Developing a framework to evaluate the performance of mobile development technologies

ZL de Beer

orcid.org/0000-0003-4138-6985

Dissertation accepted in fulfilment of the requirements for the degree *Master of Engineering in Computer and Electronic Engineering* at the North-West University

Supervisor:	Dr SGJ van Niekerk
Co-supervisor:	Dr JC Vosloo

Graduation:	May 2022
Student number:	27100456

ACKNOWLEDGEMENTS

I want to express my immense gratitude towards the following parties:

- First and foremost, I would like to thank my almighty God, Lord and Saviour. I am genuinely grateful for the life that You have given me. Without Your guidance, love and mercy, this would not have been possible.

- To my loving parents, Louw and Marelise de Beer. Your love and support have been a pillar of strength throughout my life. Thank you for all you have taught me and all the opportunities you have granted me. Without you, this study would not have been possible.

- I want to thank my brother, Wiehan de Beer. You are my best friend. Thank you for your tremendous emotional support throughout this study. Your wit and humour were able to lift my spirits on even the darkest of days. Without you, this study would not have been possible.

- I want to thank my supervisor, Dr Sybrand van Niekerk, for your leadership throughout this study. Your inputs were highly valued, and it was a privilege to work under your guidance.

- I want to thank my co-supervisors, Dr Jaco Prinsloo, Dr Jan Vosloo, and Dr Jean van Laar for their advice and recommendations throughout this study. Thank you for sacrificing your time to assist me in this study.

- To all my co-workers, thank you for helping me grow and develop as a person and an engineer.

- I want to thank my friends for their patience and understanding. I would not have completed this study without your motivation and support.

- Finally, I would like to thank Prof. Eddie Matthews and Enermanage (Pty) Ltd for their financial support during this study.

ABSTRACT

Title: Developing a framework to evaluate the performance of mobile development technologies

Author: ZL de Beer

Supervisor: Dr SGJ van Niekerk

Co-supervisor: Dr JC Vosloo

Assistant-supervisor: Dr J Prinsloo

Keywords: Mobile development technologies; Performance evaluation framework; Cross-platform.

Mobile devices and applications have become invaluable tools in modern society. There are many approaches to mobile development, and for each one, there are multiple mobile development technologies. Performance is a crucial feature that can influence the success of a mobile application. There are many studies in literature evaluating mobile development technology performance.

There is, however, no framework available to assess the performance of mobile development technologies. The mobile industry is rapidly advancing, and the results of many performance-evaluation studies are no longer representative of the mobile development industry. For the academic community to keep up with industry, a continuous effort should be exercised to research the performance of mobile development technologies.

This study aims to develop a framework to evaluate the performance of mobile development technologies. The framework consists of four phases: selection, implementation, measurement, and evaluation. The developed framework is then applied by conducting a performance evaluation that compares five popular modern mobile development technologies. The mobile technologies comprise Android Native and four cross-platform technologies: Ionic, Flutter, React Native, and Xamarin. In addition, four benchmark tests were created, with CPU usage, RAM usage, and execution time being the selected performance metrics.

The results showed that Android Native was the best-performing technology overall; however, using the statistical analysis methods of the framework, it was determined that in tests where

Android Native performed the best, it never performed significantly better than the second-ranking technology.

The study concludes with a discussion on the work done. The application of the developed framework has shown that it can indeed be used to evaluate mobile development technology performance and can be used in future performance-evaluation studies. The research objectives of this study were met, and recommendations for future work are given.

TABLE OF CONTENTS

LIST OF FIGURES

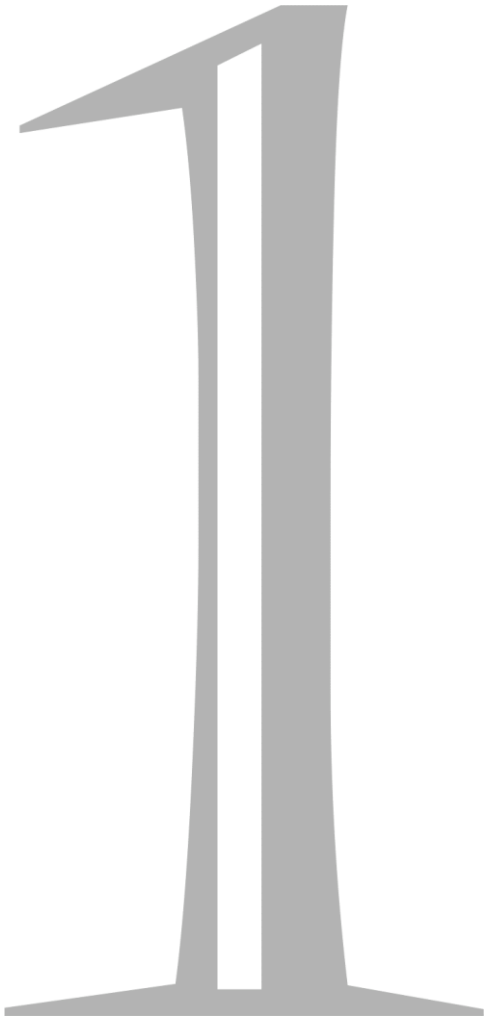
LIST OF TABLES

LIST OF ABBREVIATIONS

CPU	-	Central processing unit
CSS	-	Cascading style sheets
FPS	-	Frames per second
GPS	-	Global positioning system
GPU	-	Graphics processing unit
GUI	-	Graphical user interface
HTML	-	Hypertext markup language
IDE	-	Integrated development environment
MDA	-	Mobile development approach
MDT	-	Mobile development technology
RAM	-	Random-access memory
SDK	-	Software development kit
UX	-	User experience

CHAPTER 1

Introduction



1 INTRODUCTION

1.1 PREAMBLE

This chapter serves as an introduction to the study. It provides a background on the mobile industry, mobile development approaches, mobile development technologies, and the importance of performance in mobile applications. Following the background, a review of previous performance-evaluation studies is conducted. The background and analysis of prior studies found in literature will provide the required knowledge to understand the need for the study and the problem statement, which is also formulated in this chapter. Finally, the research objectives will be defined, and an overview of this dissertation will be given.

1.2 BACKGROUND ON THE MOBILE INDUSTRY

1.2.1 OVERVIEW

Mobile applications have become invaluable tools in the modern world. Mobile development has experienced significant growth in the last decade due to the general availability and popularity of smartphones [1]. In addition, advancements in mobile technology have increased mobile device features and performance, resulting in more capable and sophisticated mobile applications entering the market. The impact of the mobile industry will be highlighted and discussed in this section.

1.2.2 THE IMPACT OF THE MOBILE INDUSTRY

Mobile applications are changing the way we live our daily lives. An example of this would be mobile banking. According to a study conducted by Thusi *et al.* [2], 43% of the South African mobile population uses mobile devices to do their banking. The study also mentioned that 58% of millennial customers would consider switching to a competitor bank if offered a superior mobile experience.

The effects of mobile applications in education are also evident. The ubiquitous nature of mobile devices and applications provides new opportunities for educators to deliver educational content [3]. In a study conducted by Drigas *et al.* [4], the authors stated that the future of mobile application use in education looks promising. They argued that educational

applications allow students to learn anywhere and at any time, and could also lead to a more personalised learning experience.

Mobile gaming is a particularly successful branch of the mobile industry. Augmented reality and location-based mobile games, such as Pokémon™ Go, have been hugely successful. Pokémon™ Go has been downloaded 500 million times in just two months since its launch in 2016 [5]. Mobile games have recently also gained the attention of the Institute of Electrical and Electronics Engineers (IEEE); on 11 June 2020, they announced their Mobile Gaming Working Group (MGWG), the purpose of which is to optimise user experience in mobile games. They aim to do this by defining criteria and mechanisms for performance optimisations and evaluation methods for game fluency [6].

The success of social media applications, such as Instagram[1] and Tik Tok[2], further proves how integrated mobile applications have become in our lives. Instagram had approximately 1 billion active monthly users in 2019, and approximately 25 million of these accounts were official brand accounts [7]. Tik Tok is a short video-sharing application released in 2016. By the end of 2017, Tik Tok users had exceeded 700 million. It has since become the most downloaded application on Apple Inc. App Store [8].

At the time of this study, revenues generated by streaming services had also grown in recent years. Annual revenues for video streaming have grown from $30.3 billion in 2016 to an expected $70 billion in 2021. For music streaming, revenues have grown from $2.89 billion in 2015 to an expected $16.4 billion in 2021 [9]. Some giants in the streaming industry include YouTube, Netflix, and Spotify, with approximately 47% of mobile application users watching online videos using the YouTube video streaming service [10].

Ride-sourcing applications, such as Uber and Lyft, have revolutionised the way we approach transportation. Travellers who wish to get from one location to another without having to drive themselves request a ride through their mobile devices. Their location is then broadcasted to nearby potential drivers who are able to collect them within a few minutes of their request [11].

With mobile applications becoming critical to many industries, the demand for mobile application development is high. Millions of applications are available for download via application stores provided by Google and Apple[12]. In 2018, there were 2.5 million

[1] https://www.instagram.com
[2] https://www.tiktok.com/

applications available on the Google Play Store and 2 million applications on the Apple App Store [13].

The mobile application industry is projected to continue to proliferate. According to a research report by Statista [14], total global mobile application revenues are expected to grow from $365.2 billion in 2018 to a remarkable $935.2 billion from paid downloads and in-app advertising in 2023. With the current success of the mobile industry, it is logical for companies, businesses, and individuals to invest in mobile application development. In addition, a presence in mobile application stores could provide exposure to millions of potential customers.

1.2.3 Summary

This section highlighted the impact of the mobile industry, including how mobile applications have changed the way banking, education, gaming, transport, and entertainment are approached. It was emphasised that companies and individuals have a financial incentive to invest in mobile application development.

1.3 Background on mobile development approaches

1.3.1 Overview

The mobile development industry has grown significantly over the past few years. Consequently, multiple approaches to mobile application development have emerged [15]. These comprise (1) the native approach, (2) the hybrid approach, (3) the interpreted approach, (4) the cross-compiled approach, (5) the model-driven approach, and (6) the progressive web approach.

Each approach is fundamentally different in the type of mobile application it creates as well as the methods used to develop mobile applications. This section will give a background on the different approaches to mobile development most frequently encountered in literature.

1.3.2 Native approach

The mobile platform market is dominated by Google's Android and Apple's iOS. Google and Android provide software development kits (SDKs) and integrated development environments (IDEs) as well as other tools to develop mobile applications specifically for their platforms.

Native applications are not cross-platform, meaning they are restricted to their targeted platform. A native application targeting the Android platform cannot be used on the iOS platform and vice versa [15]. The advantage of this approach is having more control over each platform's native features [16].

If a company decides to use the native approach, they will need to develop distinct versions for each mobile platform. The development and maintenance of native applications for targeting multiple platforms are costly and labour-intensive. In addition, numerous native applications with separate codebases would require more developers with platform-specific experience.

An overview of the native approach to mobile development can be seen in Figure 1-1. Developing native Android applications requires developers to be familiar with the Java or Kotlin languages [15]. Google also provides an official IDE called Android Studio. The languages needed to develop native iOS applications are Swift or Objective-C, and the official IDE provided by Apple is called Xcode [17]. Through Android Studio and Xcode, installable native Android and iOS applications are created. These native applications are then natively executed by each platform.

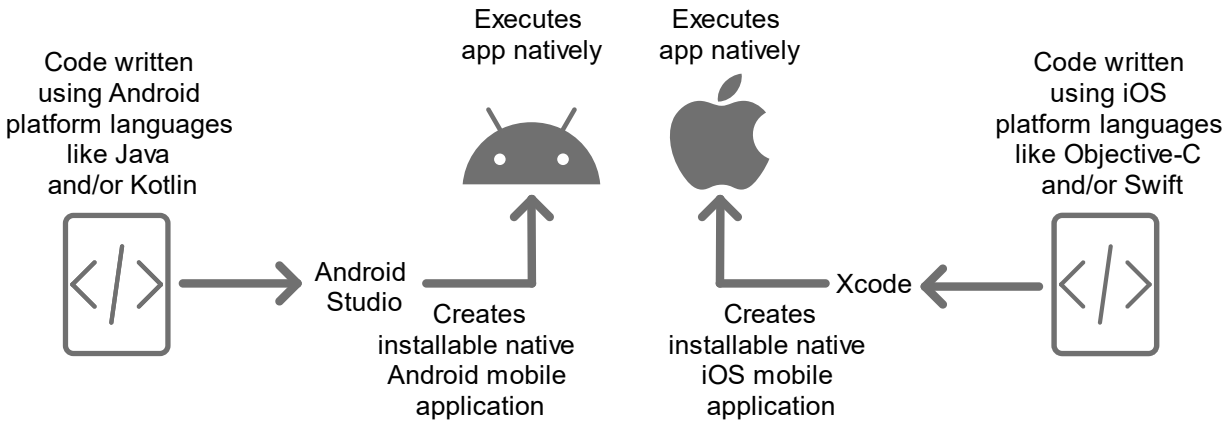


Figure 1-1- Overview of the native approach to mobile development [18].

1.3.3 HYBRID APPROACH

Hybrid applications are cross-platform, meaning there is no need to develop multiple applications for the numerous platforms available [19]. Instead, hybrid applications are built using standard web technologies such as HTML, CSS, and JavaScript. This allows developers with experience in web development to create mobile applications using their existing knowledge and skills.

Hybrid applications combine native technologies and web technologies by using a component called WebView [20]. The WebView component is an embedded web browser, allowing code written in HTML, CSS, and JavaScript to be executed as if it were a website.

Most hybrid applications make use of a tool called Cordova [18]. Cordova is the underlying tool responsible for controlling the WebView as well as the managing and packaging of code. It also provides multiple plugins to access each platform's native features. Although most hybrid applications are based on Cordova, new alternative tools like Capacitor are entering the scene [18].

An overview of the Hybrid approach can be seen in Figure 1-2. As previously mentioned, code is written using common web technologies like HTML, CSS, and JavaScript. An MDT based on Cordova or Capacitor generates an installable mobile application for both the iOS and Android platforms. These applications are then executed within an embedded WebView controlled by Cordova or Capacitor.

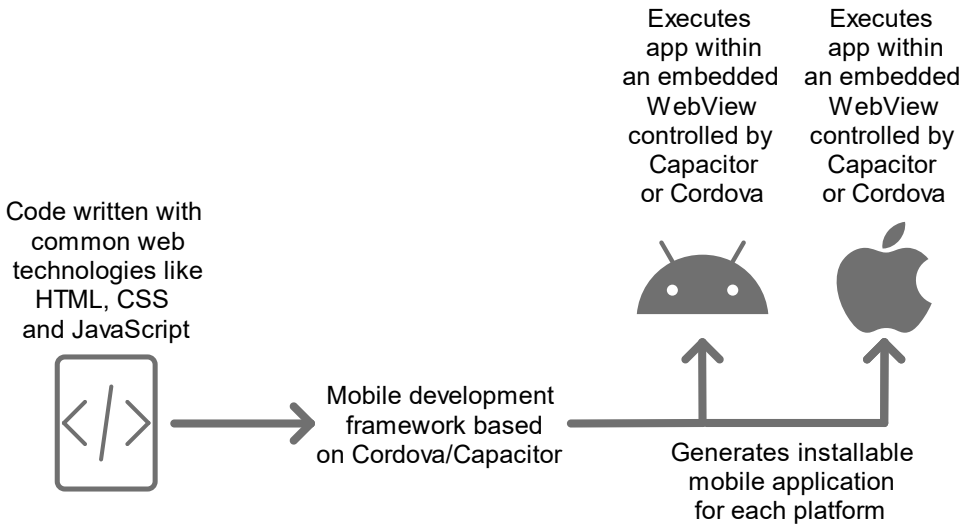


Figure 1-2- Overview of the hybrid approach to mobile development [18].

1.3.4 INTERPRETED APPROACH

Interpreted applications are like hybrid applications in that they are both cross-platform and can be developed using JavaScript; however, the fundamental difference is that interpreted applications do not use a WebView component that renders a website. Instead, interpreted applications render native elements to the screen during runtime.

With interpreted applications, there is a dedicated interpreter that interprets code during runtime [21] using on-device JavaScript interpreters like JavaScriptCore for iOS and V8 for

Android [22]. In addition, interpreted applications gain access to native application features and sensors through a data-abstraction layer.

An overview of the interpreted approach can be seen in Figure 1-3. Code is written using a non-native language like JavaScript. Through the interpreted MDT, an installable mobile application is generated for each mobile platform. These mobile applications are then installed and executed on each platform. On Android platforms, the V8 interpreter is used during runtime, while on iOS platforms, the JavaScriptCore interpreter is used during runtime.

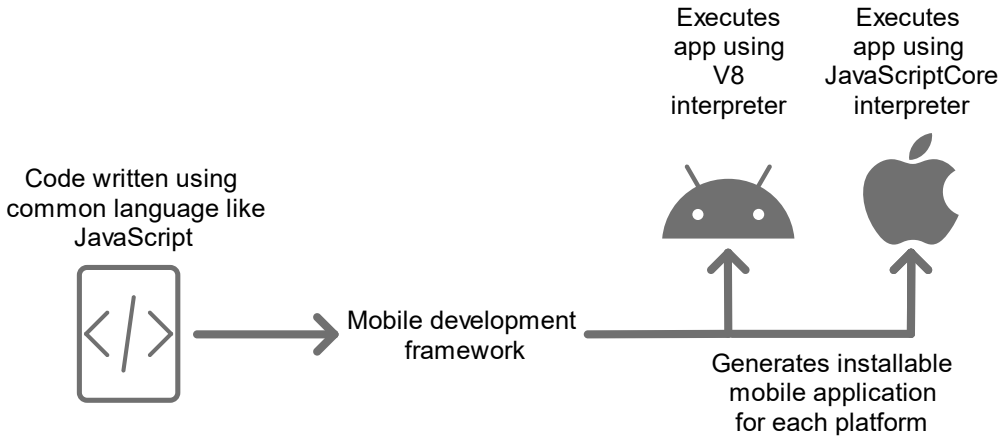


Figure 1-3- Overview of the interpreted approach to mobile development [18].

1.3.5 CROSS-COMPILED APPROACH

A cross-compiled application converts code written in a non-native language into native binaries [19] [23]; in other words, the cross-compiler generates native code for each platform. Due to the cross-compiler, there is no data-abstraction layer present in cross-compiled applications as in the case of the hybrid and interpreted applications [18].

An overview of the cross-compiled approach can be seen in Figure 1-4. Code is written using common non-native languages like C# or JavaScript. The cross-compiled MDT then creates installable native applications for each platform using native code generated by the MDT. These applications are executed natively on each mobile platform as if they were created using native tools.

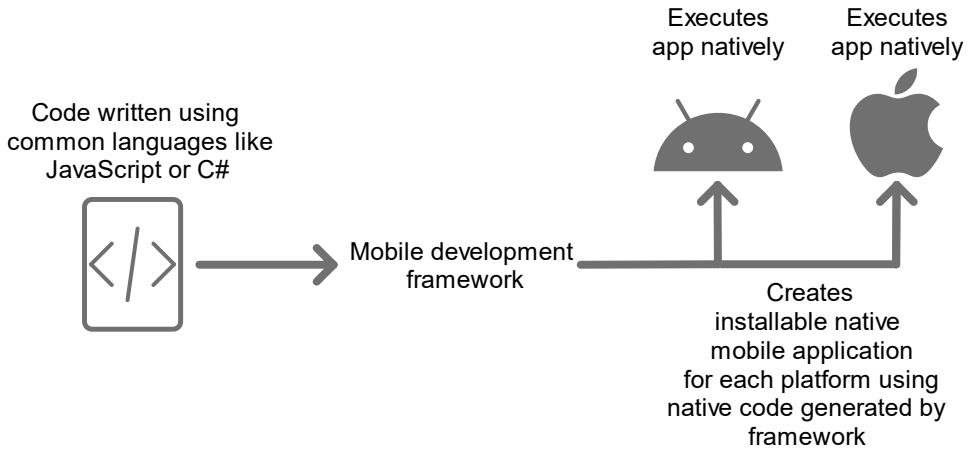


Figure 1-4 - Overview of the cross-compiled approach [18].

