Lappeenranta University of Technology
School of Engineering Science
Software Engineering
Master's Programme in Software Engineering and Digital Transformation

Anna Osipova

CROSS-PLATFORM MOBILE DEVELOPMENT FRAMEWORKS: CHOOSING THE MOST SUITABLE OPTION FOR SELECTED USE CASE

Examiners: Associate professor Jouni Ikonen
Assistant professor Antti Knutas

Supervisors: Assistant professor Antti Knutas
Associate professor Jouni Ikonen

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
School of Engineering Science
Tietotekniikan koulutusohjelma
Master's Programme in Software Engineering and Digital Transformation

Anna Osipova

Alustariippumattomat mobiilikehitysohjelmistokehykset: parhaan vaihtoehdon valitseminen tiettyyn käyttötapaukseen

Diplomityö

94 sivua, 18 kuvaa, 9 taulukkoa, 1 liite

Työn tarkastajat: Tutkijaopettaja Jouni Ikonen

Apulaisprofessori Antti Knutas

Hakusanat: ohjelmistokehitys, mobiilisovellukset, mobiilikehitys, ohjelmistokehykset
Keywords: software engineering, mobile applications, mobile development, software frameworks

Mobiilisovellusten kehitys on haastavaa johtuen laitteiden ja alustojen pirstoutumisesta. Tämä diplomityö tutkii vaihtoehtoisia ohjelmistokehysvaihtoja tiimille, joka tuottaa B2B mobiilisovelluksia Cordova:n avulla. Tavoitteena on valita ohjelmistokehys, joka parantaisi kehitysprosessia ja tuottaisi toiminnallisempia sovelluksia nopeammin. Kehitysprosessit arvioidaan tuottamalla sama demosovellus usealla eri ohjelmistokehyksellä.

Ohjelmistokehykset arvioidaan niiden keskeisten ominaisuuksien ja demosovelluksen kehityksen tulosten perusteella. Lupaavimmat vaihtoedot esitellään.

ABSTRACT

Lappeenranta University of Technology
School of Engineering Science
Software Engineering
Master's Programme in Software Engineering and Digital Transformation

Anna Osipova

Cross-platform mobile development frameworks: choosing the most suitable option for selected use case

Master’s Thesis

94 pages, 18 figures, 9 tables, 1 appendix

Examiners: Associate professor Jouni Ikonen

Assistant professor Antti Knutas

Keywords: software engineering, mobile applications, mobile development, software frameworks

Mobile application development is challenging due to device and platform fragmentation. This thesis looks into alternative framework options for a team that develops B2B mobile applications using Cordova framework. The goal is to select a framework that would improve development process and provide more functional end applications faster. Development process of several frameworks is evaluated by implementing a similar demo application with each one. Frameworks are evaluated based on their key characteristics and demo application implementation results and the most promising options are presented.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisors Antti Knutas and Jouni Ikonen for their help and support with this work and for sticking with me despite my terrible scheduling skills.

Thanks goes to my family and friends
who believed in me when I didn't
and kept kicking me until I did it.

TABLE OF CONTENTS

NOMENCLATURE

API – Application Programming Interface

B2B – Business-to-business

Bluetooth – Wireless technology standard for exchanging data over short distances

C# - Multi-paradigm programming language

CLI - Command-line Interface

CSS – Cascading Style Sheets

DOM - Document Object Model

GPS – Global Positioning System

GSM – Global System for Mobile Communications

HTML – Hyper Text Markup Language

HTTP – Hypertext Transfer Protocol

IDE - Integrated Development Environment

JS - JavaScript

JSX - JavaScript XML

LTE - Long-Term Evolution is a standard for wireless broadband communication for mobile devices

LTS – Long Term Support

MDA – Model Driven Architecture

.NET – Software framework

Npm – Node Package Manager

OOP – Object Oriented Programming

OS – Operating System

PC – Personal Computer

REST - Representational State Transfer

Scrum – Agile methodology used in software development

SDK – Software Development Kit

UI – User Interface

UX – User Experience

VM – Virtual Machine

Wi-Fi – Radio technologies used for Wireless Local Area Networking (WLAN)

WORA – Write Once Run Anywhere

XAML – Extensible Application Markup Language

XML – Extensible Markup Language

1 INTRODUCTION

This chapter covers background for this research. Section 1.1 describes the main reasons and motivation behind this case study. Section 1.2 presents the research questions and describes how the solution to the presented problem will be found. Section 1.3 covers research methods used in this case study. 1.4 presents the scope of this research.

1.1 Motivation for this research

The practical purpose of this research is to determine if a development team in question should change their existing cross-platform mobile development framework (CPMDF) for a better alternative, and which of the available frameworks should be chosen. At the moment the development team uses Backbone.js HTML5 framework to build responsive Web applications and Cordova to package these applications for mobile devices. Back-end is done using Java and Tomcat, however evaluating or changing back-end technologies is out of scope of this research.

Development team consists of roughly 20 people – the exact number varies based on several circumstances – and is further divided into 3 smaller SCRUM teams. In turn the development team is part of a bigger corporation and this corporation sets financial limits and requires separate approval for bigger executive decisions. The three SCRUM teams do not have strict boundaries and might "lend" members to each other based on project requirements. All major decisions regarding technology choices and ways of working are made together for all the three teams so that development processes would stay as unified as possible.

One of the important considerations in choosing a framework is utilization of existing knowledge, so that costs for training team members or hiring new ones could be minimized. At the moment all the team members have knowledge and experience with HTML5. Even though only Backbone.js is used in development process, most team members have experience with other modern HTML5 frameworks, such as React and Angular, from previous employments, hobby and open-source projects. Three people have experience in writing native Android applications, one person has experience with creating iOS applications with Objective-C, one person has experience with Qt. Because the team in question is part of a bigger IT organization, it may be possible to utilize resources from other teams and get specialists in other technologies

to help out with training or aid in some part of project development, e.g. to create external native plugins or integrations.

Another problem to consider is the hardware used by developers. Currently 19 out of 20 team members use Windows machines with Linux virtual machines for development. If the team were to switch to a framework that would require different hardware, e.g. Apple machines, there would be considerable investment costs (MacBooks with similar computational power to the laptops currently being used start at 1500 €). On top of the cost aspect MacBooks are not supported by the corporation's technical support department, which would mean that team members would have to do their own technical support tasks, which would take time from development. Another problem is that MacBook acquisition is not officially supported and has its own process inside the corporation, and it can be lengthy.

Main product development focus is on Business-to-business applications. Most projects are mobile application clients for pre-existing B2B systems that have been produced by other business units within the company. The team is simultaneously working on several projects – each SCRUM team has at least one project in development phase, several projects in planning phase and around ten projects in support. For most of these projects, security is the main concern due to handling of sensitive business data. Delivery speed is a close second due to pressure from the management. Usability, user experience and UI design are not a high priority, because the B2B systems already have existing user bases and these users would effectively be forced whatever mobile client is offered. Due to this the executive decision is to focus on delivery speed instead seeing that potential users will not have alternative options available. Management has a tendency of prioritizing short-term goals over long-term ones and speed over quality.

User base needs to be considered when designing mobile applications and their UX. Most clients are Finnish firms and public sector organizations, e.g. municipalities and government organizations. In the case of Finnish public sector more than half of employees are women and the biggest age groups are 55-59 for women and 50-54 for men. [1] This means that the targeted user group is above average age compared to median smartphone user (the age group to spend most time on smartphone applications monthly in the US in 2016 was 18-24 years old [2]) and may have different smartphone usage habits and expectations for how an application should

work. This needs to be taken into consideration when designing UX. It is important to retain as much of UI of original PC system as possible, so that users can clearly associate the mobile client with its host system and intuitively figure out how to perform the same actions. This can be challenging because smartphone screens are considerably smaller than laptop screens: some functionality will have to be omitted from the mobile client completely. It is important to distinguish early in the planning phase which functionality is essential for the mobile client and which is not necessary, because users would prefer to access it on a bigger screen anyway. Platform guidelines also need to be considered so that mobile application looks unified with the rest of the mobile operating system (OS) and with other applications on the same device; however, budget constraints must be kept in mind since they might limit how much work can be put into the UI.

Often the development team has to work on several projects simultaneously; changing priorities from time to time based on external circumstances: requirements and cooperation capabilities of those units that own the source system. Project schedule has to adjust based on the state of readiness of the source system integration. At the same time there are several existing projects in support phase that are also part of team's responsibility. It is the team's aim to use the same technology stack for each project to avoid maintenance nightmares, where knowledge of tens of different technologies is required to support existing products, but due to varying functional and non-functional requirements between projects that is not always possible. Some projects can have different supported platform requirements (e.g. when the client happens to have a specific outdated smartphone model in use) or when other development teams participate in the development process and set requirements for the technologies being used in the project. However as much as is possible the focus has to stay on having as little technology fragmentation as possible between different projects. A situation needs to be avoided where only specific people know how a specific project works and if they are not available (reallocated or have changed positions) no one else is able to do work on a project without spending large amounts of time figuring out its architecture.

1.2 Solution to the described problem

The main objective of this master's thesis is to solve the problem presented above by answering the following questions:

- How do CPMDF differ from native frameworks?
- What benefits does usage cross-platform mobile development frameworks bring compared to using native frameworks?
- What drawbacks does usage cross-platform mobile development frameworks bring compared to using native frameworks?
- Which of existing cross-platform mobile development frameworks would be the most suitable for the specific use case described in this thesis?

The main goal of this research from the point of view of the development team is to assess the technology stack currently being used against other options and potentially find a framework that would improve the speed of projects' development phase, produce faster and more functional applications with better and more user-friendly UIs, provide support for at least Android and iOS platforms, have an easy learning curve, would capitalize on team's existing skills and knowledge and not be too expensive.

1.3 Research methods

A case study is conducted to find answers to the research questions described in section 1.2. According to Runeson [3], "Case study is an empirical method aimed at investigating contemporary phenomena in their context." Case study is the most appropriate choice when there is no possibility to conduct a controlled experiment [4].

A case study research consists of several phases [3]. First phase, design phase, defines the case and the research objectives. Seconds phase consists of collecting data for the research. In analysis phase this data is analyzed, and some conclusions are drawn based on findings.

Different kinds of evidence are collected for the case study. Gillham [5, p. 20] presents a list of main types of evidence. While these types are not strictly usable for a case study in Software Engineering, they can be easily adapted. For this case study the main types of evidence are the following: framework documentation and information provided by framework authors and

maintainers; existing research on these frameworks; developer experience with using these frameworks; developer community observation; characteristics and usability of demo software applications created with these frameworks.

Gillham [5, p. 15] also states the importance of context when conducting a case study: it is not only important to do a thorough literature review of the field and study related work, it is also equally important to get to familiarize oneself with the context of the case. When choosing a software developing method it is of utmost importance that the software development context be taken into account and that the method is tailored to that context [6]. Similarly, software development technologies need to take the context into account. In this case study context consists of the development environment, the business and non-functional requirements and the software development practices.

At the same time technologies used in development need to be compatible with the software development methodology in use, which in this case is agile development and SCRUM. With agile methodologies selected technology stack has to support fast development cycles and continuous deployments. From the point of view of agile development methods it is important that these core values are followed: valuing individuals and interactions over process, valuing working software over documentation, valuing customer collaboration over contract negotiation and adapting to change over following a plan [7]. Any potential software frameworks also have to adhere to these core concepts.

Singer et al. [4] describe collecting data in a case study: "Data collection is always performed with respect to a well-defined unit of analysis. In software engineering, the unit of analysis might be a company, a project, a team, an individual developer, a particular episode or event, a specific work product, etc." For this case study the chosen unit of analysis are all the CPMDFs that are marketed and available for public use, for free or for purchase. For evaluation of developer experience when using these frameworks, the unit of analysis is the single developer that tries each framework and does the development work with it.

Easterbrook et al. [4] stress that the important part of doing an empirical case study is ensuring the validity of the conclusions that are drawn from the research. One of the risks of this case study is the possibility of the researcher's bias towards specific technologies and programming

languages due to personal tastes and preferences. It is important to focus on consistency when conducting the practical part of implementing demo applications and to view the development process objectively. Some ways to ensure validity would be to use another developer for peer debriefing or as an external auditor [4]. Unfortunately, due to budget constraints that is not possible.

In practice this case study is conducted by collecting and researching the available information on different cross-platform development frameworks and comparing their main characteristic and features against each other. Key characteristic evaluation will be used to determine which frameworks would be the most beneficial for the presented case and development team in question. The currently used framework option will also be evaluated and compared against other potential options. If a suitable alternative is found, it will potentially replace the current technology stack. At the moment the development team is using HTML5 and Cordova for mobile application development.

Main characteristics to compare between different platform will be the following:

- developer experience
- programming language and developer tools
- learning curve of the framework
- modifiability (how easy it is to create features not directly supported by the framework)
- modularity (how easy it is to reuse code and pieces of the application in development of application for another platform)
- speed of development process
- look and feel of the UI and UX (how similar to native it is)
- responsiveness of the application (empirical evaluation)
- size of developer community and popularity among developers
- existing knowledge base
- pricing

After researching the main characteristics of all the currently available CPMDFs a short-list will be formed. The short-listed options will be put under more in-depth study. To assess framework

usability from the developer perspective and to evaluate steepness of the initial learning curve, a demo application will be implemented using each of the short-listed frameworks and the resulting applications and development experiences will be compared.

End product of this research should be a list of all or most options in cross-platform development frameworks, a short-list of options that seem to best fit the use case and development teams' current goals and business plan with a more in-depth look into these options, and a recommendation for whether or not to switch to a new framework.

1.4 Scope

This research will focus on the two main mobile platforms: Android and iOS. Windows Phones are no longer manufactured as of 2016 and Microsoft's support for Windows Mobile 10 will end in 2019 [8]. Microsoft released the last device that runs a Windows 10 Mobile operating system, Lumia 950, in November 2015 [9]. It's successor, Lumia 960, planned for release in 2017, was cancelled [10]. Even though there are more Windows Phones still in use in Finland, than there are elsewhere in the world, due to loyalty to Nokia and preference for locally produced smartphones, the number of users for this platform is too small and reducing constantly; the development and especially support costs for an extra mobile platform support are too high.

1.5 Structure of the thesis

Following this introduction section, the thesis consists of the following sections. The second chapter is the literature review which presents literature related to the topic of mobile applications and mobile development and goes over related research of comparing different CPMDFs. The purpose of this chapter is to familiarize with the topic and to research existing studies that can serve as a pillar for this case study. The third chapter focuses on listing frameworks options and their key characteristics, evaluating different frameworks, describing the demo application implementation process and evaluating this process. The fourth chapter discusses the advantages and disadvantages of researched approaches and compares these results to other related studies. The final chapter offers conclusions that have been made in this research.

2 LITERATURE REVIEW

This chapter examines existing literature related to the topic of CPMDFs. Information comes from both scientific and non-scientific references. Section 2.1 introduces mobile and smartphone device history and evolution. Section 2.2 describes the current state of mobile applications. Section 2.3 introduces mobile application development and the problems that arise in that field. Section 2.4 presents the two main mobile application platforms that are currently used and briefly describes development process for these platforms. Section 2.5 describes the taxonomy of cross-platform development approaches. Section 2.6 offers information on popularity of different CPMDFs among developer community. Section 2.7 describes comparison and selection criteria for this case study. Section 2.8 goes through the available research on the topic of comparing CPMDFs.

2.1 Evolution of mobile devices

With smartphone devices becoming more wide-spread every year and gradually replacing personal computers not just for leisure, personal communication and everyday activities, but for business purposes as well, mobile application development market is growing faster than ever. According to Statista [11], in 2017 there were a total of 178.1 billions of mobile application downloads worldwide, in 2018 a total of 205.4 billions and by 2022 they predict this number to grow to 258.2 billions. It has become expected that a PC application or a Web service should also have a mobile application client so that users could seamlessly access the same data on the go.

Growing popularity of smartphones has opened the market of mobile software development to the public and created new business opportunities. Previously software development was mostly limited to mobile phone manufacturers and carries companies [12]. Now mobile software development APIs are open to the public. This has resulted in rapid growth in mobile software development.

Modern smartphones have come a long way from Ericsson's car phone that had to be connected with wires and poles to the telephone lines running overhead, to the invention of GSM and first mobile phones that were so big they were actually car phones, to wide-spread "handportables"

in the 1990s [13, pp. 22, 76-78], and finally to the smartphones of today that can do anything a personal computer can and even more. Mobile phones started out as semi-portable devices for making calls and very rapidly evolved into a multitool of seemingly unlimited possibilities.

Early mobile phones in the 1990s were expensive and thus only available to a small audience. By 2012 46% of Americans owned a smartphone [14]. In 2017, when smartphone sales slowed for the first time after growing for a decade, Hodgson [15] expects most people to have access to a smartphone or a tablet device. Smartphone sales picked up again with 455 units being sold just in first quarter of 2018 [16]. A major factor in growing smartphone popularity, apart from them becoming more affordable and publicly available, is the growing amount of available software, which has made smartphones versatile and multi-functional [17].

Smartphones are mobile phones with considerably more computational power and more features. They normally come with Wi-Fi connectivity, GPS capabilities, big amounts of internal storage, cameras, touch screens and other capabilities. They have multifunctionality and capabilities that used to be limited to personal computers [18]. There are less and less limits imposed by insufficient hardware as the industry grows and smartphones evolve further: modern day smartphones can even have more computational power than a lower-end computer. Connectivity issues and insufficient bandwidth are gradually becoming issues of the past with the expansion of high-speed mobile LTE networks [19], growing number of free public Wi-Fi networks and better roaming data access.

Consumers are putting more and more emphasis on investing in higher end smartphone devices and staying “up to date” with the latest models. Even if the old device is still functional it is replaced periodically with a newer model due to frequent release of faster and more functional devices. According to Xun Li [20], people upgrade their phones on average every 18 months. This is encouraged by provider contracts, aggressive marketing from the smartphone vendors and lack of long-term support for older devices. After a few years on the market older devices no longer receive operating system updates and may thus be lacking newer features and important security fixes. For example, currently the oldest supported iPhone is model 5S which was released in 2013. Most likely it will stop receiving updated by autumn of 2019.

People have come to rely more and more on smartphones. Landlines are no longer used for the lack of need – everybody has a mobile phone. Smartphones provide functionality that was unthinkable just a couple of decades ago: easy-to-use and accurate GPS and location sharing with other people, instant access to the Internet whenever you are, unlimited access to books, movies and other entertainment, instant communication, photo and video camera that fits in a pocket and huge amounts of data storage.

2.2 Mobile applications

Each smartphone platform has its own application distribution platform, e.g. App Store for iPhones and Google Play for Android phones. When these were first launched, many applications were distributed for free and funded with advertisements. In recent years users have gotten accustomed to the idea of paying for applications. First the applications were relatively cheap, but more and more software vendors are switching from a one-time-purchase-fee to a software-as-a-service model with recurring weekly, monthly or yearly payments. This makes mobile applications even more expensive to the end user. Figure 1 shows how the mobile software market keeps growing: global app store revenue was $ 74 billion in 2017 and is predicted to reach $ 139 billion by 2021.

