVC), Model-View-ViewModel(MVVM), Model-View-Presenter(MVP).

When it comes to interpreting the results of the Android developer survey, it is seen that the outputs are mostly the same but with slight differences. According to the result of this Android developer survey conducted amongst more than a hundred Android developers, MVVM is the presentational design pattern that dominates the tendencies of Android developers. As an answer to the question "What presentational design pattern do you apply to your Android apps?", 76.7 percent of all participants responded that MVVM is their presentational design pattern of choice. MVVM is followed by MVC(10%) and MVP(2%). It seems that the recently introduced library called "Android Jetpack" which was designed by Google to help Android developers structure Android applications based on the MVVM design pattern has had a huge impact on the Android community lately. Consequently, MVVM is the top choice of Android developers according to the Android developer survey. And this outcome is different from the results identified by the academic literature review, where MVP was the most attributed one. This situation might be interpreted in such a way that the academic literature has been following the changes in the Android environment a bit slower.



**Figure 5.** Design pattern choice by Android developers according to the developer survey conducted by the study.

As a part of the Android developer survey, the question "Do you apply Uncle Bob's CLEAN Architecture to your Android applications?" was also asked. The answers show that only 43% of all participants apply Clean Architecture and its principles when developing Android applications. 33.3% of the participants responded that they might consider applying and 16.7% of the participants say that they do not apply Clean Architecture and its principles when developing Android applications.



**Figure 6.** Usage of Clean Architecture by Android developers according to the developer survey conducted by the study.

**Problem Statement -> concise version goes to intro, details go to background**

**Add questions about effectiveness of the tools but specific questions**

**How did I come up with these questions? What methods did you use? Defansta bu sorular gelebilir**

**I need some questions to comply with the other questionnaire**

**E.g. Why do you use MVVM? Do you agree that MVVM is the best one? Why? Why not?**

**Initial analysis of the survey**

**Explain the survey**

**Start writing about the results -> FIRST TASK**

**Maintainability**

***"Good programmers write code that humans can understand." - Martin Fowler***

In ancient times, when computers were big, heavy, and slow, programmers were limited to use low-level programming languages that are working close to computer CPUs. These were imperative programming languages, and the programs written in these programming languages were following the procedural programming paradigm. Although that approach works fine, the biggest problem was that these programming languages were designed to be understood by computers, not humans. The main reason for this situation was that, back in that time, computers lacked proper hardware, resources, and speed. Consequently, the priorities back then were different. The computer programs had to be fast and less memory consuming. However, this situation has changed in the current day. Even a low-quality mobile device is much stronger and smarter than the computers that we were using a couple of decades ago, and software systems became complex. Although this change brought a positive impact on the end-user side as it also brought more functionality and ease, the impact it brought the software development side is complexity. Especially when developing large-enterprise software products, ignoring that fact and not considering how to overcome this complexity may cause significant failures. In this context, new programming languages and paradigms were born, the priorities have altered, and this new reality brought different challenges and new quality standards to software development(17).

When the priorities of modern software development are analyzed today, the maintainability of software systems emerges as one of the most critical priorities, perhaps even the most important. 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(28). In the context of software engineering, maintainability means how well a software system is understandable, repairable, and extendable. Maintenance is one of the most important parts of the software development life cycle because the time spent on maintaining software systems requires more time and resources than the rest of the process. The relative expense for maintaining software and dealing with its development speaks to over 90% of its absolute expense (4). The level of maintainability that a software system depends on several different factors. Overall, a software system can be considered as maintainable if it is simple to grasp how it works and what it does, and making changes such as adding new features and fixing bugs is easy. 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(31).

Considering the life cycle of software systems might help to understand the importance of maintainability. Software systems are born, they live, they change and eventually they die. However, their lifetime is generally long, and 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. In other words, developers might need to make a change to the software system, which they worked in weeks or months before. In such cases, 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. Also, when a new developer joins the development team, onboarding should be smooth, and the new developer should be able to understand the purpose of the software system easily. Ignoring all these might cause companies a significant amount of time and money. That is what makes maintainability that important. Developing maintainable software systems is the way to tackle such issues.

The importance of maintainability for software systems can also easily be seen when looking at its role in the software development lifecycle and its effect on software development costs. The time has to be spent on the maintenance of complex software products is comparatively more extended 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(18). 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(30). 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 prominent 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 promising product from the market and went out of the business. Lousy code and consequently, maintainability was the reason for this company to go out of the business(16). This real-life example clearly shows how vital maintainability is to software systems and what fatal consequences it can cause if ignored.

The importance of the maintainability for software systems is evident and this situation is no different for Android Applications. In fact, the importance of maintainability for Android applications is even higher since Android applications have a very active software development life cycle. Given that the growing user demands and business needs making the Android applications more and more complex and Android applications having frequent update rates, it is not hard to see how important the maintainability is for Android application development. This situation is one of the main sources of motivation for this study.

In addition to that, in the context of Android, when the main challenges mentioned in this study are evaluated together, the importance of maintainability as a non-functional requirement becomes even more evident for Android application development because the high level of maintainability is the way to overcome the challenges and complexities mentioned in this study while developing Android applications. In consequence, the question is how to achieve the goal of developing Android applications with high maintainability. From the software development point of view, the Android platform does not have strict rules on how the applications are developed. Developing maintainable applications is not an obligation. However, developing Android applications with high maintainability is a need to solve the difficulties mentioned in this study in a timely and cost-efficient manner, facilitate the development processes for the Android developers, and increase the quality of the Android applications. The methods to be followed and the technologies to be used in Android applications for meeting these requirements constantly evolve, and the topic is still controversial among the Android community. Different solutions have been proposed and tried since the birth of the platform. However, the unchanged reality is that the most important criteria for building reliable software systems in a timely and cost-efficient manner is maintainability and of course, this reality is not different for Android applications.