1.3.6 MODEL-DRIVEN APPROACH

The model-driven approach addresses the need for businesses to create mobile applications even when they lack employees with programming experience. With the model-driven approach, a mobile application is described on a higher level without worrying about writing low-level code [24], [25]. A defining characteristic of the model-driven approach is that both developers and non-developers can create mobile applications.

The model-driven approach is also cross-platform, using platform-agnostic models to generate native platform-specific mobile applications. These models are described through either the Unified Modelling Language (UML) or a Domain Specific Language (DSL) [17]. It is interesting to note that the model-driven approach is frequently discussed in literature but is not as popular among developers [22]. MD² is an example of a model-driven cross-platform framework that originated from an academic context [26].

An overview of the model-driven approach can be seen in Figure 1-5. Through models written using DSL that describe UI and business logic, the model-driven MDT generates native installable mobile applications for each platform. This is done through native code generated by the framework. Each application is then natively run by the platforms.

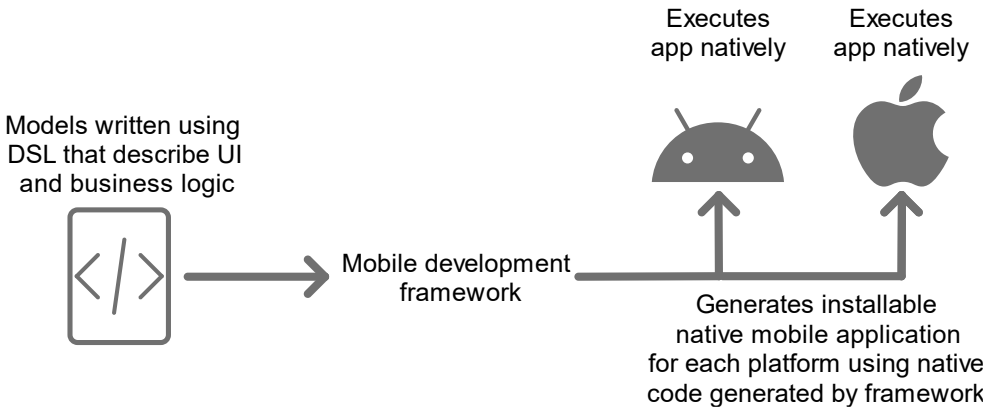


Figure 1-5- Overview of the model-driven approach [18].

1.3.7 PROGRESSIVE WEB APPROACH

Progressive applications are a novel cross-platform mobile development approach. Advocated by Google, this technology aims to combine the benefits of the web approach and the native approach, and seeks to bridge them by introducing home-screen installation, offline functionality, and push notifications [27].

Since progressive web applications are essentially hosted websites, there is no need for distribution through an app store or regular updates. Progressive web applications differ from the hybrid approach because they use the Service Worker API rather than an embedded WebView. Upon initial access to the website from the web host, the progressive web application will be downloaded and installed on the device.

The Service Worker is a background script written in JavaScript that manages the application's life cycle, data synchronisation, and push notifications [18]. The Service Worker also handles the proxy of network connections, background tasks, and provision of an offline experience. Whenever content is displayed, the Service Worker will decide whether cached content is still relevant or if new content should be loaded from the hosted website.

An overview of the progressive web approach can be seen in Figure 1-6. Code is written using common web technologies like HTML, CSS, and JavaScript. The code is then hosted on the web. The web host serves a web application that can be accessed through a PWA-compliant mobile browser. The PWA will be installed on the device when first accessed, after which internet connection is no longer needed. The mobile application is then executed inside the PWA-compliant browsers on each platform with the Service Worker managing functionality.

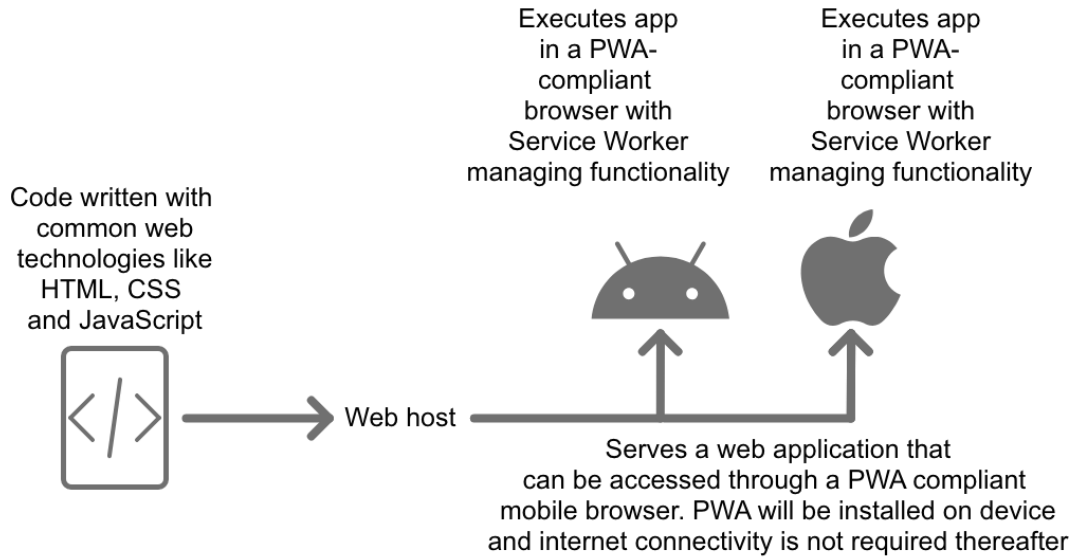


Figure 1-6 – Overview of the progressive web approach to mobile development [18].

1.3.8 SUMMARY

In this section, a background to the different mobile development approaches found in literature was provided. The approaches that were discussed include: (1) the native approach, (2) the hybrid approach, (3) the interpreted approach, (4) the cross-compiled approach, (5) the model-driven approach, and (6) the progressive web approach.

1.4 BACKGROUND ON MOBILE DEVELOPMENT TECHNOLOGIES

1.4.1 OVERVIEW

This section will give a brief background on mobile development technologies and how they differ from mobile development approaches. Examples of mobile development technologies used in industry will also be provided.

1.4.2 MOBILE DEVELOPMENT TECHNOLOGIES

For every mobile development *approach* (discussed in Section 1.3), multiple mobile development technologies utilise said approach to allow developers to create mobile applications.

It is essential to understand the difference between mobile development *technologies* and mobile development *approaches*. Mobile development technologies provide the tools, software libraries, and environments needed to create mobile applications. Mobile development

approaches describe how a mobile application is produced and what type of mobile application is produced. Each mobile development technology falls under a mobile development approach.

In Table 1-1, 74 mobile development technologies used in industry can be seen. Note that some mobile development technologies, such as *Ionic*[3], can be configured to utilise different approaches and are thus present in multiple categories. Table 1-1 emphasises the sheer number of MDTs available that can be used to create mobile applications. Observe how many cross-platform MDTs there are compared to native MDTs. This emphasises developers' interest in one codebase that targets multiple platforms.

Developers will ultimately have to choose an MDT. Several factors will influence this choice, including the application's requirements, performance, and features as well as the time-to-market, costs, skills, and developers' knowledge [28], [29].

Table 1-1- Mobile development technologies used in industry [18].

Native (Section 1.3.2)	**Hybrid (Section 1.3.3)**	**Interpreted (Section 1.3.4)**	**Cross-compiled (Section 1.3.5)**	**Model-driven (Section 1.3.6)**	**Progressive web (Section 1.3.7)**
Android Native	AppGyver	Adobe AIR	Apportable	Appian	Angular
iOS Native	AppStudio	Fusetools	Codename One	Applause	Ember.js
	Capacitor	Jasonette	Corona	Automobile	Glimmer.js
	Cocoon	Kony	DragonRAD	AXIOM	Ionic
	Cordova	LuaView	Flutter	MAML	Mithril
	EvoThings	MoSync	Marmalade	Mdsl	Moon.js
	Framework7	NativeScript	MonoCross	MD²	Lit (Polymer)
	Intel App Framework	React Native	MoSync	Mendix	Preact
	Intel XDK	Smartface Cloud	Qt Mobile	MobDSL	React.js
	Ionic	Tabris.js	RAD Studio	MobiaModeler	Sencil.js
	Kony	Titanium Appcelerator	Rhodes	Mobl	Svelte
	Onsen UI	Weex	RoboVM	MobML	ViperHTML
	PhoneGap		Xamarin	WebRatio	Vue.js
	Quasar		XojoMobile	XIS-Mobile	Zuix
	RhoMobile			Xmob	
	SenchaTouch				
	Trigger.io				

[3] https://ionicframework.com/

1.4.3 SUMMARY

In this section, a background on mobile development technologies was given. Mobile development technologies were then described and distinguished from mobile development approaches. Finally, mobile development technologies used in industry were listed and categorised by mobile development approach.

1.5 BACKGROUND ON MOBILE APPLICATION PERFORMANCE

1.5.1 OVERVIEW

The previous sections established that there are multiple mobile development approaches and multiple development technologies utilising each approach. In this section, background on user experience and the importance of mobile-application performance will be discussed.

1.5.2 THE CHALLENGES OF MOBILE APPLICATIONS AND PERFORMANCE

With mobile applications becoming an integral part of society, developers need to provide a high-quality user experience. User experience plays a significant role in the success of a mobile application. Some of the critical factors influencing the user experience of a mobile application [30] include:

- UI design
- Battery efficiency
- Utilisation of mobile features
- Data costs
- Performance

UI design refers to the design of an interface through which the users interact with the application. This includes the position of buttons and other UI elements. Battery efficiency refers to how much battery the application consumes while in use. Poor battery efficiency influences user experience since it limits device usage. Utilisation of mobile features refers to the ability of an application to use native mobile features such as camera access or contacts access. Data costs are another factor influencing user experience. High data costs prevent users from using or experiencing applications. Finally, performance refers to how well the application runs on their device.

From the factors listed above, we can see that performance is a critical factor influencing user experience. A higher level of performance thus leads to a higher rate of user acceptance. In the mobile industry, developers will need to choose the mobile development technology they want to use to create their applications. An important choice is choosing between native and cross-platform development technologies [31].

It is commonly understood that the native approach allows for better performance and greater control and access to native mobile features [28], [32], [33]. Consequently, native development technologies require more development time, costs, and knowledge. On the other hand, mobile development technologies based on cross-platform approaches (Sections 1.3.3 - 1.3.7) require fewer development resources, but generally at the cost of performance.

In a study conducted by Joorabchi *et al.* [34], the authors outlined the challenges of mobile application development for multiple platforms. 76% of their survey participants reported that developing the same applications for each platform is a considerable challenge, as each platform has different standards and expectations regarding interfaces and user experience. The authors remarked that if cost and time-to-market are a priority, a cross-platform framework should be used; alternatively, if user experience (or *performance*) is a priority, the authors recommend the native approach.

It would be beneficial for mobile developers to choose a mobile development technology that produces performant mobile applications. There are numerous studies found in literature that study the performance of mobile development technologies and approaches. Despite this, Majchrzak *et al.* [35] and Gaouar *et al.* [36] have mentioned that more empirical studies are required to broaden the knowledge of performance-related concerns between different mobile development technologies and approaches.

In a recent study conducted by Biørn-Hansen *et al.* [18], the authors describe the challenges regarding mobile development research's state of the art. They concluded that new mobile development research relies too heavily on older studies that are no longer representative of the current state of mobile development. Mobile development approaches and technologies have significantly progressed in recent years, and challenges commonly discussed in literature might no longer be relevant or accurate.

The mobile development industry is changing at a rapid pace. If the academic community wants to keep up with the industry, a continuous effort should be exercised to research the performance of mobile development technologies.

Mobile developers and researchers are, therefore, left with a challenge. Performance is key to user experience, user experience leads to user acceptance, and user acceptance influences the success of the mobile application. The performance results of many studies are no longer relevant, so mobile developers and researchers must conduct performance evaluations themselves.

The following question can be asked:

How can they *evaluate the performance of mobile development technologies* when choosing one for their next project?

1.5.3 SUMMARY

In this section, a background to the challenges of mobile development and performance was given. Performance was established as being key to user experience and thus user acceptance and the overall success of a mobile application. Literature has shown that new research relies too heavily on studies that are no longer representative of the current state of mobile development. The mobile development field is changing at a rapid pace, and more empirical studies are needed. A question was raised as to how the performance of mobile development technologies could be evaluated.

1.6 REVIEW OF PREVIOUS PERFORMANCE-EVALUATION STUDIES

1.6.1 OVERVIEW

In this study so far, the background sections provided the knowledge required to conduct a literature review. The following ideas were established:

- The mobile industry is flourishing, with clear incentives for individuals and companies to invest in mobile application development (Section 1.2).
- There are multiple *approaches* to mobile application development, namely cross-platform and platform-specific approaches. Each approach differs in how it produces an application and the type of application it produces (Section 1.3).

- For each *approach*, there are multiple mobile development *technologies*. Mobile development technologies provide the necessary tools, software libraries, and environments to create mobile applications (Section 1.4).
- Performance influences the overall success of a mobile application, but many performance evaluation study results are dated. Developers should evaluate the performance of mobile development technologies themselves. A question was raised as to how they would be able to accomplish this (Section 1.5).

A literature review will be conducted in this section, where previous performance-evaluation studies will be examined. The literature review aims to offer an answer to the question asked in Section 1.5.2, namely, *How can the performance of mobile development technologies be evaluated?* Each study will be summarised using the following main discussion points:

- Mobile development technologies used in the study
- Overview of the study
- The method used in the study
- Results of the study
- A brief commentary on the study

After each study is examined, a section will follow where literature will be discussed.

1.6.2 PREVIOUS PERFORMANCE-EVALUATION STUDIES

Common framework: A hybrid approach to integrate cross-platform components in mobile application (2014) [37]	
MDTs:	Android Native, COMMON, Titanium Appcelerator
Overview:	Perchat *et al.* investigated the performance of a mobile development technology they created called COMMON. The authors compared COMMON with other mobile development technologies.
Method:	They created a fully-featured benchmark application using the mobile development technologies and compared lines of code, execution times, application sizes, and RAM usage. Their tests involved GPS, networks access, and computational tests.
Results:	The results showed that their COMMON framework performed better than Titanium Appcelerator in all comparisons and had comparable performance to Android Native.
Commentary:	This study compared MDTs by implementing a fully-fledged application using each MDT. This implementation method requires a significant amount of time, especially when multiple MDTs are compared.

Evaluating impact of cross-platform frameworks in energy consumption of mobile applications (2014) [38]

MDTs:	Android Native, Titanium Appcelerator, PhoneGap
Overview:	Ciman and Gaggi studied the impact of cross-platform MDTs on energy consumption. They compared cross-platform MDTs with native MDTs and measured the energy consumption of mobile sensors and features.
Method:	The cross-platform MDTs, Titanium Appcelerator and PhoneGap, were compared with Android Native. Their benchmark tests included measuring the impact of using the accelerometer, compass, microphone, GPS, and camera on energy consumption. In addition, energy consumption was measured using a hardware device called Monsoon Power Monitor and Power Tool software.
Results:	The authors concluded that the use of cross-platform mobile development technologies would increase energy consumption.
Commentary:	This study gave compelling insight into the energy consumption of mobile applications created using different MDTs. They also showed how the frequency of sensor activities influences energy consumption, urging developers to reduce sensor readings in their applications. Their method did involve repeating the tests, but the authors did not specify how many times their benchmark tests were repeated.

An evaluation framework for cross-platform mobile application development tools (2015) [39]

MDTs:	Android Native, iOS Native, PhoneGap, Titanium Appcelerator, Adobe AIR, WebWorks, MoSync
Overview:	This study proposed an evaluation framework for cross-platform MDTs. The proposed evaluation framework aims to assist in evaluating features, performance, and development experience for current and future MDTs.
Method:	Multiple benchmark tests were performed where execution times were measured. The benchmarks tests were grouped under processor-intensive, data-driven, device access, and user-experience categories.
Results:	The results of the study indicated a disparity among the MDTs used. The authors concluded that there are apparent differences between the MDTs, with performance and lack of features being the limitations of cross-platform MDTs.
Commentary:	The authors provided a helpful measurement process that can be used to measure performance. The authors specified how many times tests were repeated, and their tests were thoroughly described. A shortcoming of this framework is that it is not generic in nature and thus not suitable as a standardised framework for MDT performance evaluations.

A quantitative assessment of performance in mobile app development tools (2015) [40]	
MDTs:	Android Native, iOS Native, PhoneGap, Xamarin
Overview:	In this study, the authors claim that cross-platform MDTs have a performance overhead as compared to native MDTs, but the exact overhead size is unclear. Their study aimed to compare cross-platform MDTs with native MDTs.
Method:	Benchmark applications were created using each MDT. They compared response times, load times, RAM usage, CP usage, application sizes, and energy consumption.
Results:	The results of their study showed that there is a performance penalty when using cross-platform MDTs, but that this penalty is acceptable for most cases. Xamarin performed better than PhoneGap, but both frameworks performed worse than Android Native and iOS.
Commentary:	PropertyCross was used for their benchmark applications and tests. PropertyCross is a deprecated open-source project where the same fully-fledged application was created using different MDTs. Since the project is no longer being supported, it cannot be used for future studies, especially when new MDTs are required for comparison.

Measuring energy consumption of cross-platform frameworks for mobile applications (2015) [41]	
MDTs:	Android Native, Titanium Appcelerator, PhoneGap
Overview:	Ciman and Gaggi conducted another study measuring the energy consumption of cross-platform frameworks for mobile applications; however, in this study, they included web applications accessed through browsers.
Method:	The cross-platform MDTs and web applications (run on Firefox, Chrome, and Opera browsers) were compared with Android Native. Their benchmark tests were the same as in their previous study [38].
Results:	Their findings were predominantly the same as their previous study [38]. Cross-platform MDTs had increased energy consumption when compared to native MDTs. They made the statement that web applications were not ready for widespread adoption at the time, as energy usage was much higher compared to the MDTs.
Commentary:	Apart from describing the energy impact of web applications running in mobile browsers, this study provided no additional insight into the energy consumption of MDTs, as the tests, methods, and MDTs are similar to their previous study.

A comprehensive comparison between hybrid and native app paradigms (2016) [42]	
MDTs:	Android Native, Cordova
Overview:	Que *et al*. performed a comparison between hybrid and native application approaches. They compared the two approaches by the ease of coding, testing, distribution, ease of use, functionalities, and performance.
Method:	The benchmark tests they used targeted the camera, accelerometer, GPS, and media playback. They measured the installation time, start-up time, CPU usage, RAM usage, battery temperature, and network flow.
Results:	The native application had faster installation times, shorter start-up times, lower CPU usage, lower RAM usage, and lower battery temperature.
Commentary:	This study used a cloud-testing platform based on real hardware devices to measure performance data. This method allowed the authors to gather the performance data of 200 mobile devices. This method of testing could be used to save considerable time in future studies.

