VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Multiplatform Mobile Application Development Methodology

Metodika vývoje multiplatformní mobilní aplikace

2017 Bc. Ľubomír Sokolovský

Zadání bakalářské práce

Jiří Dvorský

Ukázka sazby diplomové nebo bakalářské práce

Diploma Thesis Typesetting Demo

+++

Podpis vedoucího katedry

+++

Podpis děkana fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. dubna 2016 .....

Zde vložte text dohodnutého omezení přístupu k Vaší práci, chránící například firemní know-how. Zde vložte text dohodnutého omezení přístupu k Vaší práce, chránící například firemní know-how. A zavazujete se, že:

1. podle § 5 o práci nikomu neřeknete,
2. po obhajobě na ni zapomenete a
3. budete popírat její existenci.

A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. Konec textu dohodnutého omezení přístupu k Vaší práci.

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 1. dubna 2016 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Táto práca sa snaží vyriešiť problém vyberania najvhodnejšieho vývojového nástroja pre multi-platformnú mobilnú aplikáciu. Zameriava sa na frameworky umožňujúce všeobecný vývoj aplikácií pre Android, iOS a, voliteľne aj, Windows. Práca analyzuje rozhodujúce faktory pri výbere vývojových nástrojov z teoretického aj praktického hľadiska. Praktická časť analýzy spočíva v demonštratívnej implementácii prípadov užitia, ktoré sú kritické pri vývoji mobilných aplikácií. Všetky prípady užitia sú implementované v 3 rozličných nástrojoch - Apache Cordova, React Native a Xamarin. Táto analýza vyústila v zostavení série metodických krokov, ktoré sprevádzajú svojho používateľa procesom výberu najvhodnejšieho mobilného multi-platformového vývojového nástroja. Metodika bola úspešne overená na skupine existujúcich alebo plánovaných mobilných aplikácií. Hlavným prínosom tejto práce je jednoduchý, no zároveň veľmi presný spôsob ohodnotenia vhodnosti ľubovoľného mobilného multi-platformového frameworku pre takmer akýkoľvek projekt.

Klíčová slova: multi-platformný, mobilná aplikácia, mobilný vývoj, Android, iOS, Windows, Apache Cordova, React Native, Xamarin, metodika

Abstract

This thesis tries to tackle the issue of choosing the most suitable development tool for multi-platform mobile app. It focuses on frameworks enabling general app development for Android, iOS and, optionally, Windows. The thesis analyses decisive factors in development framework selection both theoretically and practically. The practical analysis lies in demonstrative implementation of use cases crucial in mobile app development. All use cases are implemented in 3 distinct frameworks - Apache Cordova, React Native and Xamarin. The analysis resulted in composing a series of methodological steps which guide its user through the process of selecting the most suitable mobile multi-platform development tool. The methodology was succesfully verified on a set of existing or planned mobile apps. The primary benefit of this thesis is a simple, yet very precise way of evaluating the suitability of an arbitrary mobile multi-platform framework for almost any project.

Key Words: cross-platform, mutli-platform, mobile app, mobile development, Android, iOS, Windows, Apache Cordova, React Native, Xamarin, methodology

Contents

List of symbols and abbreviations

2D	–	2-dimensional
3D	–	3-dimensional
AI	–	Artificial Intelligence
API	–	Application programming interface
BLOB	–	Binary Large OBject
CI	–	Continuous Integration
CLI	–	Command-line Interface
CRM	–	Customer Relationship Management
CSS	–	Cascading Style Sheets
DB	–	Database
DevOps	–	software DEVelopment and information technology OPerationS
DOM	–	Document Object Model
ERP	–	Enterprise Resource Planning
GB	–	GigaByte
GHz	–	GigaHertz
GPS	–	Global Positioning System
GUID	–	Globally Unique Identifier
HAXM	–	Hardware Accelerated eXecution Manager
HDD	–	Harddisk Drive
HTML	–	HyperText Markup Language
IDE	–	Integrated Development Environment
IL	–	Intermediate Language
IoT	–	Internet of Things
IT	–	Information technologies
JDK	–	Java Development Kit
JS	–	JavaScript
MB	–	MegaByte
MBaaS	–	Mobile Backend as a Service
MMS	–	Multimedia Message Services
MVVM	–	Modelview-View-Model (architecture)
NDK	–	Native Development Kit
NFC	–	Near Field Communication
OS	–	Operating system
OTA	–	Over-the-air
PC	–	Personal Computer
PCL	–	Portable Class Library

PDA	–	Personal Digital Assistant
QR (code)	–	Quick Response (code)
RIM	–	Research in Motion - BlackBerry operating system
SD (card)	–	Secure Digital (card)
SDK	–	Software Development Kit
SMS	–	Short Message Services
SQL	–	Structured Query Language
SSD	–	Solid State Drive
UI	–	User Interface
UWP	–	Universal Windows Platform
UX	–	User Experience
VoIP	–	Voice over Internet Protocol
WinRT	–	Windows Runtime
WIP	–	Work In Progress
WORA	–	Write Once Run Anywhere
WYSIWYG	–	What You See Is What You Get
XML	–	eXtensible Markup Language

List of Figures

List of Tables

Listings

1 Introduction

Competition is a natural trait of any market. It offers customers with the possibility to choose between multiple variants. At the same time, it forces the producers to innovate and outweigh disadvantages of their products with bonus features.

Just like any other market, this is true also for smartphones and mobile operating systems. However, the benefits and variety for customers on one hand represent challenges for mobile app developers on the other. They stand before a difficult decision - to either implement their application multiple times for each operating system, or stay exclusive to a single platform and ignore all the others.

Another problem, besides the fragmentation, are the perpetual changes in operating systems themselves and their market shares. While only few years ago, Symbian was the dominant platform, since 2010 Android is the king of the smartphone world. Between years 2009-2016 there have been 12 major version changes and 24 API changes for Android. It is similar for iOS, currently the second strongest mobile OS, with 10 different versions since 2008. While Android and iOS changed their versions, they remained faithful to their respective development technologies. This cannot be said of Windows, which from Windows Phone 7 to Windows 10 Mobile swapped several different technologies (XNA, WinRT, Silverlight and UWP).

Implementing the same app over and over again using different languages and APIs is a boring and tedious task for developers, and a waste of time and money for IT companies. Soon, solutions and tools allowing development for multiple platforms, while writing the code just once, began to emerge. As of September 2016, there can be found more than 100 of these multi-platform development tools for mobile operating systems. There is a common fear that multi-platform applications are inferior compared to native development. However, according to several surveys, 81% developers claim multi-platform applications being as good as native (or even better), while saving 50+% of development time (compared to developing 2-3 native apps)[14]. However, the same study reveals that majority of multi-platform projects are planned for short term (up to 3 months of development).

Some of these tools offer code-free programming. Others provider optimization of web applications for mobile browsers. There are solutions for truly native apps developed with a single code base, or hybrid apps that are programmed as web apps, but have access to device hardware. And for game developers, there are multiple frameworks and engines for both 2D and 3D development.

However, choosing the right development tool can be a difficult task. Often, many products seem to provide the same functionality. The devil is always in the details, and discovering a missing framework capability in the middle of development process can result in wasting of several months of work. The purpose of this thesis is to create a methodology, that will guide the developers, architects and managers and help them to find the most suitable development tools, that will fulfill all their criteria.

1.1 Thesis overview

Because there are so many different multi-platform development solutions and not all of them can be covered, we will make the target group a little narrower. In the first chapters, this thesis will try to determine which operating systems are still relevant for the multi-platform development. We will analyze the global mobile operating systems market, smartphone sales, application revenues and developers' focus.

After that we will take the relevant operating systems and focus only on multi-platform development tools which enable creating apps for the most used OSs. Since we want to focus on the possibility of using device hardware and native API, we will ignore all web-based solutions and game engines. The thesis will omit also all tools which do not enable general development options (e.g. tools focused only on a single company or activity).

From the products which passed the filter, we will pick a few ones and test their usability on a couple of pre-defined use cases. The thesis will discuss and compare the implementation details, strengths, weaknesses and limitations of individual development tools. The results of these implementations, as well as theoretical research, will be helpful in establishing a set of factors crucial in the creation of the resulting methodology.

The methodology itself will consist of several parts. The first part will validate projects and requirements, to which the methodology is relevant. The second part will be the core of the methodology, helping the user determine the right multi-platform development tool. The last part will contain suggestions for additional research which can be performed by the user if the methodological results were indefinite.

1.2 Remarks

The reader may be familiar with the term "cross-platform" development. This term is interchangeable with "multi-platform" [9], which is used in this thesis. Both cross-platform and multi-platform software development refer to the process of creating a piece of software that can be run on multiple platforms.

Considering the Windows operating system, Microsoft had several products for mobile devices. From 2000 to 2010 they delivered several versions of the Windows Mobile product line. It was succeeded by Windows Phone 7, 8 and 8.1. In 2015, Microsoft introduced their Windows 10 operating system with common core for desktop, tablet, smartphone and IoT development [48]. The version for smartphones is called Windows 10 Mobile [49], similar to the old product line from the previous decade, although being an iteration of Windows Phone 8.1. To avoid long and confusing names and distinguish the mobile and desktop versions, we will call the Microsoft operating system for smartphones simply Windows. In a more narrow context, under Windows we will understand Windows 10 Mobile and the previous version, Windows Phone 8.1, which has forward-compatible applications [6]. Regarding tables and charts, if not stated otherwise, the data are presented for the year 2016.

2 Relevant mobile operating systems

In order to create a methodology for choosing the right mobile multi-platform development tool we first need to specify which platforms will be targeted. Throughout the time there have been many more or less successful mobile operating systems. The trend shifts in mobile market are one of the most dynamic compared to other market types. This is caused mainly by two following factors:

1. The mobile market is very young. Mobile phones became commercially available to the wide public only about 30 years ago, in the late 1980s and early 1990s.

2. They became increasingly popular, making them an interesting technology for development and investing, offering high profit.

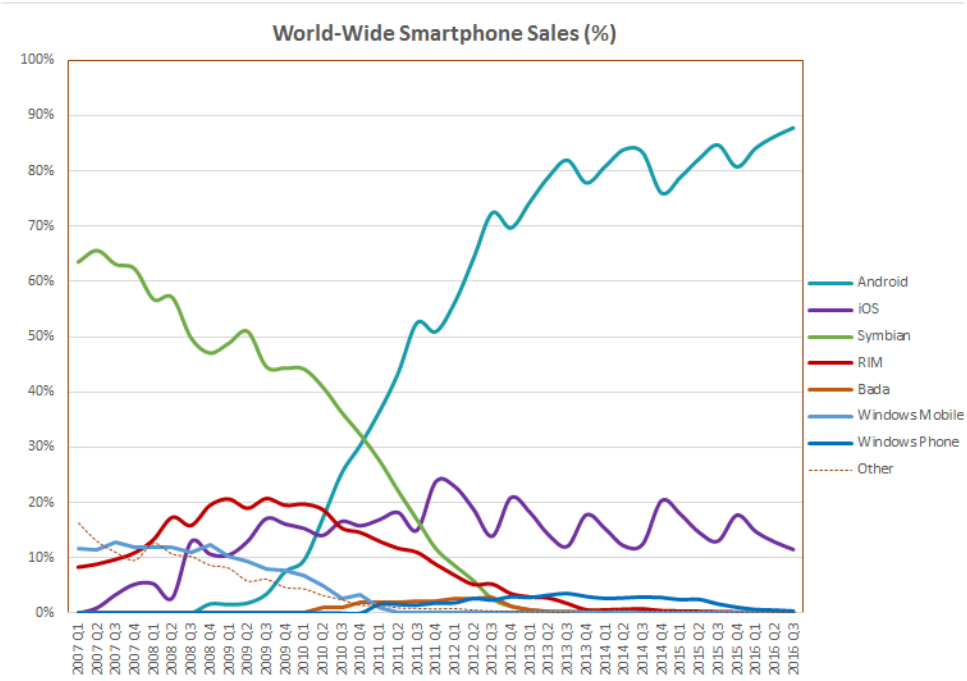

Figure 1: World-wide smartphone sales [28]

By the end of millennium, mobile phones were almost in every family in the western hemisphere [25]. However, they were still used mainly for telephony and sending SMS, with occasional MMS in the first years of 2000s. There were only few devices, which tried to combine mobile phones with PDAs - small personal computers, targeted mainly for business and enterprise environments. They were running on operating systems such as Palm OS, BlackBerry OS and Windows CE, which later evolved into smartphone operating systems.

These hybrids of mobile phones and PDAs were the first predecessors of today's smartphones. Probably, the most known series are Nokia 9000 Communicator devices. In 2000, Ericsson

released E380, which was the first device marketed as "smartphone". It was also the first device running Symbian OS, the dominant mobile operating system until 2010.