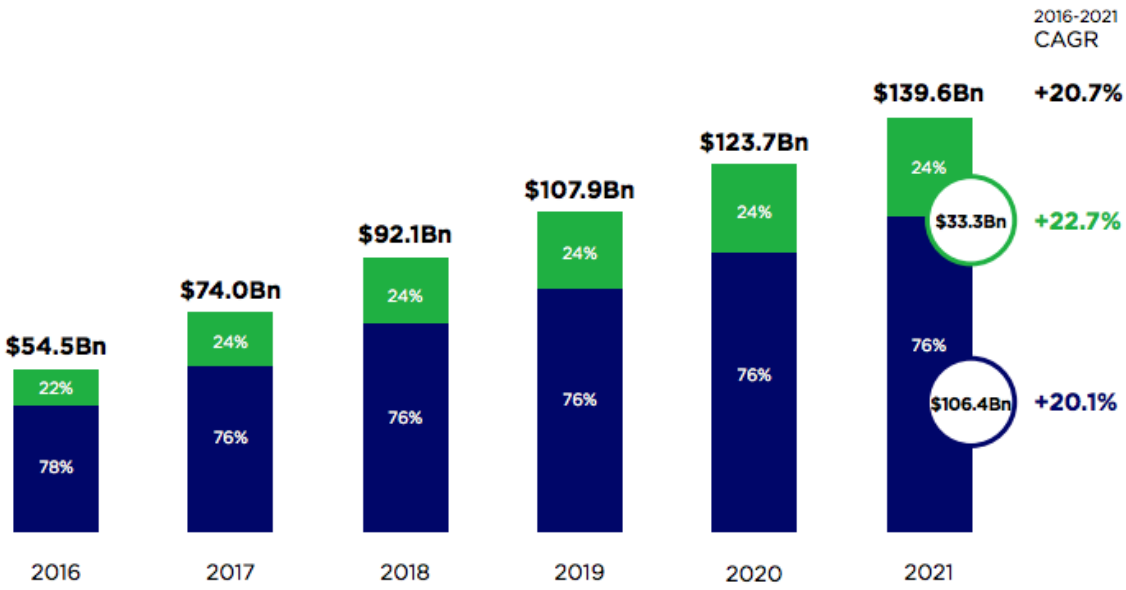

Figure 1. Global app store revenue [21]

Mobile applications have several key differences to traditional applications. For security concerns each mobile application runs in its own sandbox inside the phone's OS and can only access interfaces provide by the native SDKs, without direct access to hardware, the file system or other applications' data. This may create constraints for developers that cannot be worked around. Android ecosystem is more allowing regarding application access to the OS and the underlying hardware, while Apple is stricter and is known both for its better security and for greater restrictions in application development.

Another major difference between mobile and traditional applications is the smaller screen and different style of interaction. Instead of a mouse and keyboard smartphones are equipped with a touchscreen. According to Wasserman, "The mobile user interface paradigm is based around widgets, touch, physical motion, and keyboards (physical and virtual), rather than the familiar WIMP (Windows, Icons, Menus, Pointer) interface style" [22]. Smartphone screens range from 3" to 6" and screen space is very limited compared to even the smaller laptops, which start at 10".

2.3 Mobile application development and related concerns

Mobile application development has existed for more than 20 years, but its exponential growth started after Apple opened its App Store to the public in 2008 and the spike in smartphone popularity after Apple released its iPhone 3G in 2009 [22]. Since then many other manufacturers have entered the smartphone market with their own devices (Blackberry, Nokia, Windows Phones). They each use different operating systems and require different programming languages and tools for application development. This diversity in technologies and operating systems produces what Charkaoui [23] calls "fragmentation" in the mobile area. He states that "Recognizing the importance of defragmentation and wanting to optimize the design process of mobile applications, the idea of developing a single application that works everywhere (or almost everywhere) became a goal that was much more difficult to achieve – but remains as attractive as ever."

According to Wasserman [22], "From the standpoint of the application developer, it's quite expensive to support multiple platforms, especially when there are multiple versions and

variants of each of them". In worst case adding support for a second platform can double the amount of development work, even though the application and its functionality is essentially the same.

In some cases, e.g. when targeting a specific user group that might not be using state-of-the-art smartphones, it might be required to consider adding Windows Phone to the list of supported devices. Larger corporations can be slow to adapt to changes in the technology market and many large corporations in Finland were still ordering Microsoft (former Nokia) phones even after these phones had lost popularity in most other parts of the world and Microsoft was about to halt their production. At the time there was a big gap in prices between iPhones and Windows phones, and some companies refused to fill it with Android phones due to mistrust towards the open-source Android OS and related security concerns. Some companies were still ordering Windows Phones for their employees in 2017 due to the price difference compared to iPhones and supposedly better security compared to Androids. As a result, these phones might still be in use until 2020.

According to Abrahamsson [24], mobile applications are usually small-sized compared to traditional applications, they are usually less than 10000 lines of code. They are released in short cycles and usually at lower prices. Mobile development teams tend to be small.

2.4 Mobile application platforms

At the beginning of this decade there were five bigger mobile operation systems being used by consumers: Android, iOS, Windows Mobile, Blackberry and Symbian. In the past few years this number has shrunk to mainly two: only devices running Android and iOS operating systems are still being manufactured and sold. There is a number of pre-existing devices running other operating systems that are still in use, but this number is decreasing every year as older devices are replaced by new ones running Android or iOS. Figure 2 shows rapid decline in market share of BlackBerry OS after 2011 and of SymbianOS after 2012, and the current state of almost complete lack of market share for these two operation systems and Windows.

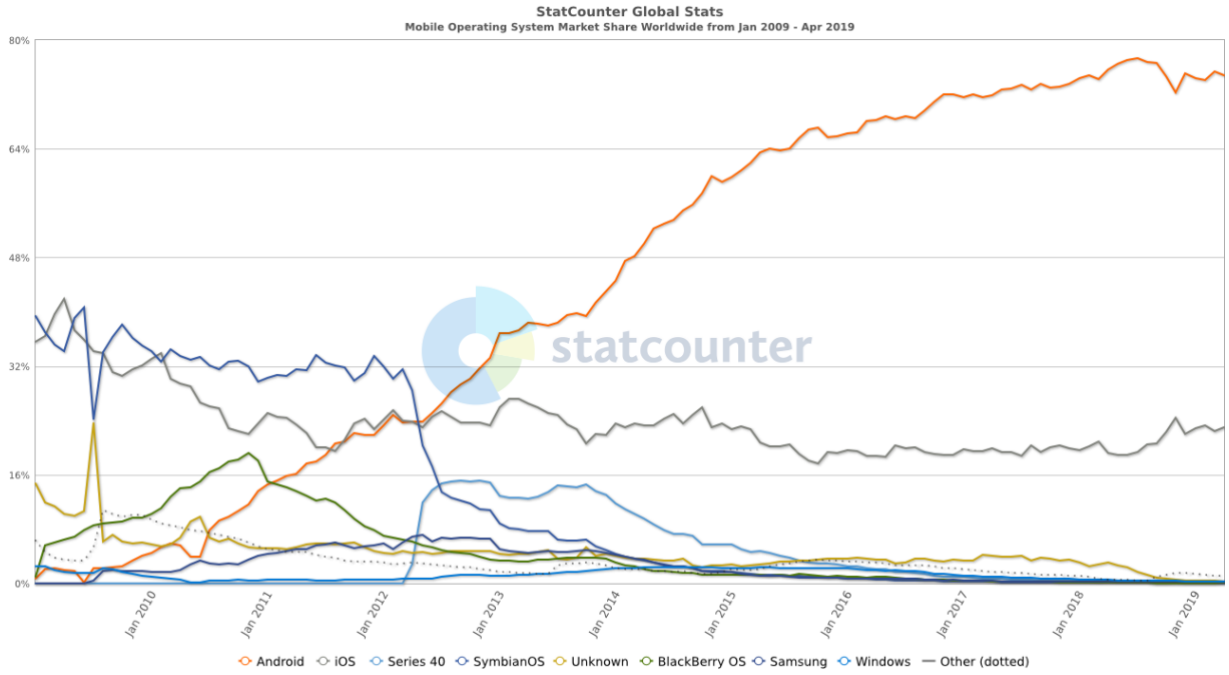

Figure 2. Mobile OS market share 2010 to 2019 [25]

According to Gartner's [26] report in 2017 Android and iOS operating systems account for 99.6% of all smartphones. Android has 81.7% of the market as opposed to iOS with 17.9%. However, these numbers are not really comparable, because unlike several Android phone vendors Apple is not present in the lower end smartphone market and does not compete with cheaper Android devices. Apple's cheapest smartphone is currently sold for $449 (iPhone 7) in the United States, while the cheapest Samsung smartphone (J3) costs just $150.

From the point of view of application development different mobile platforms have several major differences in terms of how development is done and what tools and languages are used [27]. Key differences are presented in Table 1.

Operating system	**Virtual machine**	**Program. Language**	**User interface**	**Memory management**	**IDE**	**Development on**	**Devices**
iOS	No	Objective-C/Swift	Cocoa Touch	Reference counting	Xcode	macOS	homogeneous
Android	Dalvik	Java/Kotlin	XML files	Garbage collector	Android Studio	Multi-platform	heterogeneous

Windows	CLR	C# and .NET	XAML files	Garbage collector	Visual Studio	Windows	homogeneous

Table 1. Some differences between several mobile operating systems [27]

2.4.1 iOS platform

iOS application development is done using Apple's Xcode IDE and Objective-C or Swift programming language. Swift is a replacement for Objective-C and is used for all the new applications, however older legacy applications and their extensions and plugins may still need to be developed in Objective-C.

According to Developer Survey by Stack Overflow [28] in 2019 the second most dreaded technology among developers is Objective-C. It was acquired by Apple in 1996 and has thus been in use for almost 20 years. In June 2014 Apple released Swift, a new modern language combining object-oriented, functional and imperative programming paradigms, as a long-term replacement for Objective-C for both macOS and iOS application development. Apple promised that Swift would be faster, more secure and easier to learn than its predecessor [29]. According to an empirical study on Swift's usage the programming language has become very popular in a short amount of time since its release and developers seems to find it "easy to understand and adopt" [30].

As can be seen from Stack Overflow Trends graph [31], Swift has surpassed Objective-C in popularity among developers in 2015 (Figure 3), less than a year after it was released. Interestingly, Objective-C had been on decline since 2011, even before the release of Swift. One possible reason for this could have been the number of emerging cross-platform development frameworks at that time (e.g. PhoneGap was released in 2009 [32], Xamarin was released in 2011 [33], Qt added iOS support in 2013 [34]). Another possible reason could have been the growing popularity of Stack Overflow and the growth of overall number of different technologies discussed on the platform, which would have caused each one individual topic to amount to a smaller share of all questions even if the number of questions on particular topic had stayed the same.

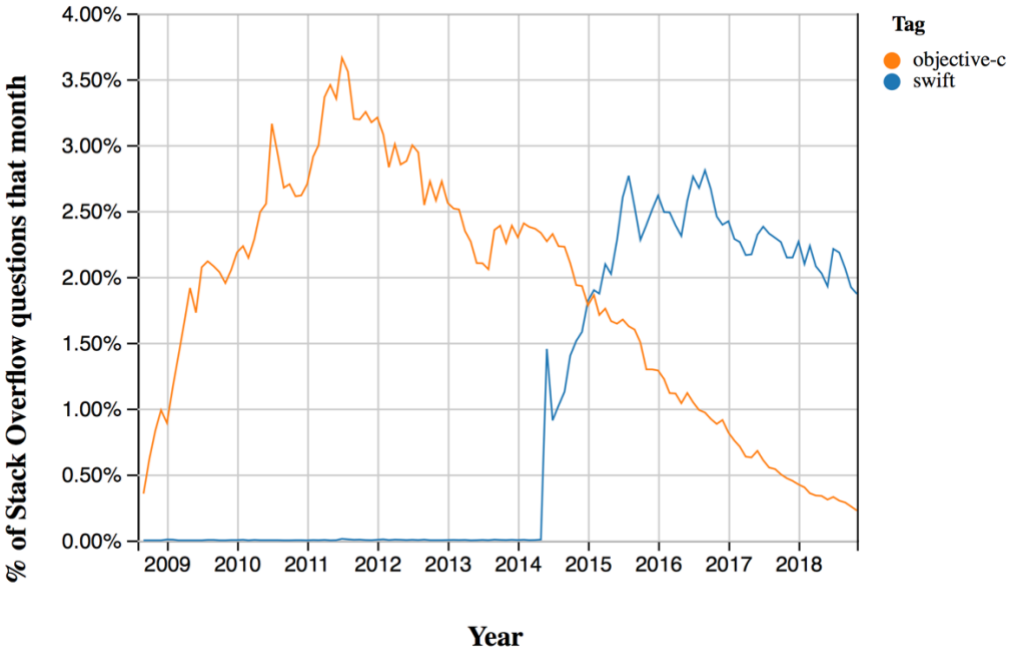

Figure 3. Stack Overflow trends of Swift and Objective-C in early 2019 graph [31]

iOS applications are distributed to end users through the App Store which Apple opened in 2008, shortly after the release of their first smartphone. All applications are manually tested and screened by Apple's review team before becoming available for download or purchase. This way Apple ensures not only a high level of security by identifying and rejecting all potentially harmful and malware applications, but also a high level of quality. From the end user's perspective bad user experience with App Store applications will be associated not just with the software manufacturer, but with the device and mobile OS itself, thus making Apple lose credibility on the mobile market. Application approval process can take a considerable amount of time, if Apple's reviewers have concerns and questions regarding application's capabilities. Every consequent update also needs to go through the approval process, but it is not as long and thorough for updates as it is for the initial release.

Participation in the iOS Developer program, which costs $99 a year, is required to release any applications built with Xcode on the App Store. Xcode IDE itself is available for free. It requires a macOS machine to run it. iOS application development is possible without opting into iOS Developer Program, since just installing Xcode gives access to all the IDE features, the UIKit

SDK and emulators for all supported versions of iOS devices, but without participating in the program it is not possible to install built applications on actual iOS devices or to release them to the App Store.

Xcode, which is provided by Apple, is the dedicated IDE for iOS development. There is a third-party alternative, AppCode, developed by JetBrains. It is part of JetBrains's bigger line of code editors (which Android Studio is also a part of), and their products are known to excel at code editing [35]. AppCode offers more customization options, better code completion and recommendations, better refactoring functionality and easier navigation through code and project files. It surpasses Xcode in line-by-line debugging capabilities, which Xcode is known to lack in. On the other hand, AppCode falls behind in support for some of the application UI editing features, such as Storyboards. AppCode requires a separate license at 199 € per developer per year. It should also be noted that even when using AppCode developer needs to have Xcode installed to access all the SDK features necessary for building an iOS application [36].

2.4.2 Android platform

Android OS originally derived from Linux and is open source. It is being used by several mobile device manufacturers, because Android does not have strict hardware requirements and can be run on almost any device. This leads to Android ecosystem being more complex than the iOS ecosystem because there are various devices from different manufacturers and different hardware combinations. Bigger vendors of Android phones, such as Samsung or Huawei, modify the core Android before using it on their phones, adding their own software on top, and ship phones with slightly differing versions of the OS. This leads to the problem of Android fragmentation [37].

According to a report on Android fragmentation from OpenSignal [38], there were over 24 000 distinct Android devices in 2015. The amount of different Android devices is illustrated in Figure 4. Of all Android phone vendors Samsung had the biggest market share at 43% and manufactured 12 out 13 most popular devices, but there are at least 10 other popular brands, such as HTC, Google, Huawei and others, and over a hundred smaller ones. An even bigger problem from the development point of view is the fragmentation in OS versions illustrated in

Figure 5 [38]. Developers are forced to support various versions of the OS simultaneously and deal with inconsistencies between devices from different vendors. OS fragmentation also creates problems for larger corporations that implement “bring your own device” policies, because of different security issues in different OS versions.

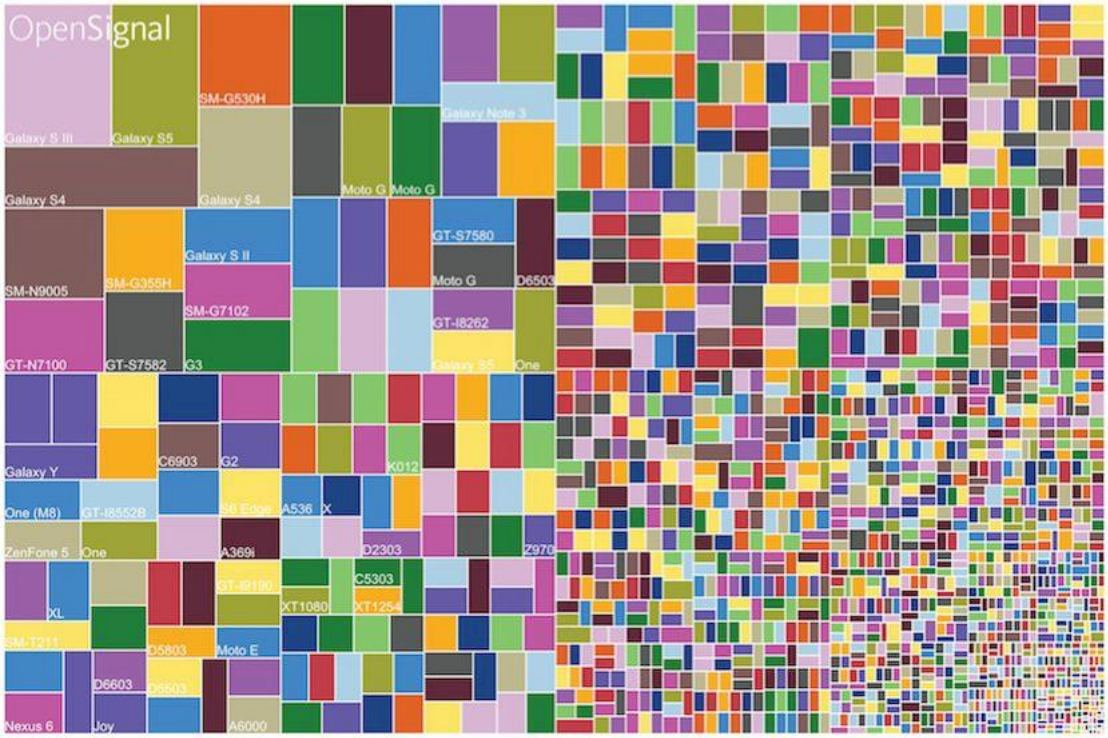

Figure 4: Android device fragmentation [38]

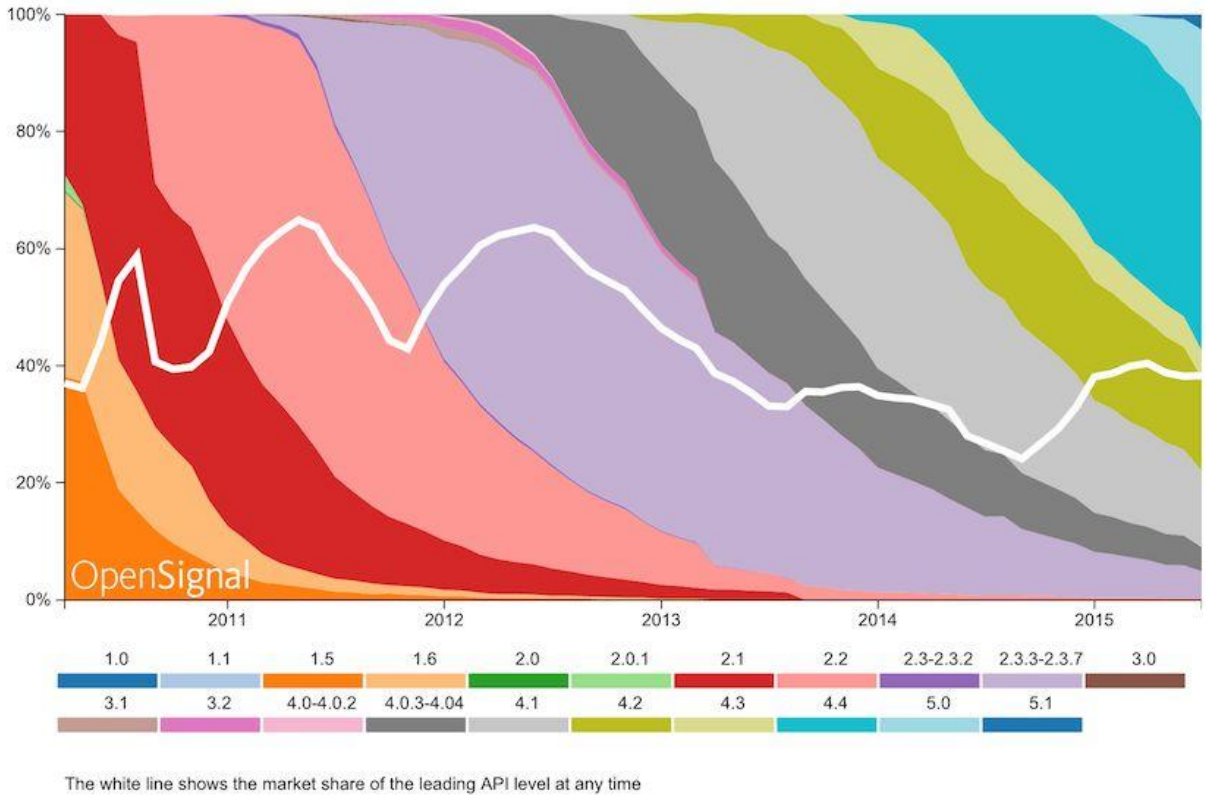

Figure 5. Android OS fragmentation [38]

Development of Android applications is done using the Android Studio IDE, which is available for free, Java or Kotlin as programming languages, and Gradle as a project build tool. Android SDK, which is a software component library for Android, is required to compile and build Android applications.

