



Analysis of Approaches to Developing Kotlin Multiplatform Applications and Their Impact on Software Engineering

Nikola Stanić^{1*}  [0000-0002-8306-0273] and Stefan Ćirković¹  [0009-0004-6775-1543]
¹ University of Kragujevac/ Faculty of Technical Sciences, Čačak, Serbia
* nikola.stanic@ftn.kg.ac.rs

Abstract: *This paper explores the concept of Kotlin Multiplatform and analyzes the approach to developing multiplatform applications using this tool. The research aims to analyze the key features of Kotlin Multiplatform, including its ability to share code across different platforms such as Android, iOS, and web. Through a detailed analysis of existing literature and case studies, the paper will explore the advantages and challenges of developing multiplatform applications using Kotlin Multiplatform, as well as their impact on software engineering. Special attention will be paid to performance, scalability, and code management within Kotlin Multiplatform projects. Based on the gathered data, the paper will also explore future perspectives of multiplatform application development, including potential trends, technological advancements and other cross platform solutions.*

Keywords: *Kotlin Multiplatform; multiplatform; software engineering; performance; code management*

1. INTRODUCTION

In today's digitally-driven world, the online presence of businesses has become more than just a trend, it's a necessity. With the rapid advancements in technology and the widespread accessibility of the internet, consumers are increasingly turning to online platforms to discover, interact with, and ultimately make purchasing decisions regarding products and services.

This would mean that every company has to develop a separate mobile application, web application, and desktop application. This leads to significant costs and inconsistencies across platforms. It requires having several teams of developers to build the application on each platform and later maintain it.

One way to solve this problem is by creating a single application that would work on all platforms. In 2017, the first version of Kotlin Multiplatform was released, aiming to enable the development of a mobile application that would work on both the Android and iOS systems.

Kotlin Multiplatform is intended to streamline the creation of cross-platform projects, reducing the time spent writing and maintaining separate codebases for different platforms while still allowing for the advantages of native programming. In Figure 1, all platforms supported by Kotlin Multiplatform are shown.

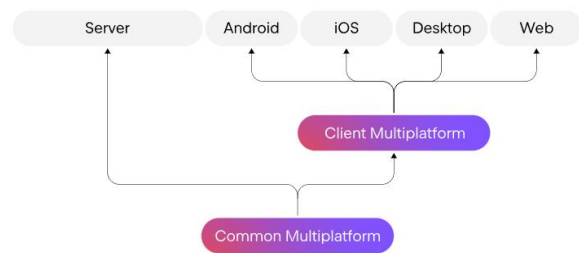


Figure 1. All platforms supported in Kotlin Multiplatform [1]

In the paper [2], a comprehensive methodology for developing multi-platform applications using the Kotlin programming language is presented. It highlights the use of Kotlin Multiplatform and Compose Multiplatform frameworks, which allow for the creation of universal logic code and user interfaces for Windows, Android, macOS, and Linux, thereby reducing development time and minimizing errors. The study emphasizes the principles of declarative programming and the MVI architectural pattern, alongside essential tools such as Kotlin Coroutines for asynchrony, Gradle Kotlin DSL, the Decompose library, and the MVIKotlin framework. A modular project architecture is proposed, divided into a common module with core application logic and platform-specific modules for application initialization and launch. Dependency Injection is effectively managed using Koin module files. This methodology offers a streamlined approach to developing user interfaces and application components across multiple platforms using Kotlin.

In the paper [3], the authors address the challenges faced by software developers in supporting multiple platforms, particularly mobile platforms, due to significant platform differences. They propose a native approach for developing multiplatform applications that run on both Java and Android. This approach tackles practical software engineering concerns, including tool configuration and the software design and development process. It enables sharing 37% to 40% of application code between the two platforms, which enhances the quality of the applications. The authors also suggest that this approach can be adapted for transforming existing Java applications into Android applications.