Several other companies have seen the potential of these hybrid devices, which combined mobility, telephony, computing power and allowed connection to the ever-growing Internet. Soon, Symbian, Palm, BlackBerry and Windows Mobile got new rivals. It was iOS in 2007, Android in 2008 and Bada in 2009. These new players shattered the mobile market - and two of them even surpassed the old operating systems.

Today, the situation is very different than it was ten years ago. Symbian OS, Palm OS and Bada are all discontinued projects. BlackBerry OS (known as RIM) is at the brink of existence. Windows Mobile was replaced by Windows Phone, but has lost a large portion of the market. The dominant roles are held by iOS and Android.

2.1 Current situation

Specifying, which operating systems are relevant to our methodology is not a trivial task, if we want the methodology to be accurate also for years to come. Yet, there are several factors, which may help us determine the trends to come:

1. Is the OS still officially supported?
2. What is the current market share of active users?
3. What are the market shares of the sales?
4. What are the revenues for developing applications for a particular OS?
5. How many developers support the OS?

Answering these questions will help us assign priority to individual smartphone platforms.

2.2 Supported operating systems

As of the August 2016, the following smartphone operating systems are still being developed and supported:

- Android (with multiple modifications, such as Cyanogen, Fire OS or MIUI)
- BlackBerry 10 (RIM OS)
- Firefox OS
- H5OS
- iOS
- Sailfish OS

- Tizen
- Ubuntu Touch OS
- Windows 10 Mobile (previously Windows Phone 8.1)

2.3 Smartphone OS usage market share

The following table shows the percentage of smartphone users for each smartphone OS (for the year 2016) [29]:

Table 1: Smartphone OS usage market share

Platform	Market share (%)	Devices (approximate, in millions)
Android	65.33%	1371.93
iOS	27.8%	583.8
Windows	2.64%	55.44
BlackBerry	1.18%	24.78
Symbian	1.15%	24.15
Other	1.9%	39.9

The approximate number of active devices is based on the figure that there is about 2.1 billion smartphones in use worldwide [33]. With more than a billion active devices, Android is clearly the most used mobile operating system. iOS is also very strong, having more than a quarter of the market share. The other platforms are far behind. Windows users are just a tenth compared to Apple's iPhone and iPad users. The portion BlackBerry's RIM users is just slightly above 1%. Symbian, having a similar share although being discontinued, is still used by more than 20 million users.

2.4 Smartphone OS sales market share

The smartphone sales market share can help us predict the future growth or decline of a certain platform. As seen in Figure 1 earlier in this chapter, there was a dramatic shift in 2010 and 2011, when Android surpassed Symbian in the percentage of sold smartphones.

By the end of 2013, Symbian and Bada were pronounced discontinued. Windows Mobile transformed to Windows Phone in 2010. However, the transformation was not very successful. Windows Phone's sales peaked at 3.2% in 2013 and have been decreasing ever since. Compared to the 12% market share of Windows Mobile in 2007, this situation looks very bleak for Microsoft.

Even worse, however, are the numbers for another former major smartphone OS. BlackBerry's RIM had almost 20% market share in 2009. Now, for two years its sales have dropped below 1%. iOS on the second place has its sales market share fluctuating around 15%, but there is a large gap between the second and the first place. Android, now with stunning 84.1% of all smartphone sales is the major force in the industry.

Table 2: Smartphone OS sales, various reports from [20]

Year	RIM	Symbian	iOS	Android	Bada	Windows	Other
2016	0.19%	-	14.78%	84.1%	-	0.7%	0.23%
2015	0.37%	-	16.26%	80.52%	-	2.47%	0.38%
2014	0.6%	-	15.4%	80.7%	-	2.8%	0.5%
2013	1.9%	0.1%	15.6%	78.4%	0.2%	3.2%	0.6%
2012	5%	4.2%	19.1%	66.4%	2.3%	2.5%	0.5%
2011	10.9%	18.74%	18.92%	46.53%	2.01%	1.65%	1.21%

Table 3: Regional smartphone sales in 2016

%	Android	BlackBerry	iOS	Windows	Other
USA	58.2	0.1	39.1	2.6	0
Mexico	90.5	0.7	5	2.6	1.2
Brazil	92.4	0	3.3	4.1	0.1
Argentina	83.5	3.4	0.9	9.1	3.2
UK	52.6	0.2	38.6	8.6	0
Germany	74.2	0.6	19.3	5.9	0.1
France	71.8	0.5	19.3	7.8	0.5
Spain	87.8	0	11.4	0.8	0
Italy	78.1	0.2	14.4	7.2	0.1
Russia	71.2	0.6	14.8	10.6	2.7
China	73.9	0	25	0.9	0.3
Japan	48.7	0	50.3	0.5	0.5
Australia	52.6	0.2	41.2	5.4	0.6
World	84.1	0.19	14.87	0.7	0.23

Yet, the raw sales percentages may not necessarily correspond to the change of user base in absolute numbers. There is the possibility that Android users are buying new devices more often compared to iOS or Windows users, resulting in higher sales. But it definitely shows the trends whether a certain platform is experiencing its rise or fall.

For certain businesses, the global data may not be relevant, since regionally the sales percentages differ substantially.

2.5 Smartphone app stores revenues

The sheer numbers of mobile platform users or device sales is one thing. But many developers - and almost all businesses - are motivated by something else entirely. Money may be a very decisive factor in choosing, which platforms will be targeted and which will be omitted.

For some time now, it is common knowledge that Android users are not as willing to pay for apps as their iOS colleagues [31]. This has been true also in recent years. Although the

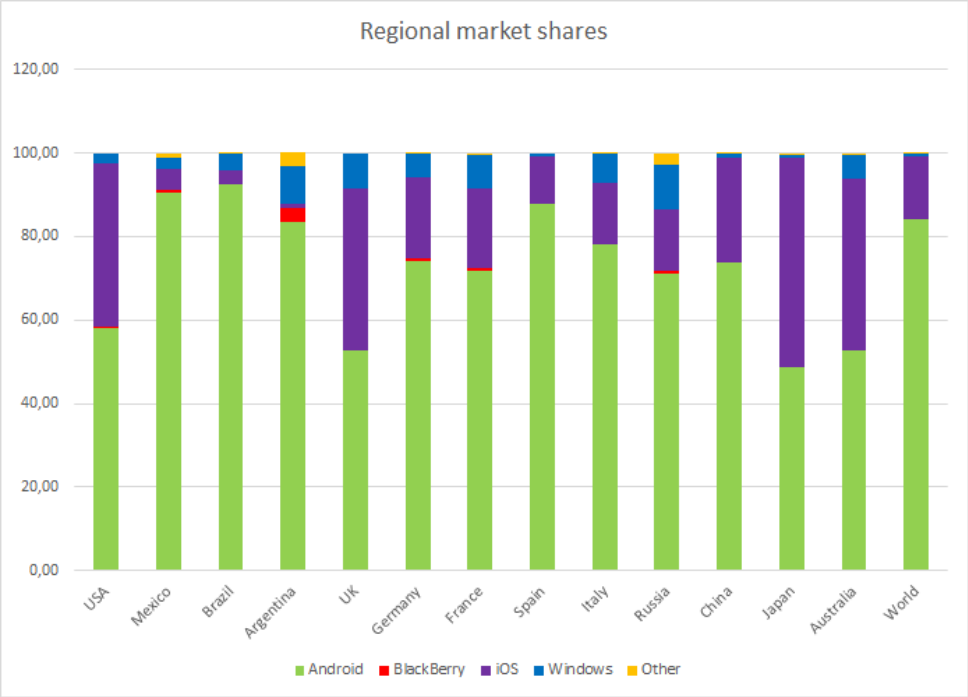

Figure 2: Regional market shares [40]

number of downloaded apps in Google Play is twice as high compared to Apple's App Store, iOS is creating 70% more revenue compared to Android [24].

These data are backed also by a survey performed by InMobi [22]. While on average an Android developer[1] makes $4900 per month, an iOS developer earns $8100. However, there is a much more interesting discovery made by the survey. Developers targeting Windows earn the most - on average $11400 per month.

The article explains this by the small amount of apps found on Windows Store. Smaller market means less competition and this has dual effect on the market. The discoverability factor of your application is much higher and the chance there will be a free alternative is much smaller. With no competition, a developer is free to increase the cost of an application [32].

However, some [17] suggest the survey numbers may be skewed due to smaller sample of Windows developers. Even if the Windows monthly revenue is not accurate, the numbers make it still a very interesting platform. This cannot be said of other platforms. From the ones not discontinued, BlackBerry is the strongest one. However, its revenues do not figure in many recent surveys. In older articles, BlackBerry app development seems to be the least rewarding [34].

[1]The survey does not distinguish between individual developers, developer groups or companies - all three are represented by the term "developer"

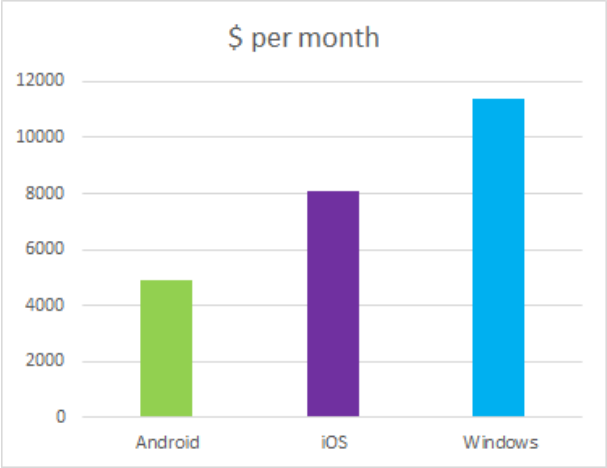

Figure 3: Average app revenues from individual app stores, per month

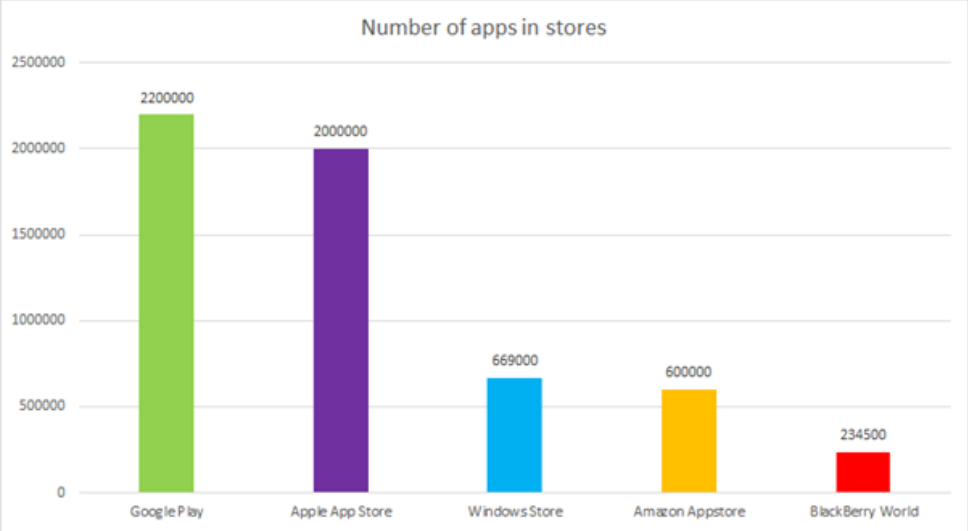

Figure 4: Estimated number of apps in individual app stores.

Table 4: Portion of developers creating apps for individual operating systems

	Q1 2014	Q3 2014	Q1 2015	Q3 2015
Android	71%	70%	71%	71%
BlackBerry	14%	11%	13%	unavailable
iOS	55%	51%	54%	51%
Windows	26%	28%	30%	27%

Table 5: Primary platform for individual developers

	Q1 2014	Q3 2014	Q1 2015	Q1 2016
Android	35%	40%	40%	41%
BlackBerry	3%	2%	2%	1%
iOS	38%	38%	37%	39%
Windows	4%	7%	8%	9%
Web & others	20%	13%	13%	11%

2.6 Operating systems targeted by developers

There is one more factor that may decide about the future of a mobile operating system, and that is the number of developers creating new apps. A rich and healthy application store environment may help to attract new customers.

As we have seen in figure 4, both Google Play and Apple App Store are very rich application markets. The numbers exceeding 2 million may even be discouraging for some developers, since they present both low discoverability and high competition. This can be vastly different in Windows and Amazon Stores, where the numbers are smaller by 2/3. For BlackBerry, the number is even smaller. And unlike other stores, BlackBerry World does not seem to grow with a comparable rate [5].

So, how many developers create apps for the individual operating systems? According to VisionMobile [39] more than two thirds develop for Android. About one half creates applications for iOS and only one quarter for Windows.

As it seems, part of developers is slowly abandoning the mobile web apps development and choosing one of the three dominant platforms as their primary. It is interesting that the number of developers having Windows as their primary platform is almost the same as the number of developers targeting Windows exclusively. And even though there is much more developers creating apps for Android compared to iOS, the number of developers claiming both operating systems as their primary is roughly the same.