Comparing performance parameters of mobile app development strategies (2016) [32]	
MDTs:	Native Android, Native iOS, Ionic, Sencha Touch 2, jQuery Mobile, Intel App Framework, MGWT, Famou.us, Adobe AIR, NeoMAD, Titanium Appcelerator, Xamarin
Overview:	In this study, the performance of 10 cross-platform MDTs was compared with two native MDTs.
Method:	Benchmark applications were implemented using each MDT. The benchmark applications were based on the community-driven PropertyCross initiative. They measured launch times, navigation times, CPU usage, RAM usage, disk space, and battery usage.
Results:	The study showed that MDTs using the same approach have similar behaviours. Even though there is a performance penalty for using cross-platform MDTs, this penalty is acceptable, especially on high-end devices.
Commentary:	This study showed how the performance of cross-platform applications differs based on the platform (iOS or Android). Platform performance should not be assumed based on other platforms' performances. If a platform's specific performance is essential, then all significant platforms should be included for study.

An empirical analysis of energy consumption of cross-platform frameworks for mobile development (2017) [43]	
MDTs:	Android Native, iOS Native, PhoneGap, Appcelerator Titanium, MoSync
Overview:	In this study, Ciman and Gaggi performed another empirical analysis of energy consumption. They made use of multiple devices for this study and included more platforms.
Method:	The selected MDTs were used to implement benchmark applications. The impact of accelerometer, device orientation, compass, proximity, light, GPS, camera, and audio recording on energy consumption was measured.
Results:	Their results again confirmed a cost in terms of increased energy consumption when using cross-platform MDTs.
Commentary:	This study provided additional insight into the performance of MDTs, especially since the authors included more platforms and MDTs. This study showed that cross-platform MDTs always lead to increased energy consumption. Another important observation is that the choice of MDT language will influence performance, as seen with MoSync (JavaScript and C++).

Performance analysis of native and cross-platform mobile applications (2017) [44]	
MDTs:	Android Native, iOS Native, Xamarin
Overview:	This study analysed the performance of cross-platform and native applications. The aim was to compare the performance of mobile applications created in Xamarin with native Android and iOS.
Method:	Benchmark tests included numerical calculations, reading and writing a text file, network access, and GPS location. Execution times were then measured.
Results:	Their results confirmed that native iOS and Android performed better in most tests; however, the authors concluded that Xamarin performs well, and if developers are aware of the performance impact, it is worth using.
Commentary:	This study showed, contrary to many previous studies, that a cross-platform MDT can perform better than a native MDT. In their text file reading test, Xamarin outperformed Android Native.

Progressive web apps: the possible web-native unifier for mobile development (2017) ***[27]***

MDTs:	React Native, PWA, Ionic
Overview:	This study argued that progressive web applications are a possible unifier for web applications and native applications. The features of progressive web applications compared with the performance of two other cross-platform frameworks were discussed.
Method:	Installation sizes and activity launch times were compared. The study also measured the time it took to render the application toolbar from the icon tap.
Results:	The PWA had the smallest installation size and the quickest launch time. React Native rendered the toolbar 3.5 times faster than the PWA. There was a remarkable difference between the hybrid application and the PWA. The hybrid application rendered the toolbar 10 times slower than the PWA.
Commentary:	This study provided insight to PWAs as a new standard for bridging the gap between native mobile applications and web applications and showed that PWAs are competitive to the standard mobile application. The authors highlighted that their benchmark tests relied on error-prone human reactions. This error-prone nature emphasises a need for benchmark tests that are not dependent on human responses.

A performance evaluation of cross-platform mobile application development approaches (2018) [45]

MDTs:	Android Native, iOS Native, Xamarin Forms, Xamarin Android, Cordova, Titanium Appcelerator
Overview:	This study aimed to measure the performance characteristics of mobile applications created using different approaches and tools to give insight into the trade-offs and designs of the various mobile development technologies.
Method:	Three benchmark applications were designed. The authors made multiple observations regarding the building times, rendering times, total UI response times, RAM usage, and application sizes.
Results:	The results of their study showed that there are significant performance differences when using different mobile development technologies
Commentary:	Even though this study discussed the performance of the specified MDTs relative to each other, no empirical data was provided, making it difficult to assess how significant the differences in performance were.

An evaluation of cross-platform frameworks for multimedia mobile applications development (2018) [46]	
MDTs:	Android Native, iOS Native, PhoneGap, Sencha Touch, Titanium Appcelerator
Overview:	In this study, the authors wanted to determine whether cross-platform mobile development technologies have enough resources to develop multimedia and sensor applications.
Method:	The first test involved measuring the RAM usage of taking a picture, applying a filter, sharing the picture, and finally sharing the device's location. The second test measured the execution time and RAM usage of 100 interactions of applying filters to an image.
Results:	The results showed that the cross-platform applications had similar performances to their native counterparts in RAM usage. The cross-platform applications lagged in execution times.
Commentary:	This study used an interesting method to select the MDTs for comparison. The authors chose the cross-platform MDTs by their popularity on GitHub, emphasising the importance of evaluating MDTs used by the community.

Bridging the gap: investigating device-feature exposure in cross-platform development (2018) [22]	
MDTs:	Ionic, React Native
Overview:	In this study, device-feature exposure in cross-platform development was investigated. The authors assessed and compared how the hybrid approach and the interpreted approach facilitate native hardware features within a JavaScript context.
Method:	Two benchmark applications were implemented. Upon pressing a button, the applications fetch an image from the device storage and display it to the user. They measured the execution time for each application.
Results:	The Ionic application was five times more efficient at communication between the native side and application side than React Native.
Commentary:	This study is unique in that the authors wrote their plugins themselves. They also used language-based timer functionality to measure execution time, meaning that human reactions do not influence their results, as is the case in a previous study [27].

Development approaches for mobile applications: comparative analysis of features (2019) [47]	
MDTs:	Native Android, Native iOS, Ionic, Cordova, Titanium Appcelerator, NativeScript, Xamarin, Corona
Overview:	In this study, development approaches for mobile applications were discussed. They performed a comparative analysis of features that will affect the choice of development approach to be used.
Method:	The authors compared performance, installation modes, battery usage, application sizes, image rendering, and initial boot times. They used a scale ranging from very high to very low.
Results:	No quantitative results were given. Only scores from very high to very low were assigned to each MDT.
Commentary:	The authors extensively compared features across multiple categories such as non-functional features, developer features, and project management. Their research is very informative and provides excellent insight into different approaches to mobile development; however, even though they discussed performance, more understanding could have been obtained if quantitative results had been discussed.

Development frameworks for mobile devices (2018) [48]	
MDTs:	Native Android, Cordova, Titanium Appcelerator, Xamarin, Corona
Overview:	This study focused on the energy consumption of MDTs, mainly on the Android platform.
Method:	Three benchmark applications focused on features such as intensive processing, video playback, and audio playback. Energy consumption, CPU usage, and execution times were measured.
Results:	Titanium was the only cross-platform MDT that performed well in all three benchmark tests. The performance of the other MDTs showed many variances.
Commentary:	A significant result of their study is that the MDTs' performance depended on the type of benchmark tests, suggesting that a single benchmark test cannot infer an MDT's performance and that overall performance depends on multiple tests.

Animations in cross-platform mobile applications: an evaluation of tools, metrics and performance (2019) [49]	
MDTs:	Native Android, Native iOS, React Native, Ionic, Xamarin
Overview:	This study aimed to investigate transitions and animations in mobile-user interfaces developed using native and cross-platform MDTs.
Method:	Three animation benchmark applications were created for the side menu, page navigation, and Lottie star animations. They measured FPS, CPU usage, RAM usage, and GPU memory usage.
Results:	The CPU usage on Android was consistently lower than iOS across all three animation benchmark tests, regardless of the framework used. The authors also noted that the performance of cross-platform MDTs is different for each platform.
Commentary:	This study provided insight into the animation capabilities of different MDTs. This study was the first study in this literature review that investigated GPU performance and the measuring of FPS. This study also suggested that MDT performance is dependent on the platform.

Performance analysis of mobile cross-platform development approaches based on typical UI interactions (2019) [50]	
MDTs:	Android Native, Ionic, React Native
Overview:	This study analysed the UI performance of MDTs. They chose MDTs based on popularity among the developer community. Apart from the cross-platform MDTs, they also included native Android for comparison.
Method:	A benchmark application with 1000 contacts in a scrollable virtual list was implemented. They measured the performance impact of three bottom-to-top swipe gestures on CPU usage and RAM usage.
Results:	The authors stated that cross-platform applications had substantial performance disadvantages compared to native applications, but further study is required.
Commentary:	The authors of this study provided a detailed step-by-step description of the execution of their benchmark test. This detailed description is valuable for reproducibility and comprehension. In addition, this study emphasised the performance impacts of cross-platform MDTs, and they still have a way to go before they can compete with native MDTs.

A comparative study of Java and Kotlin for Android mobile application development (2020) [51]	
MDTs:	Android Native
Overview:	This study compared the Java and Kotlin languages for Android mobile application development.
Method:	Benchmark applications were implemented to fetch and display information about a famous person. Compilations times, APK sizes, lines of code, number of classes required, ecosystem differences, and programming language constructs were compared.
Results:	The Java application was superior in compilation times and APK size, while Kotlin allowed more clean and concise code.
Commentary:	The aim of this study, unlike most other studies in this literature review, was to compare the performance impact of different programming languages on the same MDT. The performance metrics focused more on language performance instead of hardware-related performance like RAM usage and CPU usage.

An empirical investigation of performance overhead in cross-platform mobile development frameworks (2020) [16]	
MDTs:	Android Native, Ionic, React Native, NativeScript, MAML/ MD²
Overview:	This study investigated performance overhead in cross-platform mobile development technologies. They wanted to analyse popular cross-platform frameworks' ability to communicate with underlying native hardware features and APIs.
Method:	Benchmark applications that target specific native mobile features were implemented. These features were accelerometer, contacts, file system, and geolocation. The performance metrics they measured were execution time, RAM usage, and CPU usage. After they collected their data, they performed statistical analysis to identify the variances in technology performance and whether these variances were statistically significant.
Results:	In their results section, they ranked the frameworks in terms of performance. Overall, native Android had the best performance; however, there were cases in their study where the cross-platform applications outperformed the native application. An example of this would be NativeScript having better CPU usage and execution times compared to Native Android in specific benchmark tests.
Commentary:	This study could be considered the most thorough study in the literature review. The authors compared the performance of a wide range of MDTs across multiple MDAs. They also provided a valuable method to rank MDTs by performance after completing multiple benchmarking tests.

1.6.3 DISCUSSION

As evident from literature in Section 1.6.2, many studies have evaluated and compared mobile development technologies' performance; however, it is interesting that most studies aimed to compare and assess *approaches* rather than the *technologies* themselves. These studies start by selecting mobile development approaches for comparison and then selecting representative mobile development technologies. Even though Willocx *et al.* [32], [40] mentioned that MDTs of the same approach tend to perform similarly, there are still differences, and the results and conclusions of many of the studies discussed in Section 1.6.2 might have been dependent on the MDTs used and not the approaches as a whole.

Many mobile development technologies have been studied throughout the years. In Table 1-2, a summary of the MDT studies in literature can be seen. Android Native featured sixteen times,

and iOS Native featured eight times, respectively, likely due to studies comparing the performance of cross-platform approaches to the native approach. The most featured cross-platform MDTs were Titanium Appcelerator, PhoneGap, Xamarin, Ionic, and React Native.

Table 1-2 - MDTs studied in literature

MDT	#	Studies
Android Native	16	[16], [32], [46]–[51], [37]–[43], [45]
Titanium Appcelerator	9	[32], [37]–[39], [41], [43], [45]–[48]
iOS Native	8	[32], [39]–[41], [46], [47], [49]
Ionic	6	[16], [22], [27], [47], [49], [50]
PhoneGap	6	[21], [38]–[41], [46]
Xamarin	6	[40], [44], [46]–[49]
React Native	5	[16], [22], [27], [49], [50]
Cordova	4	[42], [45], [47], [48]
Corona	2	[47], [48]
MoSync	2	[21], [39]
NativeScript	2	[16], [47]
Adobe AIR	1	[39]
COMMON	1	[37]
Flutter	1	[16]
MAML/ MD²	1	[16]
Sencha Touch	1	[46]
WebWorks	1	[39]

Figure 1-7 indicates popularity trends of Titanium Appcelerator, Cordova, Flutter, and React Native from the popular developer community Stack Overflow[4]. Observe how Titanium Appcelerator, the most studied cross-platform MDT in literature, had very little historical representation in industry and has faded into obscurity by the time of this study. Newer MDTs such as Flutter and React Native have meanwhile risen to the top.

[4]https://insights.stackoverflow.com/trends

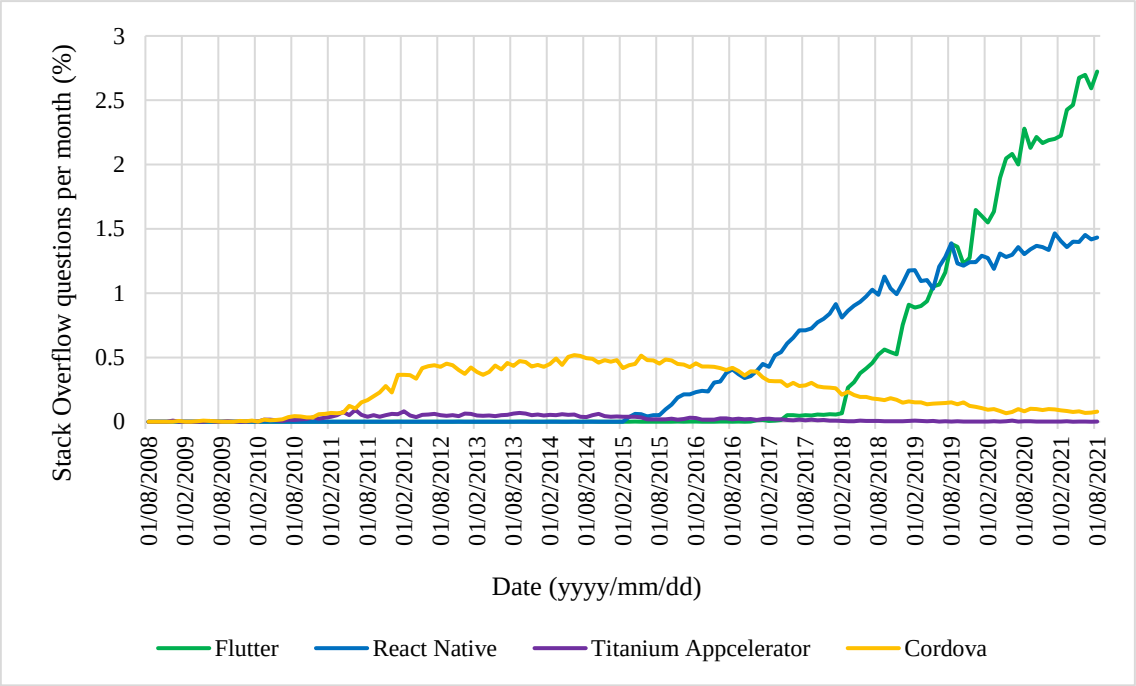


Figure 1-7- Popularity trends of different MDTs over time.

Compared to Titanium Appcelerator, especially considering their popularity, Flutter and React Native have been underrepresented in studies, emphasising the need for continuous research. By analysing Table 1-2 and Figure 1-7, it can be confirmed that Biørn-Hansen *et al.* [18] correctly suggested that the rapid evolution of MDTs are a challenge to mobile research.

Another observation that can be made from literature is the various performance metrics used to evaluate performance. For example, in Table 1-3, the different measured performance metrics to compare MDTs are reported.

Table 1-3 - Performance metrics studied in literature.

Performance metric	#	Studies
Time	15	[16], [22], [46]–[49], [51], [27], [32], [37], [39], [40], [42], [44], [45]
RAM usage	9	[16], [32], [37], [40], [42], [45], [46], [48], [50]
CPU usage	7	[16], [32], [40], [42], [48]–[50]
Size	6	[27], [37], [40], [45], [48], [51]
Battery usage	4	[21], [38], [41], [47]
Code	2	[37], [51]
Network	1	[42]
Temperature	1	[42]
FPS	1	[49]

Time (execution time, compile time, render time) was the most frequently observed performance metric in the studies obtained from literature. RAM usage, CPU usage, and size are also commonly measured in literature. Battery usage also featured frequently, but most battery usage studies were conducted by Ciman and Gaggi [21], [38], [41].

A problem is, therefore, observed from literature. The question formulated in Section 1.5.2, namely, *How can the performance of mobile development technologies be evaluated?* could not be answered by this literature review. The likely reason is that every study used a different method to assess performance.

The primary observation from this literature review is that there is no standard method or framework used to evaluate mobile development technologies' performance. This observation is in agreement with a conclusion reached by Rieger and Majchrzak [52]. The lack of a standardised method or framework limits reproducibility and leads to multiple studies in literature with incomparable results and findings [52].

After analysing the methods used in the performance-evaluation studies introduced in Section 1.6.2, a set of standard features shared by many studies could be identified. These features show what is required for a successful performance evaluation study.

When referring to Section 1.6.2, many studies focus on comparing cross-platform technologies and native technologies. Many studies started by first choosing or *selecting* the mobile development technologies for comparison, along with other tools and tests that were used to compare performance.

After selecting the mobile development technologies for comparison, they implemented benchmark applications based on the benchmark tests; however, as seen in Section 1.6.2, the methods of *implementing* benchmark applications differ amongst the studies.

The implemented benchmark applications are then used to gather performance data by *measuring* the performance impact of the benchmark tests. As with the implementation of benchmark applications, this focus point differs amongst studies.

After the data is gathered, it is *evaluated* and interpreted to determine which mobile development technology has the best performance. Some studies also made use of statistical analysis during the evaluation of results.

To summarise, the following points are essential to performance-evaluation studies:

- The selection of mobile development technologies and other tools and tests that will be used in the performance evaluation.
- The implementation of benchmark applications based on the tests selected.
- The measurement of performance-related data using the implemented benchmark applications.
- The evaluation of performance measurement results to determine which mobile development technology performed the best.

These standard features will be critical going forward, constituting what a performance evaluation is and what is considered essential by researchers. It is, however, in the interpretation of these standard features where studies differ, and these different interpretations lead to studies with limited reproducible results.

1.6.4 SUMMARY

In this section, a literature review was conducted. An overview of 19 performance-evaluation studies was given. It was observed that no common framework exists to evaluate the performance of mobile development technologies. The standard features shared amongst many performance-evaluation studies were also identified and discussed.

1.7 PROBLEM STATEMENT AND RESEARCH OBJECTIVES

Mobile applications have become invaluable tools in the modern world, and the demand for mobile application development is consequently high. There are multiple approaches to mobile development, and for each approach, there are numerous mobile development technologies available.

For mobile applications to be successful, developers need to ensure that they provide a quality user experience. Performance is mentioned as a critical factor influencing the quality of experience of mobile applications. Multiple studies have shown that there are performance differences in mobile applications created using different MDTs. A question was raised as to how the performance of mobile development technologies can be evaluated.

A literature study revealed that multiple studies have looked at the performance of mobile development technologies; however, a problem was identified. There is no standardised method or framework that can be used to carry out these performance-evaluation studies,

leading to multiple performance-evaluation studies with different techniques and limited reproducibility.

From the discussion above, a clear gap in the current mobile development research field can be identified. This gap can be formulated as a single statement below:

There is no framework that can be used to evaluate the performance of mobile development technologies. Failure to address this knowledge gap will lead to more mobile development technology studies with incomparable methods and results.

Having defined a clear knowledge gap in the knowledge spectrum, a problem statement can be formulated and is stated below:

Currently, in the MDT research field, multiple studies have been conducted focusing on performance; however, these studies use different methods of evaluation, leading to numerous incomparable studies. These incompatibilities are due to the lack of a standardised framework that can be used to evaluate mobile development technology performance.