2. METHODOLOGY

To gather the necessary literature, electronic databases such as ResearchGate, Google Scholar, and ScienceDirect were utilized, as these are considered the most efficient tools for comprehensive literature searches. The focus is on papers published in the period from 2018 to 2024 in journals and collections of papers, papers written in English and based on qualitative and quantitative methods. Keywords such as "Kotlin Multiplatform," "cross-platform development", "Kotlin Multiplatform performance," and "Kotlin Multiplatform vs native development" were used. Selected papers have been chosen to investigate the advantages and disadvantages of Kotlin Multiplatform applications, as well as those examining the performance of this tool compared to similar ones. The selected papers were reviewed in full to extract detailed information about Kotlin Multiplatform. This included its technical capabilities, performance metrics, and case studies of its application. The extracted data were synthesized to provide a coherent narrative about the current state and future potential of Kotlin Multiplatform.

A SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis was conducted to evaluate the overall potential and challenges associated with Kotlin Multiplatform. Visual aids, such as charts and tables, were created to illustrate key findings and comparisons.

By employing this detailed methodology, the review aims to provide a thorough and objective analysis of Kotlin Multiplatform, offering valuable insights for developers, researchers, and industry professionals.

3. RESULTS AND DISCUSSION

The traditional approach to programming, known as the native development, involves using a language specific to the platform. For example, Kotlin for Android, Swift for iOS, Java for desktop applications, HTML for web applications, etc.

To avoid this, the industry has shifted towards cross-platform development, where by creating one application and using a single programming language, we obtain an application that works on all platforms.

3.1. Advantages and disadvantages

The main advantages of the native development are [4]:

- Best user experience,
- Great app performance,
- Leverage full platform capabilities

The main advantages of the cross-platform development are [4]:

- Reduce development time by reusing the same code for different platforms,
- Consistent behavior across platforms,
- Fewer bugs

Table 1. Kotlin SWOT analysis [1]

Kotlin - SWOT	
Strengths:	Kotlin Multiplatform enables high code reuse, provides native performance, and offers access to platform-specific APIs. Strong support from JetBrains and Google, along with an active developer community, further enhances its robustness and reliability.
Weaknesses:	Weaknesses include a limited number of libraries available for all platforms, initial setup complexity, and a steeper learning curve for new developers unfamiliar with the framework.
Opportunities:	Kotlin Multiplatform offers promising opportunities, with growing library support and increasing adoption in the industry. There is potential for expansion into more platforms, which could further enhance its appeal and utility.
Threats:	Despite its many advantages, Kotlin Multiplatform faces threats from other cross-platform solutions such as Flutter, Xamarin, and React Native, as well as the rapid pace of technological changes that could impact its relevance and efficiency.

With the Kotlin Multiplatform, developers can maintain a unified codebase for the application logic across various platforms. You also get advantages of native programming, including great performance and full access to platform SDKs. Kotlin provides the following code sharing mechanisms [1]:

- Share common code among all platforms used in your project.
- Share code among some platforms included in your project to reuse much of the code in similar platforms.

In Figure 2, the architecture of Kotlin Multiplatform is shown.

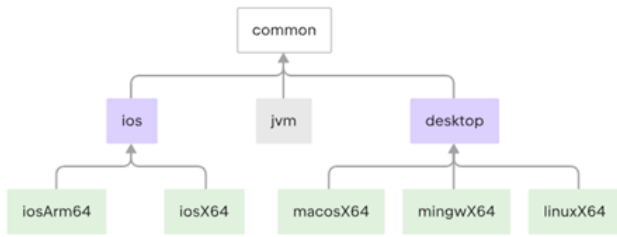


Figure 2. The Kotlin Multiplatform Mobile (KMM) architecture [1]

The main challenge that developers encounter in developing such applications is the limited number of libraries that can work across all platforms. The main task is to utilize a shared codebase across all platforms and libraries that function on each platform. If a library isn't supported on one of the platforms, we'd have to locate a platform-specific library and implement its functionality separately

Due to the accelerated development, this is becoming less of an issue, and this year we can even see that many libraries have released their beta versions of plugins that work on multiplatform applications like ktor, library that helps you build servers and clients that can handle tasks asynchronously.

