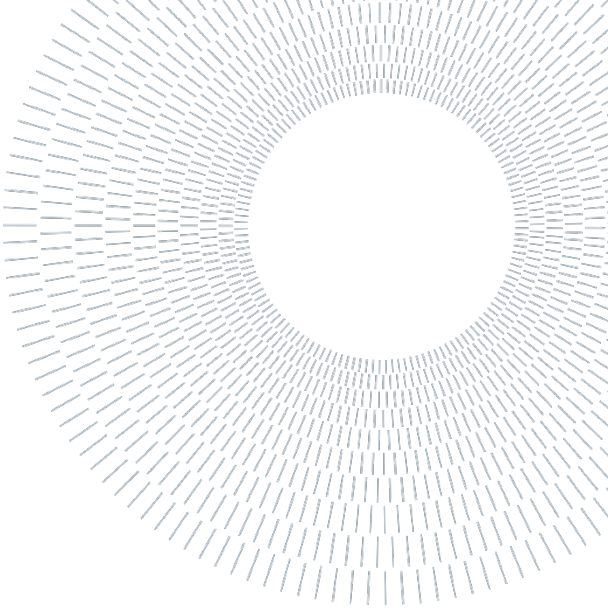
POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE



EXECUTIVE SUMMARY OF THE THESIS

Analysis of the Usage of Specific Technologies in Android Development

TESI DI LAUREA MAGISTRALE IN

COMPUTER SCIENCE AND ENGINEERING -

INGEGNERIA INFORMATICA

AUTHOR: ROBERT MEDVEDEC

ADVISOR: LUCIANO BARESI

ACADEMIC YEAR: 2021-2022

1. Introduction

Android devices make up more than two-thirds of the smartphone market as of 2022. [1]
With the vast majority of the human population in developed countries owning such a device, applications that run on smartphones are not only shaping the way people use their phones but also shape how everyone leads their lives.

Development of Android applications ever since Android OS inception in 2008 has been rapidly evolving and quite often rashly changing due to quick technological advancements in both mobile and computer hardware capabilities. Defining the
the current state of Android development and pinpointing the most used languages, technologies, IDEs, architectural patterns, and other elements have never been easy tasks due to such rapid changes.
In recent years Android OS creators, a consortium led by Google, managed to slow down the evolution of the development by sticking to a certain approach and technologies, however good or bad they might be in a general sense, create some sort of stability in the Android app development world. The thrust from other developers to make development for other systems, most notably iOS, as

closely as possible connected to the Android development goes hand in hand with
Google's intention and is further stabilizing the technologies used. All of this is to
allow developers to create better and more innovative apps with their focus shifted to execution rather than catching up with recent technologies.

The goal of this work is to define the current state of the most used technologies in Android application development and to provide the reader with enough background to understand them. The addition to main goal is to determine which technologies are used the most often but also to determine which combination of them achieves the best results when it comes to performance, usability, and code readability. Based on these results a snapshot of the current state in Android development will be created along with an approximation of what the future of Android applications is going to look like.

The research is done on the most downloaded and rated open-source applications that still have active repositories and recent releases. Many of these applications are used by big companies and often function as companion
apps for selling their main product.

2. Technological background

The first version of Android came out in 2008. It has always been characterized as a widely available open-source operating system for mobile devices. Being based on Linux kernel the idea of being available for such a big number of devices was existent from the start. The initial versions were very buggy and slow, didn't work in the same way on different devices, and didn't offer much support for application development. In 14 years on the market and 12 versions later the situation changed, and the current Android 13 is superior in many aspects to its direct competitor iOS, while supporting exponentially higher number of devices, not only from the mobile world, and having a great support both from the development community and Google itself. Applications are natively written in Java or Kotlin and can use a number of different technologies depending on the intended use and functionalities.