From the problem statement above, a clear need for the study can be formulated and is stated below:

There is a need to develop a framework to evaluate the performance of mobile development technologies.

To address the need of the study, the following research objectives are formulated and stated below:

1. *Develop a framework to evaluate the performance of mobile development technologies.*
2. *Verify the developed framework by ensuring that it meets the requirements and specifications of an MDT performance evaluation.*
3. *Apply the framework by conducting performance-evaluation studies using popular modern mobile development technologies.*
4. *Validate that the developed framework addresses the research objectives.*

1.8 DISSERTATION OVERVIEW

Chapter 1: *Introduction*

This chapter provided a background to the mobile industry, mobile development approaches, and mobile development technologies. The need for performant mobile applications was also established. Following the background, a literature study was conducted on performance-evaluation studies. A gap in literature was identified, and the formulated problem statement emphasised the lack of a standardised performance evaluation framework. This chapter was concluded by listing objectives that should be met to address the problem.

Chapter 2: *Methodology*

In this chapter, the framework to evaluate mobile development technology performance is established. The framework comprises four phases, namely selection, implementation, measurement, and evaluation. The sub-phases of each phase are also discussed.

Chapter 3: *Results*

In this chapter, the developed framework is applied by evaluating the performance of five MDTs. These MDTs comprise Ionic, Flutter, React Native, Xamarin and Android Native. Four benchmark tests were selected, resulting in 20 benchmark applications that were used for comparison.

Chapter 4: *Conclusion*

This chapter concludes the study by giving a summarised overview of the study. Recommendations for further research are then provided. Finally, the chapter concludes by referring to the objectives formulated in Chapter 1 and how they were achieved.

CHAPTER 2

Methodology

2 METHODOLOGY

2.1 PREAMBLE

From the literature study in Chapter 1, it was established that numerous studies had been conducted evaluating and comparing the performance of mobile development technologies; however, these studies did not follow a standardised method or framework. Therefore, the problem statement in Chapter 1 emphasised the need to develop a generic framework to evaluate the performance of mobile development technologies.

In this chapter, a framework is developed to evaluate the performance of mobile development technologies. Furthermore, the framework aims to provide a structured generic method that can be followed to produce more comparable results for future performance-evaluation studies.

2.2 REQUIREMENTS AND SPECIFICATIONS

As discussed in Section 1.6.3, there are standard features that are shared amongst performance-evaluation studies. To ensure that the problem statement in Section 1.7 is addressed, these features must be defined as requirements and specifications that the framework should address. Table 2-1 shows the requirements and specifications of the framework based on the standard features identified in Chapter 1.

Table 2-1 - Requirements and specifications of the framework.

Requirement	**Specification**
Selection	The framework should include a selection phase. The necessary MDTs, benchmark tests, metrics, software, tools, and devices required for a performance evaluation should be given and discussed in the selection phase.
Implementation	The framework should include an implementation phase. The benchmark applications required for a performance evaluation should be discussed in the implementation phase.
Measurement	The framework should include a measurement phase. How performance metric data is measured should be discussed in the measurement phase.
Evaluation	The framework should include an evaluation phase. How the performance metrics results can be evaluated and used to rank the MDT included in the performance evaluation should be discussed in the evaluation phase.

As per Table 2-1, the framework will be divided into four phases, namely (1) selection, (2) implementation, (3) measurement, and (4) evaluation. The diagram in Figure 2-1 depicts the flow of the four phases.

Figure 2-1 – The four phases of the framework to evaluate the performance of mobile development technologies.

Each phase contains sub-phases. Each of these sub-phases will be discussed thoroughly throughout this chapter. The phases and sub-phases combined will form the framework to evaluate the performance of mobile development technologies.

2.3 SELECTION

2.3.1 OVERVIEW

The first phase of the proposed framework is selection. Figure 2-2 shows the sub-phases for the selection phase. The selection phase aims to select and document the necessities required for performance evaluation, including choosing the MDTs, performance metrics, plugins, testing devices, and profiling tools. Each sub-phase will be discussed in detail in the following sections.

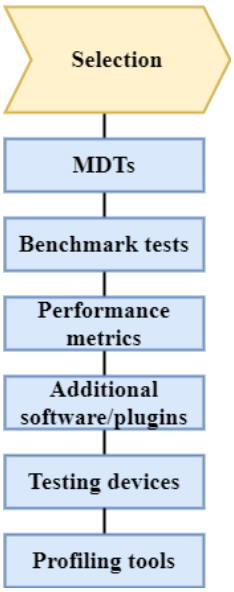


Figure 2-2- Sub-phases for the selection phase.

2.3.2 MDTs

The first sub-phase of selection is selecting the MDTs whose performance will be evaluated. The literature study discovered that many performance-evaluation studies focused on performance differences between mobile development approaches instead of comparing specific mobile development technologies. Even though this was the case, the studies still selected specific mobile development technologies to represent each approach and compared their performance directly.

For this framework, it does not matter how or why mobile development technologies are chosen. Researchers or developers can define their own MDT selection methodology. Some of the reasons for MDT selection in literature included popularity [46] and novelty [22], but most was due to MDA comparison. However, the selected mobile development technologies must be adequately documented for replication, specifying the MDTs, versions, and languages.

2.3.3 Benchmark tests

The second sub-phase of selection involves benchmark tests. During this phase, benchmark tests used to evaluate the selected MDTs in Section 2.3.2 are chosen and properly defined. It is essential to declare benchmark tests clearly to improve the reproducibility of performance-evaluation studies.

There were studies in literature where benchmark tests were not clearly defined. For each benchmark test, a benchmark application will be developed using each selected MDT. There is no limit to the number of benchmark tests that can be used. Benchmark tests in literature included reading data from mobile sensors [21], animations [49], and performing processing-intensive algorithms. UI rendering focussed benchmark tests are also a possibility. However, a benchmark test must consist of a repeatable sequence of steps with a clear beginning and end.

2.3.4 Performance metrics

The third sub-phase of selection involves specifying the performance metrics that will be measured. Performance metrics are also dependent on the type of evaluation study that is to be conducted. For example, if the aim is to study how cross-platform MDTs impact battery, consider energy consumption and CPU usage as performance metrics.

From literature, it is clear that the measurement of time is a highly valued performance metric. Of all the studies reviewed in the literature study, 84% measured some form of time. Other performance metrics frequently measured in literature included RAM usage and CPU usage. If animations and graphics performance are to be studied, metrics such as GPU usage and FPS should be considered.

2.3.5 ADDITIONAL SOFTWARE / PLUGINS

The fourth sub-phase of selection is selecting and specifying all software and plugins used in addition to the MDTs. Most mobile development technologies have plugins available that can be added to projects. These plugins are either officially supported by the MDT developers themselves or by a third party. Plugins provide specific functionality to mobile projects, particularly when accessing mobile hardware such as the camera or storage.

Plugin selection is often ignored in literature. Even comprehensive performance-evaluation studies, such as the one conducted by Biørn-Hansen *et al.* [16], fail to mention the plugins used. If researchers wanted to replicate the study, they would have to look at the source code provided to determine which plugins were used.

Plugins can add performance overhead to benchmark testing and thus it is important when choosing plugins, to prioritise first-party plugins. Sometimes, an MDT does not provide any first-party plugins to access sensors or other mobile features. In such cases, third-party plugins will have to be used. When third-party plugins are used, it is important to prioritise the highest rated or most popular third-party libraries to simulate how it would have been used in industry.

2.3.6 TESTING DEVICES

The fifth sub-phase of selection is device selection. Ideally, performance evaluations should be conducted on physical mobile devices. Most studies in literature made use of physical devices. Emulators can also be used, but it should be noted that there are subtle detectable differences between emulators and real physical devices [34].

In addition to physical devices, it is important to consider multiple devices across the hardware spectrum. MDTs may perform well on high-end devices, but poor on low-end devices. Thus in order to get a better general sense of MDT performance, multiple test devices should be considered.

2.3.7 PROFILING TOOLS

The sixth sub-phase of selection is profiling tool selection [16], [38], [41], [43]. Measuring performance metrics will require profiling tools, especially when measuring device-resource usage. Both iOS and Android provide profiling tools to measure RAM usage, CPU usage, and network usage. Time can also be determined through built-in consoles and logging [16].

For this framework, official profiling tools are recommended for these standard metrics; however, certain studies will require hardware measurement devices such as the Monsoon Power Monitor. Therefore, no matter which profiling or measuring tools are used, it is essential to specify which tools were used and how they were used. Using profiling tools can result in performance overhead, so it is important to make use of the same profiling tools consistently for all benchmark tests. Doing so will ensure that the performance overhead is consistent and present in all performance metric results so that no MDT gets an unfair advantage.

2.3.8 SUMMARY

The purpose of the selection phase is to select MDTs, benchmark tests, additional software or plugins, testing devices, and profiling tools used for performance evaluation. Doing so will make it easier to reproduce the performance evaluation and is the essential first step required for a performance evaluation.

2.4 IMPLEMENTATION

2.4.1 OVERVIEW

The second phase of the framework is implementation. Implementation involves creating benchmark applications using the selected benchmark tests from the previous phase. Figure 2-3 illustrates the implementation phase of the proposed framework.

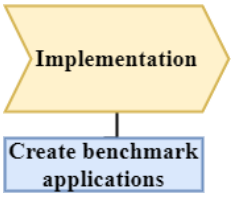


Figure 2-3 - Sub-phases for the implementation phase.

2.4.2 CREATION OF BENCHMARK APPLICATIONS

The first sub-phase of implementation is to create benchmark applications using the selected MDTs. For example, in the literature study, it was discovered that some performance-evaluation studies made use of fully featured mobile applications for benchmarking. An example of this would be the study by [37], where the authors created a complete shopping application using multiple technologies.

This approach to benchmarking applications is unnecessarily complicated and time-consuming. For example, if a developer or researcher wanted to compare the performance of five mobile development technologies, they would have to implement a fully featured mobile application using each technology. This approach may, therefore, be more viable if a project such as PropertyCross can be used [40].

Other studies used more sustainable approaches to benchmarking. An example of this would be the study conducted by Biørn-Hansen *et al*. [16]. First, they created benchmark applications using their selected MDTs. Then, within the benchmark application, they created separate user interfaces (or *pages*) for each specified task. A possible problem with this approach is the performance impact of loading additional pages and UI elements.

For the proposed framework, it is recommended that a single benchmark application be used to execute a single benchmark test. In Figure 2-4, the user interface of a benchmark application can be seen. The simplicity of the UI will allow for rapid development of benchmark applications using each MDT and limit the potential effect of UI elements on performance. Many MDTs provide so-called *starter templates*, with an entry page and a few UI elements, which can quickly be repurposed for benchmarking.

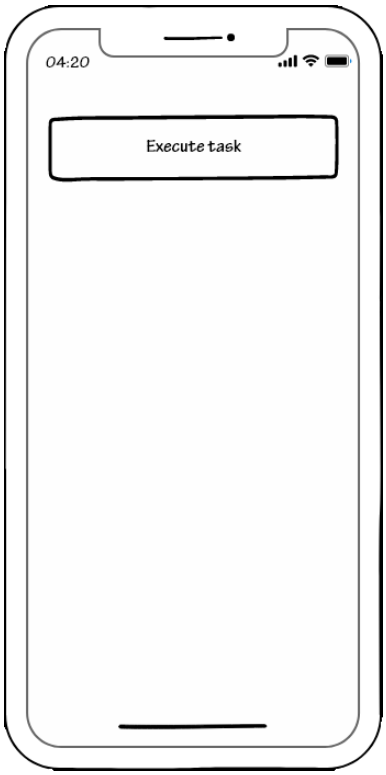


Figure 2-4 - The user interface of a proposed benchmark application.

The user will be presented with an *execute benchmark test* button. Upon pressing the button, the benchmark test is performed. After the benchmark applications are implemented, they should be compiled into installable packages for the test devices (APKs for Android test devices and IPAs for iOS). These installable packages should then be transferred to the testing device(s). The packages will be installed during the measurement phase.

2.4.3 SUMMARY

The implementation phase focuses on the creation of benchmark applications. A simplistic UI with only one *execute task* button is proposed for the implementation of benchmark applications.

2.5 MEASUREMENT

2.5.1 OVERVIEW

The third phase of the proposed framework is measurement. In this phase, the impact of the benchmark tests on the specified performance metrics can be measured. Figure 2-3 illustrates the implementation phase of the proposed framework.

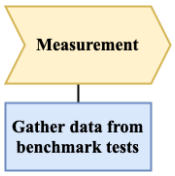


Figure 2-5 - Sub-phases for the measurement phase.

2.5.2 GATHER DATA FROM BENCHMARK TESTS

This first and only sub-phase of measurement is gathering performance metric data while executing the benchmark tests [16], [39]. A detailed diagram of the measurement process can be seen in Figure 2-6. First, a benchmark application was created using one of the specified MDTs. The installable packages (APKs and IPAs) compiled during the previous phase will be needed for this. After a benchmark application is installed, the selected profiling tools should be attached. The benchmark applications should then be opened. If the UI were implemented as recommended in Figure 2-4, a button would be displayed.

The benchmark test will be executed upon the pressing of the button. While the benchmark test is being performed, the impact on performance metrics should be measured and recorded using the profiling tool(s). After the benchmark test is completed, the profiling tools should be detached. Thus, the act of attaching profiling tools, opening a benchmark application, executing the benchmark test, extracting the performance metric data, closing the benchmark application and detaching profiling tools can be referred to as a measurement cycle.

Continue conducting measurement cycles until the number specified in the selection phase is reached. After the total measurement cycles with a benchmark application made using one MDT is done, the next benchmark application created using another MDT should be installed, and the measurement cycles should be repeated. Finally, this sub-phase is complete after completing all the measurement cycles using all the benchmark applications for all the selected MDTs.

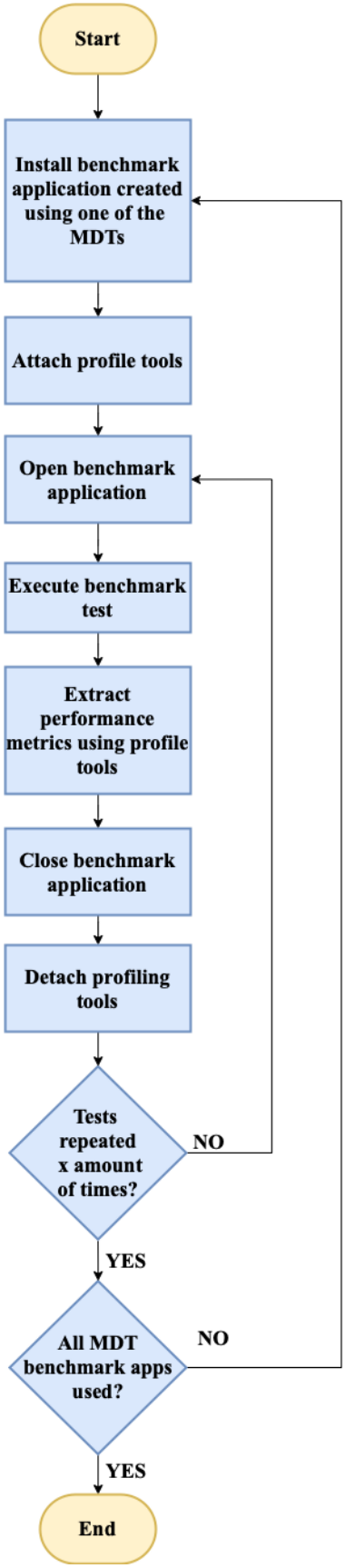


Figure 2-6 - Measurement process of performance metrics.

2.5.3 SUMMARY

The measurement phase focuses on the execution of the benchmark tests and gathering performance metric data. A measurement process diagram is provided and discussed in detail. After completion of the measurement phase, all performance-related data should be available for evaluation and comparison.

2.6 EVALUATION

2.6.1 OVERVIEW

The fourth phase of the proposed framework is evaluation. After measuring the performance impact of the benchmark tests, the results can be analysed and the MDTs compared. Figure 2-7 illustrates the sub-phases of the data-analysis phase. Firstly, descriptive statistics will be calculated using the results obtained from the measurement phase. Secondly, Analysis of Variance (ANOVA) tests will be conducted to analyse the difference in means among the performance of the selected MDTs. Finally, for each benchmark test, the MDTs will be ranked by their performance results.

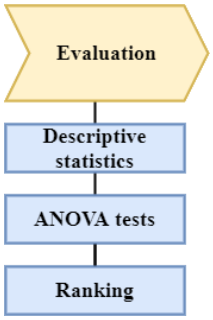


Figure 2-7 – Sub-phases for the evaluation phase.

2.6.2 DESCRIPTIVE STATISTICS

Descriptive statistics that will be calculated for this proposed framework include (1) mean, (2) standard deviation, (3) maximum value, and (4) minimum value. These descriptive statistics can provide an overview of the measured performance results and insight into the differences in performance metric data.

Mean

The mean or *average* of performance metric data can be calculated using Equation (Equation 2-1.

$$\bar{x} = \frac{\sum x}{n} \qquad \textit{(Equation 2-1)}$$

Here,

$\bar{x}$ = sample mean,

$\sum x$ = sum of each value in the sample, and

n = number of values in the sample.

A mean value can be used to measure central tendency [53]. The literature study shows that the mean value descriptive statistic has been used in many performance-evaluation studies [16], [27], [39], [44], [46], [47], [50]. The mean will be calculated based on the number of benchmark tests performed. For example, if a performance metric is RAM usage and 25 benchmark tests were performed, the mean or *average* RAM usage can be calculated by summing the RAM usage of the 25 benchmark tests and dividing by 25.

Standard deviation

The standard deviation of a data sample is the average amount of variability in that sample [54]. In other words, standard deviation shows on average how far each value lies from the sample's mean. A higher standard deviation value will indicate that the performance values are spread far from the mean value. A lower standard deviation value will indicate that performance values are clustered closer to the mean value. Equation 2-2 shows how the standard deviation can be calculated.

$$s = \sqrt{\frac{\sum(X - \bar{x})^2}{n-1}} \qquad \textit{(Equation 2-2)}$$

Here,

s = sample standard deviation,

X = each value in the sample,

$\bar{x}$ = sample mean, and

n = number of values in the sample.

Minimum and maximum values

The minimum and maximum values refer to the lowest and highest values in the data sample. Minimum and maximum values are typically used to determine the range of the sample, which gives an overview of how far apart extreme values are from each other. The range can also be used as an indicator of variability; however, this is only the case when dealing with a data sample without outliers.

2.6.3 ANOVA TESTS

Analysis of Variance (ANOVA) tests can be used to analyse the difference in mean values of more than two MDTs. If the performance of only two MDTs is compared, then t-tests should be used instead of ANOVA tests. Performing an ANOVA test is helpful for MDT performance evaluations because it reveals whether the performances of various MDTs differ from each other.

The null hypothesis (H_0) of the ANOVA test states that there is no difference in the mean value for the selected performance metrics. The alternative hypothesis (H_a) is that at least one of the MDTs will differ significantly from the overall mean values. There are multiple software packages available that can perform one-way ANOVA tests, with many providing summaries that are useful for interpreting the results.

After performing the ANOVA test, some post-hoc testing will be required. The ANOVA test will reveal differences in mean values, but it will not show whether these differences are *significant*. Determining whether there is a statistically significant difference in performance among MDTs will provide helpful insight. That is why for this proposed framework, Tukey's Honestly-Significant Difference (Tukey HSD) post-hoc test is recommended and will be used [16].