Google's Android team is actively supporting Kotlin Multiplatform by releasing experimental multiplatform versions of Jetpack libraries. So far, they have made several libraries, including Collections, DataStore, Annotations, and Paging, compatible with Kotlin Multiplatform [5].

Kotlin Multiplatform uses the Compose Multiplatform framework for creating user interfaces. With the Compose Multiplatform UI framework, you can push the code-sharing capabilities of Kotlin Multiplatform beyond application logic. You can implement the user interface once and then use it for all the platforms you target – iOS, Android, desktop, and web. By combining Compose and Kotlin Multiplatform you can achieve your codebase to consist of Kotlin for 80–95% depending on the project's complexity. In Figure 3, the growth of the number of libraries supported by Kotlin Multiplatform over the years is shown.

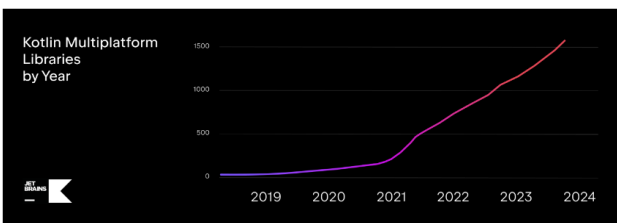


Figure 3. Kotlin Multiplatform Libraries by Year [3]

3.2. Performance test

In this chapter, several studies comparing the performance of applications developed using KMM versus native are presented.

In the 2019, Evert [6] published a study on the impacts on development productivity, application size, and startup time for Android and iOS applications developed in Kotlin Multiplatform.

Study [6] shows:

As for if the startup time is affected by using Kotlin Multiplatform instead of native development, the results speak partly for and partly against. The startup times seems to be significantly longer for the multiplatform Android, than the native Android, application. No significance can be seen in the difference in the startup times for the iOS applications.

As for the application sizes, they are shown to be larger in the multiplatform applications, than in the native applications. However, the larger size might very well be due to an initial application size overhead, and the difference in size might decrease with a more extensive application.

The results indicated that the Kotlin Multiplatform framework could make it possible for a developer to write less code but larger application size compared to developing natively.

In the study [7], the performance of Kotlin Multiplatform Mobile was compared to Swift for iOS development in terms of execution time, memory consumption, and CPU usage.

The results demonstrate generally faster execution times for KMM, yet with an overhead in higher memory consumption and CPU usage. This suggests that Kotlin Multiplatform can be opted for in performance-critical applications, whereas Native is suggested for apps that prioritize efficient resource usage [7].

4. DETAILED CASE STUDIES: PRACTICAL IMPLEMENTATION OF KOTLIN MULTIPLATFORM

In the case study [8], companies are showcased that have implemented various code-sharing strategies, including integrating into existing apps and sharing a portion of app logic, as well as building new applications based on Kotlin Multiplatform and Compose Multiplatform.

McDonald's leverages Kotlin Multiplatform for their Global Mobile App, enabling them to build a codebase that can be shared across platforms, removing the need for codebase redundancies [8].

After experimenting with both Flutter and React Native, 9GAG decided to implement Kotlin Multiplatform. They gradually adopted the technology and now ship features faster, while providing a consistent experience to their users [8].

Kotlin Multiplatform helps tech giant Netflix optimize product reliability and delivery speed, which is crucial for serving their customers' constantly evolving needs [8].

To understand the practical implementation of Kotlin Multiplatform across different types of applications, we will consider several detailed case studies. We analyze the use of sensors, games, and background service applications, and evaluate their suitability for Kotlin Multiplatform based on literature and practical examples.

4.1. Use of sensors in applications

The use of sensors in applications is an important aspect for many modern mobile applications, especially in the domains of health, fitness, and geolocation services. Kotlin Multiplatform allows sharing business logic between iOS and Android applications, but it has certain limitations when it comes to direct interaction with sensor hardware. Integration of platform-specific functionalities, such as sensors, requires additional efforts and the use of platform modules to ensure full functionality on both platforms [1]. Therefore, applications that heavily utilize sensors may be less suitable for Kotlin Multiplatform due to the complexity of implementation.