Java, which was first released by Sun Microsystems in 1995, is one of the most popular programming languages today. Java code is compiled into intermediate bytecode, which is executed by a Virtual Machine built-in into every platform that runs Java (the Java VM on Android is Dalvik) [39]. This provides the benefit of not having to recompile for every possible device that the application will be run on. Java is also popular for Android development, because it is fairly uncomplicated and does not have pointer arithmetic (as opposed to C language).

Kotlin is a relatively new statically-typed general-purpose open-source language initially developed by JetBrains and released in 2016; it later gained official endorsement from Google which increased its popularity. Kotlin was designed with a focus on interoperability, it works

seamlessly with Java and they can be used alongside inside the same project [40]. According to Madhurima Banerjee et al. [39], Kotlin is a better choice for Android development from a developer point of view, due to it being a more modern language, providing an easier development process, better prevention of bugs occurring in the code, being safer and more concise compared to Java.

Unlike with iOS release process, where distribution of applications is only possible through the App Store, building an Android project yields a binary, an APK file, which can be installed directly to an Android device. Android also has its own distribution platform for applications, Google Play. Similarly to Apple Store, there is a review process for all applications released to the Google Play Store, however the process is not nearly as thorough or strict as with App Store applications, and the amount of malware and software with security vulnerabilities is a growing concern on Android [41]. Google is trying to address this issue. For example, they released a "Google Play Protect" security tool that automatically scans users' phones for malware. To ensure that no malware is downloaded in the first place, they are vetting all Google Play applications [42].

2.5 Cross-platform development approaches

Clark defines a framework as a "reusable set of code that provides a collection of objects and functions you can use to get a head start on building your application. The main goal of a framework is to keep you from re-inventing the wheel each time you build an application." [43, p. 8] A framework also promotes a standard structure for all applications and guides developers into following best practices when creating code. In case of cross-platform development frameworks it is essential to have applications developed for different platforms follow the same structure. According to Riehle, "Frameworks are a central concept of large-scale object-oriented software development. They promise increased productivity, shorter development times, and higher quality of applications" [44].

Mobile software vendors want to target the largest possible number of users, which means targeting multiple mobile operation system, while simultaneously keeping development costs in check and project timelines reasonable. In conventional mobile development iOS applications

are developed natively using Xcode IDE in Swift or Objective-C languages; Android applications are developed using Android SDK and Java or Kotlin languages. The tasks of developing similar applications with the same functionality for these two different platforms are often split between different development teams, because these platforms require completely different technology knowledge [45]. This effectively doubles development time for the project, and also results in a lot of repetitive work being done, because the applications for different platforms share some of the business logic. There is a risk that insufficient communication between the two development teams will lead to inconsistencies in the functionalities between different versions of the same application.

Cross-platform mobile development frameworks (CPMDF) attempt to optimize the process of developing the same application for different platforms. Ideally such a framework would provide the means to develop one application that would run seamlessly on all the existing mobile platforms. In reality an ideal CPMDF has not yet been invented, nethertheless there are frameworks that do a good job of optimizing some parts of the development process. CPMDF's main purpose is to increase the speed of the development process by reusing some of the code between platforms. Figure 6 illustrates the difference in traditional and cross-platform models of mobile development: in the cross-platform model only one development cycle is needed to create applications for all the supported platforms [46].

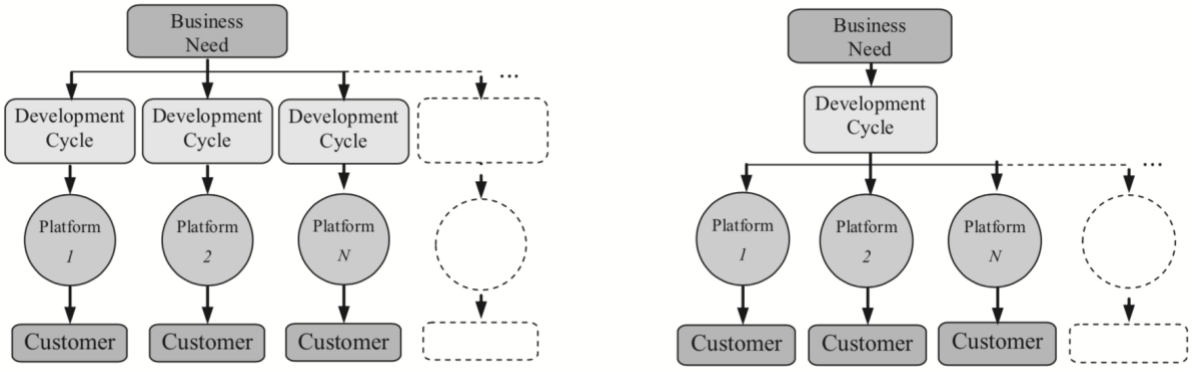

Figure 6. Traditional development model on the left and multi-platform development model on the right [46]

CPMDF can be categorized into 4 methodologies based on the approach they use to make the same code run on multiple platforms: Web, Hybrid, Interpreted and Cross compiled. Some of these methodologies concentrate on the application development phase, others on the execution phase. Each methodology has their own advantages and disadvantages. Mobile applications produced by these methods can also be categorized by the type of resulting application: the Web approach produces a Web page, the Hybrid approach produces a hybrid application, Interpreted and Cross compiled approaches produce a native application [45].

Web approach produces a Web application which is run in a browser and has a responsive UI designed for the mobile screens. It does not produce an actual mobile application, which would be distributed through respective application stores for each platform. Web applications are developed using HTML5, CSS and JavaScript and optionally any of the available Web frameworks (such as React or Angular). This is the fastest method, because the application only needs to be developed once and deployed once. Web application architecture is shown in Figure 7.

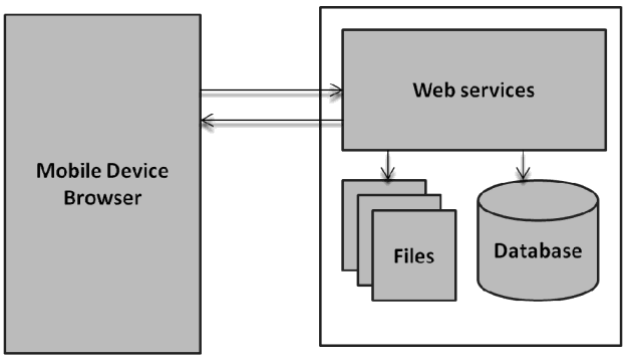

Figure 7. Architecture of a Web application [47]

2.5.1 Hybrid approach

Hybrid approach (also referred to as WebView approach) is a combination of a Web and native application. Like in the Web approach, the application is developed using HTML5 technologies, and it is executed by the device's browser engine in a full-screen WebView component. The device's native APIs are exposed to the application through an abstraction layer and can be accessed by the JavaScript code, however developers are limited by what APIs have been implemented in the abstraction. Application performance is very much limited by the

capabilities of platform's browser component. Application UI uses HTML5 components instead of native components. Hybrid application is packaged as a native application and distributed through the appropriate marketplace for each platform. Hybrid application architecture is shown in Figure 8.

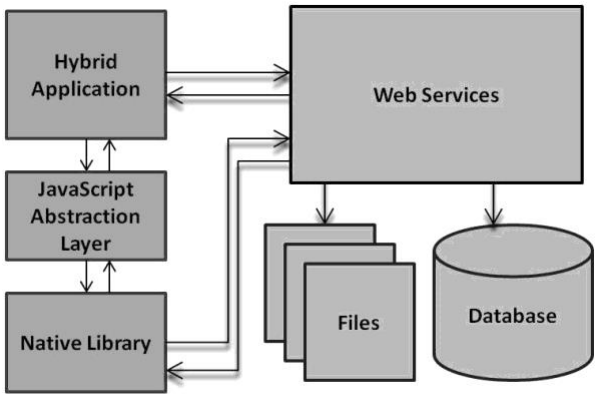

Figure 8. Architecture of a Hybrid (WebView) application [47]

WebView approach wins in speed of development due to several reasons. Firstly, any HTML5 stack can be used for the WebView approach. Picking the stack that developers are already familiar with will greatly improve development speed. HTML5 applications can be developed and tested in the browser and no platform specific modifications are necessary (although they are recommended to improve look and feel and to make applications blend in with the rest of the mobile OS). If the goal is to port an existing HTML5 application to mobile, it is fast and easy to do this with Cordova. Ionic shares similar functionality, but it is designed to be developed using Angular as the framework. The drawbacks of using HTML5 are the lack of native look and feel and slowness of the user interface and animations.

"WebView API enables developer to reap the benefit of web application in native mobile programming because not only allows applications to display web content, but it also enables applications to interact with the web content itself." [48] Web technologies and HTML5 is a fast and easy way to develop responsive applications with a wide array of features.

2.5.2 Interpreted approach

In the Interpreted approach application code is deployed to the device and executed by the interpreter at runtime. Native APIs of the device are accessible through an abstraction layer but are limited to what abstractions have been implemented in the framework. This approach allows using device's native UI components for interface and user interaction, which makes it possible to achieve a native look and feel. Interpreted application architecture is shown in Figure 9.

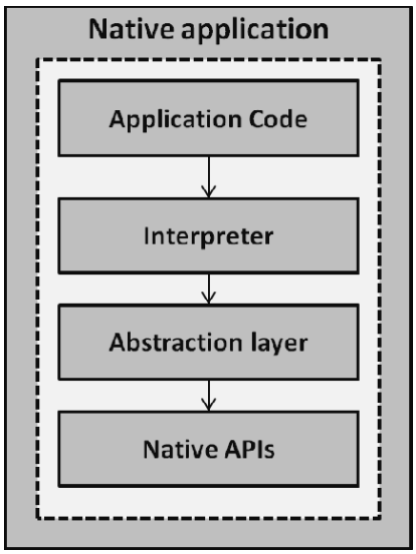

Figure 9. Interpreted application architecture [47]

React Native and NativeScript are slightly different implementations of the interpreted approach: they run JavaScript code inside a Virtual Machine and use it to call native APIs and access UI components. Implemented level of abstraction allows reusing a large amount of code, depending on how much emphasis is put on making the UI look "native". Overall this approach produces smoother and faster applications compared to WebView and allows to achieve similar development speed. Interface speed and smoothness is comparable to native implementations in simpler applications but decline when working with heavy graphics e.g. when developing games. While Interpreted approach would require writing a big portion of the application separately for each platform, it allows to do so in the same programming language thus reducing the need to learn multiple programming languages.

2.5.3 Cross compiled approach

In the Cross compiled approach, all versions of the application are created using the same common language and are then compiled into respective binaries for each platform. All the

native APIs and UI components are accessible, but this code is completely platform specific and cannot be reused, it has to be implemented separately for each mobile OS. This approach yields the highest performance of all the cross-platform development approaches, but also provides the least code reusability. Cross compiled application architecture is shown in Figure 10.

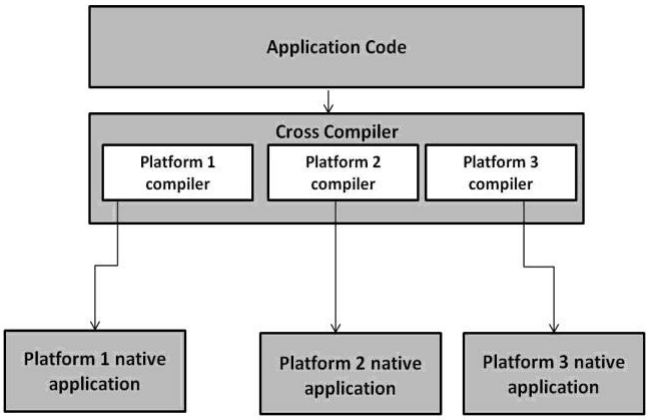

Figure 10. Cross compiled application architecture [47]

Xamarin and RubyMotion are examples of cross compiled approach. These frameworks allow developers to create both iOS and Android versions of their applications in a common language and then compile it into Swift or Java respectively. This approach allows reusing business logic code but requires for developers to write all of the user interface related code separately and to learn to use specific user interface libraries and SDKs for each platform. The result is 100% native application using native components.

2.6 CPMDF popularity among developers

Developer community, its size and activity are all important considerations when evaluating frameworks, programming languages or other development tools. A good community can be source of answers to questions and problem solving and can provide tools and best practices on top of what the actual framework provides [43, p. 12]. For this reason, when evaluating different development tools community support is an important factor to consider.

Stack Overflow is the biggest Q&A platform for software developers. According to Merchant et al. [49] Stack Overflow has almost 4 million unique active users and over 15 million existing questions. Developer community views Stack Overflow as a trusted and reliable source for

information. Normally Q&A Web-sites have answer rates between 66% and 90%, but on Stack Overflow over 92% of all questions are answered. The median answer time is only 11 minutes [50].

Stack Overflow is popular among developers because it is so effective: not only do developers get their questions answered in a short amount of time, their questions might have already been asked and answered; and its popularity is what makes it even more effective by attracting a bigger user base. Stack Overflow employs gamification with a point system that rewards users for asking, answering and rating questions, which improves participation rates and content quality. One of Stack Overflow users interviewed by Mamykina [50] compares Q&A platform to a popular computer game: "Stack Overflow — it's like World of Warcraft, only more productive."

Because many developers rely on peer communities for information and technical expertise and because a large number of these developers use Stack Overflow, the platform's data can be used to get an understanding of the popularity and the current trends in technologies and programming languages [50]. Stack Overflow has a tag system where each question gets tagged based on the topic of the question and the popularity data for these tags is openly available in Stack Overflow Trends.

According to Stack Overflow trends [51] Cordova has been the top Cross-platform mobile development framework and stayed consistently popular from 2012 to 2016 but started losing popularity in the past two years. Xamarin emerged in 2013, Ionic in 2014 and React Native in 2015. All of these new frameworks quickly became as popular as Cordova, React Native showing the steepest popularity growth. These tendencies can be viewed in Figure 11.

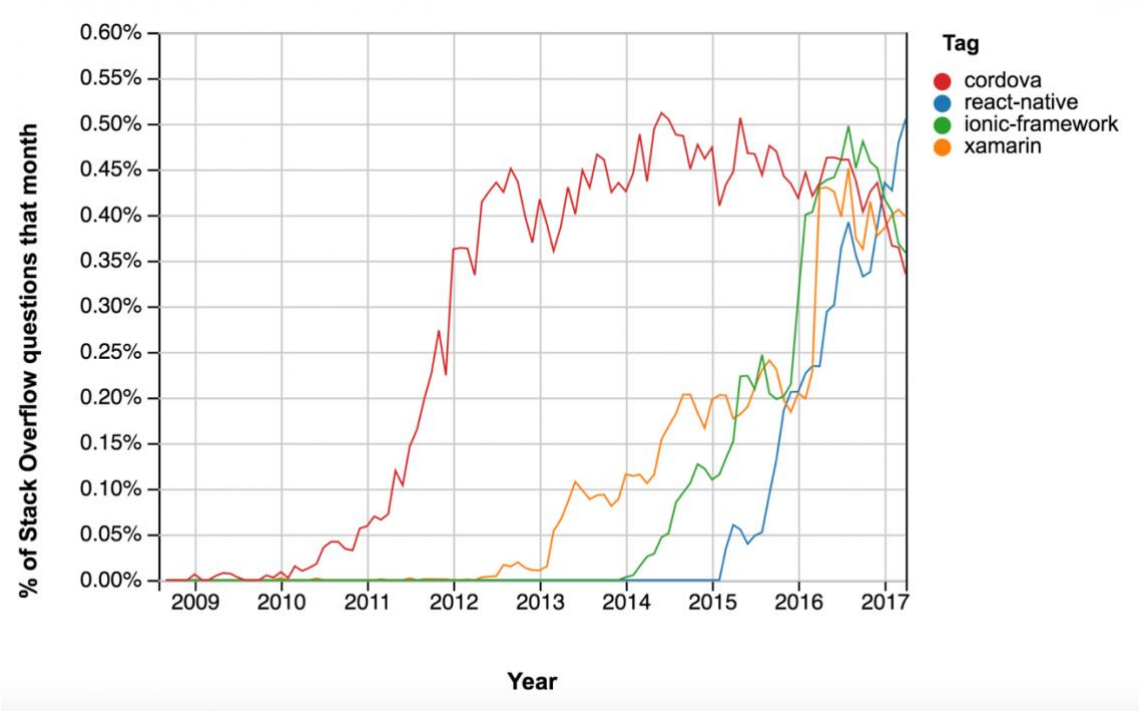

Figure 11. Stack Overflow trends of cross-platform mobile development frameworks in early 2017 graph [31]

As seen from Stack Overflow Trends graph from early 2019 (Figure 12) the main three cross-platform mobile development framework has kept their popularity level – with only Cordova gradually declining – while React Native's popularity has more than doubled since 2017. One of the reasons for such fast growth can be the growing popularity of React – the HTML5 framework that React Native is based on. However, that does not explain why Ionic has not seen the same level of growth, since it is based on Angular HTML5 framework, which has also experienced a fast growth, even surpassing React (Figure 13).