When put in relation with the market shares and revenues, we can get some interesting data. Although Apple App Store is producing 70% more revenue than Google, only 12% developers create iOS exclusive apps, compared to 28% Android exclusives. Android beats iOS also in the number of developers picking it as primary platform and overall number of developers. It

Table 6: Developers creating apps exclusively for particular OS

Android	iOS	Windows
28%	12%	8%

does not seem probable that developers and IT companies would favor a larger user base, which produces smaller profit. However, there might be other factors discouraging developers from targeting iOS:

- ObjectiveC is more complex and difficult to learn, compared to Java (this might change with the introduction of Swift)
- iOS can be built only on MacOS devices (Android apps can be developed on MacOS, Windows and Linux)
- Publishing apps on Apple App Store is more complex and expensive compared to publishing on Google Play

Another interesting pair to compare is BlackBerry and Windows. Both have small user bases and low sales. But while BlackBerry has small revenues and only 1% of developers targeting it as primary platform, Windows has the highest revenues and 9% of developers targeting it as primary platform and 27% developing also Windows apps. This can be seen also on their app stores - Windows Store has 3-times more applications than BlackBerry World and is still growing, while the latter stagnates for several years. With Windows 10 unifying the development for desktop, tablet and mobile, the numbers can grow even faster and eventually, it might be the developers who will save the Windows mobile platform.

2.7 Tablets

So far the thesis has been concerned mainly by smartphones. Yet, there is another major group of mobile devices - tablets. In the current market, the lines between individual device categories is often blurred. Between smartphones and tablets there are phablets and the gray zone between tablets and notebooks is composed of netbooks, ultrabooks and tablets with detachable keyboard. Moreover, with the introduction of Continuum for Windows [47], it is hard to tell whether there is a border at all.

Yet, the development environment for iOS distinguishes between apps for iPhone and iPad [23] and also Android has special guidelines for adjusting apps for tablets [41]. Therefore, let us take a look at the OS market shares of tablets as well.

Inferring from smartphone market shares, it is no surprise that Android is the most used operating system, followed by iOS and Windows respectively. BlackBerry PlayBook does not figure in the statistics at all. However, very interesting is Windows on tablets compared to

Table 7: Global market share of tablet operating systems [44]

	2013	2014	2015	2016
Android	62.36%	67.33%	67.4%	66.2%
iOS	33.93%	27.57%	23.9%	22.4%
Windows	3.5%	5.09%	8.6%	11.3%

smartphones. While the Windows smartphone share was below 3% and decreasing, in tablet world Windows is on the rise. It is estimated, that by 2020 Windows will have almost 20% of the market share, similar to iPads [51]. Already now, Windows is the dominant operating system, when it comes to 2in1 devices, like detachable tablets [16]. With the introduction of the Windows universal platform development paradigm, Windows starts to be supported also by tools, which had not taken it into consideration before [13].

2.8 Conclusion

Based on the previous factors, we can filter out three mobile operating systems, which will be relevant to our methodology - Android, iOS and Windows. The support of two of them - Android and iOS - will be a crucial factor, when selecting suitable development tools.

Android has clearly the largest market share, both for smartphones and tablets. With the exception of Japan, it has also the highest sales, biggest app store and is targeted by the largest portion of developers. iOS has less than a half of Android's smartphone market share, and even smaller numbers for tablets and future sales. Still, it takes up 1/4 of the market, is targeted by more than half of the developers and Apple App Store has 70% higher revenues than Google Play.

Windows is a debatable operating system. In smartphone world, it has low market share and sales below 1%. However, it has the highest relative revenues, has growing tablet market share and more than 25% developers create apps for Windows Store as well. Moreover, around 36% of multi-platform tool users wish their tool to support Windows development as well [14]. Therefore, we will not omit development tools that do not support Windows. Yet, for those that do, we will compare Android and iOS capabilities with Windows as well.

There are two more operating systems - BlackBerry and Symbian - that have around 1% share of smartphone users. Although these OSs may be interesting for some niche markets, the thesis will not consider them as primary points of interest. Symbian is a discontinued project and BlackBerry seems to be transforming into a hardware company, producing Android devices. Occasionally, these platforms may be referred in relation with individual multi-platform development tools.

3 Mobile multi-platform development tools

For each software platform there exists one or two official tools, languages, APIs and supporting software making up a development stack. And then there exists a ton of various modifications, customizations, frameworks and corporate tools to enhance and speed up the development for a narrow range of implementation problems.

The same is true for mobile development. There are the official development tools [45]:

Table 8: Official mobile OS development tools

Platform	IDE	Language	User interface	Desktop development	App market
Android	Android Studio	Java	XML	Linux, macOS, Windows	Google Play
BlackBerry (RIM)	Momentics	C++	Qt & QML	Linux, macOS, Windows	BlackBerry World
iOS	XCode	ObjectiveC	ObjectiveC or Cocoa Pods	macOS	Apple iTunes
Windows	Visual Studio	C#	XAML	Windows	Windows Store

That were the official tools, initially created by the owner companies to enable developers create apps for their respective platforms. But already now, things start to get complicated. You can create C++ apps also for Android and Windows. And, although you will need to implement an Objective C wrapper, you can create C++ libraries for iOS, as well.

It seems, we have found the perfect cross-platform tool. As the reader will learn later, C++ can really be used for cross-platform development, but it is not that straightforward. The C++ support for Android and iOS is limited. And each platform has a different application lifecycle, different hardware peripheries, different APIs and libraries. In the end, developing in C++ would result in developing 4 different applications - but in a much more difficult way.

In the list above, there is one more language known for its “multiplatformicity” - Java. However, its “Write once, run anywhere” is not so true in the mobile world. While BlackBerry was forced by its declining market share to support Android apps written in Java, iOS and Windows are not so supportive. Yet, even for Java there are tools which enable it to spread even to those platforms. So, let us discover those cross-platform development options up close.

3.1 Multi-platform development approaches

As we have seen above, there is no common development tool for all mobile platforms. Even if we ignored the differences in programming languages, there are still various paradigms in application lifecycle, access to native APIs, interactions between processes, etc.

For many companies - or individual developers - it is too expensive to have an expert for each platform and each language. They have to struggle, to either omit certain platform, or learn new skills for each OS. However, certain aspects of every mobile application can be abstracted and standardized. And this is where multi platform development tools come into play. They take a single language, single development tools and abstract application aspects across all mobile operating systems to create a unified code based. The level of abstraction is a crucial factor which can help us differentiate between individual multiplatform tools. If the level of abstraction is high, we get a "Write once, run anywhere" approach, but lose the control over the specifics of individual platforms. If the level of abstraction is lower, we can access the platform specific features, but we have to implement them individually and the shared code base is smaller.

There exist multiple ways how to divide and categorize multi-platform development tools. Research2guidance [14] classifies multi-platform tools according to how the app is created into following classes: Web app toolkits, App factories, Cross-platform integrated development environments, Cross-platform integrated development environments for Enterprise, Cross-platform compilers, Cross-platform services. Silva's division [42] is a bit different: App factory, Web-to-native wrapper, Runtime and Domain-specific language.

Probably the best known division is into following 3 categories [10] [26] [45]:

- Native apps - mobile applications that are installed on the device, executed by the OS and have full access to native APIs and sensors.
- Web apps - HTML, CSS and JavaScript web apps accessed from the mobile browser. They do not have to be installed, but have no access to native APIs or sensors (with the exception of camera and file system).
- Hybrid apps - web apps that are bundled within a customized web view container. Through this web container they are able to access the native APIs and sensors. The app has to be installed, but provides all benefits of a web app.

However, this categorization does not distinguish between truly native approach and tools that mimic native behaviour with custom runtimes, virtual machines or cross-compilation [15]. Therefore, this thesis uses following categories: mobile web apps, hybrid apps, interpreted apps and cross-compiled apps. [11][2][18]. In [12], Nielsen introduces a fifth approach - source translation, or transpiling. However, this last approach is usually used in combination with one of the previous approaches.

3.1.1 Mobile web apps

It is often stated by many developers, that the only true multi platform development is possible only via standard web technologies - HTML, CSS and JavaScript. It is true that all smartphones have a web browser capable of displaying web pages and apps. Mobile web apps are exactly that - classic web apps adapted for a mobile browser. However, each browser can render the web app

in a different manner, resulting in inconsistent user experience. Moreover, the application can be slow, depending on the connection speed and browser interpretation capabilities. Plus, there is very restricted, or no connection to the native APIs and hardware tools.

3.1.2 Hybrid applications

Hybrid apps try to minimize the problems of web apps, while benefiting from their strengths. They wrap the JavaScript (and HTML + CSS) application in a lightweight native wrapper, most often a webview stripped of almost all functionality. Unlike mobile web apps, hybrid applications are available also offline and have much better access to native APIs, interfaces and peripheries. However, the native functionality may differ for each OS. The application is still in a web browser, which drastically decreases its performance. Plus, the nature of JavaScript does not allow to use multiple threads. Also, hybrid applications do not have access to native UI. A drawback for some may be a benefit to others - the application will look the same on each operating system.

3.1.3 Interpretation

Interpreted apps utilize a custom runtime environment or virtual machine (like JavaScript engine or Java Virtual Machine), while interfacing with the native platform at the same time. Interpreted apps are written predominantely in JavaScript, similar to mobile web apps and hybrid apps. However, they are not executed in a web browser, so there is no sand-box limitation and no HTML DOM. As the name suggests, the code is directly interpreted by an included JavaScript engine. Access to native APIs and sensors is almost unlimited. Various tools provide various UI rendering techniques - some use native UI elements (like Appcelerator and NativeScript), others provide a platform-agnostic unified UI (e.g. Codename One and React Native). Performance-wise, interpreted apps are in between hybrid and cross-compiled apps. However, they allow faster protoryping process, since updating code does not require a build process.

3.1.4 Cross-compilation

To the developer, creating a cross-compilation or interpreted application may seem very similar. However, cross-compilation is a much more complicated process. Cross-compilation describes the practice of developing an app using platform-agnostic API or programming language. The cross-compiler transforms the code into native platform-specific executable apps (or libraries). While some tools (e.g. Kivy or Qt) apply this process to all layers - from data layer to UI - resulting in a true WORA approach, others allow more platform-specific access (like RubyMotion or Xamarin). Cross-compiled apps have performance closest to truly native apps. Due to complicated process of cross-compilation and OS differences, some tools combine cross-compilation with interpretation (e.g. Mono.Android interprets code from .NET libraries).

3.1.5 Source translation

The last approach, source translation is based in translating one high-level programming language into another. The app is then run from the platform-specific programming language. Similar to cross-compilation and Virtual Machines, the advantages of this technique is the native performance, native UI and almost unlimited access to native API. However, no framework uses pure source translation approach. Rather, it is combined with other approaches - hybrid, interpreted or cross-compiled - where only part of code is translated.

3.2 MBaaS

For many businesses the use case of an internal company application is often the same - to collect inputs, store them, and to perform various analyses, display data and notify employees when necessary. This requires a database server, or other cloud storage, web api, web server, and often both a mobile and web application. To implement the complete work flow correctly and efficiently a whole group of software engineers, developers and coders is needed. Many companies would need to outsource development of such project, or hire several freelance developers. Further costs connected with support services and the need to reveal company secrets to 3rd parties are all risks that many are not willing to undertake.

For these purposes, complex solutions known as Mobile Backend as a Service (MBaaS) began to emerge. An MBaaS allows the design and development of a database, web api, mobile and web app in a single tool. These tools often provide visual programming, with occasional customization of code in JavaScript, or another scripting language.

Depending on the MBaaS provider, some tools allow integration with an existing backend, creating only a bridge to the client application and adding integration to social networks. The mobile applications created with an MBaaS are usually mobile web apps or hybrid apps. Most MBaaS services are commercial, with limited open source support.

3.3 Multi-platform development frameworks and tools

As mentioned in earlier chapters, as of 2016, there exists more than a hundred different tools and services for mobile multi-platform development. A crucial factor of this thesis, however, is the focus on native API usage. Thus, a lot of these tools are not relevant for further discussion. Here is the list of all criteria that a tool must pass in order to be investigated further. Each criterion contains a short description, why is it important, and a few examples of tools that did not pass.