The Tukey test runs pairwise comparisons between the mean performance metric values of the selected MDTs. A conservative error estimate is used to find MDTs that are statistically different from one another. An example of a Tukey test can be seen in Table 2-2. A p-value < 0.05 allows us to reject the null hypothesis. A rejection of the null hypothesis means that there are *significant* pairwise differences between the two MDTs. In Table 2-2 there are significant pairwise differences between MDT 1 and MDT 2 and between MDT 1 and MDT 3. There is no significant difference between MDT 2 and MDT 3.

Table 2-2 - Example of a Tukey test

Group 1	Group 2	p-value	Reject (H_0)
MDT 1	**MDT 2**	0.001	True
MDT 1	**MDT 3**	0.001	True
MDT 2	**MDT 3**	0.7846	False

2.6.4 RANKING

A ranking of MDTs by their performance is often not done in literature, especially when looking at multiple performance metrics. Instead, many studies only list the performance results and then proceed to discuss and analyse them [32], [37], [40], [41]. Other studies, such as the one conducted by Dhillon and Mahmoud [39], ranked overall MDT performance by how many times it performed the best for each benchmark test.

The only study from literature that provided a method to rank the MDTs by performance was the one conducted by Biørn-Hansen *et al.* [16]. Their approach allows for MDTs to be ranked by their performance results, even when multiple performance metrics are measured. Using this ranking method will allow for the *best overall performing MDT* per benchmark test to be identified.

For this proposed framework, the method used by Biørn-Hansen *et al*. [16] will be used and adapted to rank the performance of MDTs. For each measured performance metric, the MDTs will be scored according to their ranking for that metric. The best performing MDT will be assigned a score of 1 for the given metric. The second-best performing MDT will receive a score of 2 for the given metric and so forth. The scores will then be added up. The MDT with the lowest total score will be considered the *best overall performing MDT* for that specific benchmark test. The MDT with the highest total score will be considered the *worst overall performing MDT* for that specific benchmark test.

In the example benchmark test ranking as can be seen in Table 2-3, MDT 1 performed the best for the given performance metrics. Thus, a score of 1 point is assigned to each performance metric category for MDT 1. MDT 2 scored the second-best across each performance metric and received a score of 2 for each category. Lastly, MDT 3 had the worst performance across all performance metrics and received a score of 3 for each metric.

Table 2-3- An example of MDT ranking for an example benchmark test (lower is better).

MDT	**Metric 1**	**Metric 2**	**Metric 3**	∑
MDT 1	1	1	1	3
MDT 2	2	2	2	6
MDT 3	3	3	3	9

When adding up the scores, MDT 1 had a total score of 3, MDT 2 had a total score of 6, and MDT 3 had a total score of 9. Thus, for this benchmark test example, MDT 1 was the *best overall performing* MDT and MDT 3 was the *worst overall performing* MDT.

This framework does not provide a method to assign weights to performance metrics and assumes that each metric has the same weight assigned to them. However, if a user of this framework has independently calculated weights to assign to their performance metrics, they can multiply the scores of each performance metric with the weights, to get the weighted score as seen in Equation 2-3.

$$\boldsymbol{weighted\ score} = (\boldsymbol{score})(\boldsymbol{weight}) \qquad \textit{(Equation 2-3)}$$

2.6.5 SUMMARY

The first sub-phase involves determining descriptive statistics. The descriptive statistics include calculating mean values, standard deviation, and maximum and minimum values. The second sub-phase involves performing ANOVA and post-hoc Tukey tests to determine differences in mean values among the specified MDTs, and whether these differences are significant. Finally, in the third sub-phase, the MDTs are ranked using a method obtained from literature. This ranking method can be used to determine the best overall performing MDT as well as the worst one for a specific benchmark test.

2.7 VERIFICATION

2.7.1 OVERVIEW

This section verifies that the developed framework fulfils the requirements of the study and that the specifications set out in Section 2.2 are met. In addition, verification of the developed

framework ensures that the problem statement defined in Section 1.7 is addressed when applied.

2.7.2 SELECTION

The developed framework features a selection phase as discussed in Section 2.3. The selection phase provides a method to specify the aspects required to conduct a performance evaluation study based on techniques used in literature. For example, the selection phase involves selecting MDTs, benchmark tests, performance metrics, plugins, test devices, and profiling tools. The inclusion of a selection phase adapted from literature verifies that this requirement is met.

2.7.3 IMPLEMENTATION

The developed framework features an implementation phase as discussed in Section 2.4. The implementation phase provides a method to create benchmark applications based on the benchmark tests specified during the selection phase. Benchmark application implementation methods were compared, and a single benchmark test per MDT approach was suggested.

2.7.4 MEASUREMENT

The developed framework features a measurement phase as discussed in Section 2.5. The measurement phase provides a method to measure performance metrics data from the benchmark application created during the implementation phase. In addition, a measurement flow diagram is provided as seen in Figure 2-6.

2.7.5 EVALUATION

The developed framework features a measurement phase as discussed in Section 2.6. The evaluation phase provides a method to evaluate the results obtained during the measurement phase. The evaluation method offers statistical methods to analyse significant differences between MDTs and a ranking method adapted from [16].

2.7.6 SUMMARY

Table 2-4 summarises whether the requirements and specifications identified in Section 2.2 have been met. The meeting of these requirements (using methods adapted and obtained from literature) verifies that the framework is adequately developed, and the methods used are correct.

Table 2-4 - Verification of framework summary.

Requirement	Framework reference	Requirement met
Selection	2.3.2, 2.3.3, 2.3.4, 2.3.5, 2.3.6, 2.3.7	✓
Implementation	2.4.2	✓
Measurement	2.5.2	✓
Evaluation	2.6.2, 2.6.3, 2.6.4	✓

2.8 SUMMARY

This chapter proposed a generic framework to assess MDTs' performance when executing specified tasks. The proposed framework consists out of four phases: (1) selection, (2) implementation, (3) measurement, and (4) evaluation. Each phase comprises one or more sub-phases that were discussed in detail throughout this chapter.

In the first phase of the proposed framework (Section 2.3), the MDTs, plugins, hardware devices, profiling tools, and tasks are selected and specified. In the second phase (Section 2.4), benchmark applications are implemented using the selected MDTs. A simple UI is recommended with a single button displayed on the device screen. Pressing the button executes the benchmark test.

The third phase (Section 2.5) involves measuring MDT performance and gathering performance metric data. Finally, the fourth phase involves analysing the results obtained during the measurement phase. Descriptive statistics, ANOVA tests, and post-hoc Tukey tests are used to evaluate the MDTs. A method to rank the MDTs based on the performance metrics is also provided. The developed framework is shown in Figure 2-8.

This chapter concludes with a section discussing verification and whether the requirements and specifications defined in Table 2-1 were addressed. Table 2-4 confirms that the requirements were addressed and thus verified that the framework was developed correctly.

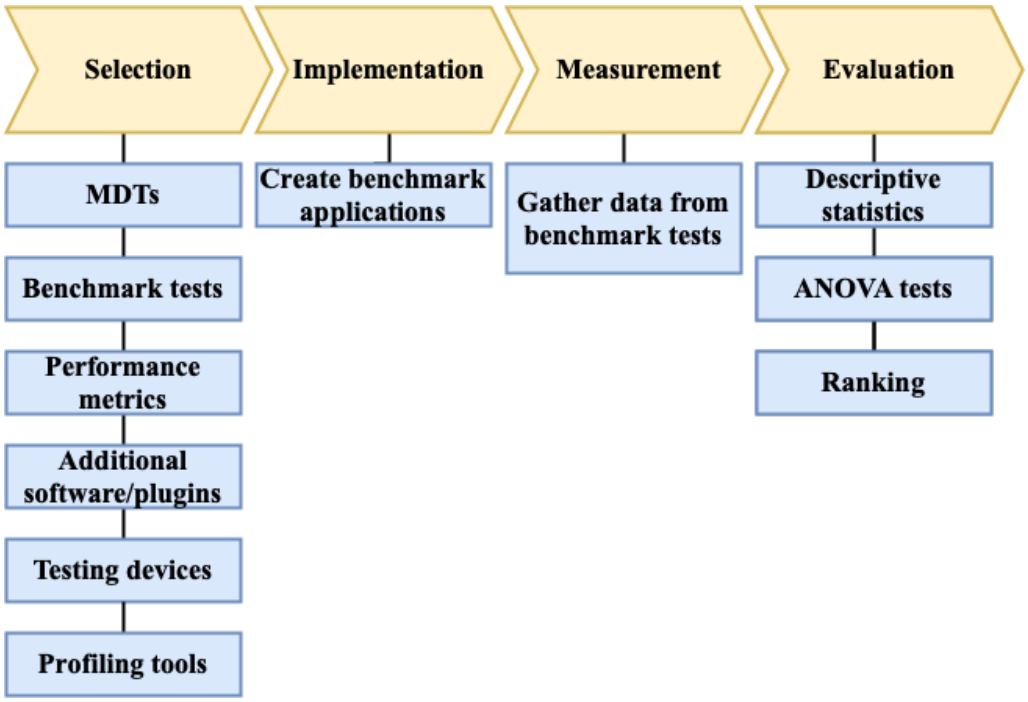
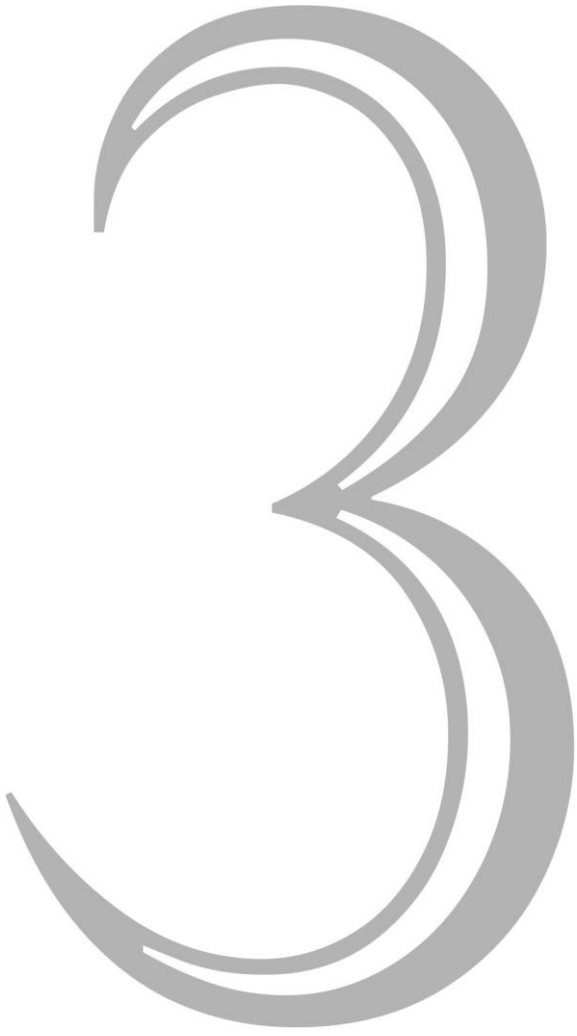


Figure 2-8- The proposed framework to evaluate the performance of mobile development technologies.

CHAPTER 3

Results



3 RESULTS

3.1 PREAMBLE

In Chapter 2, a framework (Figure 3-1) was developed to evaluate the performance of mobile development technologies. The framework comprises four phases and multiple sub-phases to achieve this goal.

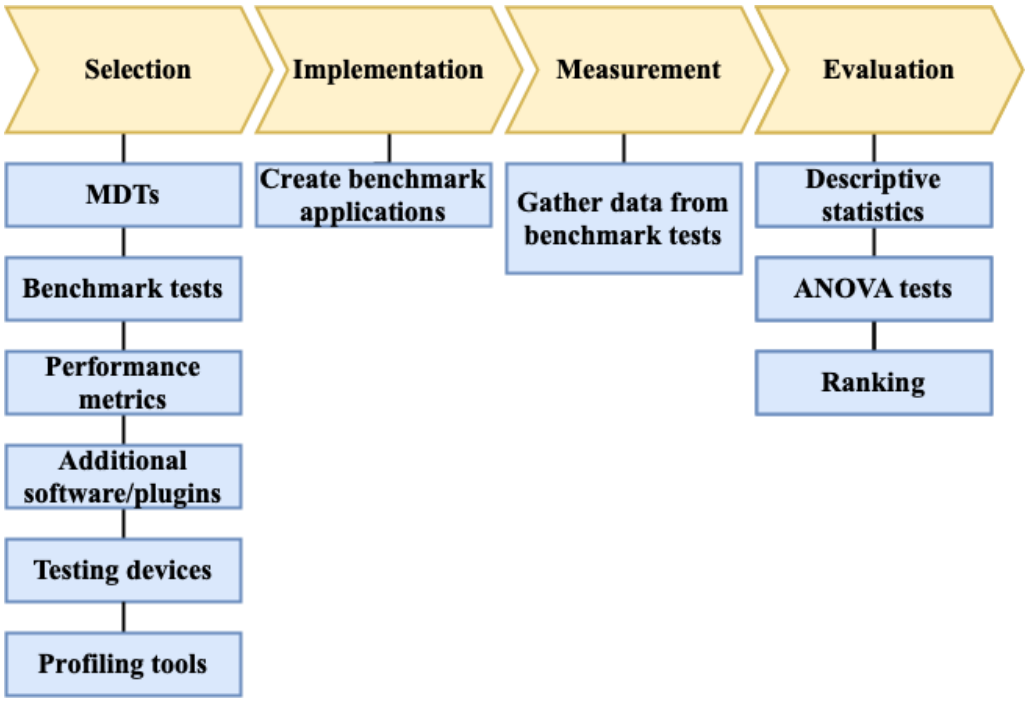


Figure 3-1 – A framework to evaluate the performance of mobile development technologies.

During the *selection* phase, the mobile development technologies, benchmark tests, performance metrics, plugins, test devices, and profiling tools are specified and recorded. Benchmark applications are created using each of the selected mobile development technologies in the *implementation* phase. Performance metric data is gathered during the *measurement* phase and is analysed and used to rank the selected MDTs by performance during the *evaluation* phase. In this chapter, the developed framework will be applied by evaluating five popular modern mobile development technologies.

3.2 SELECTION

3.2.1 OVERVIEW

In this section, the selection phase of the proposed framework will be applied. The selection phase includes discussing the selected (1) MDTs, (2) benchmark tests, (3) performance metrics, (4) plugins, (5) test devices, and (6) profiling tools.

3.2.2 MDTS

The mobile development technologies used for the performance evaluation can be seen in Table 3-1. Unlike many performance-evaluation studies, these MDTs were not chosen due to their approach but due to their popularity in the developer community.

Table 3-1 - Selected MDTs

MDT	Approach	Language	Version
Ionic	Hybrid	TypeScript, HTML, CSS	Ionic CLI 6.12.3
Flutter	Cross-compiled	Dart	Flutter 1.22.5
React Native	Interpreted	JavaScript	React Native 0.63.4
Xamarin	Cross-compiled	C#	Xamarin 16.8.000.261
Android Native	Native	Java	Android API 29

Ionic Framework[5] is a hybrid cross-platform mobile development technology that utilises standard web technologies such as TypeScript, HTML, and CSS to create mobile applications. In addition, ionic allows for popular front-end web technologies such as Angular, Vue, and React to be used. Ionic with Capacitor (providing access to native features as discussed in Section 1.3.3) and Angular as the front-end technology will be used for this performance evaluation.

Flutter[6] is a cross-compiled, cross-platform mobile development technology created by Google. Using the Dart language, Flutter allows for building natively compiled mobile applications for multiple platforms. Flutter was specifically chosen due to its popularity in the mobile development community and lack of presence in literature.

[5] https://ionicframework.com/
[6] https://flutter.dev/

React Native[7] is an interpreted cross-platform mobile development technology created by Facebook. React Native allows developers to develop mobile applications using JavaScript, with UI elements natively rendered during runtime. React Native, as with Flutter, is also a prevalent development technology for mobile applications.

Xamarin[8] is another cross-compiled cross-platform mobile development technology. Supported by Microsoft, Xamarin extends their .NET platform with tools and libraries that can be used to create mobile applications with C#.

Android Native[9] refers to native mobile applications created exclusively for the Android platform using Android Studio. Most performance-evaluation studies included native MDTs due to native applications often being regarded as the most performant method to create mobile applications. Thus, for this performance evaluation, Android Native will also be evaluated.

The popularity trends[10] of the selected MDTs can be seen in Figure 3-2. Observe how Android Native has always been the most popular MDT but has recently started trending downwards. Conversely, the two popular cross-platform MDTs, Flutter and React Native, are still trending upwards. Finally, Xamarin and Ionic are the least popular of the selected MDTs.

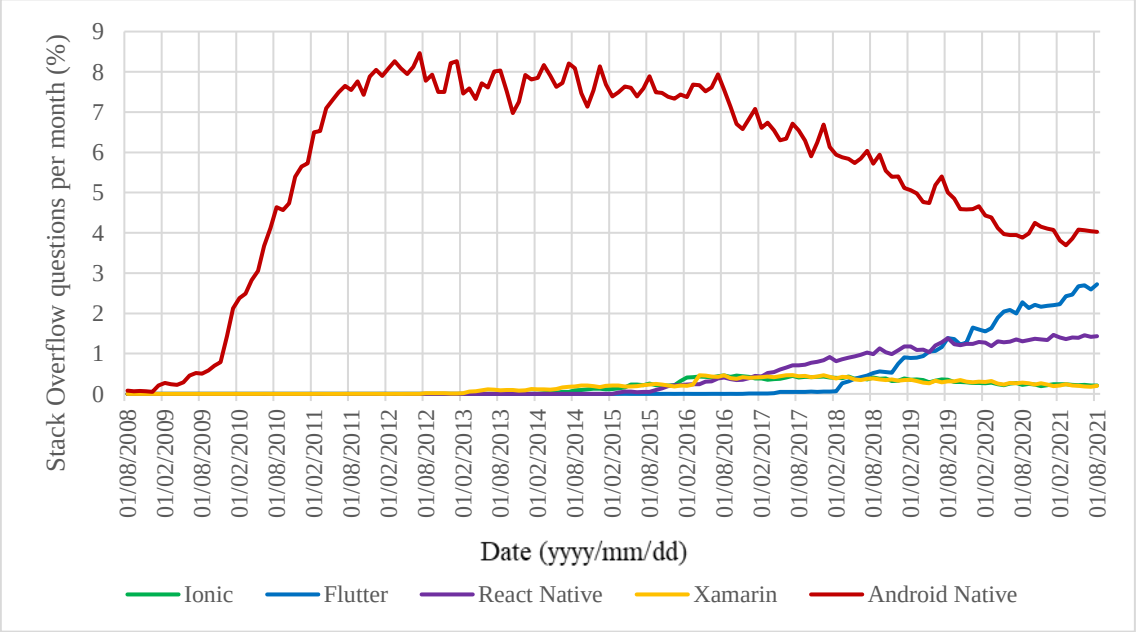


Figure 3-2 - Popularity trends of the selected MDTs.

[7] https://reactnative.dev/
[8] https://dotnet.microsoft.com/apps/xamarin
[9] https://developer.android.com/
[10] https://insights.stackoverflow.com/trends

When observing Figure 3-2, one might think that Ionic and Xamarin are poor choices since they are so unpopular when compared to Flutter, React Native and Android Native. However, as can be seen in Table 3-2, Native Android, Flutter, React Native, Ionic and Xamarin are the five most popular MDTs according to Stack Overflow. Notice how unpopular Titanium Appcelerator is compared to the top five MDTs when it comes to industry popularity.