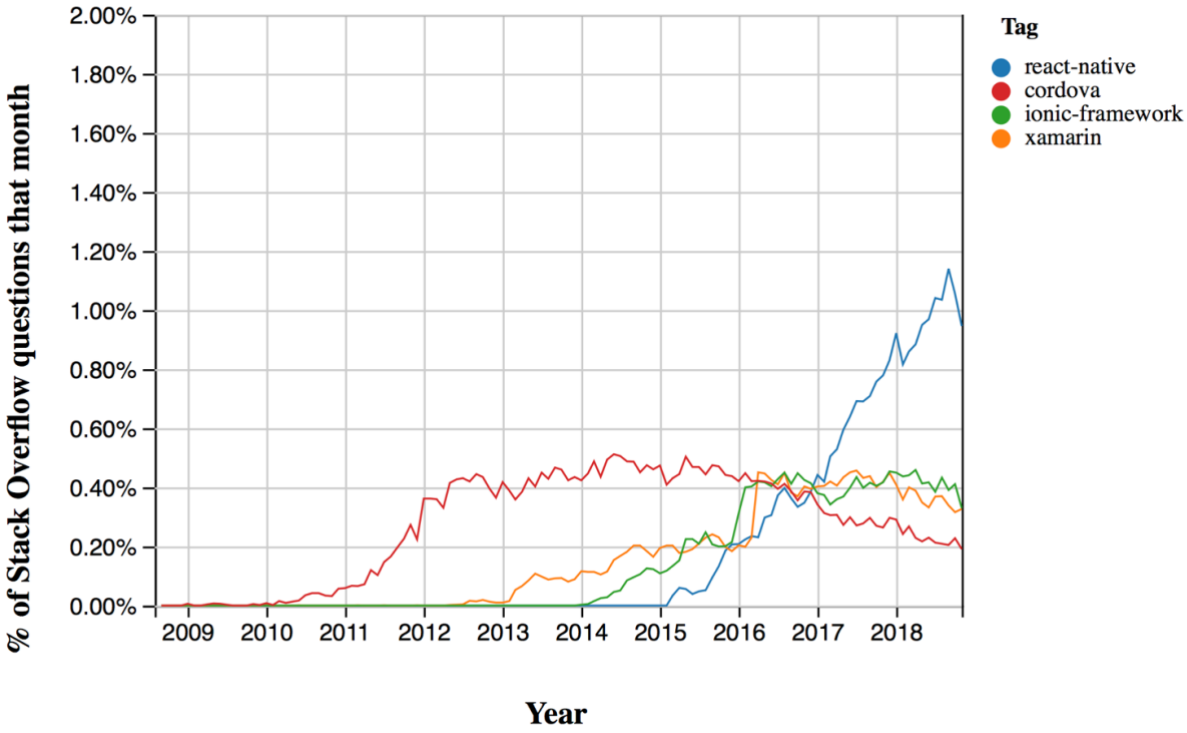

Figure 12. Stack Overflow trends of cross-platform mobile development frameworks in early 2019 graph [31]

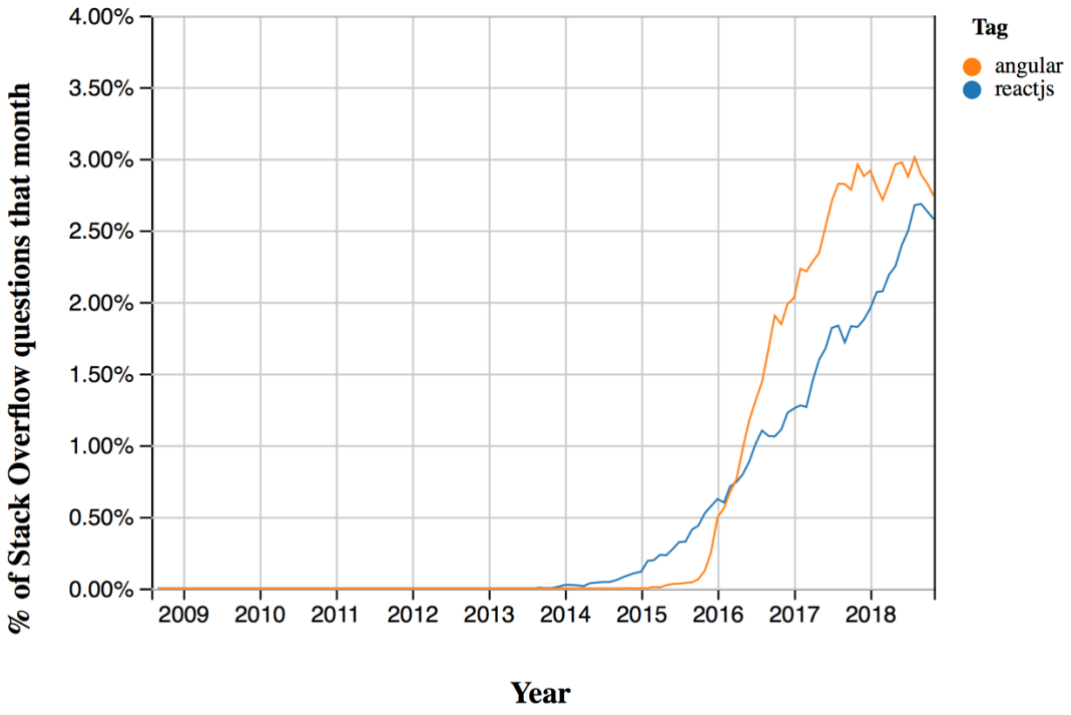

Figure 13. Stack Overflow trends of React and Angular in early 2019 graph [31]

Another possible reason for rapid growth of React Native's popularity is the change to its license that Facebook made in February 2018. Previously, despite being open source, both React and React Native have been distributed under BSD license that had a clause allowing Facebook to terminate the license and revoke rights of usage for anybody who had a patent dispute with Facebook. While this may not seem like a problem to any firm that is not planning to create a Facebook clone, it can cause problems in unexpected situations. For example, there is a possibility of Facebook buying a company that a firm has patent dispute with. Another possible problem can arise when a bigger corporation passes on buying a smaller startup, because they have a possibility of patent disputes with Facebook and do not want to buy applications written in React. In such a situation a company would immediately lose license to React and React Native (and other tools developed by Facebook) and be forced to rewrite all their applications using other frameworks. Recently Facebook has decided to put open source values above their commercial and legal interests and changed licenses of React and React Native to MIT.

Stack Overflow Trends measures the number of questions asked about each technology. According to David Robinson [52], Data Scientist at Stack Overflow, while measuring developer interest in different technologies based on number of questions asked has its problems, for example some technologies might raise more questions than others due to their complexity, data engineers at Stack Overflow have found that "it's a simple measure that gives useful insights into the developer ecosystem". He claims it is especially useful for measuring change over time.

It should be noted that growing interest in the developer community towards a specific technology might not always transfer to the business world. Bigger corporations are often slow to adopt to fast-paced changes in IT eco-system. Among business executives, who's approval is usually required for major technology decisions in software development firms, trends can be very different, because business managers are more conservative and reserved towards new technologies. Thus, Stack Overflow may not reflect real market value of using these technologies.

2.7 Comparison and selection criteria of CPMDFs

Dalmasso et al. [53] have identified the following key criteria which should be considered when choosing an appropriate cross-platform development framework: support of multiple mobile platforms, rich UI and UX, smooth back-end communication, security, support for app extensions, application battery consumption, access to built-in features and native APIs and open source code and developer community. Applications developed with different CPMDFs have their memory usage, CPU usage and battery consumption evaluated and compared.

Raj [47] proposes to consider choosing an appropriate cross-platform development approach based on application requirements. He suggests that for data driven applications the dominant approach is a Web application and for standalone and sensor applications native, Cross compiled or Interpreted should be used. WebView approach gets the middle ground: it is preferable, but not perfect, for each of these use cases.

Vilcek [54] evaluates CPMDFs using the following criteria: supported computer operating systems, supported mobile platforms, programming languages used in each framework,

documentation and community, speed and complexity of installation, complexity of development. He places the greatest emphasis on the number of mobile platforms that each framework supports.

Palmieri [55] coders several key factors when comparing CPMDFs: mobile OSs that the framework supports, developer licenses and their prices, programming languages used, availability and accessibility of native APIs, architecture of the framework and the provided IDE. He places the biggest emphasis on the number of supported mobile operating systems, even if these OSs are not widely used.

2.8 Related research

Corral et al. [46] look into advantages and disadvantages of using CPMDFs over native development tools. They consider three different points of view: those of platform providers, developers and end users and claim that the change in development paradigm will affect each of these groups. They see main advantages as bigger variety on the software application market, lower development costs and more competition between platforms leading to lower prices. Possible disadvantages are worse UX, more difficult support phase, imperfect development tools.

Redda [56] looks into benefits of using CPMDFs, formulates a method to evaluate and choose the best CPMDF, conducts the evaluation and considers the best tools. He concludes that CPMDF are changing rapidly and their maturity should be considered before taking any into use, however taking that into account he settles on Xamarin for Cross compiled choice and Appcelerator Titanium for Hybrid choice.

In another related research Corral et al. [57] analyze the performance of a PhoneGap application deployed to Android OS with the goal of assessing performance of web-based cross-platform approach. Performance evaluation is conducted by comparing execution time of different tasks and API calls between the PhoneGap application and a native application. Results show that web-based approach produced slower results in 7 out of 8 benchmarks compared to the native application. Research concludes that web-based CPMDFs have both advantages and

disadvantages and that choosing to use CPMDF is a strategic decision that should consider trade-offs as well as advantages.

In his research Lifh [58] investigates the possibilities of using React Native for cross-platform mobile development by implementing a clone of an existing mobile application using React Native and evaluating performance, functionality and code-base of the resulting application. The research found that most of the application can be implemented with a single code-base for both Android and iOS with only the Bluetooth handling requiring a separate implementation. The code-base ended up larger than each platform-specific application, but smaller than both combined. Performance-wise there was a small downgrade for Android, but a significant increase in CPU usage for iOS. The research concluded that React Native is very capable for simpler applications but could require significant amount of platform-specific code for more complicated use cases.

Abrahamsson [59] compares Android and iOS native applications with a similar one developed with React Native. Comparison focuses mainly on code-bases and modifiability of these applications, which were evaluated using the SQMMA-model. The research concludes that React Native has better modifiability, but also states that the data sample is too small and that it may be too early to draw conclusions based on it, because a larger code-base that would have been developed for a longer period of time is required for a thorough evaluation.

Adinugroho et al. [60] implemented and evaluated a mobile application which uses a variation of the WebView approach where the Web content is not packaged with the WebView wrapper, but is always loaded from the Web, which allows changing the content dynamically without having to publish an updated version of the application to respective application stores. Unlike in other similar studies Adinugroho did not use an existing WebView wrapper (such as PhoneGap or Cordova), but implemented the wrapper from scratch using native code. Resulting application was then compared against a native application and a WebView (PhoneGap) application. Adinugroho concluded that while WebView approach does not offer the same level of performance and UX quality as native approach, it does not fall far behind. He also noted that while dynamic application updates provide an advantage, it comes with the downside of bigger data transfer requirements due to having to always load the entire application over HTTP.

Charland and Leroux [61] compare the WebView and native mobile application development approaches. They note that the WebView approach supports some of the native APIs, but not all of them. However, the lack of support for some of the APIs is just a question of implementing bridging between the WebView wrapped JavaScript code and the native APIs. One of the notable problems that are yet to be solved with WebView is smooth scrolling. Charland and Leroux state that WebView approach trades some of the UI décor for a more pragmatic data-driven approach and that it is a business decision to either focus on the best possible UI or on pragmatic data representation. Their comparison concluded "The Web technology stack has not achieved the level of performance we can attain with native code, but it's getting close."

Willocx et al. [62] research various CPMDFs and compare performance of the same application implemented using native approach and ten different CPMDFs on Android, iOS and Windows platforms. Mobile application used for performance comparison is PropertyCross, which is a project aiming at helping developers select a CPMDF by providing code for the same application implemented with various frameworks. Willocx et al. measure and compare response times, CPU, memory and battery usages and disk space between different implementations; they do not compare application quality or UX. They state that cross-platform tools come with a performance penalty but agree that "The performance penalty resulting from the use of cross-platform tools is generally acceptable, especially on high-end devices". Most performance penalty was noted with Sencha Touch 2 and jQuery Mobile implementations. Research concludes that most CPMDF options are valid and that the decision of the most suitable option is mostly up to developer preference.

Nielsen [63] does an in-depth research of various cross-platform approaches and specific frameworks, comparing advantages and disadvantages of each approach. His work offers a very extensive list of available CPMDFs. A survey was conducted by Nielsen which showed that when asked to choose a mobile development approach over 80% of developers would prefer native approach over cross-platform approaches. His research includes creating an application with native frameworks and tools for Android, iOS and Windows platforms, and with Xamarin, and comparing development experiences.

Smutny [64] researches several CPMDFs as well as several HTML5 frameworks as alternatives to native development and look more closely into a practical case of using jQuery Mobile to implement a dictionary application. Resulting application is tested with Android and iOS tablet devices. Research concludes, that for a use case, where no native API communication is required by the application, HTML5 is a viable approach. Smutny states that "Audience, content and context of the mobile application are key indicators to decision for choosing the best configuration."

There exists a considerable amount of research on various CPMDFs. In some only the characteristics and capabilities are evaluated, others focus on measuring and evaluating performance parameters of the resulting applications, such as CPU load, memory usage, battery usage and application size on disk. Most research compare various frameworks not just against other frameworks, but also against native development options. Research differs widely regarding the specific frameworks being evaluated, unfortunately few of them focus on the most popular options for 2019, particularly React Native and Ionic. While research scope differs, conclusions are mostly along the same lines: CPMDFs provide multiple benefits in reducing development time in situations, where multiple platforms have to be supported. However, they come with performance penalties and the decision to use a CPMDFs or to resort to native development depends on applications requirements.

3 FRAMEWORK EVALUATION

This chapter contains research of characteristics, benefits and disadvantages of different CPMDFs, a short-list of the most promising candidates for adoption in the development process, and an objective comparison of these short-listed frameworks. A demo application is implemented with each short-listed framework; development process and the UI and usability of the end result are evaluated and compared.

Based on their feature lists, developer community sizes and projected future development these frameworks are short-listed: Ionic, NativeScript, React Native, RubyMotion and Xamarin. Cordova, the currently used framework, is also compared against these options.

Section 3.1 describes the demo application implementation process. Sections 3.2, 3.3, 3.4, 3.5, and 3.6 lists advantages and disadvantages of Ruby Motion, NativeScript, Xamarin, React Native and Ionic respectively, describes and evaluates development process of the demo application and offers developer's reflections on the development process. Section 3.7 describes Cordova framework's advantages and disadvantages. Section 3.8 lists some other CPMDFs that were not considered for the main comparison. Section 3.9 offers a comparison of the main frameworks. Section 3.10 concludes the comparison by offering the most promising option based on presented data and research.

3.1 Demo application

A small "Hello World" styled application is implemented using each of the short-listed frameworks. It is a simple to-do application with a task list, and it has the following features:

- Showing a list of tasks
- Adding a new task
- Marking a task as complete
- Deleting a task by swiping
- Filtering of task by user input

All the task data is fetched from a Mongo database using a simple REST backend which is protected by basic authentication, where hashed credentials are sent with each request in the

request headers. Tasks and their statuses are fetched using a GET request, updated or deleted with a PUT request and created with a POST request.

The application consists of one view which displays a list of tasks. Tasks can be completed by tapping them, completed tasks are highlighted with green color. Each task list element can be swiped from right to left to delete the task. User controls include a button to add new tasks, pressing it opens a prompt where user can enter the text; a search input which filters displayed tasks.

Resulting application is built into an Xcode project using the frameworks' own build tools. Xcode project is then compiled into a binary and run on an iOS emulator. Resulting demo applications are also tested on an actual iPhone device to evaluate speed, UI responsivity and UX.

Source code for each demo application is available in a public git repository. URLs for each repository can be found in Appendix 1.

3.2 RubyMotion

RubyMotion is a cross-platform development framework supporting iOS, macOS, watchOS (Apple Watch) and Android application development. It works with Ruby language which is compiled into each platform's native code. RubyMotion does not require any specific IDE (Integrated Development Environment) and is entirely driven from the terminal command-line prompt.

3.2.1 Advantages

Regardless of target platform all development is done in a single language – Ruby. This allows for a shallower learning curve when learning to develop for multiple platforms, since there is no need to learn both Java/Kotlin and Objective-C/Swift, instead it is enough to just learn Ruby to start creating applications for several platforms. Also, Ruby is considered easier to learn than Objective-C or Java. This is in part because Ruby was designed to be human-readable and in part because unlike the other two languages Ruby is a dynamic language (scripting language)

and these are considered more modern [65]. Ruby is considerably ahead of Objective-C because it has fully implemented memory management.

Using the same language when developing applications for different platforms allows reusing business logic related code, which also helps reduce overhead when testing business logic. This can help save on development and maintenance time and costs and reduce the number of software bugs.

Ruby Motion does not require any specific IDE and is fully operated from the terminal using command-line tools. This allows developers to choose a text editor or IDE of their preference, which has a positive impact on the learning curve. RubyMotion comes with a testing framework integrated into every project.

3.2.2 Disadvantages

Even though applications for all the supported platforms are developed using the same language – Ruby – their UIs are created using separate different components and native APIs. This means that while business logic code can be reused, UI code cannot. Instead the UI of each applications needs to be developed separately, and knowledge of native APIs and UI SDKs is required by developers.

Because the UI code for different platforms is completely separate, each application will have to be developed as a separate RubyMotion project. This creates repeatability in the development process and causes overhead when sharing business logic code between applications. Business logic will have to be extracted into separate modules and versioned and maintained separately from the applications themselves.

Another problem with RubyMotion is the lack of intermediate and advanced level documentation. RubyMotion has a detailed documentation for getting started and working with basic framework features, but it is lacking concerning more advanced features required to build complex applications.

Even though all the code is done in Ruby, developers are required to have knowledge of each framework's own UI component APIs. For example, to create even a simple iOS application

developer has to use UIKit and its components. Apple has documentation for these APIs available for developers, but this documentation is for Swift or Objective-C and not for Ruby, and it can prove problematic adapting it for RubyMotion.

There are not many existing well-known mobile applications that are developed with RubyMotion. It is a good thing for any framework to have a big company backing it and driving further development. In case of RubyMotion there are only smaller "indie" applications using it that do not have financial or developer resources to influence development of Ruby Motion and to contribute to the community.

RubyMotion is slow to adapt to changes to iOS and Android frameworks. When Apple or Google make changes to their frameworks or add new features, developers using RubyMotion have to wait for the framework to be updated to be able to use these new features in their applications and to update to the newer versions of operating systems.

3.2.3 Demo application

A fully-working demo application with all the planned features was implemented using the free starter version of RubyMotion. Table 2 displays a rough breakdown of how much time each part of the implementation took. Screenshot of the demo applications is shown in Figure 14.