1. **The tool is not discontinued.** For obvious reasons, only frameworks and platforms that are still supported and developed will be considered. A discontinued project may be interesting for experimental purposes, but its use in production is highly unlikely, and questionable. Examples: MoSync, RoboVM, WidgetPad

2. **At least Android and iOS are supported.** Since Android and iOS are the two dominant players in the smartphone world, any multi-platform tool must support at least these two operating systems. The support for Windows is advantageous, but optional. Examples: AML, Appinventor, Java ME, Kallipso

3. **The tool is not game-centric.** Many game frameworks work seamlessly across multiple operating systems. However, they neither intend, nor are able to use the native interfaces. Examples: Marmelade, Unity, Wave

4. **General app development is possible.** This may be considered an extension of the previous criterion. The tools must allow development of almost any kind of mobile application. All tools bound to a particular business will be rejected. This includes also all MBaaS solutions which do not allow server-less implementation of mobile application. Examples: appMobi (mobile security), Appticles (publishing), i-exceed (banking), Pegasystems (customer engagement), and rejected MBaaS tools (AnyPresence, AppGyver, Kinvey, Mobile Frame, MooFWD)

5. **The tool must allow access to most used native APIs.** Access to the camera, GPS, accelerometer, gyroscope, media, file system and local database must be granted. This criterion eliminates all mobile web app frameworks, since their access is limited by the rendering browser. Examples: AppPress, Dojo, Bootstrap Mobile, jQuery Mobile, Sencha Touch

The complete table of all mobile multi-platform development tools which passed all criteria can be found below. Out of more than 100 tools, only 25 fulfilled all necessary criteria. More information about the rejected tools can be found in [12].

As the reader can see, the table still feature several development tools using the web application approach, altough it was said, those tools will be removed. This is due to the fact that some frameworks allow the developers to create various output applications, e.g. both hybrid and web apps.

Some MBaaS providers allow creating offline apps - their tools have been added to the table. They can be easily distinguished by the word "yes" in the MBaaS column. Similarly, all tools based or supporting Apache Cordova builds and plugins have the word "yes" in the Cordova column.

Naturally, all tools support app development for Android and iOS. It is worth noting that more than a half supports also Windows mobile development, with 3 more planning to do so in the near future [50] [30] [38]. BlackBerry support is much sparses, with only 4 tools fully supporting RIM builds. 3 others have partial or unofficial BlackBerry support.

It is clear that the most popular programming language is JavaScript. Most tools take either hybrid or interpreted approach. Only a handful of them are cross-compiled.

Table 9: Selected multi-platform development tools

Product	Language	Approach	MBaaS	Apache Cordova	Windows	BlackBerry
Alpha Anywhere	Code-free, JavaScript	Hybrid	Yes	Yes	No	No
Appcelerator	JavaScript	Interpreted	Yes	No	Yes	No
Appery.io	Code-free, JavaScript	Hybrid, Web apps	Yes	Yes	Yes	No
Aquro	JavaScript	Hybrid	Yes	Yes	No	No
Codename One	Java	Interpreted	No	No	Yes	Yes
Corona Labs	Lua	Cross-compiled	No	No	Yes	No
Embarcadero	C++, Delphi	Cross-compiled, Web apps	No	No	Yes	Partially
Fuse	JavaScript	Interpreted	No	No	No	No
Instant Developer	Code-free, C#, Java	Hybrid	Yes	No	Yes	No
Ionic	AngularJS, JavaScript	Hybrid	No	Yes	Yes	Unofficial support
Kivy	Python	Cross-compiled, Hybrid	No	No	Yes	No
Kony	Code-free, JavaScript	Interpreted, Web apps	Yes	No	Yes	Only web apps
Monaca	JavaScript	Hybrid	Yes	Yes	Yes	No
NativeScript	AngularJS, JavaScript	Interpreted	No	No	Early access	No
NeoMAD	Java	Cross-compiled	No	No	Yes	Yes
NS Basic	BASIC, JavaScript	Hybrid	No	Yes	Only as web app	No
PhoneGap	JavaScript	Hybrid	No	Yes	Yes	Yes
Qt	C++	Cross-compiled	No	No	Yes	Yes
React Native	ReactJS, JavaScript	Interpreted	No	No	Planned	No
RubyMotion	Ruby	Cross-compiled	No	No	No	No
Smartface	Code-free, JavaScript	Interpreted	Partially	No	No	No
Tabris.js	JavaScript	Interpreted	No	Yes	Yes	No
Telerik Platform	AngularJS, JavaScript	Hybrid, Web apps	No	Yes	Yes	No
ViziApps	Code-free, JavaScript	Hybrid, Web apps	Partially	No	No	No
Xamarin	C#	Cross-compiled	No	No	Yes	No

4 Development tools evaluation

Already now we are able to distinguish several decisive criteria to choose a development tool for multi-platform mobile applications. The supported operating systems, programming languages, price, licencing, etc. However, to determine also other criteria, not visible on the first sight, further investigation needs to be done. We will pick a few development tools from previous chapter for evaluation and implement several use cases in them.

4.1 Development tools chosen for evaluation

To test the differences between individual tools, the most popular tool[2] from each build approach was chosen. Xamarin, representing the cross-compiled group, serves as a reference framework, due to author's previous experience with it. The largest group, hybrid apps, will be represented by its most known member - Apache Cordova. From tools using interpretation, React Native is on the rise.

4.1.1 Xamarin

Xamarin is a software company producing a mobile multi-platform development tool of the same name. It was established in May 2011 and acquired by Microsoft in February 2016. Xamarin (the platform) allows developers to code in C# for Android, iOS and Windows. Developers can use the Visual Studio IDE on Windows or Xamarin Studio on both Windows and MacOS. However, only Android applications can be built on both operating systems. To create an iOS application, developer needs a Mac device to perform the build. Likewise, Windows (Phone) app can be developed only on a Windows (Desktop) machine.

On iOS, Xamarin compiles Ahead-of-Time to native assembly, while on Android there is an intermediate compilation to IL code [27]. There are in fact two ways an application can be created. The classical approach allows sharing common business logic in a PCL (Portable Class Library - a C# library for multiple platforms). However, the individual user interfaces and platform-specific behaviour has to be implemented for each platform individually.

The second way to develop applications for Xamarin is via Xamarin.Forms. This is a custom-made UI framework, that removes the need to implement user interfaces for each platform individually. However, creating platform-specific behaviour is still possible (and in some cases required). With Xamarin.Forms the amount of code shared across platforms can be up to 100% [27].

4.1.2 Apache Cordova

One of the most popular multiplatform development tools is definitely PhonGap. PhoneGap is the original and most popular distribution of Apache Cordova [1], which is an open source devel-

[2] According to a joint popularity index from Google Trends and G2Crowd

opment framework enabling app development using HTML 5, CSS 3 and JavaScript. Cordova provides a native wrapper with a web view (or equivalent component) and accesses hardware peripheries with JavaScript API. The application itself is built within the web view wrapper. This type of app development is called hybrid.

Although the application is installed natively and runs offline, the UI elements are often different, access to native API is restricted and the application tends to have decreased performance. Plus, for hackers and copycats it is very easy to download the installation package and extract the JavaScript files, since they are generally included as standard web content [7].

Yet, developing using Apache Cordova (and other hybrid tools) is very fast, and some obfuscation tools exist. It is worth noting that PhoneGap is not the only distribution of Apache Cordova. There are many others, like Ionic, allowing development in Angular, NSBasic, developed in Basic, or Telerik, which supports JavaScript, Angular and TypeScript.

For the evaluation, the Visual Studio distribution of Apache Cordova was chosen. By default, it offers development in HTML 5, CSS 3 and JavaScript or TypeScript. However, it contains also templates for Angular projects and is directly recommended by the Ionic framework [21].

4.1.3 React Native

React Native is a multi-platform development tool created by Facebook. It uses similar technology as ReactJS and converts it into Android and iOS apps. Support for Universal Windows Platform is on the way and there already exists a unofficial plugin for Windows development[37].

Although interpreted frameworks may seem similar to hybrid tools due to the use of JavaScript, there is no WebView container and no HTML DOM. Interpreted tools use alternative approaches to UI representation: Appcelerator and NativeScript use XML, React Native utilizes JSX. Although some interpreted frameworks provide native look and feel for the UI layer, due to the nature of JSX, React Native renders a platform-agnostic UI resembling web pages.

JavaScript interpreted approach has been present for a while, thanks to Appcelerator. However, only in recent years it got more traction with the declining popularity of PhoneGap and the rise of tools like Fuse, NativeScript, Tabris.js and, most prominently, React Native. The only interpreted framework tool studied in this thesis, which does not use JavaScript, is Codename One with its Java implementation.

4.2 Use case definition

This is the list of use case scenarios, that will be implemented by each developer tool:

1. **Hello world!** - the first scenario is the simplest one. The system will provide an almost blank application, with a single button. After clicking on the button, a “Hello world” text will be toggled.

2. **Persistent storage** - this scenario will test the ability of the development tool to store and load data into a persistent storage. The following types will be stored: string, float, binary data and a more complex object.

3. **Camera & GPS** - the third scenario describes the most widely used hardware peripherals - camera and GPS. In this scenario, we will use the camera to take a photo and the GPS to add the coordinates of current location.

4. **Custom UI element and using native code** - the next scenario explores the possibility of using a custom UI element. We will try to implement a circular progress bar. If there is the possibility to render a native element from the multi-platform framework directly, we will use that. Otherwise, we will call a native random number generator and then fill the circular progress bar to generated percentage.

5. **Push notifications** - in the fifth scenario we will try to take advantage of the notification API of each platform. We will research the possibility to display push notifications sent from a web application. Because each platform handles the notifications differently, we will use only basic notifications without advanced features.

6. **Tablet optimization** - the sixth use case will test the ability of the development framework or platform to adjust the UI elements for various screen sizes.

Note, that the use cases cover only those aspects of multi-platform development, that can be achieved on all target operating systems (Android, iOS and Windows). Therefore, we will not test platform-specific features, such as fragments, widgets, life tiles, etc.

4.3 Hardware and software configuration

The evaluation of development tools is performed on on a Windows machine for Android and Windows Phone. For iOS, the built (or whole development) is done on an iMac device. The specifications can be found in following tables.

Table 10: Desktop configuration

OS	Microsoft Windows 10 Home	macOS Sierra, v 10.12.3
Model	Asus N551J	iMac 21.5-inch, Mid 2011
Processor	Intel Core i7-4720HQ x64, 2.6 GHz	Intel Core i5, 2.5 GHz
Memory	16 GB	16 GB
Storage	120 GB SSD, 950 GB HDD	512 GB HDD

Table 11: Mobile configuration

OS	Android Marshmallow 6.0.1	iOS 10	Windows 10 Mobile
Model	Sony Xperia Z5 Compat (E5823)	iPhone 5S	Nokia Lumia 735
Processor	Qualcomm Snapdragon 810, Octa-core x64, 2 GHz	Apple A7 chipset, Dual-core x64, 1.3 GHz	Qualcomm Snapdragon 400, Quad-core x64, 1.2 GHz
Memory	2048 MB	1024 MB	1024 MB
Storage	32 GB internal, 32 GB SD card	16 GB internal	16 GB internal, 16 GB SD card

Table 12: Development tools configuration

Xamarin
Microsoft Visual Studio Enterprise 2015, Update 3 Xamarin for Visual Studio 4.2, Xamarin.Android 7.0, Xamarin.iOS 10.0 Tools for Universal Windows Apps 14.0
Apache Cordova
Microsoft Visual Studio Enterprise 2015, Update 3 Tools for Apache Cordova Update 10 Tools for Universal Windows Apps 14.0 Tools for Universal Windows Apps 14.0
React Native
JetBrains WebStorm 2016.3 React Native v 0.41

4.4 Results

This subsection summarizes the results of evaluated apps. Apache Cordova was implemented via Visual Studio plugin and tested on Android, iOS and Windows. The main development machine was Windows. No special architectural style or design patterns were used.

On the other hand, React Native was developed almost entirely on macOS in WebStorm. Although there exists a custom UWP plugin for React Native, running an app on Windows device was not achieved. Therefor, React Native was evaluated only on Android and iOS. For more complicated projects, the Redux architecture was used.

Similar to Apache Cordova, Xamarin was also primarily developed on Windows desktop and tested on Android, iOS and Windows mobile. To get closer to the UI development experience of the other evaluated frameworks, Xamarin.Forms in combination with MVVM was chosen.

4.4.1 Setup

Considering Apache Cordova, the initial setup for Android and Windows is rather easy since it is a plugin within Visual Studio. The built-in installation process automatically downloads and installs Node.js with Cordova CLI and Git CLI. It also prepares Apache Ant, Oracle Java JDK and Android SDK to enable Android development. Tools for Windows mobile development are not installed automatically. However, they can be added by ticking the checkboxes for Universal Windows Platform tools for Cordova and Windows 10 Mobile Emulator.

Since iOS apps can be built only on a Mac, a separate setup is needed. Visual Studio provides the possibility to build the application remotely, so only few programs need to be installed - namely Node.js, Xcode and Xcode command-line tools. After enabling remote agent on Mac, Visual Studio can connect to it and build the iOS app remotely.

Regarding React Native, the attempt to run a Windows app with the React Native Windows plugin failed. With no reason to develop on a Windows machine, the development process was moved to a Mac. Similar to Apache Cordova, React Native required Node.js as well. In addition to the required React Native CLI, the installation of Watchman was recommended. For Android, Java Development Kit had to be installed. Next, a custom installation of Android Studio was performed, adding Android SDK, Android SDK Platform, Intel HAXM and Android Virtual Device. Since React Native does not support Android Nougat[3], Android Marshmallow SDK had to be downloaded as well (with several required sub-features). At last, the Android setup was finished by specifying the ANDROID_HOME environment variable.