Table 3-2 – MDTs in literature ranked by industry popularity

MDTs in literature	Stack Overflow questions in January 2021 (%)
Native Android	3.77%
Flutter	2.23%
React Native	1.35 %
Ionic	0.22 %
Xamarin	0.21 %
Cordova	0.08 %
NativeScript	0.04 %
Titanium Appcelerator	0.00 %
PhoneGap	0.00 %
Corona	0.00 %
MoSync	0.00 %
Adobe AIR	0.00 %
COMMON	0.00 %
MAML/ MD²	0.00 %
Sencha Touch	0.00 %

3.2.3 BENCHMARK TESTS

A brief overview of the benchmark tests used to evaluate the selected MDTs (from Section 3.2.2) can be seen in Table 3-3. These tests will be used to create benchmark applications using each MDT. In addition, the benchmark tests will focus on accessing and collecting data from native mobile features and sensors for this performance evaluation. The benchmark tests are discussed in further detail in the *implementation* section (Section 3.3).

Table 3-3 – Selected benchmark tests

Benchmark tests	Description
Accelerometer	Read acceleration from the device accelerometer
Contacts	Read the first names and surnames of contacts stored on the device
Filesystem	Read/write a text file from/to the device filesystem
Geolocation	Read geolocation data from the device GPS

3.2.4 PERFORMANCE METRICS

A brief overview of the performance metrics that will be measured to evaluate the performance of the selected MDTs can be seen in Table 3-4. CPU usage, RAM usage, and execution time were some of the most frequently used performance metrics found in literature as can be seen from Table 1-3. For that reason, they are chosen for this performance evaluation as well. Furthermore, a combination of these metrics should allow for an insightful comparison of selected MDTs' performance. Battery usage was the fifth most studied metric in literature according to Table 1-3 and is an important metric used in industry. However, since battery usage is correlated to CPU usage, it has not been chosen as a metric in this study.

Table 3-4 - Selected performance metrics that will be measured

Metric	Description
CPU usage	Measurement of how much of the CPU is utilised during benchmark-test execution
RAM usage	Measurement of how much of the RAM is utilised during benchmark-test execution
Execution time	Measurement of how long it took to complete the benchmark test

The performance metrics and precisely how they are measured will be discussed in further detail in the *measurement* section (Section 3.4).

3.2.5 ADDITIONAL SOFTWARE / PLUGINS

As mentioned in Section 2.3.5, many MDTs' functionalities can be extended using plugins; therefore, it is essential to specify any plugins or additional software used to perform the benchmark tests. Table 3-5 shows the extra software/plugins for this performance evaluation.

Table 3-5 - Selected additional software/plugins for performance evaluation

MDT	Benchmark test	Plugin	Third-party	Version
Ionic	Accelerometer	Capacitor Motion	×	2.4.5
	Contacts	Capacitor Community Contacts	✓	1.0.9
	Filesystem	Capacitor Filesystem	×	2.4.5
	Geolocation	Capacitor Geolocation	×	2.4.5
Flutter	Accelerometer	pub.dev sensors	×	0.4.2
	Contacts	pub.dev contacts_service	✓	0.4.6
	Filesystem	pub.dev path_provider	×	0.4.1
	Geolocation	pub.dev geolocator	✓	6.1.13
React Native	Accelerometer	Expo sensors	✓	10.0.0
	Contacts	react-native-contacts	✓	6.0.3
	Filesystem	react-native-fs	✓	2.16.6
	Geolocation	Expo location	✓	11.0.0
Xamarin	Accelerometer	Xamarin.Essentials: Accelerometer	×	1.5.3.2
	Contacts	Xamarin.Essentials: Contacts	×	1.5.3.2
	Filesystem	Xamarin.Essentials: File System	×	1.5.3.2
	Geolocation	Xamarin.Essentials: Geolocation	×	1.5.3.2

Note that Android Native did not require any plugins to develop the benchmark tests and were thus omitted from Table 3-5.

3.2.6 TEST DEVICES

As mentioned in Section 2.3.6, especially when evaluating MDT performance for large scale industrial projects, it is important to consider multiple test devices across the hardware spectrum. Multiple devices will give a better sense of general MDT performance.

However, for the purposes of this study, the usage of a single test device is sufficient for validating the developed framework. The selected test device used for benchmark testing can be seen in Table 3-6.

Table 3-6 - Selected test device

Device	***CPU***	***RAM***	***OS***
Nokia 7 Plus	Qualcomm Snapdragon 660 2.20 GHz	4GB	Android 10

3.2.7 PROFILING TOOLS

The selected profiling tool that will be used for measurement can be seen in Table 3-7. Android Studio provides a profiling tool called the Android Profiler. Among other performance metrics, CPU usage, RAM usage, and execution time can be accessed through the built-in console.

Table 3-7 - The selected profiling tool

Profiler	**Version**
Android Profiler (Android Studio)	4.1.1

3.2.8 SUMMARY

In this section, the *selection* phase of the proposed framework was applied. Five popular mobile development technologies were selected. In addition, benchmark tests were chosen, and the performance metrics will be measured to evaluate the performance of the mobile development technologies. The plugins for each benchmark test alongside the test device and profiling tools were also specified.

3.3 IMPLEMENTATION

3.3.1 OVERVIEW

In this section, the *implementation* phase of the proposed is applied. The *implementation* phase includes using the selected MDTs (Section 3.2.2) to create benchmark applications based on the chosen benchmark tests (Section 3.2.3). This section will also discuss the benchmark tests in further detail.

3.3.2 THE CREATION OF BENCHMARK APPLICATIONS

For each benchmark test, a benchmark application was created using the selected MDTs. The benchmark applications featured simple UI designs, with a single button being displayed upon launch. Upon pressing the button, the benchmark test begins. First, a background timer would start, then, upon completion of the benchmark test, the background timer is stopped, and the execution time is printed to a console.

It is important to note that the usage of a background timer can result in performance overhead. Thus when implementing the background timers, great care was taken to start and stop them

consistently across all MDTs and benchmark tests. Doing so will reduce the effects of the performance overhead so that no MDT gains an unfair performance of another MDT due to the effects of the timers.

Accelerometer

The accelerometer test involves retrieving acceleration data (including the effects of gravity) from the test-device accelerometer sensor. Then, using the selected MDTs and plugins, a listener was added to listen for accelerometer events. Once the first accelerometer data object was returned, it was assigned, and the benchmark test ended.

Contacts

The contacts test involves retrieving the first names and surnames of 118 contacts stored on the test device. Then, using the selected MDTs and plugins, a list of all device contacts was requested. After the list of contacts was received, the first name and surname were assigned, and the benchmark test ended.

Filesystem

The filesystem test involved writing a 10 KB text file to the file system of the test device. A text file was created using the MDTs and plugins, and 10 KB of data was written. After the writing of the text file was completed, its contents were immediately read again and assigned. After the assignment of the text file content, the benchmark test ended.

Geolocation

The geolocation test involved retrieving the location of the test device using the device GPS sensor. Then, using the MDTs and plugins, the GPS coordinates of the device were requested. Upon retrieval of the GPS coordinates data, the data was assigned, and the benchmark test ended.

3.3.3 SUMMARY

In this section, the *implementation* phase of the proposed framework was applied. Twenty benchmark applications were created using the selected MDTs to execute the chosen benchmark tests.

3.4 MEASUREMENT

3.4.1 OVERVIEW

In this section, the *measurement* phase of the proposed framework will be applied. The *measurement* phase includes measuring the CPU usage and RAM usage during benchmark-test execution and how long the test took to execute. The specifics as to how these metrics were measured will also be discussed in more detail.

3.4.2 CPU USAGE

As seen in Table 1-3, CPU usage is a prevalent metric in literature. For this reason, CPU usage was chosen as a performance metric for this application of the proposed framework. Peak CPU usage as a percentage will be measured using the selected profiling tool, Android Profiler. Figure 3-3 illustrates how peak CPU usage was measured using the profiling tool. After starting the benchmark test, the highest peak CPU usage value as a percentage is recorded.

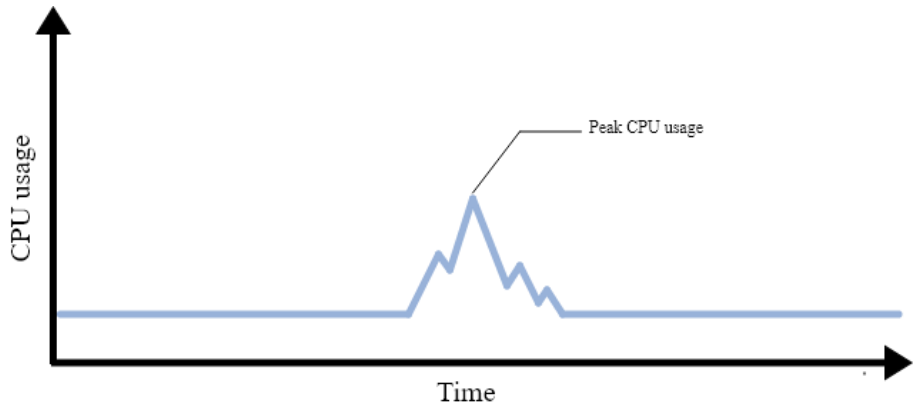


Figure 3-3 – Measuring peak CPU usage.

3.4.3 RAM USAGE

As seen in Table 1-3, RAM usage is another performance metric commonly used in performance-evaluation studies, and for this reason, it was also selected as a performance metric. For this performance evaluation, RAM usage will be measured in a similar way used by Biørn-Hansen [16]. Figure 3-4 shows how RAM usage was measured.

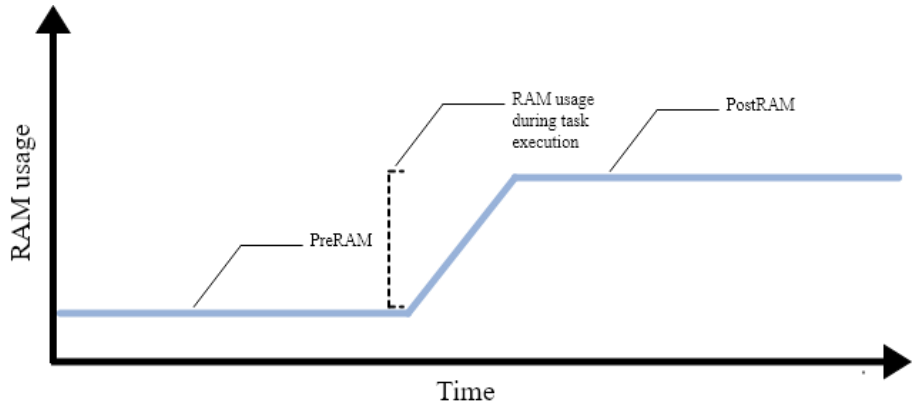


Figure 3-4 - Measuring peak RAM usage.

The RAM usage before (PreRAM) and after (PostRAM) benchmark-test execution is measured. RAM usage can be calculated using Equation (*Equation 3-1* below.

$$\boldsymbol{RAM\ usage = PostRAM\ usage - PreRAM\ usage} \quad \textit{(Equation 3-1)}$$

Here,

$\boldsymbol{RAM\ usage}$	-	*RAM usage used to execute the task,*
$\boldsymbol{PostRAM\ usage}$	-	*RAM usage after task execution, and*
$\boldsymbol{PreRAM\ usage}$	-	*RAM usage before task execution.*

3.4.4 EXECUTION TIME

As seen in Table 1-3, the most measured performance metric found in literature is execution time. For this performance evaluation, the execution time of the benchmark tests will be measured using built-in timer functionality provided by the programming language of the mobile development technology. The execution time will be printed to the console using the built-in language features and extracted using the profiling tool during task execution.

3.4.5 MEASUREMENT PROCESS

The measurement process, as seen in Figure 2-6 was used to gather the measurement data. Each benchmark application was installed separately on the test device, and the associated benchmark test was run 20 times. After each benchmark test, the CPU usage, RAM usage, and execution time was extracted from the profiling tool connected to the test device. This process was repeated until all data was collected.

3.4.6 SUMMARY

In this section, the *measurement* phase of the approached framework was applied. CPU usage, RAM usage, and execution times were recorded for benchmark applications created using the implementation phase of the proposed framework.

3.5 EVALUATION

3.5.1 OVERVIEW

In this section, the *evaluation* phase will be applied. The evaluation phase involves analysing the performance metric data and discussing descriptive statistics and the ranking of the mobile development technologies. The performance results of each task will be addressed separately. The results of the Tukey tests can be seen in the Appendix.

3.5.2 ACCELEROMETER

CPU usage

Table 3-8 contains the CPU usage results of the accelerometer benchmark test. Ionic had the most CPU usage, with a mean value of 12.7% utilisation. Flutter required less CPU usage to complete the task, with a mean value of 5.4% utilisation. Note that Flutter performed better than Android Native, with a mean value of 7% CPU utilisation. An analysis of the standard deviation and minimum and maximum values shows that React Native has more variability in CPU usage.

Table 3-8 - Accelerometer: CPU usage (%)

MDT	Mean	Std	Min	Max
Ionic	12.7	1.6	9	15
Flutter	5.4	1.4	3	10
React Native	9	3.8	6	16
Xamarin	10	2.9	6	14
Android Native	7	2	3	11

Conducting an ANOVA test revealed statistically significant differences in accelerometer CPU usage for different MDTs ($F\text{-value} = 23.9$, $p\text{-value} < 0.001$). A Tukey post-hoc test showed significant pair-wise differences between all the MDTs, except between Android Native and React Native and between Xamarin and React Native.

RAM usage

Table 3-9 contains the RAM usage results of the accelerometer benchmark test. Ionic had the highest mean RAM usage value at 6.9 MB, and Flutter required the least RAM usage at 1.3 MB. Ionic and React Native had higher variability than the other MDTs.

Table 3-9 - Accelerometer: RAM usage (MB)

MDT	Mean	Std	Min	Max
Ionic	6.9	2.5	3.9	12.8
Flutter	1.3	0.6	0.5	3.1
React Native	2.3	3.8	0.5	6.5
Xamarin	1.6	0.4	0.3	2.1
Android Native	2.2	1.7	0.9	9

An ANOVA test showed statistically significant differences in accelerometer RAM usage for the different MDTs ($F\text{-value} = 47.6$, $p\text{-value} < 0.001$); however, a Tukey post-hoc test revealed only significant pair-wise differences between Ionic and the other MDTs. The post-hoc test suggests that Ionic alone is responsible for the statistically significant differences in accelerometer RAM usage. All the other MDTs did not significantly differ from each other in terms of RAM usage.

Execution time

The results of the accelerometer benchmark test with regards to execution time can be seen in Table 3-10. Ionic took the most time to execute the task at 103 ms, while Flutter took the least time to complete the task at 30 ms. Ionic and Xamarin showed higher variation in execution time than the other MDTs.

Table 3-10 - Accelerometer: Execution Time (ms)

MDT	Mean	Std	Min	Max
Ionic	103	25	90	208
Flutter	30	12	13	51
React Native	59	11	48	91
Xamarin	86	22	61	124
Android Native	56	16	40	105

An ANOVA test revealed statistically significant differences in accelerometer execution times ($F\text{-value} = 46.9$, $p\text{-value} < 0.001$). In addition, a post-hoc Tukey test showed significant pair-wise differences in execution time for all the MDTs except between Android Native and React.

Ranking

Table 3-11 shows the ranking of the MDTs when performing the accelerometer benchmark test. Again, Flutter was the best-performing MDT followed by Android Native. This result is noteworthy as Android Native is historically considered to be better than cross-platform counterparts. React Native and Xamarin tied for third, while Ionic performed the worst.

Table 3-11 – Accelerometer ranking (where lower is better)

MDT	CPU	RAM	Time	∑
Ionic	5	5	5	15
Flutter	1	1	1	3
React Native	3	4	3	10
Xamarin	4	2	4	10
Android Native	2	3	2	7

3.5.3 CONTACTS

CPU usage

The CPU-usage results of the contacts benchmark test can be seen in Table 3-12. Ionic had the most CPU usage at 17.8%, followed closely by Flutter and React Native. Android Native performed the best, only requiring 8.4% CPU usage to complete the task. It should be noted, however, that Android Native showed the most variation in CPU usage.

Table 3-12 - Contacts: CPU usage (%)

MDT	Mean	Std	Min	Max
Ionic	17.8	2.6	12	22
Flutter	16.7	1.9	13	2
React Native	15.5	1.9	13	19
Xamarin	10	1.2	8	12
Android Native	8.4	4	4	15

An ANOVA test showed statistically significant differences in CPU usage when executing the contacts benchmark test (F-value $= 53.1$, p-value < 0.001) when using the different MDTs. In addition, a Tukey post-hoc test showed significant pair-wise differences between all the MDTs, except between Android Native and Xamarin, between Ionic and Xamarin, and between React Native and Xamarin.

RAM usage

The RAM usage results of the contacts benchmark test can be seen in Table 3-13. Android Native required the least amount of RAM to complete the task, while Ionic required the most. Ionic showed more variance RAM usage as well.

Table 3-13 - Contacts: RAM usage (MB)

MDT	Mean	Std	Min	Max
Ionic	7.5	2.7	4.9	13.4
Flutter	5.6	1.3	2.5	8.4
React Native	4.1	0.8	2.5	5.9
Xamarin	5.9	1.2	3.1	7.8
Android Native	3.1	1.9	1.1	6.8

An ANOVA test revealed statistically significant differences in RAM usage for the different MDTs (F-value $= 20.3$, p-value < 0.001). Upon closer inspection of the post-hoc Tukey results, it was revealed that there are significant pair-wise differences in RAM usage between all the MDTs, except between Android and React and between Flutter and Xamarin.

Execution time

The execution time results of the contacts benchmark test can be seen in Table 3-14. Observe how Xamarin took much longer to execute the task than the other MDTs; even the minimum Xamarin value is much higher than the maximum values of the other MDTs. Android Native had the fastest execution time.

Table 3-14 - Contacts: Execution time (ms)

MDT	Mean	Std	Min	Max
Ionic	115.3	57.9	93	365
Flutter	285.2	48.6	197	372
React Native	334.6	83.7	243	484
Xamarin	4 371.4	1 463.9	2 850	5 984
Android Native	105.5	99.4	23	277

The ANOVA test shows statistically significant differences in execution for the contacts benchmark test (F-value = 159.6, p-value < 0.001), but the post-hoc Tukey test revealed that the only significant pair-wise differences are between Xamarin and the other MDTs. The post-hoc test, therefore, suggests that Xamarin alone is responsible for the ANOVA test results. Hence, when not considering Xamarin, there are no significant differences amongst the MDTs in execution time when completing the contacts benchmark test.

Ranking

The ranking of the MDTs for the contacts benchmark test can be seen in Table 3-15. Ionic performed the worst, while Android Native performed the best. React Native ranked second place, Flutter ranked third place, and Xamarin ranked fourth place.

Table 3-15 - Contacts ranking (lower is better)

MDT	CPU	RAM	Time	∑
Ionic	5	5	2	12
Flutter	4	3	3	10
React Native	3	2	4	9
Xamarin	2	4	5	11
Android Native	1	1	1	3

3.5.4 FILESYSTEM

CPU usage

The CPU-usage results of the filesystem benchmark test can be seen in Table 3-16. Android Native required the least CPU usage at 5%, while Ionic required the most CPU usage at 16.7%. React Native also showed relatively high CPU usage at 10.6%.

Table 3-16 – Filesystem: CPU usage (%)

MDT	Mean	Std	Min	Max
Ionic	16.7	3.3	11	24
Flutter	5.4	1.2	3	8
React Native	10.6	2.5	6	18
Xamarin	6.5	1	5	8
Android Native	5	0.7	4	6

An ANOVA test showed statistically significant differences in CPU usage for different MDTs ($F\text{-value} = 118.6199$, $p\text{-value} < 0.001$). A Tukey post-hoc test showed significant pair-wise differences between all the MDT combinations except between Android and Flutter, between Android and Xamarin, and between Xamarin and Flutter. Thus, there were no significant differences in CPU usage for the three best MDTs.