Task	**Time (h)**
Setting up the project	1
To-do application business logic	4
Getting an understanding of iOS UI components	3
Total	**8**

Table 2. RubyMotion demo application development time breakdown

List of implemented features:

- Fetching a list of tasks from the backend
- Adding a new task (request to the backend, UI update)
- Marking a task as complete (request to the backend, UI update)
- Filtering of tasks

- Deleting tasks by swiping (request to the backend, UI update)

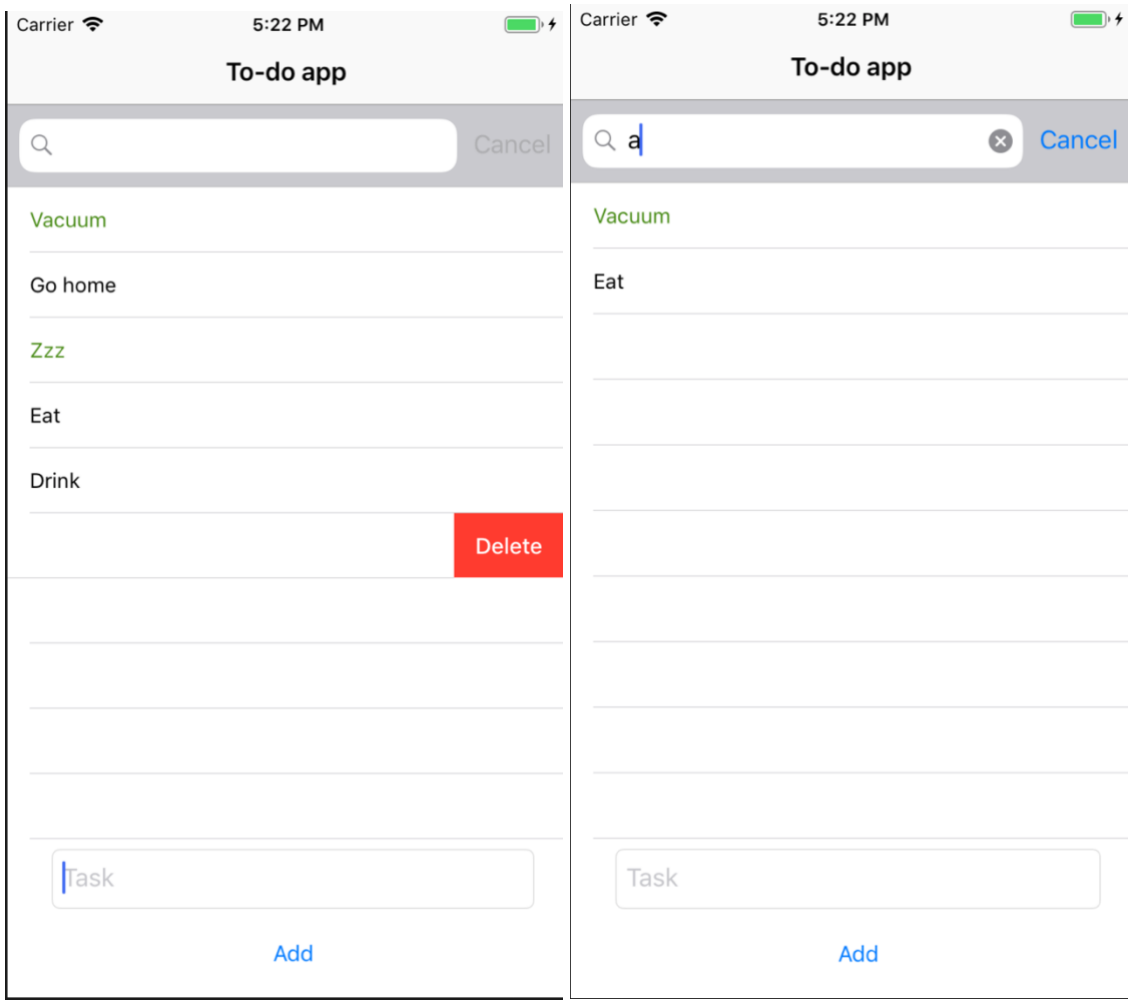

Figure 14. Screenshots of Ruby Motion demo application

3.2.4 Reflections

Ruby as a language was easy to jump into despite having very little previous experience with it. Ruby seemed to have a shallow learning curve. However, implementation of the UI related code proved to be more complicated: it required in-depth knowledge of iOS SDK and UIKit. Using search engines to find answers and usage examples also proved tricky, because most of the available information was for Swift language instead of Ruby. Overall developer experience proved positive. Ruby's own command line build utility (rake) proved to be highly usable.

Resulting application was indistinguishable from a native iOS application: its UI was composed of all the same UIKit components. The application ran as fast and there were no issues with user interactions.

3.3 NativeScript

NativeScript provides support for iOS and Android platforms. It uses the Interpreted approach. Code is written in JavaScript and executed at runtime. Unlike React Native it is not tied to a specific framework: it can be used with plain JavaScript or any HTML5 JavaScript framework. NativeScript's documentation suggests that using Angular is a popular option and they offer a separate documentation specifically covering the Angular use case. NativeScript also supports the usage of TypeScript in place of regular JavaScript (TypeScript is a typed version of JavaScript that compiles into regular JavaScript).

NativeScript is aimed at not having to write duplicate code. Their marketing slogan is "Write once, use everywhere". When creating an application for both Android and iOS only one code base is required and only one UI has to be created, which is then adapted by NativeScript to run on all platforms. This is beneficial for saving development time and costs and simplifying application maintenance, but the end result is not as smooth and fast as with native UIs for each platform.

NativeScript is a fairly developed tool with a few big backer products such as SAP. However, it is not as known and not as widely used as React Native. Its community is smaller. As a result, it has less open source libraries, less support and less help offered when running into problems during development.

3.3.1 Advantages

NativeScript provides developers with good CLI tools. Developers can easily create, build, test and deploy applications using the command line.

Developing for both mobile platforms with only one code base is a great advantage. It saves development and testing time, and greatly decreases maintenance costs. Many of the features,

especially bug fixes, can only be tested once instead of coding and testing them on both platforms separately. This improves time to deployment and improves maintainability.

NativeScript framework offers a feature called LiveSync. It is a tool for speeding up application development. It monitors changes to the source code and updates the application running on development server with the changes made to the code in real time. This greatly increases speed of development and the overall development experience.

Because the UI code is implemented using JavaScript, and not restricted by usage of native APIs, developers have freedom to choose UI libraries of their preference. Any of the many available JavaScript UI libraries can be used, including the popular ones, like Bootstrap. NativeScript also allows customizing UIs for each platform to create UIs that blend in better with the OS and its look.

3.3.2 Disadvantages

At the moment NativeScript does not offer support for debugging CSS and inspecting the DOM-tree. This complicates development of the user interfaces, because without the ability to debug it takes longer to find causes of problems and bugs. This sets NativeScript apart from React Native, which offers support to inspect the DOM-tree, and all the Hybrid approach frameworks, which offer full support of browser debugging and developer tools.

While LiveSync sound like a useful feature in theory, in practice it proved to be quite unreliable. According to both personal testing and reflections of other developers based on their experience, the LiveSync process is prone to crashing and often fails to start. At times it seems to run, but it does not update the application when the code changes. This can seriously hinder the development process, because it creates a situation where the developer does not see expected changes in the application and assumes the code is faulty, when in fact the code might be working, but the application has not been updated to reflect the changes.

Another problem with NativeScript is that even though it is supposed to work well with Angular, features that work in Angular might not work with NativeScript. This creates a problem similar to the one described above with LiveSync: developers rely on their existing knowledge on

Angular and assume that what they are doing should work and look for the cause of their code not working in the wrong place.

Some of the NativeScript components (the pieces that help implement native features for mobile platforms) are created and maintained by Telerik. Some of these components are not open source and are not free to use. They are also not well documented and are often hard to pair with the Angular version of TypeScript. Overall the number of available components to use with TypeScript is considerably lower than for React Native.

3.3.3 Demo application

A fully working demo to-do application was implemented using NativeScript, however certain native UI components had to be replaced with 3rd party ones. Table 3 displays a rough breakdown of time spent on each part of the development process. Application screenshots are shown in Figure 15.

Task	**Time (h)**
Setting up the project	0.5
To-do application business logic	4
Debugging problems with LiveSync	2
Debugging Telerik UI	3
Debugging Swipe functionality	1
Total	**10.5**

Table 3. NativeScript demo application development time breakdown

List of implemented features:

- Fetching a list of tasks from the backend
- Adding a new task (request to the backend, UI update)
- Marking a task as complete (request to the backend, UI update)
- Filtering of tasks

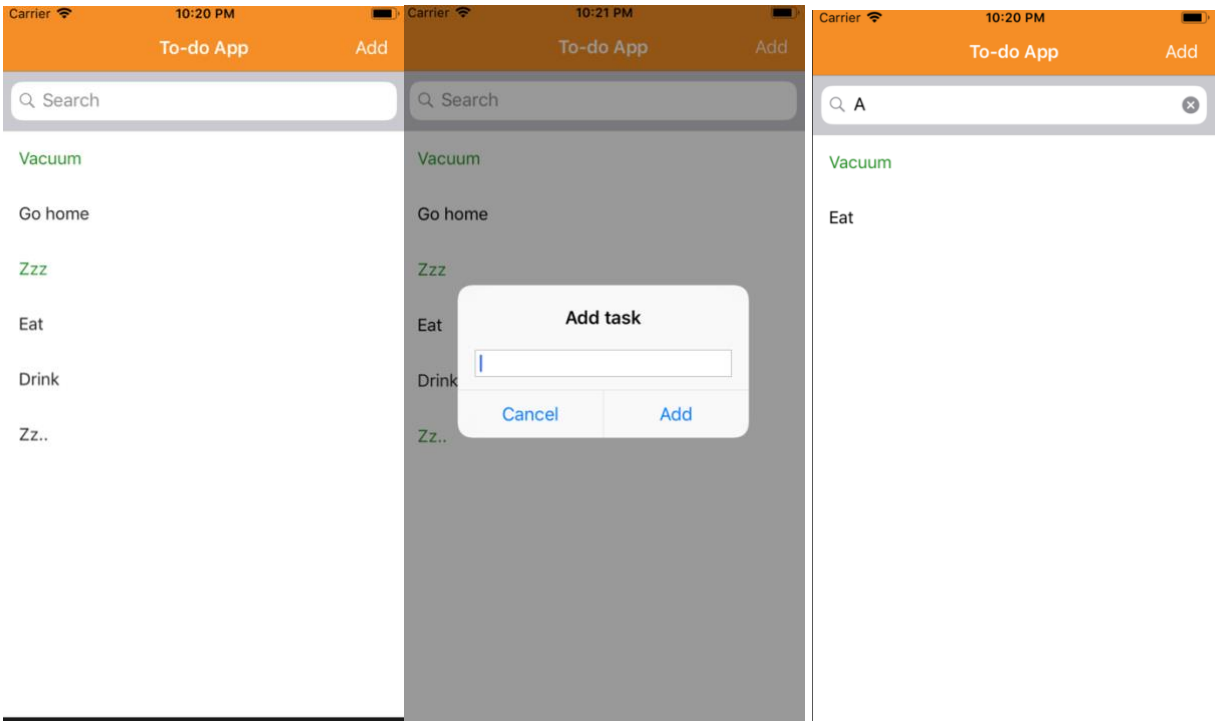

Figure 15. Screenshots of NativeScript demo application

3.3.4 Reflections

NativeScript proved easy to set up and get started. It had an easy to use installation script that downloaded and installed all the required dependencies and created an empty starter project. Online documentation describes this process in detail and the instructions are easy to follow. Package comes with a command line tool that takes care of compiling and running the application and recompiling and deploying necessary parts when code is changed.

However, a bug emerged during installation and caused problems when attempting to run the application. According to the NativeScript community this bug is solely related to npm version 5 and does not reproduce with other versions of the package manager, but it is nonetheless a big problem and is caused by NativeScript itself, not by npm. These issues made the process of initial project setup considerably slower.

Development process did not prove smooth either: there were problems when implementing swipe functionality on ListView components. NativeScript has support for UIKit components, including ListView, but the swipe functionality was not working correctly. The only solution

was to move from using UIKit components, which would have been the preferred way for a more native looking UI, and implement the list using different components.

Overall the resulting demo application run fast and smoothly on the iOS platform. There was no noticeable slowness or lagginess in the animations. The UI looked good and blended in well with the look of native applications, despite having to resort to using non-native components. The applications seemed better in quality and UX than its analogs created using WebKit.

3.4 Xamarin

Xamarin is a cross-platform mobile development framework that uses Cross compiled approach and that allows developers to create their mobile applications using Visual Studio as an IDE, C# language and the .NET framework. Code is then compiled into the respective native languages for iOS and Android resulting in a native application. Xamarin supports iOS and Android mobile platforms.

The idea behind this approach is that developers only need to know how to use one IDE, framework and programming language to develop for multiple platforms. A lot of the application logic code can be shared between platforms, and some of the UI code. This increases the speed of development process, allows for the same developers to work on both platforms and helps support and update existing applications.

However, each platform requires a separate Xamarin project, which makes sharing business-logic code cumbersome, and a lot of the UI has to be implemented separately. C# is used to access native UI libraries (UIKit for iOS and Android SDK for Android) and knowledge of their APIs is required by the developers.

3.4.1 Advantages

One of the advantages of Xamarin are the tools that come with the framework. Visual Studio is an advanced IDE which is popular among developers. Because the framework, the language (C#) and the IDE all come from the same vendor, they are very well integrated together. Visual Studio offers code completion for C# and Xamarin, visual UI designers, debugging tools, collaboration features and publishing tools.

C# is a mature modern language that has been around since 2000 and has been widely in use for more than a decade. It is the main language for Windows application development and is known and used by many developers, it is also considered easy to learn. It has a comprehensive documentation and a good resource base.

3.4.2 Disadvantages

Apart from the official Xamarin documentation it can be problematic to find good resources and code examples for Xamarin development. While resources on C# are in abundance, most information on native components and their usage is written for native languages, not Xamarin and C#. Adapting these code examples requires considerable programming skills and knowledge of both native APIs and languages (Swift/Objective-C and UIKit or Java/Kotlin and Android SDK). Acquiring good understanding of native tools might remove the need for using cross-platform development framework altogether.

Unlike some of the other tools considered in this case study, Xamarin is not free for commercial use. It has a free community license which can be used for educational and personal purposes, but for commercial development a perpetual license must be purchased. As of 2018 the price is $499 per user.

3.4.3 Demo application

A fully-working to-do demo application with all the planned features was implemented using Xamarin. Table 4 displays a rough breakdown of time used for implementing each part of the demo application. Application screenshots can be seen in Figure 16.

Task	**Time (h)**
Setting up the project	1
TableView implementation	2
Fetching data from REST	2
Edit/delete/complete functionality	2
Search functionality	1
Total	**8**

Table 4. Xamarin demo application development time breakdown

List of implemented features:

- Fetching a list of tasks from the backend
- Adding a new task (request to the backend, UI update)
- Editing an existing task (request to the backend, UI update)
- Marking a task as complete (request to the backend, UI update)
- Filtering of tasks
- Deleting tasks by swiping (request to the backend, UI update)

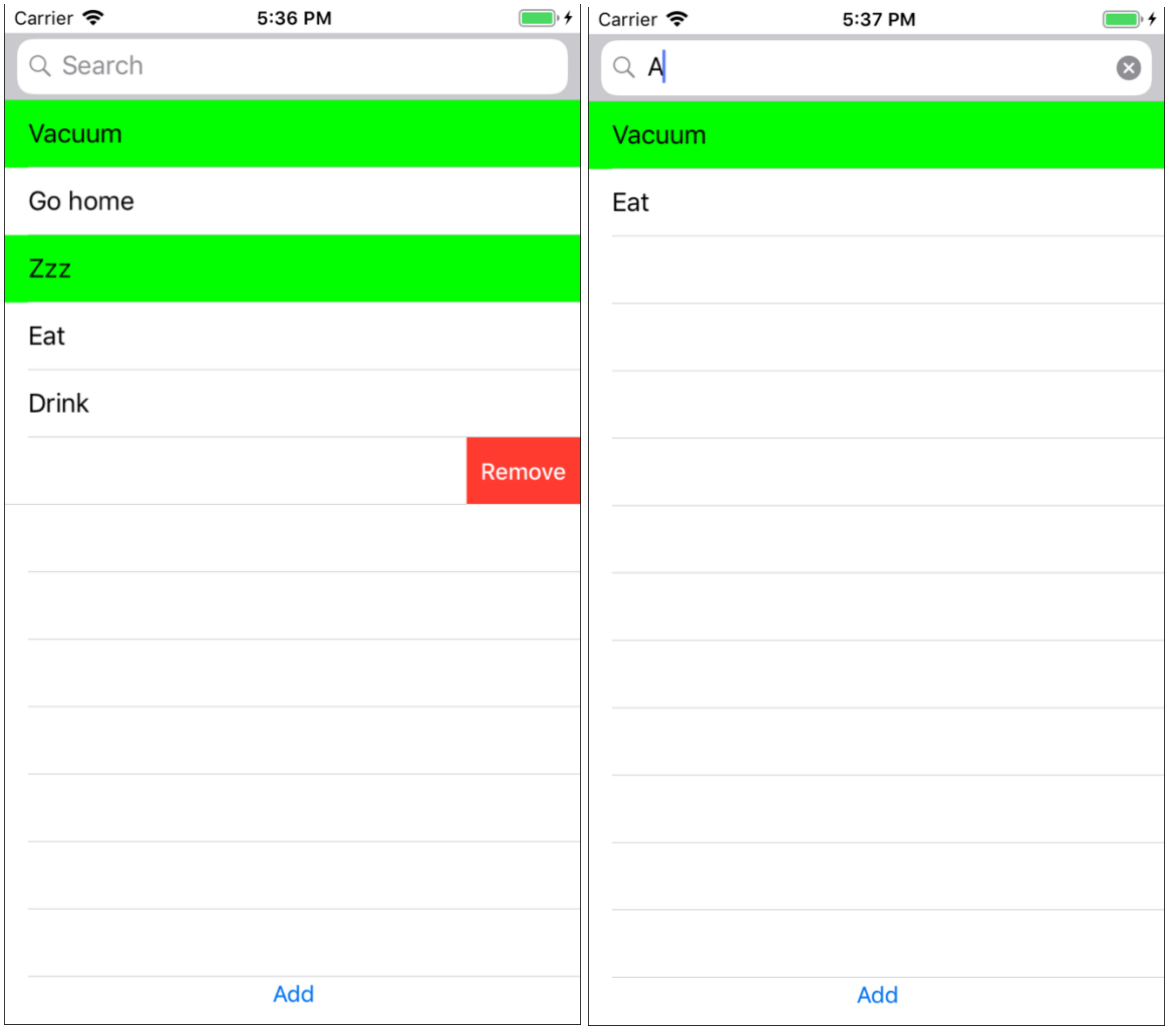

Figure 16. Screenshots of Xamarin demo application

3.4.4 Reflections

One of the advantages from developer's point of view is the use of C# language. It has built-in memory management and garbage collection, which makes it considerably easier than C++ or Objective-C. Also, C# is a relatively old and mature language and it is easy to find guides and documentation. It is widely used in many different frameworks and on different platforms. Microsoft provides a comprehensive knowledge center for C# and .NET framework.

Xamarin framework itself is more complicated to figure out and poses considerable challenges in finding information. For example, even with prior experience of writing C# it proved a complex task to implement a REST client, which took a considerable amount of time and required a lot of sifting through example code snippets.

It is even harder to seek information and examples for implementing UIKit components in Xamarin. There is a good knowledge base for UIKit provided by Apple, but it only offers examples for implementing them in Objective-C or Swift. Xamarin's own documentation and examples are lacking and it is up to the developer itself to adapt Apple's code snippets into C# language, which is often challenging and slow.

Xamarin offers a graphical storyboard editor for creating the application interface. Its aim is to automate some of the coding tasks by allowing developers to create UIs using drag-and-drop and visual tools. However, during the development process of the demo application the storyboard editor kept breaking and it took multiple tries to achieve the desired UI.

Implementing business-logic part of the application took more time in C# and Xamarin than in Ruby. This is especially alarming considering bigger pre-existing knowledge of C# than of Ruby. It is a sign of a considerably steeper learning curve for Xamarin and can be a big red flag when considering taking it into use due to the higher time and money costs for initial learning process. Overall the whole application took less time to implement with Xamarin than with RubyMotion, because of the familiarity with required UIKit components after having implemented them with RubyMotion.

3.5 React Native

React Native was created by Facebook in an attempt to make native application development flow as fast as Web application development flow. When doing Web development any code changes can be instantly reflected in the browser with tools like LiveReload. Native applications however require complete recompiling after every change.

Facebook's solution to this problem was to use React in the same fashion as in the Web, but instead of rendering DOM elements it renders higher-level platform specific components, e.g. UIKit components on iOS. React Native applications run a JavaScript engine that communicates directly with native APIs.

While the holy grail on cross-platform mobile development is the possibility to "write once, run anywhere", some are shifting from that goal after finding it unachievable [66]. Facebook's approach with React Native is "learn once, write anywhere". Their aim is not to create an application that can run on any platform, but to create a set of tools that would make it possible to easily create applications for each platform. While a basic React Native application with low emphasis on native-looking UI could achieve up to 100% code reusability between iOS and Android, performance-wise and design-wise it is usually better to use native components on each platform and this would require writing separate platform specific code.