Compared to Android, iOS installation was incomparably easier. Besides the common tools for React Native development, only Xcode was required. In addition to that, WebStorm was installed as well, because of its superior IDE capabilities for JavaScript development.

Xamarin brings us back to Windows. Although Xamarin development is possible on Mac as well (with Xamarin Studio or Visual Studio for Mac), building a Windows app is possible only on Windows. Installing Xamarin is, again, easy via a Visual Studio plugin. By default, the plugin installs Xamarin, Android NDK, Android SDK, Java JDK, Google Android Emulator and Intel HAXM. In addition to that the user may choose to install Universal Windows Platform tools for Xamarin and Windows 10 Mobile Emulator.

Again, installation on Mac has to be done separately. ???

4.4.2 Hello World

There were no issues with the first, very simplistic app. All tested frameworks passed the test almost flawlessly. The only problem was with jQuery for Windows in Apache Cordova project. For some reason, jQuery was added twice, thus the application did not work. Therefore, pure JavaScript was used for all Cordova apps.

[3] As of early 2017

4.4.3 Persistent storage

For persistent storage, SQLite database was chosen for Xamarin and Apache Cordova, while React Native app was integrated with Realm.

Apache Cordova has multiple possibilities of storing data, such as file system storage or the web-browser-based Local storage and Indexed DB. However, only SQLite plugin provides true database features and is consistent across all tested platforms. The plugin worked properly when storing strings, floats and datetimes as well as automatically incrementing the primary key. However, since JavaScript cannot work with binary data, they had to be converted to base-64, even though the SQLite database was able to store it like a BLOB. Another issue on Windows was with toolset version 141, which had do be manually downgraded to 140. After that everything went smoothly.

Similarly to Cordova, also React Native had to work with base-64 binary objects. Moreover, the Realm implementation for React Native is not yet able to automatically generate IDs [4], thus random GUID was used the primary key. Although Realm provides greater abstraction over SQLite (which forces programmers to use pure SQL), it does have considerably less functionality. Yet, for small databases that is more than sufficient.

The last tested framework, Xamarin, implemented SQLite as well. The SQLite library for .NET provides the same amount of abstraction and Realm does for React Native, yet provides the full potential of SQL statements and functions just as pure SQLite for Apache Cordova. Its implementation is superior to both alternatives. Moreover, C# has a larger variety of supported types, thus being able to manipulate with binary objects directly.

4.4.4 Camera and location

For a long time, Apache Cordova was the most popular multi-platform development tool, and it still belongs into top 3. It is therefore no surprise that it has a vast amount of 3rd party plugins and extensions, which help other developers achieve almost any desired functionality. For both camera and geolocation, Cordova offers official plugins, which worked flawlessly on all three platforms.

React Native offers geolocation functionality by default and it works perfectly both on Android and iOS. However, there is no official support for camera usage. Either a custom binding into native code (React bridge) has to be created, or a 3rd party library can be used. For our evaluation, custom library [36] was chosen, but unsuccessfully. The camera plugin did not work on neither of the tested platforms.

Xamarin for Android and Xamarin for iOS allow programmers to access the camera and geolocation just the way they would do it in a native Java or ObjectiveC application. However, Xamarin.Forms is an abstraction over platform-specific code and does not provide camera or geolocation functionality out of the box. Both of them can achieved either by custom imple-

[4] as of early 2017

mentation (injecting the platform-specific code into a common interface) or by using a 3rd party plugin. There are multiple GitHub repositories and NuGet packages offering extensibility to Xamarin.Forms, one of which was used also in this thesis [35]. On all three tested platforms both Media and Geolocator plugin worked without any issue.

4.4.5 Custom UI element and native code

For all frameworks, a circular progress bar was chosen as a custom UI element. Since hybrid apps are just web apps wrapped in a native web view, creating a custom UI element is done via HTML, CSS and JavaScript, just like in any regular web page. Creating a custom circular progress bar in Apache Cordova took almost no time at all. The more challenging part was creating a bridge between the native code and JavaScript, since this required creating a whole new Apache Cordova plugin - a process requiring several steps for each platform.

Also in React Native the custom element was done almost immediately, since a 3rd party plugin exists. Unlike Apache Cordova, also the bridge between React code and native code was done rather easily. Since the build process of React Native includes a generated native infrastructure, it is possible to extend the infrastructure with custom code. A simple React annotation than exposes the native classes and methods inside React Native.

The most complicated circular progress bar implementation was performed in Xamarin. Since the circular progress bar does not exist on any platform, it has to be implemented on each of them. Afterwards, a common element declaration is created in XAML in Xamarin.Forms. The last step is to tell Xamarin, how to render the common XAML declaration on individual frameworks - this is done via Custom Renderers. A Custom Renderer binds the common XAML element and its properties to the native element, its properties and rendering functionality.

Also, the bridge to native code can be done in two ways. The first one is very easy - dependency injection. Since C# code in Xamarin is compiled to native code, platform specific classes inheriting from a common interface can contain native methods. In the common library, the interface is then used to instantiate the platform-specific code and invoke a particular method. The second approach can use real native JAR, AAR, DLL or Objective-C library. These are then inserted into the solution via a Binding Project, which enables direct method calls into the native library.

4.4.6 PUSH notifications

To avoid the tedious task of creating a new PUSH notification service for each platform-specific service provider (Apple Push Notifications Service, Google Firebase, Windows Notification Service), a multi-platform abstraction available via Microsoft Azure was leveraged. Its Notification Hub automatically implements all necessary infrastructure in the individual notification services and reliably delivers PUSH notifications on each mobile platform.

Considering the development in the mobile frameworks themselves... [TBD].

4.4.7 Tablet optimization

To demonstrate that the UI optimization for various screen sizes can be achieved by common techniques used in responsive web design, a simple demonstration web page from W3Schools was used [5].

[TBD]

4.4.8 Summary

When summed together, it can be said that Apache Cordova is a very mature framework with great support and lots of 3rd party libraries and various guides and tutorials. It is very easy to learn and use it, especially for developers familiar with modern web technologies. Thanks to its rich set of plugins, almost any desired functionality is achievable. And the use of HTML, CSS and JavaScript the UI design and development is also straightforward. However, Apache Cordova is suitable only for small apps, which are not performance-demanding. Because the app runs in a browser, parts of native functionality is simply unavailable. The resulting applications are often bloated and rather slow. Last, but not least, the apps do not look native. This can be solved by using Ionic, which produces native-looking apps from Angular. However, it does not solve the other issues.

Considering React Native, it takes a little time to learn the way, how it works and how to implement Redux, the most commonly used React architecture. Yet, the same principles apply also for web development in ReactJS, so developers can reuse this knowledge. After getting past this initial hurdles, the development in React Native is extremely fast. Npm is growing everyday with new React Native packages. However, React Native is still in development and many of its features are not yet production-ready, e.g.: Windows development, or camera usage. Its speed performance and the resulting app size are also not optimal. Yet, it can be expected that once React Native is finished, all this issues will be solved, or at least mitigated. For fast prototyping and small to medium-sized apps, React Native might be the right framework in the near future. Its closest competitor, NativeScript, uses Angular and provides slightly better performance, yet at the cost of smaller popularity (thus smaller community and less 3rd party libraries). It might be possible that React Native and other JavaScript interpreted frameworks will once render hybrid frameworks obsolete.

The last tested development tool, Xamarin, is the most difficult to learn, but is the most powerful at the same time. Unlike Apache Cordova or React Native, Xamarin provides only partial abstraction (in case of Xamarin.Forms) or almost no abstraction (in case of Xamarin.Android and Xamarin.iOS) over the native code. A lot of functionality that is quickly achievable in the other frameworks require more coding in Xamarin. Therefore, Xamarin is not suitable for prototyping and small projects. However, it provides the greatest control over native code, enables the usage of all native APIs and allows the usage of common design patterns and best

[5] Can be found at: https://www.w3schools.com/w3css/tryit.asp?filename=tryw3css_templates_blog&stacked=h

practices. A vast amount of NuGet packages and 3rd party libraries and plugins can quickly enrich Xamarin with new functionality. Xamarin also produces apps with the smallest overhead and the best performance (there are benchmarks in which Xamarin surpassed even native Java on Android). Due to this factors, Xamarin is best suited for medium to large-scale applications, or applications with demanding and complicated functionality.

5 Methodology

There exist multiple texts comparing a couple of multi-platform development tools [12] [11] [45]. Some of them even provide guide or explanation on how the tools were chosen and how to pick the most suitable one for particular projects [8] [10]. However, there is no comprehensive methodology, generic enough to cover a wide variety of multi-platform frameworks. From the theoretical and practical information we have gathered in previous chapters, it is possible to construct a series of methodological steps that will achieve exactly this goal.

5.1 Necessary preconditions

If the methodology is supposed to be effective in the process of selecting the right tool, several preconditions need to be satisfied. The most important factor is specification. Of course, it is impossible to determine every single part of the resulting application right from the start. However, the more precise the specification, the more accurate results can be expected.

The architect, business analyst or product owner should be aware also of the long-term vision for the next iterations. It is very unlikely that a prototype will be done in one tool, which will have to be switched for another, because of changed requirements. Most customers have neither enough money nor time.

Another important factor is to collect as much data about the available resources as possible. Enough information about the size, expertise and maturity of the development team can reduce both time and costs dramatically. Also, mapping already available software and hardware can help to keep the budget.

5.2 Methodologically unsuitable projects

Before proceeding with the methodology itself, several eliminating questions should be answered. If the answer is "yes" to one or multiple questions, given project cannot be correctly assessed by this methodology.

- **Is only one operating system (even with multiple screen sizes) targeted?** If the answer is "yes", then there is no reason to use multi-platform framework. There is nothing that beats native development. The world of multi-platform tools is a world of compromises. Native development allows to use the operating system to its fullest potential, while not limiting the performance. If Java or ObjectiveC are concerns, there exist alternatives also for single-platform development, such as Swift or Kotlin.
- **Do you want to target operating systems other than Android, BlackBerry, iOS and Windows?** This methodology is aimed strictly on mobile OSs. Although some tools allow development for desktop computers as well, this possibility is not discussed

in the methodology. From those featured in the methodology, Qt, Embarcadero, Kivy, RubyMotion or Xamarin target also one or multiple desktop platforms.

The key operating systems for this methodology are Android and iOS. Attention is paid also to Windows support (namely Windows 10 UWP and Windows 8.1 WinRT). Tools enabling development for BlackBerry are examined for their full support. However, should the reader want to focus on different mobile operating systems (like Symbian, Bada or others), this methodology and the suggested results may not be suitable for that use case.

- **Will the app use extensively multiple platform-specific APIs, sensors or widgets?** Many platform-specific features (like Android widgets or Windows life tiles) are not present on other operating systems, thus several multi-platform development tools ignore them altogether. Since the focus of multi-platform tools is to run a single common code on multiple platforms, they often use the road of least common denominator and strip away features, which have no alternatives on other platforms. Some tools provide platform overrides, or allow to have platform-specific application and UI layers. Yet, if the design and functionality for each OS is significantly different[6], a multi-platform tool is useless.

  Tools with active community may have various 3rd party resources, but these may complicate the licencing, increase the cost, decrease performance and violate the security. Adding a large quantity of libraries or plugins will increase the size of the application dramatically.

- **Do you want to create a game?** Although some multi-platform development tools studied by this methodology allow game development to certain extent, for a fully-fledged 2D or 3D game it may not be appropriate. Discussing the differences between individual game framework is beyond the scope of this work. Multi-platform frameworks allowing game development include (but are not limited to) Unity 3D, Unreal Development Kit, Marmelade, MoSync or Wave. However, should the reader plan creating multiple apps and wish to utilize the invested learning effort, he/she is advised to use the methodology and focus on tools supporting game development.

- **Do you want to create a new web app for both desktop and mobile, or port an existing desktop web page to mobile environment?** If the project is a web page or web application that should be available both for mobile and desktop environment and there is no intention of using native APIs or sensors, mobile web app frameworks should be considered. They produce a common web page, adjusted for the mobile devices. For starters, Bootstrap Mobile, AppPress, Sencha Touch, Dojo or jQuery Mobile should be examined.

[6] An example of this may be something as simple as a slider. While on iOS and Windows the native slider allows arbitrary steps, the Android SeekBar is bound to 1000 steps. As a result, values not divisible by 1000 broke the application, since the slider could not find the right position. It took several days to determine the cause of the issue and write a reliable platform-agnostic solution

- **Do you want to create a business-focused enterprise application with connection to an existing cloud backend?** This thesis focuses on frameworks, which allow general-purpose app creation. However, there is a plethora of tools and services which can create a single-purpose app (e.g. for monitoring of employees, to provide ERP and CRM services). Examples of such tools can be Appticles, i-exceed, Pega Systems or Retriever Communications.