RAM usage

The RAM usage result for the filesystem benchmark test can be seen in Table 3-17. Ionic had the highest RAM usage at 7.7 MB, while Flutter had the lowest RAM usage at 1.1 MB. Consequently, Ionic showed higher variation in RAM usage when compared to the other MDTs.

Table 3-17 - Filesystem: RAM usage (MB)

MDT	**Mean**	**Std**	**Min**	**Max**
Ionic	7.7	2.1	5.3	12.6
Flutter	1.1	0.06	0	3
React Native	2.1	0.7	0.4	3.3
Xamarin	1.4	0.22	1	1.9
Android Native	1.6	0.5	0.4	2.3

An ANOVA test showed that there are statistically significant differences in filesystem RAM usage for different MDTs. The post-hoc Tukey test showed that there are significant pair-wise differences between Ionic and all the MDTs. Flutter and React Native were also significantly different. Thus, there were no significant differences in RAM usage between Flutter, Xamarin, and Android Native.

Execution time

The execution time results of the filesystem benchmark test can be seen in Table 3-18. Ionic took the most time to execute the task at 67.6 ms. React Native and Xamarin followed with 53.3 ms and 42.3 ms, respectively. Android Native had the fastest execution time at 5.1 ms.

Table 3-18 - Filesystem: Execution time (ms)

MDT	**Mean**	**Std**	**Min**	**Max**
Ionic	67.6	45.8	29	209
Flutter	11.3	1.4	8	13
React Native	53.3	12.9	43	90
Xamarin	42.3	2.2	39	47
Android Native	5.1	0.6	4	7

After conducting an ANOVA test, it was revealed that there were statistically significant differences in filesystem execution time for the MDTs (F-value $= 31.8638$, p-value < 0.001). Furthermore, the post-hoc Tukey test revealed all pair-wise differences were significant except those between Android Native and Flutter, between Flutter and Xamarin, and between React Native and Xamarin.

Ranking

The rankings of the filesystem benchmarking test can be seen in Table 3-19. Flutter and Android Native tied for first place. Xamarin performed the second best. React Native came in third, and Ionic had the worst performance.

Table 3-19 - Filesystem ranking

MDT	CPU	RAM	Time	∑
Ionic	5	5	5	15
Flutter	2	1	2	5
React Native	4	4	4	12
Xamarin	3	2	3	8
Android Native	1	3	1	5

3.5.5 GEOLOCATION

CPU usage

The CPU usage results for the geolocation benchmark test can be seen in Table 3-20. Ionic had the highest CPU usage at 16.1%, followed by Flutter at 10.6%. React Native had the lowest CPU usage at 7.5%.

Table 3-20 - Geolocation: CPU usage (%)

MDT	Mean	Std	Min	Max
Ionic	16.1	2.3	12	21
Flutter	10.6	2.1	7	15
React Native	7.5	1.5	5	10
Xamarin	8.1	1.5	6	10
Android Native	8.9	1.8	6	13

The ANOVA test showed statistically significant differences in geolocation CPU usage for the different MDTs ($F\text{-value} = 69.8177$, $p\text{-value} < 0.001$). The Tukey post-hoc test showed no significant pair-wise differences in CPU usage between Android Native and React Native, between Android Native and Xamarin, or between React Native and Xamarin.

RAM usage

The RAM usage results of the geolocation benchmark test can be seen in Table 3-21. Ionic required the most RAM to complete the task at 7.9 MB, while Xamarin required the least at 2.1 MB. React Native was a close second at 2.6 MB.

Table 3-21 - Geolocation: RAM usage (MB)

MDT	Mean	Std	Min	Max
Ionic	7.9	1.7	5.9	13.2
Flutter	4.9	2	2.3	12.1
React Native	2.6	1.4	0.7	6.8
Xamarin	2.1	0.3	1.8	3.3
Android Native	4.9	1.5	0.8	6.7

An ANOVA test showed statistically significant differences in RAM usage for the different MDTs (F-value $=$ 46.4672, p-value $<$ 0.001). A post-hoc Tukey test showed significant pair-wise differences in geolocation RAM usage for all the MDTs, except between Android Native and Flutter and between React Native and Xamarin.

Execution time

The execution time results for the geolocation benchmark test can be seen in Table 3-22. Xamarin took the most time to complete the task at 19 179.5 ms. The other MDTs were relatively close, with Android Native having the fastest execution time at 1 048.7 ms.

Table 3-22 - Geolocation: Execution time (ms)

MDT	Mean	Std	Min	Max
Ionic	1 167.5	83.5	1 108	1 479
Flutter	1 127.7	28.7	1 097	1 176
React Native	1 053.8	18.4	1 029	1 095
Xamarin	19 179.5	18 200.2	5 587	78 404
Android Native	1 048.7	56.6	987	1 183

The ANOVA test revealed statistically significant differences in geolocation execution time for the different MDTs (F-value $=$ 19.7368, p-value $<$ 0.001). The post-hoc Tukey test showed that the only significant pair-wise differences in execution time were where Xamarin was involved. Thus, when excluding Xamarin, there were insignificant differences in execution time between the MDTs.

Ranking

The rankings of the filesystem benchmarking test can be seen in Table 3-23. React Native ranked first, followed by Android Native, Xamarin, Flutter and finally, Ionic ranked last place.

Table 3-23- Geolocation ranking

MDT	CPU	RAM	Time	∑
Ionic	5	5	4	14
Flutter	4	3	3	10
React Native	1	2	2	5
Xamarin	2	1	5	8
Android Native	3	3	1	7

3.6 DISCUSSION

In this performance evaluation, four cross-platform MDTs and one native MDT were compared. In addition, four benchmark tests were specified, and, after implementation, 20 benchmark applications were used to measure performance metrics.

In literature, the purpose of many performance-evaluation studies was to compare cross-platform MDTs with native MDTs. Cross-platform MDTs are more cost- and time-efficient than native MDTs, especially when a presence on multiple platforms is required. Assume cross-platform MDTs are comparable or even equal in performance to their native counterparts; in that case, cross-platform MDTs might be the better solution to mobile application development in many cases.

With rare exceptions, most studies in literature showed that native MDTs outperformed cross-platform MDTs; even recent studies support this claim. However, many studies did not determine how significant the performance differences between MDTs are.

Table 3-24 shows the number of times an MDT performed the best for a specific performance metric. Android Native performed the best most frequently for the CPU usage and execution time performance metrics, but Flutter performed the best most frequently with RAM usage. Android Native was also the MDT that performed the best when considering overall rankings.

At first glance, it would appear as though this performance evaluation would reach the same conclusions as previous studies, that native MDTs are superior to cross-platform in terms of performance.

Table 3-24 – The number of times that an MDT performed the best

MDT	**CPU**	**RAM**	**Time**
Ionic	0	0	0
Flutter	1	2	1
React Native	1	0	0
Xamarin	0	1	0
Android Native	2	1	3

However, when considering significance, there was no single instance where Android Native performed *significantly* better than any MDT that performed the second best. There were also cases such as with the geolocation benchmark test where, excluding Xamarin, there were no significant differences in execution time. This performance evaluation has shown the worth of including ANOVA and post-hoc Tukey testing during the evaluation phase of the proposed framework.

Some issues were discovered after applying the evaluation phase. The method obtained from literature to rank MDTs by performance is intuitive and easy to use; however, the method does not consider extreme performance values. Xamarin's contacts and geolocation execution time results are examples of this issue. It took Xamarin 13 times longer to collect the contacts' data and 14.4 times longer retrieving geolocation data than the second slowest MDT in the respective test. In industry, this poor performance display would lead to a bad user experience and, consequently, a negative customer review [55]. Yet, Xamarin was only penalised one score point compared to the MDT that performed the second worst because of the ranking method.

Overall, the framework to evaluate mobile development technology provides a structured way to compare MDTs. The framework is generic and can be applied to any MDTs, performance metrics, and benchmark tests. The benchmark tests used in this performance evaluation focused on mobile features and sensors but will be viable with more complex benchmark tests.

3.7 VALIDATION

3.7.1 OVERVIEW

In this section, the developed framework will be validated. As discussed in Section 2.7, Verification focused on meeting the requirements and specifications defined in Table 2-1 and ensuring that the framework was developed *correctly*. Validation will ensure that the research objectives are met as defined in Section 1.7 and that the *correct* framework was developed. The following validation strategy will be followed:

1. Ensure that RO1, namely, *Develop a framework to evaluate the performance of mobile development technologies*, has been met.
2. Ensure that RO2, namely, *Verify the developed framework by ensuring that it meets the requirements and specifications of an MDT performance evaluation*, has been met.
3. Ensure that RO3, namely, *Apply the framework by conducting performance-evaluation studies using popular modern mobile development technologies*, has been met.
4. Ensure that RO4, namely, *Validate that the developed framework addresses the research objectives*, has been met.

3.7.2 RO1: DEVELOP A FRAMEWORK TO EVALUATE THE PERFORMANCE OF MOBILE DEVELOPMENT TECHNOLOGIES

The first research objective involves developing a framework to evaluate the performance of mobile development technologies. Sections 2.3 - 2.6 in Chapter 2 discuss the four phases of the developed framework and their sub-phases in detail. The complete developed framework can be seen in Figure 2-8 in Chapter 2, ensuring that the first empirical objective was met.

3.7.3 RO2: VERIFY THE DEVELOPED FRAMEWORK BY ENSURING THAT IT MEETS THE REQUIREMENTS AND SPECIFICATIONS OF AN MDT PERFORMANCE EVALUATION

The second objective involves verifying the developed framework by addressing the requirements and specifications in Table 2-1 in Section 2.2. Verification ensures that the framework is developed *correctly* by meeting the requirements. The requirements were defined

based on observations made during the literature review. In Section 2.7 the requirements and specifications were verified, ensuring the second empirical objective was met.

3.7.4 RO3: Apply the framework by conducting performance-evaluation studies using popular modern mobile development technologies

The third objective involves applying the developed framework by evaluating popular modern mobile development technologies. In Chapter 3, five mobile development technologies, namely (1) Ionic, (2) Flutter, (3) React Native, (4) Xamarin, and (5) Android Native, were evaluated. Sections 3.2 - 3.5 in Chapter 3 discusses the application of the developed framework from the implementation phase to the evaluation phase, ensuring that the third empirical objective was met.

3.7.5 RO4: Validate that the developed framework addresses the research objectives

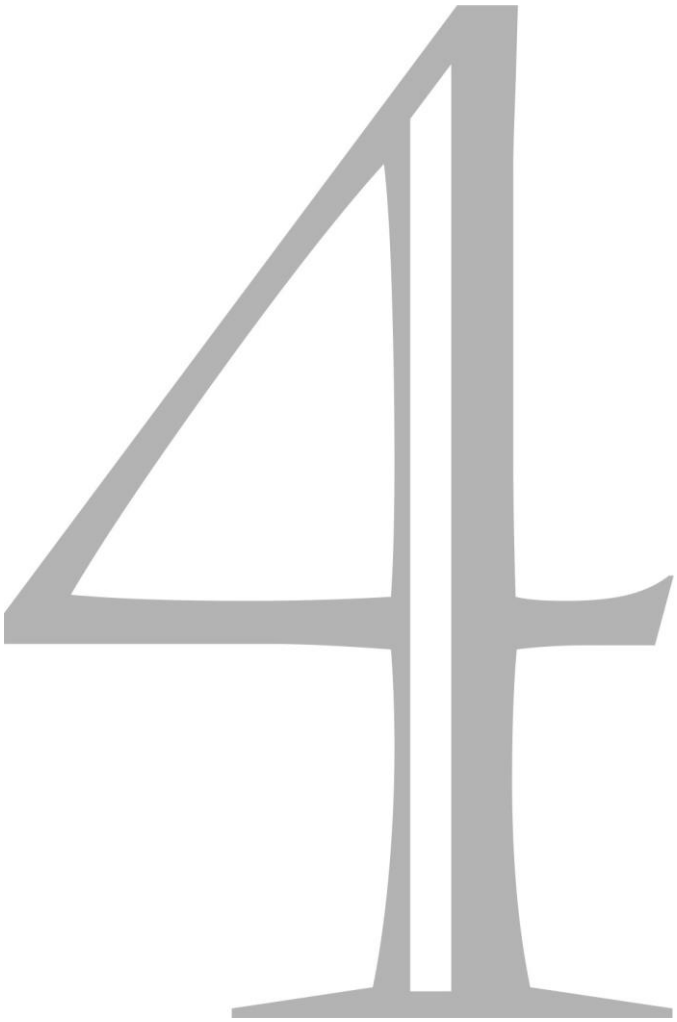
The fourth objective involves validating that the developed framework addresses the research objectives defined in Section 1.7. In this section, validation was done to confirm that the research objectives were met. This fourth and final objective was addressed after validating that the first three objectives were met.

3.8 Summary

In this section, the developed framework to evaluate the performance of mobile development technologies was applied. Four cross-platform MDTs, namely Ionic, Flutter, React Native, and Xamarin were selected, and Android Native was chosen as a native MDT. The performance metrics that were measured were CPU usage, RAM usage, and execution time. The benchmark tests included reading accelerometer data, reading geolocation data, reading and writing to the filesystem, and retrieving a list of contacts. Android Native performed the best, but Flutter came close in second place and was the best performing cross-platform MDT. After applying the developed framework, validation was done, confirming that the research objectives were met.

CHAPTER 4

Conclusion



4 CONCLUSION

4.1 REVIEW OF WORK DONE

Mobile applications have become invaluable tools in modern society. The financial success of the mobile industry increases annually and is predicted to continue to increase well into the future. There has, therefore, never been a more lucrative time for developers and companies to create mobile applications. This study introduced multiple approaches to mobile development. For each approach, there are numerous mobile development technologies. Mobile development technologies (or MDTs) provide the tools necessary to create mobile applications.

To improve a mobile application's chance of success, developers need to prioritise user experience. A quality user experience leads to more user acceptance, which increases the overall success of a mobile application. Performance is a crucial feature influencing user experience; thus, with so many MDTs available today, each with their own effect on application performance, this leads to the following question, *How does one evaluate the performance of mobile development technologies?*

A literature review was conducted to answer this question. Many performance-evaluation studies were examined. The literature review revealed that there is no standardised framework to evaluate the performance of mobile development technologies. This problem led to multiple performance evaluations using different methods and with incomparable results. A framework was, therefore, developed to evaluate the performance of mobile development technologies to address this problem.

The framework comprised four phases: *selection*, *implementation*, *measurement*, and *evaluation*. Each phase comprises one or more sub-phases. The framework was applied by comparing five MDTs, namely Ionic, Flutter, React Native, Xamarin, and Android Native. Four benchmark tests were devised and implemented using the selected MDTs, each targeting a different mobile feature or sensor.

The results showed that Android Native performed the best overall. Android Native had the best CPU performance in 50% of the benchmark tests and had the fastest execution time in 75% of the benchmark tests. Flutter was the second-best performing MDT overall and had the best RAM usage in 50% of cases. Nevertheless, there were two benchmark tests where

Xamarin performed exceptionally poor compared to the second-worst MDT of the respective test. Xamarin was 13 times slower in the contacts benchmark test and 14.4 times slower in the geolocation benchmark test than the second-worst performing MDTs of those tests.

Even though Android Native performed the best, statistical analysis showed that it never performed significantly better than the second MDT. The results obtained during this study suggest that native MDTs are not necessarily superior to cross-platform MDTs in terms of performance. An example would be Flutter, a cross-platform MDT with the best RAM performance overall. Furthermore, a statistical analysis showed that there were situations where none of the MDTs performed significantly differently from each other, for example, the geolocation benchmark test (excluding Xamarin). The results emphasised the need for statistical analysis in performance evaluations. Without statistical analysis, it would be challenging to infer significant statistical differences in performance.

There are numerous benefits to using the developed framework. Even though it was not a requirement, the framework was developed to be generic in nature. The framework does not restrict the user to MDTs, benchmark tests, or performance metrics. The user is free to design and implement their benchmark tests and select performance metrics applicable to their use case. It is also possible to adapt and use the developed framework for other technologies and not only MDTs.

The framework is valuable for future MDT performance-evaluation studies and elevates future studies to the same standard. As mentioned in Chapter 1, there is no standard structure for performance-evaluation studies in literature. The framework's usage will provide structure to future studies and result in insightful findings, especially when discussing the significance of the findings. The use of the framework will also assist in increasing the reproducibility of future studies. All selected MDTs, tools, and plugins and their versions should be declared during the *selection* phase, allowing others to use the same setup for replication purposes.

This study showed that the developed framework could successfully be applied to evaluate the performance of mobile development technologies.

4.2 MEETING OF THE RESEARCH OBJECTIVES

Verification focused on meeting the requirements and specifications defined in Table 2-1 and ensured that the framework was developed *correctly*. Validation ensured that the research objectives as defined in Section 1.7 are met, and that the *correct* framework was developed. In Section 3.7, a validation strategy was followed to ensure that the research objectives were met. The validation strategy can be seen below.

1. Ensure that RO1, namely, *Develop a framework to evaluate the performance of mobile development technologies,* has been met.
2. Ensure that RO2, namely, *Verify the developed framework by ensuring that it meets the requirements and specifications of an MDT performance evaluation,* has been met.
3. Ensure that RO3, namely, *Apply the framework by conducting performance-evaluation studies using popular modern mobile development technologies,* has been met.
4. Ensure that RO4, namely, *Validate that the developed framework addresses the research objectives,* has been met.

The first research objective was addressed after developing the framework in Sections 2.3 - 2.6. Each section discussed the four phases of the framework and their sub-phases in detail. The second research objective was addressed after verifying that the requirements and specifications defined in Table 2-1 were met. This was done in Section 2.7. The third research objective was addressed after applying the developed framework in Sections 3.2 - 3.5. The fourth research objective was addressed after validation was done in Section 3.7.

A summary of the validation done in Section 3.7, including the sections where the research objectives were met, can be seen in Table 4-1.

Table 4-1 - Validation summary

Research objectives	Section references	Objective met
Research objective 1	2.3, 2.4, 2.5, 2.6	✓
Research objective 2	2.2, 2.7	✓
Research objective 3	3.2, 3.3, 3.4, 3.5	✓
Research objective 4	1.7, 3.7	✓

The meeting of the research objectives ensured that the *correct* performance evaluation framework was developed, and in doing so addressed the defined need for the study:

There is a need to develop a framework to evaluate the performance of mobile development technologies.

The addressing of the need of the study ensured that the problem defined in the problem statement has been solved:

Currently, in the MDT research field, multiple studies have been conducted focusing on performance; however, these studies use different methods of evaluation, leading to numerous incomparable studies. These incompatibilities are due to the lack of a standardised framework that can be used to evaluate mobile development technology performance.

By solving the problem formulated in the problem statement, the clear knowledge gap in the knowledge spectrum has been filled:

There is no framework that can be used to evaluate the performance of mobile development technologies. Failure to address this knowledge gap will lead to more mobile development technology studies with incomparable methods and results.

4.3 RECOMMENDATIONS FOR FURTHER RESEARCH

This section will provide recommendations for further study. Four significant proposals will be listed and discussed below. Implementing these recommendations will either improve the framework or put its versatility to the test. The recommendations include:

- Performance metric weights
- Involve more platforms
- Apply the framework to other fields
- Use more testing devices

4.3.1 PERFORMANCE METRIC WEIGHTS

In this study, the developed framework did not provide a method of assigning weights to performance metrics. All performance metrics were treated equally during this performance

evaluation. Even though users can assign their own weights independently, it would be beneficial if the framework itself provided a method that users can follow to assign weights.