React Native showcase is impressive with widely known applications. On top of Facebook's own Instagram and Facebook clients, there are also such applications as Skype, Uber, SoundCloud Pulse and others [67].

3.5.1 Advantages

React Native provides great development tools that put it one step ahead of other frameworks for cross-platform mobile development. When running a React Native application in development mode, a packager process monitors file system for source code changes and updates the application, which is running in a mobile emulator. It is a tool similar to NativeScript's LiveSync, but it is more stable and reliable. LiveReload and HotReload functionality can be toggled on or off. With HotReload enabled changes to the source code are compiled on the fly and UI changes can be seen in real time without the need to restart or refresh the application manually.

Other useful tools in React Native are the ability to view JavaScript console output by attaching a Chrome debugger to the application. React application can be inspected using React Developer Tools that shows component hierarchy and state for all components.

Another benefit of React Native is that it makes it very easy to create mobile UI. Using JSX and composing the layout in a HTML-like syntax is very intuitive for Web developers. React Native provides good documentation on available components and their APIs.

React Native has been around for several years and it already has a relatively big number of component libraries created for it. This simplifies application development: instead of creating new components for various interface elements, it is possible to use existing libraries that are already styled according to platform guidelines.

With React Native applications all of the business logic code can be shared between platforms. Some of the UI code can also be shared: it depends on the end goals for the application. If the goal is to create an application that looks and works the same way on both Android and iOS, most of the UI code can be reused. However, it is often better to follow native look and feel guidelines for each platform, in which case separate UI code will have to be written.

3.5.2 Disadvantages

Previously, the biggest problem with using React Native for commercial development was its license. Both React and React Native were shared under BSD + patents license by Facebook, which contained a clause with the possibility to remove access. Since recently this clause has been removed and React Native is now shared under MIT license with no restrictions.

Another problem with React Native is that JavaScript knowledge is not enough: knowledge of native programming languages is required if using native modules for heavy computing or implementing UI components that are not available as part of React Native. However, this aspect can also be viewed as an advantage of React Native: the ability to build with native programming languages on top of the JavaScript code provides great flexibility. Any missing pieces that are not implemented as part of the original framework can be implemented with native code and integrated into the application.

The problem of lagging behind the changes to native APIs comes up with all the cross-platform mobile frameworks including React Native. When new versions of the native APIs are released it takes some time for the framework to adopt these changes.

3.5.3 Demo application

A fully-functional to-do demo application was implemented using React Native. The application uses native UIKit components (ListView and popup prompt) for a native looking UI. Table 5 displays a rough breakdown of time used for implementing each part of the demo application. Application screenshots can be seen in Figure 17.

Task	**Time (h)**
Setting up the project	2
Implementing REST API	1
Implementing ListView component	1
Updating ListView	2
Popup for creating new task items	1
Swipe functionality	1
Total	**8**

Table 5. React Native demo application development time breakdown

List of implemented features:

- Fetching a list of tasks from the backend
- Adding a new task (request to the backend, UI update)
- Marking a task as complete (request to the backend, UI update)
- Filtering of tasks
- Deleting tasks by swiping (request to the backend, UI update)

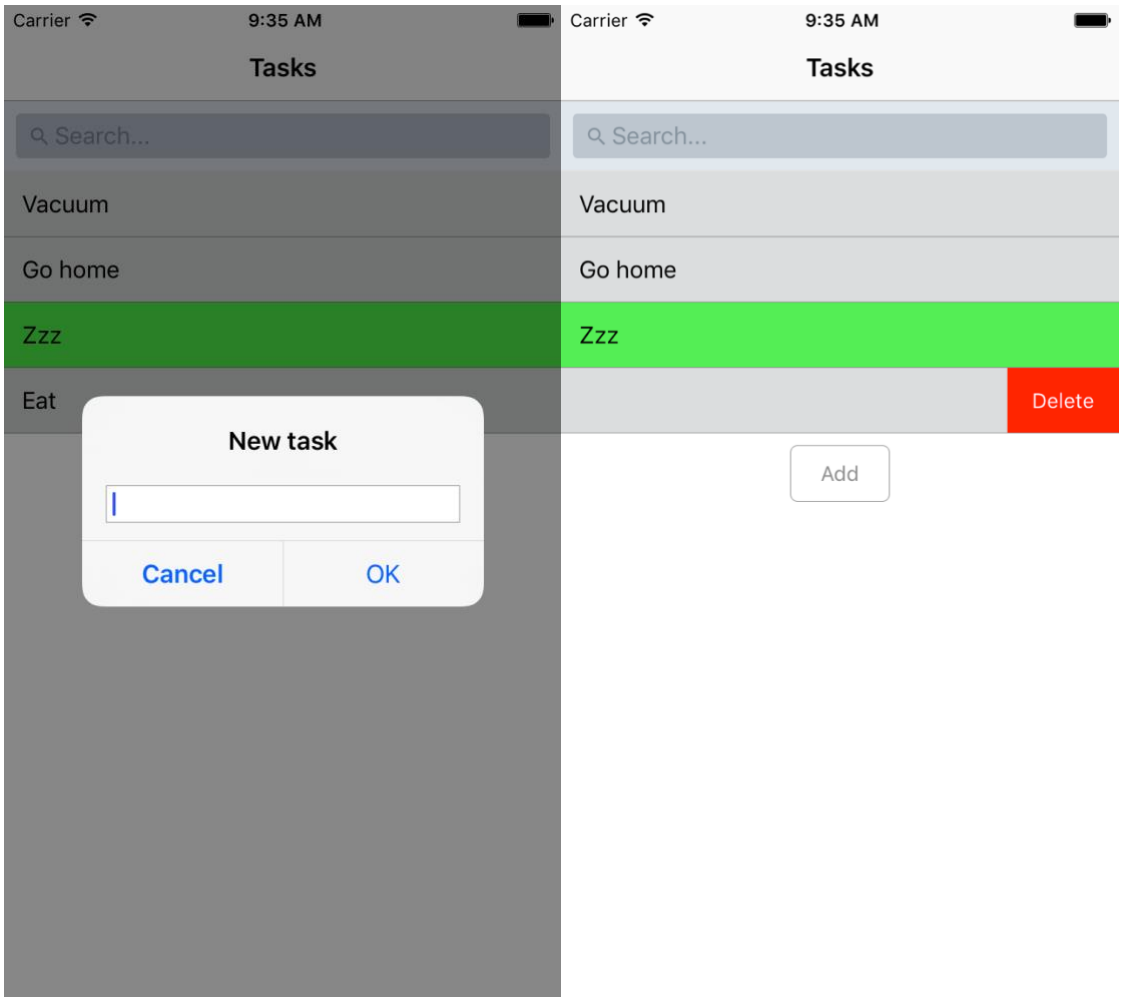

Figure 17. Screenshots of React Native demo application

3.5.4 Reflections

Of the tools tried in this research project, React Native proved to be the easiest to use. It took least time to create a fully working task list application in React Native compared to other frameworks. Part of the reason for this is possibly developer's existing familiarity with JavaScript and React. Debugging and development tools that come with React Native and development velocity they provide made a great impression. It was possible to implement native UI components with React Native, which led to a native looking UI.

3.6 Ionic

Ionic is a hybrid (WebView) development framework. It adds a layer on top of Cordova to simplify application development. Ionic applications are implemented using Angular HTML5 framework and TypeScript, which is a typed subset of JavaScript. Ionic provides a large number of pre-built APIs to access native features and UI components.

Ionic's motto is "Write once. Deploy Anywhere.". Like with Cordova it is possible to achieve 100% of code reuse between platforms. Ionic claims that their applications are better performance-wise than regular Cordova applications. This is hard to test in the scope of a simple task list demo application.

3.6.1 Advantages

Ionic's development tools are very impressive compared to what is offered with Cordova. Ionic's command line tools combine Angular's command line tools with Cordova functionality. Native application packaging, development server and live reload are set up by default. It is easy to run the application on a real device or on an emulator. Ionic provides tools for creating new wireframe applications with a single command and it is impressive how fast setting up a new project is.

Development server offers LiveReload (functionality that automatically generates new assets and reloads the browser on source code changes) which greatly speeds up development process. Development server offers error reporting: errors and stack traces are displayed neatly in the browser.

There exists a large amount of ready-made UI components that can be used in an Ionic application. The list is extensive, and components are sleek looking and polished. Ionic provides platform-specific configurations which makes it easier to develop a UI that looks differently on different platforms. For example, when using built-in icon components, iOS, Android and browser will render different icons specifically styled for these platforms.

3.6.2 Disadvantages

The main drawback of Ionic is that it is a Hybrid framework and does not achieve the performance level of a native application. For this reason, out of the frameworks looked into in

this research, feature-wise Ionic can only be compared to Cordova, which is also a Hybrid WebView-based framework. Compared to Cordova Ionic comes ahead at every turn: it is more efficient in regard to development process; projects are faster to set up and it produces better looking applications. Compared to other tested frameworks it was a clear winner in the amount of time it took to set up a new project and to create a demo application. However, this would not be the case if the application had required usage of native APIs.

3.6.3 Demo application

A fully-functional Ionic application was implemented using Ionic. Table 6 displays a rough breakdown of development time by task. Demo application screenshots can be seen in Figure 18.

Task	**Time (h)**
Setting up the project	0.5
REST API implementation	1
Task list implementation	0.5
Updating task list	0.5
Popup for adding new task items	2
Swipe functionality	1
Total	**7.5**

Table 6. Ionic demo application development time breakdown

List of implemented features:

- Fetching a list of tasks from the backend
- Adding a new task (request to the backend, UI update)
- Marking a task as complete (request to the backend, UI update)
- Filtering of tasks
- Deleting tasks by swiping (request to the backend, UI update)

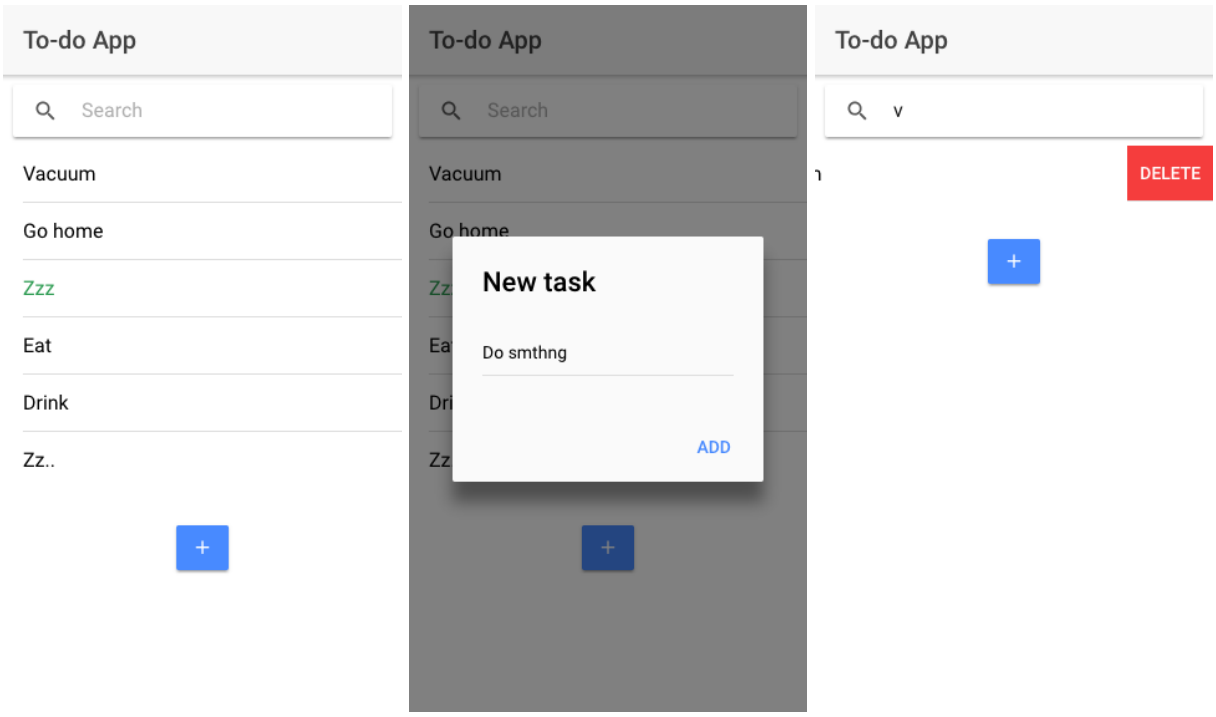

Figure 18. Screenshots of Ionic demo application

3.6.4 Reflections

As expected, creating an application with Ionic was similar to the previous experience of creating an application with Cordova. However, with the Ionic's tools this process was made smoother, faster and easier. Ionic offers tools that greatly speed up the development process, such as pre-made build and run scripts, template generators and debugging tools. There seems to be no advantage in choosing Cordova over Ionic, because Ionic has all the functionality on Cordova, but with the added advantage of many useful tools built on top.

Resulting application did not have the look of a fully native applications, but it had a fast and responsive UI. The UI components can be styled using CSS3 to achieve a look that can be relatively similar to the one of a native application, but not identical. Ionic framework's UI components attempt to follow native platforms' UI design guidelines and mimic animations and UI transitions, such as swiping gestures. This allows producing better UIs with Ionic than when using Cordova paired with some HTML5 framework.

3.7 Cordova

Cordova uses Hybrid approach; it allows running a Web application on mobile devices wrapped inside a WebView. With Cordova up to 100% of all code can be shared between platforms depending on the need for browser-specific alterations (since Android and iOS run WebView in different browsers) and the importance of native looking UIs. There is no need for frequent recompiling during development, because application can be run and tested in a browser and all the time-saving Web development tools, such as LiveReload and browser developer tools, can be used to aid development.

Applications developed with Cordova lack the look-and-feel of native applications. Its UI consists of HTML5 components styled with CSS. Using CSS to make elements in a Web application resemble native elements would take considerable amounts of time and would require separate styling for different platforms. And even then, the experience would be lacking, because the same interactions, touch gestures and animations are just not achievable with Web.

Cordova offers plugins that make it possible to use native functionality of the mobile device such as camera or GPS module. These plugins provide bridges to the native APIs and allow accessing them directly from JavaScript. More native bridges can be implemented using native code and integrated into the project.

3.7.1 Advantages

Cordova has the easiest learning curve of all the compared frameworks. Apart from Web development knowledge, it does not require any specific skills. Packaging an HTML5 application to a native application with Cordova is a fairly simple and only requires a few CLI commands. Any HTML5 framework or library can be used to create the application so knowledge of any JavaScript framework is enough.

Cordova applications can share up to 100% of all code between platforms. Platform specific code might be needed in case of browser incompatibilities and specific UI requirements. Platform specific interface elements and UI code might be needed to achieve better native look and feel.

3.7.2 Disadvantages

As with other Hybrid approach CPMDFs, the main drawback is not having a native application. Essentially, Cordova application is a Web application running in a browser window where browser controls are not visible. It does not look as good as a native application and does not work as fast.

While creating pure HTML5 applications with Cordova is straightforward, implementing native functionality can be more complicated. There are many plugins published to use with Cordova, but they are mostly created by the community, many of them have bugs, some are not up to date with the native APIs or do not support all the platforms. If a plugin for a specific feature is not available, one can be created, but it requires native coding for each platform separately (e.g. Swift for iOS, Java for Android) and a layer of JavaScript to execute that native code.

3.7.3 Reflections

An example Cordova application was not created for this comparison, because the development team in question already has knowledge on how Cordova works and of its advantages and disadvantages. The main purpose of this case study was to find a new possible alternative for the development process. Also, comparing development process and development speed of a framework used daily to development with frameworks that are new and unfamiliar to the developer would not be objective.

Judging from previous experience of using Cordova it can be estimated that the business logic of the HTML5 application would take about the same amount of time to implement as with NativeScript or React Native, since all of these use JavaScript and the code is essentially the same. UI would be easier to implement because basic HTML5 components and CSS3 would be used instead of UIKit components. Swipe functionality could prove more problematic, because it is not directly supported and would require the use of some 3rd party library. Cordova's build process is fast and smooth when using no native API addons (such as GPS or Bluetooth), so overall the to-do demo application with Cordova should be developed in least amount of time compared to all Cross compiled and Interpreted CPMDFs; and in a similar amount of time as with Ionic, which shares the same process and concepts, but offers better build tools.

3.8 Other Frameworks

Several frameworks were left out of the main comparison due to it being clear that they are not a suitable option for the needs of the development team in question, and due to the time constraints of the case study. These frameworks include 5App, AlphaAnywhere, Appcelerator Titanium, Codename One, Convertigo, Dropsource, Fuse, Kony AppPlatform, Qt, RhoMobile and Sencha. Below is a short description of each of these frameworks and the main reasons why they were not considered as a replacement for the current framework. The main reasons for not choosing these frameworks were generally the same: they are not widely used and lack a good community of developers and a knowledge base. For some of them there is a concern of the lack of long-term support.

5App is a framework with a narrow use-case, it is mostly designed to create applications by companies for their employees. Its emphasis is on data security. It has no documentation on its website, there seems to be almost no resources on how to use it and it cannot be downloaded or tried out without getting personally in contact with their sales team. There is so little information available that it is impossible to determine which cross-platform approach is used, but most likely 5App uses Hybrid approach. Stack Overflow also has no mention of 5App in their tag list or in trends. Software tools with a small user base are generally not a good choice due to lack of support, knowledge sharing between developers and active development. [68] 5App does not seem to have any meaningful developer community.

Alpha Anywhere uses the Web approach and focuses on building low-code business applications, it advertises itself as a fast tool with a possibility to build a mobile application in just 20 minutes. Low-code means that the applications are mostly built with visual tools without much coding. This greatly improves development speed but limits the configurability of an application and the number of supported features. This kind of framework can be very powerful if requirements for an application exactly match the capabilities of a framework, but at the same time the framework can be completely unusable for any use-cases outside of the predefined ones. Another downside of Alpha Anywhere is that of all the example applications on their website look outdated and have unappealing UIs. [69] Alpha Anywhere is not present in Stack Overflow tags or trends which is a sign of very low popularity among developers.

Appcelerator Titanium is a framework for developing native applications with JavaScript, which is then compiled into respective native languages. The framework has its own IDE specifically for working with it, it provides tools like a visual UI designer with code generation, API builder, native code editors and deployment tools. Unlike some of the other frameworks Appcelerator Titanium does not only provide tools to develop the front-end of mobile applications but focuses on the full development experience. It provides several tools for easier development of back-end APIs. There is a free community license, but for commercial development the price is considerably high: $99 per user per month. [70] Appcelerator Titanium's popularity peaked at 0.09% in early 2016 according to Stack Overflow Trends, has since been rapidly declining and is currently down to as low as 0.005%.

Codename One uses the Cross compiled approach. It has several pricing options including a free community version. It receives timely updates; its website has recent news and blog posts related to new framework features and the changes in native frameworks and mobile industry. Codename One's documentation is comprehensive, it offers many resources for developers including tutorials, a getting started guide and an "academy" with online courses for sale. One of Codename One's core features is a cloud platform for building and testing applications, this is an important feature since building native applications locally can be time consuming and would require a macOS machine for iOS and a Windows 10 machine for Windows mobile applications. Codename One has a drag-and-drop UI builder to speed up development; it can compile Java or Kotlin. It advertises its paradigm as "write once, run anywhere" and promises 100% code reuse. Website contains a gallery of applications built with Codename One, but the list does not contain any widely used applications and all the examples look outdated and low quality. [71] According to Stack Overflow Trends Codename One had a short-lasting peak in popularity in early 2016 with 0.07% of all questions being tagged as Codename One, but the popularity has since been rapidly declining.