  Other group of single-purpose multi-platform tools focus on providing MBaaS services. They integrate into existing cloud systems (like AWS, Azure, Oracle, IBM, SAP or Salesforce) and enable data access and management from a mobile app. Providers of these services include nypresence, AppGyver, IBM Mobile First, Kinvey, MobileFrame, MooFWD. There is plenty of others, not covered in this thesis. However, MBaaS providers who allow creating offline apps and multi-purpose apps have been added to the main methodology.

5.3 Recommended way of using the methodology

If the project has passed all questions from chapter 5.2, it means it is suitable to be subjected to the methodology. The whole methodology consists of 4 sets of questions. Each set has decreasing importance. The first set of questions is the most important. Tools not fulfilling requirements posed in this set violate the project's specification radically. Such tools should be removed from further evaluation.

It is possible that for certain projects the most suitable multi-platform development tool will be clear already after the first set of questions. However, usually the first set of question selects a small group of frameworks with similar capabilities. To distinguish which of them is the most appropriate for a certain project, the second set of questions should be taken. This set contains conditions which are fairly common in mobile app development. Failing one or two requirements in the second set is not as critical as in the first set. Usually, there exists a work-around, but it may increase the project cost, development time or negatively impact app's performance.

Projects having unusual or highly specific requirements should be subjected also to the third set of questions. This set contains conditions which are rare, or there exists an easy workaround. It also focuses on features that may help to decrease the development time or increase performance, user experience and stability of an application. Either way, after the second or third set of questions, the majority of project will get a clear recommendation on which framework to use.

If there is no clear winner even after the third set of questions, then there is the last set of questions. These supplementary questions have no clear answer. They rather point out further criteria which can be researched to discover the most suitable development tool.

If not explicitly said otherwise, each question in the first three sets can be answered in three different ways: yes, nice to have and no. Yes means the feature or condition mentioned in the question is necessary in the application and cannot be skipped. Nice to have describes a feature,

which is not crucial for the application or has not been included in the requirements, but could improve the performance or UX of the project. No means the condition is irrelevant to the project.

Similarly, each feature or condition can be fulfilled by the framework in one of three ways: yes, partially and no. Yes means the feature is fully supported by the development tool. Partially means the feature can be implemented, but it comes with a hiccup. The option "partially" usually comes with a note further describing the limitation. No means the feature is completely absent in the framework. Combination of possible answers and requirement implementations results in an evaluation matrix:

Table 13: Evaluation matrix

The user wants it: / The framework has it:	**Yes**	**Partially**	**No**
Yes	+7 / +5 / +3 points	0 / 0 / 0 points	Excluded from evaluation / -5 / -3 points
Nice to have	+7 / +5 / +3 points	0 / 0 / 0 points	-7 / 0 / 0 points
No	0 / 0 / 0 points	0 / 0 / 0 points	0 / 0 / 0 points

Particular way of assigning points is described at the beginning of each set of questions. Questions, which have a different way of assigning points, describe the process in place. A short example of assigning points can be found in the following subchapter.

5.3.1 Example score evaluation

For the purpose of brevity, we will compare only four tools - Codename One, Ionic, NeoMAD and ViziApps. Also the specification will be very simple. The application should run on Android, iOS and Windows. There should be a separate layout for tablets and phones. The app will work via WiFi, but occasional Bluetooth capability is a nice-to-have feature.

First, we will prepare a table. The columns will be filled with the names of the development tools. The rows will be used for adding points.

Table 14: Sample evaluation - step 1

	Codename One	**Ionic**	**NeoMAD**	**ViziApps**
Total				

Then we continue through the methodology and examine all necessary questions. We will write the answers in the form of points each tool scored.

Table 15: Sample evaluation - step 2

	Codename One	**Ionic**	**NeoMAD**	**ViziApps**
Can create Android app?	+7	+7	+7	+7
Can create BlackBerry app?	0	0	0	0
Can create iOS app?	+7	+7	+7	+7
Can create Windows app?	+7	+7	+7	Excluded
Total				

The first requirement for our sample project is to target Android, iOS and Windows. Since this question belongs to the first set, we operate with the step of 7 points. Let us apply the evaluation matrix from table 13. BlackBerry is not relevant to our project, so we choose the "No" row. As we can see, all frameworks gain 0 points for BlackBerry implementation - regardless of whether they support it or not. However, for Android, iOS and Windows we apply the "Yes" row. All frameworks, which support a platform gain +7 points per platform. There is no framework that would partially support a platform, so none gets 0 points for Android, iOS or Windows. However, ViziApps does not support development for Windows platform. Failing any "Yes" question in the first set results in immediate exclusion of a development tool from further evaluation. Thus we continue only with 3 frameworks in the next step.

UI responsiveness to various screen sizes is a question from the second set. Once again, we look at the "Yes" row of the evaluation matrix. Questions from the second set are evaluated by 5 points. Immediately, we can see that Codename One does not support various screen layouts. For the second set of questions, the punishment for failing a condition are not that harsh. We do not remove Codename One from further evaluation, but we subtract 5 points from its score. Ionic and NeoMAD, on the other hand, both provide functional layout adjustments, so they both gain 5 points. We proceed to the last, voluntary feature.

Implementing Bluetooth is not a necessary condition, thus we use the "Nice to have" row from the evaluation matrix. Bluetooth capability belongs to the third set of questions, rated with 3 points. Neither Codename One, nor NeoMAD provide an out-of-the box support for Bluetooth. A 3rd party library, or custom implementation has to be used instead. Thus, they

Table 16: Sample evaluation - step 3

	Codename One	**Ionic**	**NeoMAD**	**ViziApps**
Can create Android app?	+7	+7	+7	+7
Can create BlackBerry app?	0	0	0	0
Can create iOS app?	+7	+7	+7	+7
Can create Windows app?	+7	+7	+7	Excluded
Is it possible to adjust the layout for phone and tablet?	-5	+5	+5	-
Total				

gain 0 points. Ionic, which comes with Bluetooth support directly, gains 3 points. When all questions have been answered, we can continue to the final evaluation

After counting all the points, we can see that Ionic is the most suitable framework for our particular project, with NeoMAD being close in second place. Codename One is far behind and ViziApps did not even make it through the first round. However, for other projects the results can be totally different. If the reader was not still decided whether to use Ionic or NeoMAD, he/she can research further facts, such as how good is the documentation, how buggy is the tool or what other apps have been created using the particular framework.

5.4 Primary questions

This is the first and most important set of questions. Majority of the questions are crucial to all applications. Questions that are irrelevant to your project should be skipped. All tools that do not satisfy at least one condition in this first category should be removed. There might be some tools that have partial or planned support for some of the conditions. It depends on your particular use case whether this limitation can be accepted or not.

1. **Does the tool support all required mobile operating systems? (Evaluation matrix, 7 points for each required and supported platform)** The obvious question is for which operating systems will the application be developed. All tools mentioned in

Table 17: Sample evaluation - step 4

	Codename One	**Ionic**	**NeoMAD**	**ViziApps**
Can create Android app?	+7	+7	+7	+7
Can create BlackBerry app?	0	0	0	0
Can create iOS app?	+7	+7	+7	+7
Can create Windows app?	+7	+7	+7	Excluded
Is it possible to adjust the layout for phone and tablet?	-5	+5	+5	-
Is Bluetooth supported?	0	+3	0	-
Total				

this thesis (and almost all multi-platform development tools in general) allow development for Android and iOS. Development for Windows 10 is also supported by vast majority of tools, or planned, at least. However, development for BlackBerry is more difficult, since only 4 tools out of the 23 featured in this thesis fully support development for this OS.

2. **Does the tool support development on desktop operating systems you own? (Evaluation matrix, 7 points for each required and supported platform)** Most tools can be installed both on Mac and Windows desktops, while a bit smaller portion supports Linux as well. Others provide a web-based IDE, which is platform independent (but available only online). There are a few exceptions that have very strict limitations, e.g. RubyMotion can be installed and developed only on a Mac.

   (a) **Does the tool allow iOS builds on Windows/Linux? (Evaluation matrix, 7 points)** If you own at least one Mac or do not wish to target iOS, you can skip this question. While Android applications can be built on any desktop OS, projects for iPhones and iPads require compilation on a Mac device. Some tools (like Aquro, Codename One, Smartface and ViziApps) remove this obstacle by providing cloud-

Table 18: Sample evaluation - step 5

	Codename One	**Ionic**	**NeoMAD**	**ViziApps**
Can create Android app?	+7	+7	+7	+7
Can create BlackBerry app?	0	0	0	0
Can create iOS app?	+7	+7	+7	+7
Can create Windows app?	+7	+7	+7	Excluded
Is it possible to adjust the layout for phone and tablet?	-5	+5	+5	-
Is Bluetooth supported?	0	+3	0	-
Total	19	29	26	-

based builds. You do not need to buy a Mac, but you must accept that your project is being compiled somewhere in the cloud.

(b) **Does the tool allow Windows builds on Mac/Linux? (Evaluation matrix, 7 points)** If you own at least one Windows PC or do not wish to target Windows mobile platform, you can skip this question. Similar to iOS, also Windows mobile requires builds on a Windows desktop machine. Unlike macOS, however, you can install Windows on a virtual machine, side by side with your Linux or Mac, so this problem is solved easier. Yet, you still need at least one Windows license. If you want to avoid buying a Windows PC, or software license, Appery.io, Codename One, Tabris and Telerik Platform offer cloud builds for Windows mobile.

3. **How much are you willing to pay? What licensing do you need? What is the size of your team? (compare with the offered prices, do not assign any points, but remove tools which fail your requirements)** Most licences and subscriptions are based on the size of the development team. Individuals or small teams can get licences for free, or a very small cost. However, teams around 25 people and above usually require

an enterprise-grade licence. These high-end licensing options are usually in thousands of dollars per year per developer.

Some companies charge their tools monthly, others annually. In some you pay for the tool itself. Others charge you for IDE, MBaaS features or support and updates. Always examine the possibility to use discounts if you have some sort of subscription. And if the costs are still too high, look for a free and open source tool - there is plenty of them.

4. **Do you want the UI to look the same on all operating systems, or use the native look and feel? Does the tool support it? (do not assign any points, but remove tools which fail your requirements)** Even with a single shared UI code, the application may still look native on individual platforms. Many tools offer the possibility of transforming the platform-agnostic UI code into platform-specific UI elements. Others take a different approach and give the possibility for the app to look exactly the same on Android, iOS and Windows. Both approaches have their pros and cons and it depends on each project what is most suitable for them. Kivy, Kony, Qt and most of the hybrid frameworks give the opportunity to have the same look and feel across all operating systems. Appcelerator, Embarcadero, Ionic and Xamarin are examples of tools that take the native UX approach.

   (a) **If you chose native look, do you want to code a single UI layer for all operating systems, or a custom layer for each mobile OS? Does the tool support it? (do not assign any points, but remove tools which fail your requirements)** While some projects need to do rapid prototyping, others may want to facilitate the opportunities brought by each platform. If your project needs to implement different UI for each OS (e.g. to use the Android fragments, which are not available on iOS), you might want to look at tools like Appcelerator, NativeScript, React Native, RubyMotion or Xamarin. However, all of the platforms studied by this thesis provide also the possibility to write a single UI layer for all target platforms. This allows almost 100% code sharing, but reduces the available UI elements to the lowest common denominator.

5. **Does your project require multithreading and does the development tool support it? (Evaluation matrix, 7 points)** Downloading, uploading, calling remote web services, image and sound processing or other computation-heavy calculations are all examples of tasks, which take some time. During this time the app should still be responsive and, perhaps, be able to schedule also further tasks. This requires using multiple threads - one serving the UI responsiveness and the others doing the tasks in background. If the project contains a large portion of functionality involving multitasking, you should avoid all JavaScript-based tools (both hybrid and interpreted). JavaScript does not support threading on a language level, and while there are workarounds using web workers,

their implementation is inferior, compared to traditional threads. Using C++, C#, Java, Python or Ruby should handle the situation just fine.

6. **Do you want to access the media, use the camera and microphone, play audio and video? Does the tool provide the API? (Evaluation matrix, 7 points for each required and supported feature)** Capturing photos and videos is one of the most common hardware interfaces used in mobile applications. While all studied development tools allow taking photos, not all of them are able to capture video or record sounds (Kivy, Kony, NativeScript, React Native). For example, there exists a 3rd party library for Kivy, but it allows recording sounds only for Android. Even the consecutive replay of recorded sound or video is not warranted in all tools - it is completely absent in Instant Developer, and a 3rd party library is needed for NativeScript and React Native.

7. **Do you want to use geolocation, accelerometer or gyroscope? Does the tool provide the API? (Evaluation matrix, 7 points for each required and supported feature)** The global and relative positioning of the mobile device is another functionality often used in an application. Accessing GPS and navigation is offered out-of-the-box in all tools, except of Kivy and Smartface. However, using accelerometer or gyroscope needs a 3rd party library or custom implementation in Kivy, NativeScript, React Native and Smartface.