3. Analysis

Twenty-seven open-source applications have been used for the analysis, with their respective repositories being located on GitHub. Five main application categories have been defined, which include some apps that have multiple million downloads on Google Play. The analyzed categories are:

- Browsers (Brave, DuckDuckGo, Fenix,Orbot)
- Commercial applications (Bitwarden, Kickstarter, Shadowsocks, Wikipedia, Wordpress)
- Media players (Antenna, NewPipe, Phonograph, Shuttle, Timber)
- Messaging and email (K9, QKSMS, Signal, Telegram, Wire)
- Other (Google I/O, Habitica, Materialistic, Muzei, Omni Notes)

- Tech demo (Kotlin Pokedex, NotyKT, Pokedex)

The analysis was done mostly on the static segments of the application. Individual analysis consisted of recognizing the use and counting of the following aspects:

- Application size
- Selected architecture
- Programming languages
- Used design patterns
- Presentation technology
- Google and external services use
- Number of dependencies
- Android application components used

The individual results were then compared to the other applications of the same ground and similar characteristics. The analysis results are presented in the following chapters by categories.

3.1 Application complexity and size

When looking at the final numbers, several different conclusions can be drawn. In the analysis of twenty-seven apps, only four of them are bigger than 100MB when installed, with the average size after installation being 58MB.

The biggest of the four are two browser apps, Fenix and Brave. This is to be expected since browsers contain a full stack of code and have extensive features that include a lot of different libraries. The other two big apps are Signal the messaging app and Kickstarter mobile version. Signal, similar to browser apps, contains several layers of full-stack architecture and implements many security features in the messaging system. Kickstarter as an app is very exhaustive has a large number of different screens and offers many unique features to the users.

Table 3.1: Application size table

	Small (< 50MB)	Medium (50-100 MB)	Large (>100MB)
App count	17	6	4
Avg. app install size	24MB	75MB	179MB
Code size (in MB)	7MB	32MB	36MB
Kotlin apps size	21MB	77MB	179MB
Java apps size	26MB	81MB	179MB
Avg. num. of dependencies	30	41	70
Avg. num of screens	10	27	21
Avg. num of activities	13	47	40
Avg. num. of fragments	18	48	70

The final conclusion from all of the data is that the application install size mainly increases with the high number of dependencies, with the number of Activities, Fragments, screens, programming language, and code size being much less of a factor.
Despite a limited number of Compose applications, it can be determined that they are on average bigger than Views (XML) apps, mainly due to increased number of dependencies needed to support Compose UI.

3.2 Programming language

Thirteen out of twenty-seven analyzed apps use Kotlin as their primary language, with another four currently in the transition phase where most of the code is still written in other languages, mainly Java. Only one application doesn't use these languages with C# being represented once as a primary language.

Table 3.2: Programming languages used table

	Java	Kotlin	C/C++	Other
Primary language	13	13	0	1 (C#)
Secondary language	5	2	4	2 (Scala, Python)

Three of the apps made a full transition from Java in the previous years and use minimal to no Java code, with another three still using some Java code. Mozilla Fenix (Firefox) is the only large app that has made a full transition to Kotlin. This can be attributed to the large team that Mozilla has as well as the wide popularity among the developer community which helped out with the coding during the process.

Java is still a number one programming language for Android, but Kotlin is rapidly taking over. All of the apps newer than 2017 are written in Kotlin and many older ones are being translated to Kotlin from Java.

3.3 User interface (UI)

Considering that Jetpack Compose is a relatively new technology that completely changes the way the UI works it is to be expected that it hasn't completely caught on yet. Unlike making the transition from Java to Kotlin, the transition from Views to Compose is much harder to be done gradually. Views and Compose do not mesh very well together, even though it is possible, but the whole idea behind Compose and the way it works requires completely different architecture.

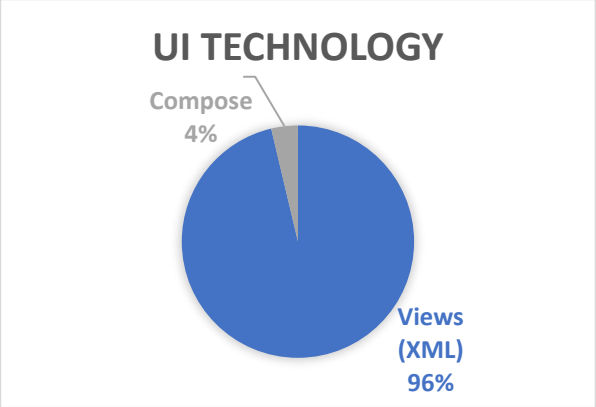


Figure 3.1: UI technology usage distribution

Out of all the analyzed apps, only two of them are using Jetpack Compose. One of those is an additional app having a Compose version next to the Views one NotyKT, and the other one already mentioned, refactored Mozilla Fenix (Firefox). Despite Compose being on the market for a few years now and having a stable version for more than a year, no applications seem to catch on.
It is even less likely that the other non-open-source apps have transitioned to it as it would take a lot of working hours for the whole operation, without any direct benefits for the user.