Convertigo is advertised as a low-code framework for building enterprise level applications. Low-code is generally good for faster development process but it sets tight constraints on the application functionality. Apart from development process itself framework takes care of resource scaling, applications distribution to end users and provides online-offline syncing capabilities. Convertigo provides cloud builds for their applications and options to do native

development in either Ionic or Angular, with mobile backend as a service. Their emphasis seems to be on applications in manufacturing support and banking. Examples of applications built with the framework look stylish and up-to-date with current mobile application design guidelines. [72] Convertigo's website provides documentation and resources for developers and even their own SDK. Outside of their documentation there seems to be no community and no knowledge base, there is no mention of Convertigo on Stack Overflow.

Dropsource is another low-code mobile development platform, it uses Cross compiled approach and generates fully native applications for iOS and Android. It provides a list of available backend integrations including SAP, Salesforce that work out of the box with little development work required. If the functionality they provide matches the requirements for an application this framework is a good choice speed and resource wise. Their pricing starts from $999 a year per developer; there is a community version available, but it has serious limitations such as no access to application's source code. Website showcases applications built with Dropsource but there are no popular applications in the list and most applications look outdated and low quality. [73] According to Stack Overflow tags there is barely any knowledge base available outside of Dropsource's own website.

Fuse is a Cross compiled approach framework with an emphasis on user experience [74]. It attempts to make the process of creating straight-forward mobile UIs for existing systems faster and more efficient. Fuse does not provide a community license or a trial version of the framework; there is no documentation openly available to research the features and limitations of the framework. Price information is also not available. According to Stack Overflow trends and tags Fuse has a non-existent community with less than 10 questions tagged; there are hardly any resources for developers available. Part of the reason is possibly the clashing names. There are several more development resources that are named similarly: FUSE stands for Filesystems in Userspace, fuse.js is a JavaScript search library, Fusebox is a coding methodology and there is a Java Web application framework named AppFuse.

Kony AppPlatform is a low-code framework that provides a range of features to improve development speed. These features include a library of reusable application components, importing UI designs directly from Photoshop and Sketch, pre-built backends and a cloud build

and application preview services and one-click deployment. Kony offers built-in analytics, targeted messaging, performance management and security features. They offer support for both iOS and Android. [75] Applications are built with JavaScript using a specific Kony IDE. Kony offers a free version of the platform with limited features and an enterprise version starting at $2500 /month. Kony AppPlatform has been named Leader in the Gartner Mobile App Development Platforms Magic Quadrant every year from 2013 to 2018 [76]. Despite this Kony does not seem to be particularly popular among the developer community with only less than 200 questions tagged on Stack Overflow.

PhoneGap is a free and open-source framework that uses Hybrid approach [77]. It is not a mobile-framework by itself, but it is a set of tools on top of Cordova to make the development and publishing processes more efficient. It was very popular when Cordova and PhoneGap had just emerged. The main distinguishing feature was, and still is, a PhoneGap mobile application that can be installed from the mobile platform's own application store, which would be connected to the development server and would run the application without the need to build and package the application. This allowed developers to set up proof-of-concepts and demos very quickly but would only work for basic HTML5 applications with no native API access. Also, this ability to set up demo applications quickly does not provide any added value when developing B2B applications with a long lifecycle. PhoneGap has since lost some of its popularity after many other tools have emerged.

Qt is a mature framework that has been around since 1993. Originally it started a framework for developing multi-platform applications with GUIs. It later expanded to support iOS, Android and Windows Phone platforms. The framework provides its own IDE, command line tool for compiling source code and building applications, tools for creating visual design and prototypes. Because it is a well-developed framework it has a big community and developer base and many compatible libraries. Development is done using C++. Based on Qt's showcase of mobile applications created using the framework it can be concluded that it was popular for Symbian development a decade ago but is no longer widely used. Qt provides a free community version for open source development; commercial licenses start at $1000 per user [78]. The framework has a large community of developers due to its wide spread back when it was one of the few available cross-platform development frameworks. It peaked in Stack Overflow trends in 2010

at 0.55% of all questions and its popularity has been declining at first slowly and then more rapidly since 2015.

RhoMobile Suite is a set of tools for creating enterprise-class mobile applications. Tools include cross-platform development framework Rhodes, an IDE RhoStudio, a framework for accessing native APIs RhoElements and a cloud service for data synchronization RhoConnect [79]. The RhoMobile Suite was open-sourced in 2016 and is free to use, paid subscription includes additional services such as cloud builds. Development in RhoMobile is done using Ruby and HTML. RhoMobile does not appear to be used widely used, has very few mentions on Stack Overflow and is not mentioned in Stack Overflow Trends.

Sencha Touch was originally built as an extension of Ext JS, Sencha's Web development framework. Later Sencha Touch was merged into Ext JS. Ext JS frameworks offers multiple tools for development, including pre-built components, theme support, drag-and-drop and configuration-based development. Ext JS as a mobile development framework is extremely efficient if the goal is to develop both Web and mobile applications, since with Sencha both can be created simultaneously with minimal extra effort to port Web application to run on mobile [43, pp. 18-21]. The result is a WebKit-based mobile application created with PhoneGap. Sencha provides a community version that is free to use for students, individuals, non-profit organizations and businesses that are just starting and have very little revenue. Commercial license is $1200 per each developer [80]. According to Stack Overflow Trends the popularity of Ext JS was highest between 2011 and 2014 when it peaked at 0.3% of all Stack Overflow questions and has been since continuously declining. No recently developed up to date showcase applications can be found.

3.9 Comparison of key characteristics

The main characteristics of short-listed CPMDFs – supported platforms, programming language used, cross-platform approach used and pricing - are presented in Table 7 below. This information was collected from each framework's own documentation.

Framework	**Platforms supported**	**Language**	**Approach**	**Pricing**

RubyMotion	iOS, Android, macOS	Ruby	Cross compiled	$499 / developer / year
NativeScript	iOS, Android	JavaScript, optionally Angular and TypeScript	Interpreted	Free
Xamarin	iOS, Android, Windows	C#	Cross compiled	Included in Microsoft Visual Studio
React Native	iOS, Android	JavaScript, React framework	Interpreted	Free
Ionic	iOS, Android	JavaScript, Angular framework	Hybrid (WebView)	Free (enterprise edition with extra features available)
Qt	iOS, Android	C++, QML	Cross compiled	$5500 / developer
Cordova	iOS, Android, Windows	JavaScript, any framework	Hybrid (WebView)	Free

Table 7. Main characteristics of short-listed CPMDFs

It can be noted that the only frameworks supporting Windows applications are Xamarin and Cordova, but this is not a key advantage, because Windows platform support is not considered important in this case study. All frameworks utilize popular and developer-friendly programming languages, except for Qt, which uses C++. Of C#, Ruby and JavaScript the latter is preferable, because the development team is already familiar with it.

When comparing frameworks by selected cross-platform approach, WebView loses to cross compiled and interpreted due to lower quality of UX. Cross compiled approach offers the best UX which is closest compared to native approach, but it is also the least modular approach,

which means that less code can be reused between platforms and more development time is required. This tradeoff between cross compiled and interpreted approaches should be considered separately from individual frameworks.

Pricing-wise the clear winner are the open source tools: React Native, NativeScript, Ionic and Cordova. On top of not having to spend money on expensive licenses there is the added benefit of not having to go through the process of acquiring them (those processes can be lengthy in bigger corporations), keeping track of them and renewing them periodically.

Table 8 presents positive and negative feedback from development process of the demo application with different CPMDFs. This is a subjective opinion of a single developer using these frameworks.

Framework	**Positive experiences**	**Negative experiences**
Ruby Motion	• Good CLI and build tools	• Lack of documentation for UI components
NativeScript	• Good CLI and build tools • Debugging in a browser, LiveSync	• Problems with debugging • Problems with components
Xamarin	• Powerful debugger, but complicated to use • Easy installation	• Complicated IDE • Unfamiliar project structure
React Native	• Good CLI and build tools • Smooth development and testing process • Debugging in a browser, live update	
Ionic	• Good CLI and build tools • Powerful starter templates	• Non-native components

Table 8. Comparison of development process with short-listed CPMDFs

Table 9 presents evaluation of the main comparison criteria for each framework. Each framework is awarded a score from 1 to 5 for the size of its community, documentation and knowledgebase and various phases of the development process. Framework's language and lines of code in the demo application implementation are also compared. It is not feasible to directly compare time spent on development of each demo application, because several of them required implementation of the same UIKit components, which use the same APIs and take less time to implement with each iteration due to becoming more familiar to the developer. Based on all of these criteria the final score from 1 to 5 is awarded to each framework.

	Ruby Motion	**NativeScript**	**Xamarin**	**React Native**	**Ionic**
Community (1-5)	2	3	3	5	5
Knowledgebase (1-5)	3	4	3	5	5
Installation process (1-5)	5	4	4	5	5
Development process (1-5)	4	2	3	5	5
Build and debugging (1-5)	4	2	3	5	5
Lines of code	230	175	315	310	235
Score	**4**	**3**	**3**	**5**	**5**

Table 9. Evaluation of development process with different CPMDFs

3.10 Most promising framework

Based on both the research of framework's qualities and the experience with creating a demo application the conclusion is that the most promising alternative to current development stack is React Native. Switching from the current Hybrid (WebView) approach to the one that is closer to native development would greatly improve application runtime speed and UX quality, yet the

team would still get the benefits of doing cross-platform development and they would continue using JavaScript as development language. React Native also has a big developer base and is growing rapidly, so there should be no concern of it becoming obsolete in the foreseeable future.

Xamarin and Ruby Motion, while powerful alternatives, would require too much changes to the development process, because cross compiled approach is very different compared to the WebView approach that is currently being used with Cordova, and because either of these two framework would require learning a new programming language; in case of Xamarin also a new IDE.

Ionic is a second candidate for the new development framework. Unlike React Native, which implements the Interpreted approach to cross-platform development, Ionic uses the WebView approach, so it is very close to the current development stack. If the team would want to make minimal changes to their development process, yet still gain the benefits of a more robust framework and better development tools, they should consider using Ionic.

4 DISCUSSION

This chapter discusses the results of this case study. Section 4.1 discusses the possible advantages of CPMDFs, section 4.2 discusses the disadvantages. Section 4.3 compares the results of this case study to the results found in similar research.

4.1 Advantages of choosing cross-platform development

According to Willocx et al. [81] CPTs (cross-platform tools) are a promising alternative to native mobile development. Parts of the code can be shared between implementations for different platforms. Also, Web development technologies can be used for mobile development, which provides the possibility for Web developers to participate in the mobile development process.

Willocx [81] states that "CPTs can also increase code maintainability for two reasons. First, a software update does not need to be implemented for multiple native implementations. Second, platform API updates are often transparent to CPT applications as these rely on the API provided by the CPT. Updates in the platform's API are tackled by the CPTs. Hence, the application source code does not necessarily have to be modified to benefit from new API features."

The main reasons to consider using cross-platform mobile development framework as an alternative to doing native development are faster development cycles and reduced costs. Reusing code across different platforms will speed up product development cycle which in turn will lower overall development costs. Ability to develop the application using existing programming language knowledge, instead of forcing developers to learn new languages and frameworks, also transfers to reduced costs.

Usage of cross-platform mobile frameworks also gives the advantage of an easier maintenance period. With fewer different technologies being used in the project its maintenance period calls for fewer narrow technology specialists. It is possible for the same developer e.g. a Ruby specialist or a team to maintain multiple supported platforms.

Further benefit of cross-platform mobile development is the possibility to save on time needed for testing. Unit tests may only be written once for all supported platforms. In case of JavaScript-

based development frameworks (Cordova, Ionic, React Native and NativeScript) functional testing is also simplified because it can be done in the browser. This doesn't provide the same test environment as native, but it allows testing of most of the application features.

Cross-platform development is a good choice for B2B (Business-to-Business) applications, because in the case of business-to-business time to deployment is often more important than a polished and sleek UI. Efficient utilization of resources is also very important for B2B. In many cases these B2B mobile applications are extensions to existing systems with a user base and thus it is not as important to focus on user experience.

4.2 Disadvantages of cross-platform mobile development

One of the disadvantages of using HTML5 hybrid (WebView) applications is the slowness of the UI on low and mid-range phones [82]. HTML5 hybrid applications are essentially a Web application running in a sandbox browser wrapped as an application. Instead of using platform's own User Interface components these applications use HTML elements. Even though these elements can be styled to mimic native components and their animations, they do not look the same and the animations are not as smooth. HTML5 animations consume considerably more computing power and use more battery than platform's native components.

When attempting to create beautiful high-quality mobile applications it is important to follow User Interface guidelines defined by each platform. Android and iOS have different guidelines, different ways of structuring application interfaces and different looks to their components. It is a clear sign of a "cheap" low quality application when for example an iOS application has the UI of an Android application. When attempting to follow all of these guidelines and best practices it is hard to fit all the code and styling into a single code base.

Another problem is having to rely on the framework being updated in a timely manner. As mobile phones are evolving a lot and changing fast, so are mobile operating systems. Mobile development environment is very fast-paced. It can be challenging for any cross-platform mobile development framework to keep up with all the changes to OSs and their APIs. If these frameworks are too slow to adopt new changes, it slows down development of cross-platform

applications even more, because developers would have to wait for the framework updates and only then start updating their software to be in line with the latest OS release.

As Choudhary [83] states, “Testing cross-platform applications may be considerably more complicated, since different platforms can exhibit slightly different behaviors or subtle bugs. This problem has led some developers to deride cross-platform development as "write once, debug everywhere", a take on Sun Microsystems' "write once, run anywhere" marketing slogan.“ Not just every platform, but every version of the same OS by different vendors and even different devices from the same vendors can have slight differences in software, which can cause a testing nightmare, especially in case of WebView approach, where there are more levels of abstraction between the device hardware and the application code.

4.3 Results compared to related work

There are several studies comparing CPMDFs and the approaches they use to achieve cross-platform capabilities. Most of this research only compare a few of the available frameworks and do not look at as wide number of available frameworks as this case study. Many researches consider some of the frameworks mentioned in this research, but omit the more recently emerged ones, that can be very promising, such as React Native.

4.3.1 Airbnb case

Gabriel Peal [84] describes in an online article the advantages and difficulties their team faced during several years of using React Native for Airbnb’s mobile applications. Airbnb is an online marketplace for short-term renting out of people’s own living space or lodging. It operates online, on Android and on iOS.

According to Peal React Native brought numerous advantages for the development process. Their development team was able to share 95%-100% of all application code between platforms. They were happy with React framework, which is a base for React Native, and with Redux for state management. React Native proved to be good performance-wise, but there were some issues with first-render time, which is the time it takes for the application to load the initial

screen. They achieved impressive iteration and build speed but had to invest heavily into React Native infrastructure.

The caveats Airbnb ran into when using React Native included the framework being immature, which forced the developers to contribute themselves to the open-source project and to maintain a separate version of the patched framework code, which is a big problem when upgrading versions. They were unhappy with some of the features of JavaScript, mostly related to it being un untyped language. They run into problems with open-source libraries for React Native, which contained unexpected bugs. Performance-wise their biggest concern was the time it took the application to render initially after launch.

In 2018 Airbnb decided to stop using React Native for their mobile application needs and to gradually transition to native development. Main reasons for this were first-render performance issues, complicated bridging infrastructure between React Native and native APIs, problems with debugging and lack of accessibility APIs in React Native. However, developers at Airbnb are mostly happy with their experience, 60% described their experience as amazing, 63% would choose React Native again, and 74% would consider React Native for a new project [85].

Airbnb's case is a good practical example of what advantages and disadvantages React Native can provide. Considering their experience, it seems that for this particular use case it would make more sense to choose React Native over native development. Firstly, the development team in question has a smaller team than Airbnb and cannot afford to dedicate many people to each individual platform. Also, unlike Airbnb they have not just one, but several mobile application projects, but each a smaller one. In this case faster deployment times that React Native offers make a big difference. Also, B2B applications are simpler UI-wise and do not rely on as much animation and gestures, and thus require less bridging with native APIs. For these reasons it is likely that React Native would work better for the case in question than for Airbnb.

4.3.2 Other studies

Huynh [86] compares Ionic framework with native development and using a mobile Web-site instead of creating a mobile application. He comes to similar conclusions as this research: CPMDFs are a viable alternative to native development. "The advance in hybrid framework in

general and the growing acceptance of open source framework, such as Ionic in particular, may provide an alternative to the native app domination and may trigger the rapid rise of hybrid apps in the years to come." He does not consider other cross-platform approaches apart from the WebView approach. Of the WebView alternatives he singles out Ionic as the most promising candidate, which is conclusive with this research.

Nunkesser [87] proposes a hierarchical approach to categorize different methods for creating native and cross-platform mobile applications and comes up with six categories. His categorization corresponds well to the one used in this research. His "Endemic" category corresponds to "native". Web Apps are HTML5 pages that run in the browser and are not technically a mobile application and were not considered a cross-platform development approach in this research. Hybrid Web Apps corresponds to the WebView approach (Cordova, Ionic), Hybrid Bridged Apps to hybrid approach (React Native, NativeScript). "Foreign Language Apps" correspond to cross compiled approach (Xamarin, RubyMotion). "System Language Apps" are applications in C/C++ which are used for graphically heavy applications such as games and were not considered an alternative for this use case.

Nunkesser concludes that "Endemic" (native) applications offer the best UX, but offer little possibility for code sharing during development, apart from sharing code used for testing the applications. "Hybrid Web Apps" (WebView approach) lack the possibility to use native UI elements but offer the most code sharing possibilities. "Foreign Language Apps" (cross compiled approach) offer possibilities for using native UI components but come with the risk of having to rely on the framework being kept up to date by the manufacturer. All of these conclusions are in line with what was found in this research. Nunkesser does not offer an opinion as to which approach would be preferable over the others.

Lifh et al. [58] compares the performance of a natively implemented and a React Native application on both Android and iOS platforms. Testing concluded that React Native decreases development time and is a good option for uses cases where not much native API access (such as Bluetooth) is required by the application. They found that Android application had similar performance on both native and React Native, and the iOS application had worse performance due to problems with Bluetooth API implementation. No end application CPU and memory

consumption analysis were done in this research, but the conclusions regarding code base and development time are similar.

Vilcek et al. [54] compare and analyze tools for development of hybrid and native mobile applications. They evaluate Android Studio, Xcode, Visual Studio (which is used for native Windows Phone development), Ionic, PhoneGap and NativeScript. Vilcek concludes that the best application quality can be achieved by doing native development, but that cross-platform solutions offer many benefits, such as faster development process, lower development costs and the possibility to use one single programming language; that these tools are constantly being developed and that the gap between native and cross-platform applications is getting smaller every day. These findings are conclusive with those produced in this research. Out of evaluated cross-platform options Vilcek singles out PhoneGap as the most promising one due to Windows Phone support; however, this research deemed Windows Phone not important and chose in favor of Ionic due to better development tools and more sophisticated framework.

Dalmasso et al. [53] evaluate a total of ten different CPMDFs. On top of similar key criteria comparison, as done in this research, they did an in-depth performance evaluation of several cross-platform applications on Android platform. They conclude that cross-platform JavaScript frameworks have generally good response time, but have a performance penalty compared to native frameworks, however, this penalty is acceptable in relation to gains during development process. This is in line with the conclusions of this research, despite the fact that no CPU or memory usage benchmarking was done and only UI and UX were evaluated.

Willocx et al. [62] compare a larger number of CPMDFs application implementations than this research: they look at performance of a total of ten CPMDFs. However, some of these frameworks are already outdated, because they are no longer being updated in accordance with the latest native APIs, for example, Intel App Framework (officially discontinued in 2017 [88]) and mgwt (has not been updated since 2014 [89]). Some are not mobile development frameworks, but HTML5 frameworks packaged with PhoneGap (e.g. jQuery Mobile and Famo.us) which were considered as the same CPMDF in this research. Some are not suitable for business application development, for example Adobe AIR is designed to create Flash animation and games.