4.2. Games

The paper [16] provides insight into the suitability of Kotlin Multiplatform for game development. Kotlin Multiplatform allows sharing business logic across different platforms, which can be useful in developing simpler games that do not require intensive graphical performance. However, for more complex games that require high performance and optimized graphics, Kotlin Multiplatform may have certain limitations.

KMP has proven to be an effective tool for rapid development and code sharing, but games that demand a high refresh rate, low latency, and complex graphical effects may benefit from native development to achieve optimal performance. For instance, games that use advanced graphics libraries and heavily rely on the GPU may face challenges when developed using Kotlin Multiplatform, as this approach might lead to performance compromises [16].

4.3. Background service applications

Background service applications often require stability, efficient resource management, and the ability to execute tasks concurrently. Kotlin Multiplatform supports asynchronous and parallel programming through coroutines, enabling efficient management of background tasks. These features make Kotlin Multiplatform suitable for developing background service applications that require reliable operation without compromising performance [17].

Kotlin Multiplatform offers a powerful tool for developing applications that share business logic between iOS and Android platforms. While its suitability for certain types of applications, such as games and sensor-intensive apps, is limited due to performance and integration complexity, Kotlin

Multiplatform excels in developing background service applications. Each application should be carefully analyzed to determine if Kotlin Multiplatform is the most effective approach for its implementation.

5. OTHER CROSS PLATFORM SOLUTIONS

Cross-platform solutions enable the development of mobile applications for various operating systems. In this section, we will analyze the following technologies through the lens of SWOT to better understand their strengths, weaknesses, opportunities, and threats.

5.1. Flutter

Flutter began as a project by the Chrome browser team at Google to explore the feasibility of building a fast rendering engine with a non-traditional layout model. They wanted to see whether it is possible to build a fast rendering engine while ignoring the traditional model of layout. In a few weeks, significant performance gains were achieved and that is what was discovered:

- Most layout is relatively simple, such as: text on a scrolling page, fixed rectangles whose size and position depend only on the size of the display, and maybe some tables, floating elements, etc.
- Most layout is local to a subtree of widgets, and that subtree typically uses one layout model, so only a small number of rules need to be supported by those widgets.

In the paper [9], the author demonstrated the development of a currency exchange application using both Flutter and Kotlin Multiplatform tools. Since the paper was published in 2018, the main issues the author faced were the unavailability of libraries, which is less of a problem today. Studies indicate that while Kotlin Multiplatform executes faster, it requires more resources. However, with decreasing technology costs and rapid evolution, execution time remains crucial for user experience. The problem with library support is diminishing due to constant adaptation efforts. Introducing Compose Multiplatform has enhanced UI development across platforms, bringing additional benefits to Kotlin Multiplatform. It's worth noting that an author of another paper suggested that these tools offer performance almost identical to native development and referred to them as a sort of "golden bullet" in the debate between native and cross-platform development. This perspective highlights the optimism in the industry regarding the future of cross-platform tools like Kotlin Multiplatform.

As technology evolves, it may become less cost-effective for companies to stick with native development. The decision to transition to Kotlin Multiplatform should be based on the project's

specific requirements and the evidence from successful case studies.

Table 2. Flutter SWOT analysis [10]

Flutter - SWOT
Strengths: Rapid application development, hot reload functionality, high performance, material design. Supported platforms: iOS, Android, Web, macOS, Windows, Linux.
Weaknesses: Partial support for native APIs, fewer libraries and packages compared to other technologies.
Opportunities: Growing support and popularity, support for web applications.
Threats: Competition from other cross-platform solutions like Xamarin, Kotlin Multiplatform, and React Native.

5.2. XAMARIN

Xamarin is a cross-platform solution that allows development of mobile applications for iOS, Android, and UWP (Universal Windows Platform) using the C# programming language and .NET ecosystem [11].

Table 3. Xamarin SWOT analysis [9]

Xamarin - SWOT
Strengths: Stable support for native APIs, high performance, developed community. Supported platforms: iOS, Android, UWP (Universal Windows Platform).
Weaknesses: Complex configuration, need to familiarize with the .NET ecosystem.
Opportunities: Integration with Visual Studio, support for Xamarin.Forms for code sharing.
Threats: Competition from Flutter, Kotlin Multiplatform, and other solutions, limitations in supporting new platform features.