8. **Do you want to access the device state (battery status, network availability, OS version, etc.)? Does the tool provide necessary APIs? (Evaluation matrix, 7 points)** Although this functionality is not usually required, particular applications may change their behaviour due to low battery power or unstable Internet connection. Yet, some tools do not provide this information (Corona, Kony, ViziApps) or rely on 3rd party libraries or custom implementation (Codename One, Kivy, Qt, React Native).

9. **Do you want to use a background process? Does the tool support them? (Evaluation matrix, 7 points)** Some applications might find it useful to have a permanent background service with no interface. This could be applied for several use cases, like tracking the position, updating stock markets data, notifying user of an emergency situation. You should take into consideration, that iOS has very limited support for background processes, allowing only geolocation updates. All other background functionality is restricted (with the exception for VoIP applications). The only possible workaround is using remote push notifications. Android, BlackBerry and Windows allow creating background processes, but their implementation and requirements may differ. In all cases, you should keep in mind that background processes on any platform are considered low priority. Therefore they are the first victims when the underlying OS needs to kill running apps and get more resources.

Because of these obstacles, only a handful of tools provide a way to create background services (Codename One, Embarcadero, NeoMAD, Qt, Xamarin). Although there exist 3rd party libraries for all tools based on Apache Cordova, their functionality is limited and inconsistent across individual operating systems.

10. **Do you want to use PUSH notifications? Does the tool support them? (Evaluation matrix, 7 points)** Unlike local notifications, which are both sent and received by the application itself, push notifications are one of the few ways how a remote server can alert your application without prior request. They are important on Android and Windows, since there are still limited ways for push-model communication. However, they are absolutely essential on iOS, because it is the only way to replace background processes. It should be noted, that notifications differ a lot on individual platform. On Android, they are the most customizable, allowing various sorts of interaction, customization and binding functionality. It is possible to create interactive notifications on Windows as well. However, iOS are very limited, offer almost no interactivity and - in case the application is in background - push notifications are handled by the operating system itself, rather than the hosting application.

    Push notifications are well established in tools providing MBaaS services, as it is one of their core functionalities. Yet, they are such an essential feature nowadays, almost all multi-platform development tools implement them (with Kivy and Qt supporting only Android and Windows notifications).

11. **Do you need to create custom plugins or invoke native libraries? Does the tool support it? (Evaluation matrix, 7 points)** Even with the richest environment of 3rd party plugins, there can still be a custom library or UI element that is only available as a native C#/Java/ObjectiveC/Swift package. You may also want to target a specific device, which has non-standard API, such as ambient lights. To do so, you need to interact with the native package either through a wrapper, or directly. Most tools provide one way or another to achieve this capability. But code-free tools, like Instant Developer, SmartFace or ViziApps have limited to no support for such behaviour.

12. **Is app monitoring and crash analytics required by the project? Are they supported by the tool? (Evaluation matrix, 7 points)** Even when the app is out, the work is not done. Monitoring usage, performance and application crashes is vital for providing continual support and improvement. While there are some free libraries and tools like ACRA, Crashlytics or Xcode Crash Reports, they may not be compatible with your chosen development tool. You should closely examine, whether integration of the multi-platform development tool and analytics framework is possible. If not, try to find out, whether an alternative is provided by the tool (e.g. Telerik Analytics for NativeScript or HockeyApp for Xamarin).

5.5 Secondary questions

This set of questions contains conditions that are important for some applications, while irrelevant to others. Failing a condition in this category may seriously affect the development process, but a sort of workaround is usually possible.

1. **How would you characterize the complexity of your app? (Add +5 if the framework has desired performance. Add 0, if the performance is higher. Subtract -5 if the performance is lower.)** Complexity of the application can help to determine which tool and programming language to choose. Simple, straight-forward applications can take advantage of the rapid prototyping provided by hybrid mobile applications and tools using scripting languages. Larger and more complex projects should be programmed in a strongly-typed programming language featuring object-oriented programming and separation of code into modules.

2. **Is designer or previewer tool available? (Evaluation matrix, 5 points)** A design or preview tool allows to see the changes in UI code immediately on a simulated mobile device screen. Some allow also WYSIWYG editing. This can highly improve the communication between a designer, product owner and the developer. It can be also very helpful in rapid prototyping and agile development.

3. **Do you want to have different UI for phone and tablet? Does the tool support multiple or responsive layouts? (Evaluation matrix, 5 points)** There are apps that look good on all screen sizes. And then there are apps that need radical layout changes to ensure good user experience both on smartphones and tablets. Most tools offer the possibility to create different layout for smaller and larger screens, or use the CSS responsive design in case of web technologies. Corona, for example, uses conditional compilation to render the right layout on a device. However, there is a number of tools (Aquro, Codename One, Instant Developer, Kivy, Qt, Tabris, ViziApps) that have no support for different UI layouts. The only way, to create a tablet-specific design is to create a separate application.

4. **Does the design include any platform-specific overrides? Does the tool support them? (Evaluation matrix, 5 points)** Even if most parts of the application will be the same on all operating systems, there might be one or two things that will differ in order to maintain the native UX. While almost all tools achieve this either by replacing code files, conditional compilation or dependency injection, there are a few exceptions which do not support platform-specific overrides, namely Instant Developer, Smartface and ViziApps.

5. **Will the application work with the file system? Is the tool able to access and manage it? (Evaluation matrix, 5 points)** All studied development tools provide access to the file system, although there might be some restrictions for Qt and hybrid

apps. It is worth mentioning, that the file system structure differs a lot on Android, iOS and Windows.

6. **Will the application require persistent storage? Can the tool work with app properties or mobile database? (Evaluation matrix, 5 points for each required and supported database)** Excluding file system, Android, iOS and Windows offer 2 different approaches to data persistence. Simple key-value data pairs, such as the current application state or settings are usually saved in the app properties (SharedPreferences/NSUserDefaults/ApplicationDataContainer). More complex data structures are usually stored in a database. The most common database used in mobile world is SQLite, which has almost universal support across the studied tools. Couchbase and Realm are examples of other popular mobile database engines, also supported by several tools. Aquro and ViziApps are the only two multi-platform development tools not allowing any client-side data storage, since they rely on all data being stored in the cloud.

7. **Do you want to use Bluetooth, NFC, fingerprint or barcode scanner? Is their API available in given tool? (Evaluation matrix, 5 points for each required and supported feature)** Unlike the other sensors, Bluetooth, NFC and fingerprint scanners are not software-related, but depend on the hardware outfit. This makes their availability fragmented even within an operating system. Only a small portion of studied tools provide the functionality out-of-the box. The majority relies on 3rd party libraries, or custom wrappers invoking native functionality. This is true also for the barcode or QR scanner - although it is captured by camera, it is processed usually by the native API.

8. **Is integration with assistant service (Alexa, Cortana, Google Assistant, Siri) possible? (Evaluation matrix, 5 points for each required and supported assistant)** Although the usage of AI-powered virtual assistance is still limited to several use cases and language, their possibilities are growing with each new update. Already now, virtual assistants are able to pick the best suited application for a particular request. If you want your application to be popular, it should communicate with the virtual assistant. Not all tools provide necessary support for this behaviour.

9. **Does the tool support debugging? (Evaluation matrix, 5 points)** Debugging is one of the earliest tools that can help developers find and fix possible errors. It allows to closely examine every line of code. For any application that has medium to high complexity, debugging capability is a must.

10. **Does the tool support unit testing? (Evaluation matrix, 5 points)** Unit tests are the first tests done after (or sometimes before) a piece of code is written. While not examining the system as a whole, they test each method or class individually. This helps to increase the clarity and readability of code. Unit tests are then the first indicator whether new or refactored code is functional and satisfies all required standards.

11. **Does the tool support automated UI testing? (Evaluation matrix, 5 points)** While new functionality is generally thoroughly tested manually, repeated tasks, such as smoke tests or regression tests, can be automated. Automated UI tests substantially increase the volume of tests performed for complex business- and enterprise-ready applications. They can be also incorporated into the continuous integration process.

12. **Does the tool support app profiling? (Evaluation matrix, 5 points)** Neither unit tests, nor manual tests are able to discover low-level or long-running issues, like memory leaks or taking up too much resources. If high quality and high performance are important for your project, you should definitely use a tool or IDE that provides app profiling.

5.6 Tertiary questions

Following questions have either low impact on the application development or relate to a very small niche of projects and a workaround is possible. Yet, they still may balance the scales in one way or another. They serve as a finishing touch after the coarse separation in first two categories.

1. **Does the tool allow development in a programming language you already have experience with? (Evaluation matrix, 3 points for each required and supported language)** Even if the developers have no previous experience with a multi-platform development tool, high skills in a particular programming language might help them to learn using it more quickly.

   Currently, the most wide-spread programming language for multi-platform development is JavaScript, or some of its derivatives (Angular, TypeScript). However, there exists at least one or two tools for each major programming language (BASIC, C++, C#, Delphi, Java, Lua, Python, Ruby). There are even a few tools which allow code-free visual app building.

2. **Are MBaaS features or cloud connection necessary? Are they available? (Evaluation matrix, 3 points)** Many project rely on at least some basic cloud connectivity. Interaction with backend server or social networks is exactly what MBaaS providers offer. If the development tool does not feature an out-of-the-box MBaaS support, check the possibility of custom implementation. Especially code-free and JavaScript-based tools are prone to inability to add custom functionality.

3. **Do you need an embedded web browser? Does the tool allow its use? (Evaluation matrix, 3 points)** Using an embedded web browser is one of the less common functionalities. Its use is very peculiar in case of the hybrid apps, since they themselves are ran from within a web browser - therefore, it is a browser inside a browser. Yet, with the exception of Instant Developer and Smartface, all tools support this functionality. Ionic,

Kivy and NeoMAD require custom implementation and PhoneGap does not feature web browser for BlackBerry.

4. **Do you want to access the address book and be able to make a call, or send SMS? Does the tool provide all necessary APIs? (Evaluation matrix, 3 points)** While contemporary smartphones are often more smart than phones, accessing the contacts list, making calls or texting is still used, from time to time. It is therefore a bit of relief that all examined development tools support this functionality, although some may require a 3rd party library.

5. **Is a testing environment for multiple devices required? Does the tool feature any? (Evaluation matrix, 3 points)** Even automated tests on an emulator or real device have one major shortfall - they test only a single screen size. Screen fragmentation is a long lasting and well known issue on Android devices, but with the support for older hardware it is relevant also in the iOS and Windows world. Amazon Web Services provide a service known as Device Farm [3], which allows simultaneous UI testing on hundreds of physical Android and iOS devices in the cloud. Similar services are offered also by BitBar [4], Firebase [19] and TestObject [46]. Some of the multi-platform development tools featured in this thesis provide also a cloud testing service, arguably with better integration with their product line (e.g. Telerik and Xamarin). If your app has the potential of being installed by millions of users, testing it on a Device Farm, or similar service, may be highly beneficial.

6. **Does the tool allow continuous integration, if necessary? (Evaluation matrix, 3 points)** Continuous integration is a crucial part of DevOps, enabling small but valuable and rapid updates of an product or service. The whole process consists of various steps, like from coding, collaboration, versioning, unit and automatic testing to packaging, releasing and configuration management. A tool that can be easily incorporated into the process can result in shorter development cycles and higher customer satisfaction.

7. **Are closed test groups required and supported? (Evaluation matrix, 3 points)** While all major application stores feature the possibility to share a WIP app to a tester group (Apple TestFlight, Google Play Testing, Windows Package Flights), some multi-platform development tools offer incorporated features of app distribution, giving you even more control. ViziApps, for example, offers the possibility of OTA (Over the Air) updates of installed apps.

8. **Is OpenGL or WebGL required? Is it supported? (Evaluation matrix, 3 points)** There are some applications that may find use in hardware-accelerated 3D graphics. A model catalogue, design studio or a game are some examples of this kind of apps. Rendering and managing a 3D object requires the use of OpenGL library specialized for mobile or

web environment. Most tools support the integration of either OpenGL ES or WebGL. However, Aquro, Instant Developer, Smartface and ViziApps are exceptions to this rule.

9. **Does the tool support creating 2D games, if necessary? (Evaluation matrix, 3 points)** Although for a full-featured game development a mobile game framework would be more suitable, some multi-platform development tools are mature enough to host a simple 2D game. Examples of them can be Corona Labs, Kivy, Qt or Xamarin.

10. **Does the tool support augmented or virtual reality, if necessary? (Evaluation matrix, 3 points)** Although augmented and virtual reality applications are dominated by games, various utility and even business apps are getting still more popular as well. Implementing one of these technologies might be exactly the thing that will set your app apart from the competition.

5.7 Supplementary questions

This last set of questions does not provide simple answers. However, they may help in investigating the differences between two frameworks that still have similar amount of points, even after passing the first three sets of questions.