The research of Willocx et al. also look at some of the same frameworks as this research, particularly Ionic and Xamarin. Their research focused on comparing performance metrics of different application implementations, which this research only touched at the most basic level. Their conclusion that native implementation offers the best performance, interpreted implementations come second while WebView approach has the highest performance penalty is in line with finding of this research.

5 CONCLUSIONS

This chapter looks at the scientific value and limitations of this research. It looks at meeting the research goals, considers implications of this research on the software development field, discusses the differences between this and existing research and describes what future work could be done on this topic.

This research presented a comparison of all the currently available CPMDFs, a short-list of the most promising alternatives based on several key metrics, a demo application created with these short-listed frameworks and a comparison of the development process. Short-listed frameworks were further evaluated and two most promising alternatives, that use different cross-platform approach, were proposed as the replacement for the current technology stack for the considered use case.

Research of CPMDFs, especially those that use JavaScript language, can open new paths for mobile application development. Up until recently mobile development has been viewed as a field that is hard to get into and has a steep learning curve, as opposed to Web development, where it is relatively easy to learn to build something concrete. With these new tools many existing Web developers can branch out into mobile development and many smaller companies can afford to implement mobile applications on top of their Web applications.

This paper focused on researching and evaluating as many different CPMDFs as possible. Many researches have done comparisons of native tools against cross-platform tools, however in this particular use case there was a set need for a cross-platform framework specifically, so only these were evaluated against each other. Most researches that have done similar comparisons only focused on a small number of available frameworks [58] [87] [54] [55]. Some research has mostly focused on outdated frameworks that are no longer being maintained or updated [62].

This research evaluated at all the available CPMDFs today and identified several more promising alternatives. However, more research is needed to identify the clear leaders in application performance. Comprehensive benchmarking of CPU usage, memory and battery consumption is required to clearly identify which framework produces more optimal end binaries. Such benchmarking should be done not just for simpler applications that fetch and

display data, such as presented demo application, but also for complicated applications which utilize native APIs, such as GPS, camera or Bluetooth. More testing of development process is also required, such where a larger group of developers get to test and evaluate selected frameworks.

A future research is needed on what are the consequences of making a switch to a new technology stack. For this purpose, each member of development team could be surveyed prior to adopting the new stack with questions such as what their expectations and what problems they hope to have been solved when the switch is done. After the switch, while still in the early adoption process, developers could be surveyed on what problems they are facing with the new technology stack. Afterwards, when a considerable amount of time has passed, the developers could be surveyed on how they feel the new stack is performing.

REFERENCES

[1] A. Sirenius, "Julkishallinnon raportointipalvelun kokeilu tiedon esittämistavasta," Tietokiri, 2019. [Online]. Available: https://tietokiri.fi/mika-tietokiri-2/julkishallinnon-raportointipalvelu/kerro-mielipiteesi-raportointinakymista/. [Accessed 25 04 2019].

[2] Statista, "Number of mobile app hours per smartphone and tablet app user in the United States in June 2016, by age group," Statista, 2016. [Online]. Available: https://www.statista.com/statistics/323522/us-user-mobile-app-engagement-age/. [Accessed 25 04 2019].

[3] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering.," *Empirical software engineering*, vol. 14, no. 2, p. 131, 2009.

[4] S. Easterbrook, J. Singer, M.-A. Storey and D. Damian, "Selecting empirical methods for software engineering research.," in *Guide to advanced empirical software engineering*, London, Springer, 2008, pp. 285-311.

[5] B. Gillham, Case study research methods, Bloomsbury Publishing, 2000.

[6] B. Fitzgerald, N. Russo and T. O'Kane, "Software development method tailoring at Motorola," *Communications of the ACM*, vol. 46, no. 4, pp. 64-70, 2003.

[7] M. L. Drury-Grogan, K. Conboy and T. Acton, "Examining decision characteristics & challenges for agile software development.," *Journal of Systems and Software*, vol. 131, pp. 248-265, 2017.

[8] M. Parmar, "Microsoft plans to support Windows 10 Mobile devices until December 2019," 25 10 2017. [Online]. Available: https://www.windowslatest.com/2017/10/25/microsoft-promises-support-windows-10-mobile-devices-december-2019/. [Accessed 05 03 2019].

[9] K. Tofel, "AT&T to start selling Microsoft Lumia 950 on November 17," 16 11 2015. [Online]. Available: https://www.zdnet.com/article/at-t-to-start-selling-microsoft-lumia-950-on-november-17/. [Accessed 05 03 2019].

[10] Z. Bowden, "Photos of Microsoft's canceled 'Lumia 960' flagship leak," 31 05 2017. [Online]. Available: https://www.windowscentral.com/photos-microsofts-canceled-lumia-960-flagship-leak. [Accessed 05 03 2019].

[11] Statista, "Number of mobile app downloads worldwide in 2017, 2018 and 2022 (in billions)," 05 2018. [Online]. Available: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. [Accessed 25 04 2019].

[12] T. Yamakami, "Mobile application platform strategies: Business model engineering for the data intensive mobile age," *International Conference on Mobile Business (ICMB'05)*, pp. 333-339, 2005.

[13] J. Agar, Constant Touch: A Global History of the Mobile Phone, Icon Books Ltd, 2013.

[14] A. Smith, "46% of American adults are smartphone owners," Pew Research Center's Internet & American Life Project, Washington, D.C., 2012.

[15] C. Hodgson, "Global smartphone sales fall for first time," *FT.com*, 22 02 2018.

[16] K. Costello, "Gartner Says Worldwide Sales of Smartphones Returned to Growth in First Quarter of 2018," Gartner, 29 05 2018. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2018-05-29-gartner-says-worldwide-sales-of-smartphones-returned-to-growth-in-first-quarter-of-2018. [Accessed 26 05 2019].

[17] H. Heitkötter, "Cross-Platform Model-Driven Development of Mobile Applications with MD2," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013.

[18] X. Li, "Smartphone Evolution and Reuse: Establishing a More Sustainable Model," in *2010 39th International Conference on Parallel Processing Workshops*, 2010.

[19] OpenSignal, "The State of LTE (February 2018)," OpenSignal, 2018. [Online]. Available: https://www.opensignal.com/reports/2018/02/state-of-lte. [Accessed 27 04 2019].

[20] F. Richter, "How Long Does Apple Support Older iPhone Models?," Statista, 17 09 2018. [Online]. Available: https://www.statista.com/chart/5824/ios-iphone-compatibility. [Accessed 24 04 2019].

[21] A. Freier, "App revenue reaches $92.1 billion in 2018 driven by mobile gaming apps," Business of Apps, 13 09 2018. [Online]. Available: http://www.businessofapps.com/news/app-revenue-reaches-92-1-billion-in-2018-driven-by-mobile-gaming-apps/. [Accessed 26 04 2019].

[22] A. Wasserman, "Software engineering issues for mobile application development.," in *FoSER 2010*, 2010.

[23] C. Salma and A. I. Marzak Abdelaziz, "Cross-Platform Mobile Development Framework Based On MDA Approach.," *International Journal of Technology Diffusion*, vol. 9, no. 1, pp. 45-59, 2018.

[24] P. Abrahamsson, "Keynote: Mobile software development–the business opportunity of today," *Proceedings of the International Conference on Software Development,* pp. 20-23, 2005.

[25] StatCounter GlobalStats, "Mobile Operating System Market Share Worldwide," StatCounter, 2019. [Online]. Available: http://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-200901-201904. [Accessed 28 04 2019].

[26] U. Egham, "Gartner Says Worldwide Sales of Smartphones Grew 7 Percent in the Fourth Quarter of 2016," 15 02 2017. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2017-02-15-gartner-says-

worldwide-sales-of-smartphones-grew-7-percent-in-the-fourth-quarter-. [Accessed 20 04 2019].

[27] J. Perchat, M. Desertot and S. Lecomte, "Component Based Framework to Create Mobile Cross-platform Applications," in *Procedia Computer Science*, 2013.

[28] Stack Overflow, "Developer Survey Results," 2019. [Online]. Available: https://insights.stackoverflow.com/survey/2019. [Accessed 06 04 2019].

[29] G. Wells, "The Future of iOS Development: Evaluating the Swift Programming Language," 2015.

[30] M. Reboucas, G. Pinto, F. Ebert, W. Torres, A. Serebrenik and F. Castor, "An Empirical Study on the Usage of the Swift Programming Language," *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, vol. 1, pp. 634-638, 2016.

[31] Stack Overflow, "Stack Overflow Trends," 2019. [Online]. Available: https://insights.stackoverflow.com/trends. [Accessed 16 05 2019].

[32] J. Fermoso, "PhoneGap Seeks to Bridge the Gap Between Mobile App Platforms.," 05 04 2009. [Online]. Available: https://gigaom.com/2009/04/05/phonegap-seeks-to-bridge-the-gap-between-mobile-app-platforms. [Accessed 20 03 2019].

[33] M. D. Icaza, "Announcing Xamarin," 15 05 2011. [Online]. Available: https://tirania.org/blog/archive/2011/May-16.html. [Accessed 20 03 2019].

[34] Qt, "New Features in Qt 5.1.," 22 11 2016. [Online]. Available: https://wiki.qt.io/New_Features_in_Qt_5.1. [Accessed 20 03 2019].

[35] H. Fakhruddin, "Xcode Vs AppCode – Which IDE Is Better For iOS Development?," 04 12 2015. [Online]. Available: http://teks.co.in/site/blog/xcode-vs-appcode-which-ide-is-better-for-ios-development/. [Accessed 07 04 2019].

[36] Prolific Interactive, "Xcode vs AppCode," 15 05 2015. [Online]. Available: https://www.prolificinteractive.com/2015/05/15/appcode-vs-xcode. [Accessed 07 04 2019].

[37] A. Khaliq, "Android Fragmentation: The Story So Far," 22 01 2019. [Online]. Available: https://www.hongkiat.com/blog/android-fragmentation/. [Accessed 26 04 2019].

[38] OpenSignal, "Android Fragmentation Visualized," 2015. [Online]. Available: https://www.opensignal.com/sites/opensignal-com/files/data/reports/global/data-2015-08/2015_08_fragmentation_report.pdf. [Accessed 16 05 2019].

[39] M. Banerjee, S. Bose, A. Kundu and M. Mukherjee, "A COMPARATIVE STUDY: JAVA VS KOTLIN PROGRAMMING IN ANDROID APPLICATION DEVELOPMENT," *International Journal of Advanced Research in Computer Science,* vol. 9, no. 3, pp. 41-45, 2018.

[40] P. Schwermer, "Performance Evaluation of Kotlin and Java on Android Runtime," 2018.

[41] I. Majocha, "Report: Top Android Security Problems in 2017," 23 12 2017. [Online]. Available: https://dzone.com/articles/report-top-android-security-problems-in-2017. [Accessed 27 04 2019].

[42] J. Fingas, " Google Play protects your Android phone against rogue apps," *Engadget,* 17 05 2017.

[43] J. E. Clark and B. P. Johnson, Sencha Touch 2 Mobile JavaScript framework, Packt Publishing Ltd, 2013.

[44] D. Riehle and T. Gross, "Role model based framework design and integration," *ACM SIGPLAN Notices,* vol. 33, no. 10, pp. 117-133, 1998.

[45] W. S. El-Kassas, "Taxonomy of Cross-Platform Mobile Applications Development Approaches," *Ain Shams Engineering Journal,* vol. 8, no. 2, pp. 163-190, 2017.

[46] L. Corral, A. Janes and T. Remencius, "Potential advantages and disadvantages of multiplatform development frameworks – A vision on mobile environments," *Procedia Computer Science,* vol. 10, pp. 1202-1207, 2012.

[47] C. R. Raj and S. B. Tolety, "A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach," *2012 Annual IEEE India Conference (INDICON),* pp. 625-629, 2012.

[48] T. Y. Adinugroho and J. B. Gautama, "Review of multi-platform mobile application development using WebView: Learning management system on mobile platform.," *Procedia Computer Science,* vol. 59, pp. 291-297, 2015.

[49] A. Merchant, D. Shah, G. S. Bhatia, A. Ghosh and P. Kumaraguru, "Signals Matter: Understanding Popularity and Impact of Users on Stack Overflow.," *The World Wide Web Conference,* pp. 3086-3092, 2019.

[50] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak and B. Hartmann, "Design lessons from the fastest q&a site in the west.," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2011.

[51] D. Robinson, "Exploring the State of Mobile Development with Stack Overflow Trends," 16 05 2017. [Online]. Available: https://stackoverflow.blog/2017/05/16/exploring-state-mobile-development-stack-overflow-trends/. [Accessed 26 05 2019].

[52] D. Robinson, "Introducing Stack Overflow Trends," 05 09 2017. [Online]. Available: https://stackoverflow.blog/2017/05/09/introducing-stack-overflow-trend. [Accessed 10 02 2019].

[53] I. Dalmasso, "Survey, comparison and evaluation of cross platform mobile application development tools.," *9th International Wireless Communications and Mobile Computing Conference (IWCMC),* pp. 323-328, 2013.

[54] T. Vilcek and T. Jakopec, "Comparative analysis of tools for development of native and hybrid mobile applications.," *40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO),* pp. 1516-1521, 2017.

[55] M. Palmieri, I. Singh and A. Cicchetti, "Comparison of cross-platform mobile development tools.," *16th International Conference on Intelligence in Next Generation Networks,* pp. 179-186, 2012.

[56] Y. A. Redda, "Cross platform Mobile Applications Development," Institutt for datateknikk og informasjonsvitenskap, 2012.

[57] L. Corral, A. Sillitti and G. Succi, "Mobile multiplatform development: An experiment for performance analysis," *Procedia Computer Science,* vol. 10, p. 736 – 743, 2012.

[58] O. Lifh and P. Lidholm, "Recreating a Native Application in React Native," 2018.

[59] R. Abrahamsson and D. Berntsen, "Comparing modifiability of React Native and two native codebases," 2017.

[60] T. Y. Adinugroho and J. B. Gautama, "Review of multi-platform mobile application development using WebView: Learning management system on mobile platform.," *Procedia Computer Science,* vol. 59, pp. 291-297, 2015.

[61] A. Charland and B. Leroux, "Mobile application development: web vs. native.," *Communications of the ACM,* vol. 54, no. 5, pp. 49-53, 2011.

[62] M. Willocx, J. Vossaert and V. Naessens, "Comparing performance parameters of mobile app development strategies.," *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft),* pp. 38-47, 2016.

[63] B. Nielsen, "Cross Platform Mobile Development," 2015.

[64] P. Smutný, "Mobile development tools and cross-platform solutions.," in *Proceedings of the 13th International Carpathian Control Conference (ICCC)*, 2012.

[65] N. Bezroukov, "A Slightly Skeptical View on Scripting Languages," 2005.

[66] C. Crawford, "React Native and the New Dream of Learn Once, Write Anywhere," 14 04 2016. [Online]. Available: https://thenewstack.io/react-native-learn-write-anywhere. [Accessed 01 05 2019].

[67] Facebook, "Who's using React Native?," 2019. [Online]. Available: https://facebook.github.io/react-native/showcase.html. [Accessed 30 04 2019].

[68] Marketwatch.com, "5app Launches Its Reliable Mobile app Toolkit for Mobile Enterprise app Development," 2019. [Online]. Available: https://computesys.com/5app-launches-its-reliable-mobile-app-toolkit-for-mobile-enterprise-app-development/. [Accessed 20 05 2019].

[69] Alpha Software, "Alpha Anywhere: Mobile App Development Platform," 2019. [Online]. Available: https://www.alphasoftware.com/mobile-app-development-platform. [Accessed 20 05 2019].

[70] Appcelerator, "Native apps. Mobile APIs. Real-time analytics. One Platform.," 2019. [Online]. Available: https://www.appcelerator.com/mobile-app-development-products/. [Accessed 20 05 2019].

[71] Codename One, "Codename One," 2019. [Online]. Available: https://www.codenameone.com. [Accessed 20 05 2019].

[72] Convertigo, "Convertigo," 2019. [Online]. Available: http://www.convertigo.com. [Accessed 20 05 2019].

[73] Dropsource, "Dropsource," 2019. [Online]. Available: https://www.dropsource.com. [Accessed 20 05 2019].

[74] R. Pedersen, "How Fuse differs from React Native and NativeScript," 09 03 2016. [Online]. Available: https://blog.fusetools.com/how-fuse-differs-from-react-native-and-nativescript-525344f02aaf. [Accessed 18 02 2019].

[75] Kony, "Kony Quantum," 2019. [Online]. Available: https://www.kony.com/products/quantum/. [Accessed 20 05 2019].

[76] J. Wong, V. Baker, A. Leow and M. Resnick, "Magic Quadrant for Mobile App Development Platforms," Gartner, 2018.

[77] Adobe, "Adobe PhoneGap," 2019. [Online]. Available: https://phonegap.com. [Accessed 20 05 2019].

[78] Qt, "Mobile App Development with Qt," 2019. [Online]. Available: https://www.qt.io/mobile-app-development/. [Accessed 20 05 2019].

[79] RhoMobile, "RhoMobile Suite Documentation," 2019. [Online]. Available: http://docs.rhomobile.com/en/5.4/guide/welcome. [Accessed 20 05 2019].

[80] Sencha, "Sencha Ext JS," 2019. [Online]. Available: https://www.sencha.com/products/extjs/. [Accessed 01 05 2019].

[81] M. Willocx, J. Vossaert and V. Naessens, "A quantitative assessment of performance in mobile app development tools.," *2015 IEEE International Conference on Mobile Services*, pp. 454-461, 2015.

[82] S. Diwakar, "Titanium vs Phonegap vs Native application development," 21 06 2012. [Online]. Available: http://www.sapandiwakar.in/api-research-study-iphone-and-android-applications. [Accessed 11 03 2019].

[83] S. Roy Choudhary, "Cross-platform testing and maintenance of web and mobile applications," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014.

[84] G. Peal, "React Native at Airbnb: The Technology," Airbnb, 19 06 2018. [Online]. Available: https://medium.com/airbnb-engineering/react-native-at-airbnb-the-technology-dafd0b43838. [Accessed 18 05 2019].

[85] G. Peal, "Sunsetting React Native," Airbnb, 19 06 2018. [Online]. Available: https://medium.com/airbnb-engineering/sunsetting-react-native-1868ba28e30a. [Accessed 18 05 2019].

[86] M. Q. Huynh, P. Ghimire and D. Truong, "Hybrid app approach: could it mark the end of native app domination?," *Issues in Informing Science and Information Technology,* no. 14, pp. 49-65, 2017.

[87] R. Nunkesser, "Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development," *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft),* pp. 214-218, 2018.

[88] Intel, "intel/appframework: The definitive HTML5 mobile javascript framework," 22 05 2017. [Online]. Available: https://github.com/intel/appframework. [Accessed 27 05 2019].

[89] mgwt, "mgwt - Making GWT work with mobile," 2014. [Online]. Available: http://www.m-gwt.com. [Accessed 27 05 2019].

[90] P. R. De Andrade, A. B. Albuquerque, O. F. Frota, R. V. Silveira and F. A. d. Silva, "Cross platform app: a comparative study.," arXiv preprint arXiv:1503.03511, 2015.

APPENDIX 1. Source repositories

Source code for all the implemented demo applications is stored in publicly accessible git repositories.

Platform	**URL**
Ionic	https://github.com/anna-osipova/cpmdf-ionic
NativeScript	https://github.com/anna-osipova/cpmdf-native-script
React Native	https://github.com/anna-osipova/cpmdf-react-native
RubyMotion	https://github.com/anna-osipova/cpmdf-ruby-motion
Xamarin	https://github.com/anna-osipova/cpmdf-xamarin