3.4 Google libraries and services

Many applications use a large number of native and non-native services developed by Google in order to improve the app dependability and shorten the development time. A number of developers seems to be trusting Google almost completely with their application, but data shows that there are still some who prefer other services and libraries that provide more freedom and complexity.

Table 3.3: Libraries and DI usage table

Library	Number of apps using it
Room (Google)	13
ProGuard/R8	25
Dagger	5
Dagger – Hilt (Google)	6
Koin	3

One of the main Google services is Firebase, which offers a range of different components to help users with many aspects of app development. These components are less complex than their non-Google counterparts but are still used quite extensively.

Table 3.4: Firebase services usage table

Firebase service	Number of apps using it
Cloud Messaging	8
Analytics	6
Crashlytics	6
Remote Config	4
Any	15

3.5 Architectural and design patterns

The most popular architecture among the apps is the MVVM. This is in no way connected with Compose, which is almost exclusively used with MVVM, as almost no apps use it. MVVM has gradually taken over the market and is most often found as a recommended architecture in guides and tutorials. There are no clean architecture usages, which could be attributed to the fact that it rarely works well with medium and large apps due to bad scalability.

Transition from MVC through MVP to MVVM is also visible from the numbers as only two applications are using MVC, with seven using MVP, and thirteen using MVVM. Seven applications are still in the transition phase or are just using more than one pattern to better suit the needs.

Table 3.5: Architecture patterns usage table

Architecture patterns	Number of apps using it
MVC	2
MVP	7
MVVM	13
Hyrbid	5

Developers seem to be following modern trends as MVVM is the most modern architectural pattern and is the most adapted to the newest methods in programming and libraries such as Jetpack Compose.

Almost all of the most popular design patterns have been featured in majority of

the apps. The developers are clearly using the advantages they bring in faster development and dependably code. The most common design patterns are Singletons, Adapters, Builders, Dependency injection, State, and Iterator.

4. Conclusions

The ultimate goal of the work was to find out which technologies are being used the most from the open-source applications. The conclusion can be made that while Google is pushing hard for all of the new technologies, like Kotlin, Jetpack, and Firebase to be implemented into all of the applications, some of the technologies still haven't made the breakthrough. Kotlin as a programming language and MVVM as an architecture are slowly taking over the spot as the most common technologies mainly to the vast improvements they have brought when compared to their predecessor. Google native services are also being used quite often and most of the design patterns can be found in all of the applications. However, the developers still have not caught on to the Jetpack Compose train and are sticking to the older View with XML approach with no real indication that this might change in the near future.

It appears that developers are more than glad to learn new approaches and adapt to new technologies if they offer substantial advantages to their development, which could ultimately bring only the highest quality updates to the development and allow for making of the highest quality applications.

Based on Kotlin and Android Jetpack usage with regards of their first appearance in the Android world several years back, it appears that most of the developers are likely to make the switch to Jetpack Compose somewhere in the next few years, which would be three to five years after the first stable release in 2022. Google's hard push for many of its services paid off and it would be a surprise if they didn't repeat this success with Compose, providing developers with a very quality programming experience that will lead to better and quicker application development.

5. Bibliography

[1] "https://gs.statcounter.com/os-market-share/mobile/worldwide," 2022. [Online].

[2] H. J. V. Gamma, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.

[3] C. K. Bass, Software Architecture in Practice (2nd Edition), Addison-Wesley, 2003.

[4] Kouraklis, MVVM in Delphi, Berkeley, CA: Apress, 2016.

6. Acknowledgements

I dedicate this work and my whole tenure in Milan to the one person without whom I would have probably never had come here, Ricardo. And of course, to the people without whom my time in Milan would have been lame – Andrea, Angelo, Emma, Francesca, Franco, George, Gosia, Heitor, Ivana, Kristina, Laura, Margot, Martìn, Pedro, Philipp, Rebeka, Sanja, Theresa, Tom, and most importantly, Toma, just to name a few.