5.3. .NET MAUI

.NET MAUI (Multi-platform App UI) is a framework that enables development of multiplatform mobile applications using C# and XAML. It is announced by Microsoft as the successor to Xamarin.Forms, providing a modern, simple, and productive platform for creating applications that work on different devices and operating systems [12].

Table 3. .NET MAUI SWOT analysis [12]

.NET MAUI - SWOT
Strengths: Integration with the .NET ecosystem enables easier app development for various platforms. High productivity and code sharing capabilities between iOS, Android, Windows, and macOS. Stable support for native APIs and tools provided by Microsoft.
Weaknesses: Being in development, it may take time for the framework to stabilize and provide all functionalities. Adoption of new technology by development teams may require time.

Opportunities:

.NET MAUI promises improvements in performance, tools, and user experience compared to previous versions of Xamarin.Forms.
 Integration with Visual Studio IDE offers additional support and tools for development.

Threats:

Competition from other popular cross-platform solutions like Flutter, which already have a large community and support.
 Need to adapt to new trends and market demands to remain competitive.

5.4. React Native

React Native is an open-source platform for developing native mobile applications, utilizing standard web technologies such as JavaScript (JSX), CSS, and HTML, but the result is a fully native application. This means that the application runs fast, smoothly, and is equivalent to any native application built using traditional iOS technologies like Objective-C and Swift [13].

Table 4. React Native SWOT analysis [13]

React Native - SWOT
Strengths: Active community, hot reload support, access to native APIs. Supported platforms: iOS, Android.
Weaknesses: Performance can be variable, issues with library versioning.
Opportunities: Use of existing knowledge in JavaScript and React, support for a large number of platforms.
Threats: Competition from other cross-platform solutions, potential performance issues for complex applications.

5.5. NativeScript

NativeScript is an open-source framework for mobile app development that allows developers to use JavaScript, TypeScript, or Angular to build high-performance applications for iOS and Android platforms [14].

Table 5. NativeScript SWOT analysis [14]

NativeScript - SWOT
Strengths: Full support for native APIs, direct integration with Angular, TypeScript support. Supported platforms: iOS, Android.
Weaknesses: Larger resources needed for application development, smaller community compared to other solutions.
Opportunities: Code sharing between web and mobile applications, access to native components.
Threats: Competition from other cross-platform solutions, limitations in supporting certain platforms.

5.6. Electron

Electron is an open-source framework for developing desktop applications using JavaScript, HTML, and CSS. It enables the creation of cross-platform applications that run on Windows, macOS, and Linux using the same JavaScript codebase. Electron integrates Chromium and Node.js,

providing access to a rich set of tools and functionalities for developing complex applications. This framework is a popular choice for creating tools, editors, and other desktop applications [15].

Table 6. Electron SWOT analysis [15]

Electron - SWOT
Strengths: Easy integration with web technologies, support for various platforms. Supported platforms: Windows, macOS, Linux.
Weaknesses: Larger resources required for installation and execution, higher memory consumption.
Opportunities: Development of desktop applications using web technologies, access to system resources.
Threats: Competition from other technologies, potential performance issues for complex applications.

6. CONCLUSION

In this paper, we analyzed the concept of Kotlin Multiplatform, its performance, and its potential to replace native approaches. According to studies [6] and [7], Kotlin Multiplatform executes faster but requires more resources. Today, this isn't a major issue because the cost of technology is decreasing, and it's rapidly evolving, while execution time is crucial for user experience.

The problem with libraries supported for all platforms is diminishing as constant efforts are made to adapt them. Nonetheless, it's possible to separately implement functionality using another plugin if one isn't supported on a particular platform until a compatible one emerges for all.

By introducing Compose Multiplatform, it's now possible to develop user interfaces in Kotlin for all types of platforms, which brings additional advantages to Kotlin Multiplatform.