4.3.2 INVOLVE MORE PLATFORMS

The conducted performance evaluation only featured benchmark applications created for the Android platform. Performance evaluations using benchmark applications developed for iOS and Windows will provide additional insight into cross-platform MDT performance and native MDT performance.

4.3.3 APPLY THE FRAMEWORK TO OTHER FIELDS

The developed framework was developed to be generic; it is not necessarily limited to the mobile industry. The framework can be adapted to evaluate web development technologies, programming languages, or computational algorithms. The *selection* and *implementation* phases would require more adaptation than the *measurement* and *evaluation* phases.

4.3.4 USE MORE TESTING DEVICES

For the validation of the developed framework only one test device was used. Even though it was acceptable for validation, it would not be suitable when using the framework for large industrial projects. In such cases it is essential that more test devices spread across the hardware spectrum is used to get a better sense of general performance.

REFERENCES

[1] A. Ahmad, K. Li, C. Feng, S. M. Asim, A. Yousif, and S. Ge, "An Empirical Study of Investigating Mobile Applications Development Challenges," *IEEE Access*, vol. 6, pp. 17711–17728, 2018, doi: 10.1109/ACCESS.2018.2818724.

[2] P. Thusi and D. K. Maduku, "South African millennials' acceptance and use of retail mobile banking apps: An integrated perspective," *Comput. Human Behav.*, vol. 111, p. 106405, Oct. 2020, doi: 10.1016/j.chb.2020.106405.

[3] Z. Karabatzaki *et al.*, "Mobile application tools for students in secondary education. An evaluation study," *Int. J. Interact. Mob. Technol.*, vol. 12, no. 2, pp. 142–161, Mar. 2018, doi: 10.3991/ijim.v12i2.8158.

[4] A. S. Drigas and P. Angelidakis, "Mobile applications within education: An overview of application paradigms in specific categories," *Int. J. Interact. Mob. Technol.*, vol. 11, no. 4, pp. 17–29, 2017, doi: 10.3991/ijim.v11i4.6589.

[5] J. Hamari, A. Malik, J. Koski, and A. Johri, "Uses and Gratifications of Pokémon Go: Why do People Play Mobile Location-Based Augmented Reality Games?," *Int. J. Hum. Comput. Interact.*, vol. 35, no. 9, pp. 804–819, 2019, doi: 10.1080/10447318.2018.1497115.

[6] H. Wang, S. Xu, H. Wu, Y. Zhang, G. Hu, and S. Mei, "Towards Unified Ultimate Gaming Experience," *IEEE Consum. Electron. Mag.*, vol. 2248, no. c, pp. 10–12, 2021, doi: 10.1109/MCE.2021.3061059.

[7] E. Kocak, V. A. Nasir, and H. B. Turker, "What drives Instagram usage? User motives and personality traits," *Online Inf. Rev.*, vol. 44, no. 3, pp. 625–643, 2020, doi: 10.1108/OIR-08-2019-0260.

[8] H. Zuo and T. Wang, "Analysis of Tik Tok User Behavior from the Perspective of Popular Culture," *Front. Art Res.*, vol. 1, no. 3, pp. 1–05, 2019, doi: 10.25236/FAR.20190301.

[9] A. Oyedele and P. M. Simpson, "Journal of Retailing and Consumer Services Streaming apps : What consumers value," vol. 41, no. April 2017, pp. 296–304, 2018.

[10] K. Tao and P. Edmunds, "Mobile APPs and Global Markets," *Theor. Econ. Lett.*, vol. 08, no. 08, pp. 1510–1524, 2018, doi: 10.4236/tel.2018.88097.

[11] U. S. Henama and P. P. S. Sifolo, "Uber: The South Africa experience," *African J. Hosp. Tour. Leis.*, vol. 6, no. 2, 2017.

[12] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, "Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation," in *Proceedings - 2nd ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft 2015*, 2015, pp. 56–59, doi: 10.1109/MobileSoft.2015.15.

[13] Z. Aydin Gokgoz, M. B. Ataman, and G. H. van Bruggen, "There's an app for that! understanding the drivers of mobile application downloads," *J. Bus. Res.*, vol. 123, no. December 2019, pp. 423–437, Feb. 2021, doi: 10.1016/j.jbusres.2020.10.006.

[14] "Total global mobile app revenues 2014-2013." [Online]. Available: https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/. [Accessed: 15-Mar-2021].

[15] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Evaluating Cross-Platform Development Approaches for Mobile Applications," in *Lecture Notes in Business Information Processing*, vol. 140 LNBIP, 2013, pp. 120–138.

[16] A. Biørn-Hansen, C. Rieger, T.-M. Grønli, T. A. Majchrzak, and G. Ghinea, "An empirical investigation of performance overhead in cross-platform mobile development frameworks," *Empir. Softw. Eng.*, vol. 25, no. 4, pp. 2997–3040, Jul. 2020, doi: 10.1007/s10664-020-09827-6.

[17] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba, "Taxonomy of Cross-Platform Mobile Applications Development Approaches," *Ain Shams Eng. J.*, vol. 8, no. 2, pp. 163–190, 2017, doi: 10.1016/j.asej.2015.08.004.

[18] A. Biørn-Hansen, T.-M. Grønli, and G. Ghinea, "A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 1–34, Jan. 2019, doi: 10.1145/3241739.

[19] C. P. Rahul Raj and S. B. Tolety, "A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach," *2012 Annu. IEEE India Conf. INDICON 2012*, pp. 625–629, 2012, doi: 10.1109/INDCON.2012.6420693.

[20] T. Y. Adinugroho, Reina, and J. B. Gautama, "Review of Multi-platform Mobile Application Development Using WebView: Learning Management System on Mobile Platform," *Procedia Comput. Sci.*, vol. 59, pp. 291–297, 2015, doi: 10.1016/j.procs.2015.07.568.

[21] M. Ciman and O. Gaggi, "An empirical analysis of energy consumption of cross-

platform frameworks for mobile development," *Pervasive Mob. Comput.*, vol. 39, pp. 214–230, Aug. 2017, doi: 10.1016/j.pmcj.2016.10.004.

[22] A. Biørn-Hansen and G. Ghinea, "Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development," in *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018, doi: 10.24251/HICSS.2018.716.

[23] L. Delia, N. Galdamez, P. Thomas, L. Corbalan, and P. Pesado, "Multi-platform mobile application development analysis," *Proc. - Int. Conf. Res. Challenges Inf. Sci.*, vol. 2015-June, no. June, pp. 181–186, 2015, doi: 10.1109/RCIS.2015.7128878.

[24] R. Acerbis, A. Bongio, M. Brambilla, and S. Butti, "Model-Driven Development of Cross-Platform Mobile Applications with Web Ratio and IFML," in *Proceedings - 2nd ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft 2015*, 2015, pp. 170–171, doi: 10.1109/MobileSoft.2015.49.

[25] M. Latif, Y. Lakhrissi, E. H. Nfaoui, and N. Es-Sbai, "Cross platform approach for mobile application development: A survey," in *2016 International Conference on Information Technology for Organizations Development, IT4OD 2016*, 2016, pp. 1–5, doi: 10.1109/IT4OD.2016.7479278.

[26] H. Heitkötter, H. Kuchen, and T. A. Majchrzak, "Extending a model-driven cross-platform development approach for business apps," *Sci. Comput. Program.*, vol. 97, no. P1, pp. 31–36, Jan. 2015, doi: 10.1016/j.scico.2013.11.013.

[27] A. Biørn-Hansen, T. A. Majchrzak, and T.-M. Grønli, "Progressive Web Apps: The Possible Web-native Unifier for Mobile Development," in *Proceedings of the 13th International Conference on Web Information Systems and Technologies*, 2017, no. Webist, pp. 344–351, doi: 10.5220/0006353703440351.

[28] I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein, "Survey, comparison and evaluation of cross platform mobile application development tools," *2013 9th Int. Wirel. Commun. Mob. Comput. Conf. IWCMC 2013*, pp. 323–328, 2013, doi: 10.1109/IWCMC.2013.6583580.

[29] L. Corral, A. Janes, and T. Remencius, "Potential advantages and disadvantages of multiplatform development frameworks - A vision on mobile environments," *Procedia Comput. Sci.*, vol. 10, pp. 1202–1207, 2012, doi: 10.1016/j.procs.2012.06.173.

[30] S. Ickin, K. Wac, M. Fiedler, L. Janowski, J. H. Hong, and A. K. Dey, "Factors

influencing quality of experience of commonly used mobile applications," *IEEE Communications Magazine*, vol. 50, no. 4, pp. 48–56, Apr. 2012, doi: 10.1109/MCOM.2012.6178833.

[31] S. Xanthopoulos and S. Xinogalos, *A comparative analysis of cross-platform development approaches for mobile applications*. 2013.

[32] M. Willocx, J. Vossaert, and V. Naessens, "Comparing performance parameters of mobile app development strategies," *Proc. - Int. Conf. Mob. Softw. Eng. Syst. MOBILESoft 2016*, pp. 38–47, 2016, doi: 10.1145/2897073.2897092.

[33] B. S.Thakare, D. Shirodkar, N. Parween, and S. Parween, "State of Art Approaches to Build Cross Platform Mobile Application," *Int. J. Comput. Appl.*, vol. 107, no. 20, pp. 22–23, Dec. 2014, doi: 10.5120/18868-0389.

[34] M. E. Joorabchi, A. Mesbah, and P. Kruchten, "Real Challenges in Mobile App Development," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013, pp. 15–24, doi: 10.1109/ESEM.2013.9.

[35] T. Majchrzak and T.-M. Grønli, "Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks," in *Proceedings of the 50th Hawaii International Conference on System Sciences (2017)*, 2017, doi: 10.24251/hicss.2017.745.

[36] L. Gaouar, A. Benamar, and F. Tarik Bendimerad, "Desirable Requirements of Cross Platform Mobile Development Tools," in *Eletronic Devices*, 2016, vol. 5, no. 1, pp. 14–22.

[37] J. Perchat, M. Desertot, and S. Lecomte, "Common framework: A hybrid approach to integrate cross-platform components in mobile application," *J. Comput. Sci.*, vol. 10, no. 11, pp. 2165–2181, 2014, doi: 10.3844/jcssp.2014.2165.2181.

[38] M. Ciman and O. Gaggi, "Evaluating impact of cross-platform frameworks in energy consumption of mobile applications," *WEBIST 2014 - Proc. 10th Int. Conf. Web Inf. Syst. Technol.*, vol. 1, pp. 423–431, 2014, doi: 10.5220/0004857604230431.

[39] S. Dhillon and Q. H. Mahmoud, "An evaluation framework for cross-platform mobile application development tools," *Softw. Pract. Exp.*, vol. 45, no. 10, pp. 1331–1357, Oct. 2015, doi: 10.1002/spe.2286.

[40] M. Willocx, J. Vossaert, and V. Naessens, "A Quantitative Assessment of Performance in Mobile App Development Tools," in *2015 IEEE International Conference on Mobile*

Services, 2015, vol. 32, no. 3, pp. 454–461, doi: 10.1109/MobServ.2015.68.

[41] M. Ciman and O. Gaggi, "Measuring Energy Consumption of Cross-Platform Frameworks for Mobile Applications," vol. 226, V. Monfort and K.-H. Krempels, Eds. Cham: Springer International Publishing, 2015, pp. 331–346.

[42] P. Que, X. Guo, and M. Zhu, "A Comprehensive Comparison between Hybrid and Native App Paradigms," in *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*, 2016, pp. 611–614, doi: 10.1109/CICN.2016.125.

[43] M. Ciman and O. Gaggi, "An empirical analysis of energy consumption of cross-platform frameworks for mobile development," *Pervasive Mob. Comput.*, vol. 39, pp. 214–230, 2017, doi: 10.1016/j.pmcj.2016.10.004.

[44] P. Grzmil, M. Skublewska-Paszkowska, E. Łukasik, and J. Smołka, "Performance analysis of native and cross-platform mobile applications," *Informatics Control Meas. Econ. Environ. Prot.*, vol. 7, no. 2, pp. 50–53, Jun. 2017, doi: 10.5604/01.3001.0010.4838.

[45] X. Jia, A. Ebone, and Y. Tan, "A performance evaluation of cross-platform mobile application development approaches," in *Proceedings - International Conference on Software Engineering*, 2018, pp. 92–93, doi: 10.1145/3197231.3197252.

[46] C. M. S. Ferreira *et al.*, "An evaluation of cross-platform frameworks for multimedia mobile applications development," *IEEE Lat. Am. Trans.*, vol. 16, no. 4, pp. 1206–1212, 2018, doi: 10.1109/TLA.2018.8362158.

[47] L. Delia *et al.*, "Development Approaches for Mobile Applications: Comparative Analysis of Features," in *Advances in Intelligent Systems and Computing*, vol. 857, no. January, 2019, pp. 470–484.

[48] L. Corbalan *et al.*, "Development frameworks for mobile devices," in *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, 2018, no. May, pp. 191–201, doi: 10.1145/3197231.3197242.

[49] A. Biørn-Hansen, T.-M. Grønli, and G. Ghinea, "Animations in Cross-Platform Mobile Applications: An Evaluation of Tools, Metrics and Performance," *Sensors*, vol. 19, no. 9, p. 2081, May 2019, doi: 10.3390/s19092081.

[50] S. Huber and L. Demetz, "Performance Analysis of Mobile Cross-platform

Development Approaches based on Typical UI Interactions," in *Proceedings of the 14th International Conference on Software Technologies*, 2019, no. Icsoft 2019, pp. 40–48, doi: 10.5220/0007838000400048.

[51] B. P. D. Putranto, R. Saptoto, O. C. Jakaria, and W. Andriyani, "A Comparative Study of Java and Kotlin for Android Mobile Application Development," in *2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 2020, pp. 383–388, doi: 10.1109/ISRITI51436.2020.9315483.

[52] C. Rieger and T. A. Majchrzak, "Towards the definitive evaluation framework for cross-platform app development approaches," *J. Syst. Softw.*, vol. 153, pp. 175–199, 2019, doi: 10.1016/j.jss.2019.04.001.

[53] S. Manikandan, "Measures of central tendency: The mean," *J. Pharmacol. Pharmacother.*, vol. 2, no. 2, pp. 140–142, 2011, doi: 10.4103/0976-500X.81920.

[54] S. Lange and R. Bender, "Measures of variability," *Dtsch. Med. Wochenschr.*, vol. 132 Suppl, no. 3, pp. 5–6, 2007, doi: 10.1055/s-2007-959026.

[55] Florian, Rösler, André, and Nitze, "Towards a Mobile Application Performance Benchmark," *ICIW 2014 Ninth Int. Conf. Internet Web Appl. Serv.*, vol. 54, no. c, p. 5, 2014.

APPENDIX

Table A-1 – Accelerometer CPU Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.0248	True
Android	**Ionic**	0.001	True
Android	**React**	0.0677	False
Android	**Xamarin**	0.0248	True
Flutter	**Ionic**	0.001	True
Flutter	**React**	0.001	True
Flutter	**Xamarin**	0.001	True
Ionic	**React**	0.0066	True
Ionic	**Xamarin**	0.0207	True
React	**Xamarin**	0.9	False

Table B-2 – Accelerometer RAM Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.2728	False
Android	**Ionic**	0.001	True
Android	**React**	0.9	False
Android	**Xamarin**	0.6893	False
Flutter	**Ionic**	0.001	True
Flutter	**React**	0.1785	False
Flutter	**Xamarin**	0.9	False
Ionic	**React**	0.001	True
Ionic	**Xamarin**	0.001	True
React	**Xamarin**	0.5631	False

Table C-3 – Accelerometer Time Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.001	True
Android	**Ionic**	0.001	True
Android	**React**	0.9	False
Android	**Xamarin**	0.001	True
Flutter	**Ionic**	0.001	True
Flutter	**React**	0.001	True
Flutter	**Xamarin**	0.001	True
Ionic	**React**	0.001	True
Ionic	**Xamarin**	0.0485	True
React	**Xamarin**	0.001	True

Table D-4 – Contacts CPU Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.001	True
Android	**Ionic**	0.001	True
Android	**React**	0.001	True
Android	**Xamarin**	0.2399	False
Flutter	**Ionic**	0.5873	False
Flutter	**React**	0.5873	False
Flutter	**Xamarin**	0.001	True
Ionic	**React**	0.0375	True
Ionic	**Xamarin**	0.001	True
React	**Xamarin**	0.001	True

Table E-5 – Contacts RAM Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.001	True
Android	**Ionic**	0.001	True
Android	**React**	0.355	False
Android	**Xamarin**	0.001	True
Flutter	**Ionic**	0.0067	True
Flutter	**React**	0.0352	True
Flutter	**Xamarin**	0.9	False
Ionic	**React**	0.001	True
Ionic	**Xamarin**	0.0273	True
React	**Xamarin**	0.009	True

Table F-6 – Contacts Time Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.9	False
Android	**Ionic**	0.9	False
Android	**React**	0.7806	False
Android	**Xamarin**	0.001	True
Flutter	**Ionic**	0.9	False
Flutter	**React**	0.9	False
Flutter	**Xamarin**	0.001	True
Ionic	**React**	0.9	False
Ionic	**Xamarin**	0.001	True
React	**Xamarin**	0.001	True

Table G-7 – Filesystem CPU Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.9	False
Android	**Ionic**	0.001	True
Android	**React**	0.001	True
Android	**Xamarin**	0.1616	False
Flutter	**Ionic**	0.001	True
Flutter	**React**	0.001	True
Flutter	**Xamarin**	0.4714	False
Ionic	**React**	0.001	True
Ionic	**Xamarin**	0.001	True
React	**Xamarin**	0.001	True

Table H-8 – Filesystem RAM Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.5233	False
Android	**Ionic**	0.001	True
Android	**React**	0.6967	False
Android	**Xamarin**	0.9	False
Flutter	**Ionic**	0.001	True
Flutter	**React**	0.0464	True
Flutter	**Xamarin**	0.9	False
Ionic	**React**	0.001	True
Ionic	**Xamarin**	0.001	True
React	**Xamarin**	0.2462	False

Table I-9 – Filesystem Time Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.8869	False
Android	**Ionic**	0.001	True
Android	**React**	0.001	True
Android	**Xamarin**	0.001	True
Flutter	**Ionic**	0.001	True
Flutter	**React**	0.001	True
Flutter	**Xamarin**	0.001	True
Ionic	**React**	0.2204	False
Ionic	**Xamarin**	0.0028	True
React	**Xamarin**	0.4868	False

Table J-10 – Geolocation CPU Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.0392	True
Android	**Ionic**	0.001	True
Android	**React**	0.1109	False
Android	**Xamarin**	0.5919	False
Flutter	**Ionic**	0.001	True
Flutter	**React**	0.001	True
Flutter	**Xamarin**	0.001	True
Ionic	**React**	0.001	True
Ionic	**Xamarin**	0.001	True
React	**Xamarin**	0.8292	False

Table K-11 – Geolocation RAM Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.9	False
Android	**Ionic**	0.001	True
Android	**React**	0. 001	True
Android	**Xamarin**	0.001	True
Flutter	**Ionic**	0.001	True
Flutter	**React**	0. 001	True
Flutter	**Xamarin**	0. 001	True
Ionic	**React**	0.001	True
Ionic	**Xamarin**	0.001	True
React	**Xamarin**	0.8162	False

Table L-12 – Geolocation Time Tukey Tests

Group 1	Group 2	p-value	Reject (H_0)
Android	**Flutter**	0.9	False
Android	**Ionic**	0.9	False
Android	**React**	0.9	False
Android	**Xamarin**	0.001	True
Flutter	**Ionic**	0.9	False
Flutter	**React**	0.9	False
Flutter	**Xamarin**	0.001	True
Ionic	**React**	0.9	False
Ionic	**Xamarin**	0.001	True
React	**Xamarin**	0.001	True