With the increasing development and improvement of Kotlin Multiplatform, it won't be cost-effective for programming companies to continue with the native approach. Whether it's better to transition to this technology is evidenced by case studies [6] of those who have adopted this approach.

Based on the SWOT analysis presented in Chapter 5, each of these technologies has its strengths and weaknesses. When choosing a cross-platform technology, it's important to carefully consider the specific requirements of the project and the goals of the application development.

Future research should focus on several key areas to enhance Kotlin Multiplatform. This includes advancing library support by developing new multiplatform libraries and improving existing ones. Performance optimization is crucial, with emphasis on compiler optimization, memory management, and execution speed. Enhancements in tooling, particularly in integrated development environments (IDEs) and automated testing tools, are also necessary. Additionally, expanding Jetpack

Compose for multiplatform applications will improve user interface consistency. Finally, gathering real-world case studies and fostering community engagement will provide valuable insights and accelerate Kotlin Multiplatform adoption.

ACKNOWLEDGEMENTS

This study was supported by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia, and these results are parts of the Grant No. 451-03-66 / 2024-03 / 200132 with University of Kragujevac - Faculty of Technical Sciences Čačak.

REFERENCES

- [1] Kotlin Language Documentation 1.9.20. URL: <https://kotlinlang.org/docs/kotlin-reference.pdf> (visited on 19.04.2024).
- [2] Kozub, Y., & Kozub, H. (2023). FEATURES OF MULTIPLATFORM APPLICATION DEVELOPMENT ON KOTLIN. Herald of Khmelnytskyi National University. Technical sciences. <https://doi.org/10.31891/2307-5732-2023-317-1-224-229>.
- [3] Cheon, Y. (2019). Multiplatform Application Development for Android and Java. 2019 IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA), 1-5. <https://doi.org/10.1109/SERA.2019.8886800>.
- [4] JetBrains Documentation 1.9.20. URL: <https://www.jetbrains.com/kotlin-multiplatform/> (visited on 19.04.2024).
- [5] *JetBrains Blog* URL: <https://blog.jetbrains.com/kotlin/2023/11/kotlin-multiplatform-stable/> (visited on 19.04.2024).
- [6] Anna-Karin Evert. "Cross-Platform Smartphone Application Development with Kotlin Multiplatform: Possible Impacts on Development Productivity, Application Size and Startup Time." MA thesis. KTH Royal Institute of Technology, 2019. URL: <https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1368323&dswid=6605>
- [7] Skantz, Anna. Performance Evaluation of Kotlin Multiplatform Mobile and Native iOS Development in Swift. <https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1793389&dswid=2208>
- [8] *JetBrains Case studies:* <https://www.jetbrains.com/help/kotlin-multiplatform-dev/case-studies.html> (visited on 19.04.2024).
- [9] Wasilewski K, Zabierowski W. A Comparison of Java, Flutter and Kotlin/Native Technologies for Sensor Data-Driven Applications. *Sensors*. 2021;21(10):3324. <https://doi.org/10.3390/s21103324>
- [10] R. Payne, "Beginning App Development with Flutter," 2023.

- [11] N. Mazloumi, "Building Xamarin.Forms Mobile Apps Using XAML: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals," 2019.
- [12] Microsoft, "Introducing .NET Multi-platform App UI (MAUI)," [dotnet.microsoft.com](https://dotnet.microsoft.com/en-us/apps/maui), Available: <https://dotnet.microsoft.com/en-us/apps/maui>. (visited on 19.04.2024).
- [13] A. Paul and A. Nalwaya, "React Native for Mobile Development," 2019.
- [14] NativeScript, Available: <https://nativescript.org/>. (visited on 19.04.2024).
- [15] Electron, "Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS," [electronjs.org](https://www.electronjs.org/). Available: <https://www.electronjs.org/>. Accessed: Apr. 21, 2024.
- [16] S. M. Lucas, "Cross-Platform Games in Kotlin," Game AI Research Group, School of Electronic Engineering and Computer Science, Queen Mary University of London.
- [17] Ю. Козыб, Г. Козыб, "Features of Multiplatform Application Development on Kotlin," doi: 10.31891/2307-5732-2023-317-1-224-229.