1. **Do the developers have experience with a multi-platform development tool?** If the answer to this question is yes, then you should consider consulting your development team. Their past experience might be very helpful in resolving the questions with no simple answers, like: What is the learning curve? How long does the development take? What is the performance of the app? Is the development tool buggy? How busy is the developer community?

   If the tool used by the developers both satisfies all the customer's requirements and is advised to be used by the development team itself, it can greatly boost the development speed. They will be familiar with all the pitfalls. Components and design patterns from past projects may be reused. You might find out, that the software licences are still valid and you have enough hardware to test on. All these factors may help to decrease the development time and costs and still satisfying your customer.

2. **Is there a source of 3rd party libraries? How rich is it?** Even if a tool does not implement certain features, it can often be substituted by a 3rd party library. Tools with rich environment of 3rd party libraries often come also with tools enabling easy plugin management. Examples of these can be: Hyperloop by Appcelerator, npm for Apache Cordova and NativeScript plugins, Nuget for Xamarin packages or motion-toolbox for RubyMotion gems.

3. **Do you have specialists for individual mobile operating systems?** As mentioned already multiple times in this thesis, each mobile OS has a different way of handling

the app lifecycle. There are differences in communication with the kernel, interactions between apps and handling navigation. There is inconsistency in available APIs, sensors, widgets and varying guidelines to UX and UI design. Some development tools shield you off these differences, focusing on the least common denominator. Others let you leverage the platform-specific features to the full extent.

Having developers specialized for individual operating systems allows you to use the platform-specific approach. While most development tools allow platform-specific overrides, it is default only for RubyMotion and Xamarin (partially also for Appcelerator, React Native and NativeScript). The former approach, where platform-specifics are shielded off (known also as the WORA approach) is used by all other tools, as well as Xamarin.Forms.

4. **What is the size of the resulting apps?** In the early days of Swift, there were many reports that the apps are much larger than apps written in Objective-C[43]. This was due to the fact that Swift was not included in iOS, thus Swift runtine, libraries and API had to be bundled within the app. This is true also of many multi-platform development tools. They often bundle their own runtine, custom libraries, UI styles and APIs together with the application itself, making its installation size bigger. It could be worthwhile to investigate, how large the resulting apps are for individual frameworks.

5. **What is the learning curve? How long does it take to create a simple app? How long does it take to master the tool?** Some programming languages are inherently easier to learn, while others require much greater effort. There are tools which require a lot of initial configuration and settings and then there are frameworks which are fully functional out-of-the-box. Part of the frameworks mentioned in this thesis lie great emphasis on architecture of the code base, yet other allow the developer to patch the code as he/she pleases. Depending on what type of application you wish to achieve, there are tools which are more or less suitable for your project.

6. **How often is the tool updated? How long does it take to react to a new OS release?** As mentioned in the beginning of this thesis, each mobile operating system has already multiple versions. Each version brings new API, increases performance, while may cut off part of redundant functionality from the past. If you app should stay competitive and fresh, it should be prepared for those changes. Try to search how long does it approximately take for a framework to support new OS versions. Some provide 0-day compatibility, while others may require several months to be fully prepared for the new version.

7. **How comprehensive is the documentation? Are tutorials, courses or books available?** To learn something new, each of us needs a source of knowledge. It is almost impossible to learn a new framework without the knowledge how to use it. Thorough

documentation is crucial, yet there are tools which provide specification only for basic use cases. However, good courses or books can be even better, quick-starting your project.

8. **How buggy is the tool? How are bugs handled?** There is no perfect piece of software - bugs are simply everywhere. However, some tools are more stable than others. Very important is also the attitude towards bug solving. Open-source rely on the community support. Some commercial tools are also very open about their bug-solving policy, allowing to track all reported, confirmed and solved bugs. However, there still can be found a few shady examples, which believe that pretending no bug exists will solve the problem.

9. **What are some prominent apps developed using the tool?** If you still are not sure which tool is most suitable for your case, try to look at some apps that were written in that tool. If you find multiple similar apps, it probably means the tool is best suited for your use case.

6 Methodology verification

In order to test the methodology out, it has been subjected to several verifications on real projects.

6.1 FloLogic

The first project is called FloLogic. It consists of a mobile application and cloud backend, monitoring and controlling a water valve. The project was started more than a year ago. The original specification required to use Xamarin, because the product owner had an MSDN subscription. Other requirements included:

1. Focus on iOS and Android, with later implementation for Windows Phones and Windows 10 devices
2. Implementation of a Microsoft Azure cloud service, communicating with the mobile app and the water valve
3. Integration of a highly reliable PUSH notification system

However, during the year several requirements were changed, added or removed. Together with other complications, this resulted in extending a project, originally estimated for 3 months, to more than a year. Changes included 3 different UI designs, starting with platform-agnostic UI to native-like platform-specific and then back to a unified UI.

The project manager of FloLogic wanted to know, whether Xamarin is still a suitable choice, since the development team was struggling with implementation of individual requirements. He provided the current requirements and compared them against individual sets of questions in this methodology. The same questions were then answered by one of FloLogic developers, to provide a slightly different point of view. The results can be found below.

As the reader can see, the answers of the project manager and developer have differed in a few points, namely:

- Desktop operating systems - the developer has stated that even though the company does not currently possess any Linux machines, Ubuntu can be installed on any Windows PC. He mentioned also the possibility to develop in a web environment, which was refused by the project manager due to security considerations.
- Multithreading - according to the developer, multithreading is not really necessary, since the only possible point of its use are calls to the cloud server, which can be handled by asynchronous callbacks. The UI is rather static, so slight lags should not impact the user experience. Moreover, the absence of true multithreading can be compensated with web workers in most JavaScript-based tools, or by Lua's co-routines.

Table 19: FloLogic - Primary questions

Question	**Project manager's answer**	**Developer's answer**
Which mobile operating systems do you want to target?	Android, iOS	Android, iOS
Which desktop operating systems do you have available for development?	macOS, Windows	Linux, macOS, Windows, web environment
What is the size of your team?	Up to 5 developers	Up to 5 developers
How much are you willing to pay for software licences per developer per year?	$0	$0
Do you want the app to have native look and feel, or same across individual platforms?	Same	Same
Do you require multithreading?	Yes	Nice to have
Will you use multimedia sensors and APIs (camera, microphone, video and audio player)	No	No
Will you use location sensors (GPS, gyroscope, accelerometer)?	No	No
Do you want to get information about device status?	No	Yes
Will you use a background process?	No	Yes
Will you use PUSH notifications?	Yes	Yes
Do you need to invoke native libraries or create custom plugins?	No	No
Would you like to use app monitoring and get crash analytics?	Nice to have	Nice to have

- Device status information - the developer expressed the need to check network availability, connection stability, as well as the version of operating system.
- Background process - according to the developer, background processes, although unavailable on iOS, are much more reliable on Android, then PUSH notifications. He would therefore prefer a tool allowing the creation of background services.

Only 4 tools have managed to pass the primary set of questions according to the project manager's answers. The developer was more demanding, leaving space only to 3 tools. Here are the results with following score:

As the reader can see, Xamarin was eliminated from further consideration already in the first set of questions. Both Xamarin and Xamarin.Forms target native look and feel of applications.

Table 20: FloLogic - Primary questions results

Tool:	**Kivy**	**Kony**	**PhoneGap**	**React Native**
Project manager's score:	28	42	40	35
Developer's score:	42	-	56	49

Although it is possible to bypass this with various workarounds (as the FloLogic development team has proven), it takes more than a month, compared to a couple of days done in other tools. Let us continue with the second set of questions, which were answered by both the project manager and the developer:

Table 21: FloLogic - Secondary questions

Question	**Project manager's answer**	**Developer's answer**
How complex is your application?	Business-grade	Business-grade
Would you like to use a designer or preview tool?	Nice to have	No
Do you need different layout for tablet and phone?	No	No
Will you use some platform-specific overrides?	No	No
Will you access the file system?	No	No
Do you need any client-side database?	No	No
Will you use Bluetooth, NFC, finger- or barcode scanner?	No	No
Do you require integration with virtual assistants?	No	No
Do you need debugging tools?	Yes	Yes
Will you use unit tests?	Yes	Nice to have
Would you like to use automated UI tests?	Nice to have	No
Would you like to use app profiling?	Nice to have	Yes

Even in the second set some answers differ. The developer was confident enough to create user interfaces purely in code, without the need of a designer tool. Also, the preferences for individual testing techniques differed from the project manager to the developer. After answering the secondary questions, the score table looks like the following:

Table 22: FloLogic - Secondary questions results

Tool:	**Kivy**	**Kony**	**PhoneGap**	**React Native**
Project manager's score:	40	79	47	62
Developer's score:	42	-	56	59

Already now, the winner is very distinguishable. Kony leads with 79 points, with React Native being far on the second place. The project manager of FloLogic was familiar with React Native, but Kony was new to him. He wanted to take also tertiary questions in order to see, whether Ract Native comes closer to Kony, or whether the gap widens. Here are the answers for the last set of questions:

Table 23: FloLogic - Tertiary questions

Question	**Project manager's answer**
Which programming languages do you have experience with?	AngularJS, C#, JS, Java, Ruby
Do you require MBaaS features or cloud connection?	Yes
Do you need embedded web browser?	No
Would you like to access the contact list, call or send SMS?	No
Is a testing environment for multiple devices required?	Nice to have
Do you need continuous integration features?	Nice to have
Would you like to test in closed groups?	No
Do you need OpenGL support?	No
Would you like to develop games?	No
Is the app targeted for augmented or virtual reality?	No

Table 24: FloLogic - Tertiary questions results

Tool:	**Kivy**	**Kony**	**PhoneGap**	**React Native**
Project manager's score:	34	88	47	65

The manager was surprised with the results and displayed deep interest in the Kony framework. He expressed himself that he would discuss the results with the product owner. However, he added he is rather sceptical about changing the framework after more than a year of development and starting from scratch.

6.2 Project W

The next project belongs to an indie startup which tries to fill the "App gap" in Windows market. They realize that Windows is not particularly strong in the mobile field, but thanks to the concepts of Universal Windows Platform they would like to create an app that runs on smartphones, tablets and desktops as well. Although this methodology does not focus on multi-platform desktop development tools, some frameworks mentioned in this thesis allow it as well.

The goal of Project W is to create an app that helps the user to track his/her daily water intake. Unlike similar apps, this app calculates the required water intake dynamically based on local weather, user statistics and data from other health-tracking apps the user might have installed. The answers for the first set of questions can be found below.

Table 25: Project W - Primary questions

Question	**Answer**
Which mobile operating systems do you want to target?	Android, iOS, Windows
Which desktop operating systems do you have available for development?	Windows (maybe macOS)
Do you require iOS builds without the need of a Mac?	Nice to have
What is the size of your team?	Indie
How much are you willing to pay for software licences per developer per year?	$0
Do you want the app to have native look and feel, or same across individual platforms?	Native look and feel
Do you want to code a single UI layer for all operating systems, or a custom layer for each mobile OS?	Single
Do you require multithreading?	Nice to have
Will you use multimedia sensors and APIs (camera, microphone, video and audio player)	No
Will you use location sensors (GPS, gyroscope, accelerometer)?	GPS
Do you want to get information about device status?	Yes
Will you use a background process?	No
Will you use PUSH notifications?	Yes
Do you need to invoke native libraries or create custom plugins?	No
Would you like to use app monitoring and get crash analytics?	Nice to have

The startup has quite limited resources. Although they would like to create apps for iOS, they currently do not own a Mac. They had planned to buy one, but as soon as they heard some tools allow iOS builds without a Mac, they showed deep interest for this possibility. Even with this limitation, 6 frameworks have passed the primary questions.

Table 26: Project W - Primary questions results

Tool:	**Embarcadero**	**Ionic**	**Monaca**	**NativeScript**	**Tabris.js**	**Xamarin**
Score:	56	42	42	42	56	49

We can see that all selected frameworks have fairly equal score, with Embarcadero and Tabris.js slightly standing out. It is interesting though that the filtered tools contain all types of approaches - hybrid, interpreted and even cross-platform. In this case, passing another set of questions is absolutely necessary.

Table 27: Project W - Secondary questions

Question	**Answer**
How complex is your application?	Native-like
Would you like to use a designer or preview tool?	Nice to have
Do you need different layout for tablet and phone?	Yes
Will you use some platform-specific overrides?	No
Will you access the file system?	No
Do you need any client-side database?	SQLite or Realm
Will you use Bluetooth, NFC, finger- or barcode scanner?	Barcode scanner
Do you require integration with virtual assistants?	No
Do you need debugging tools?	Yes
Will you use unit tests?	Nice to have
Would you like to use automated UI tests?	Nice to have
Would you like to use app profiling?	No

Although the Project W application will be rather simple considering the UI, it will have a lot of backend functionality, synchronization with other apps and API calls - thus it was categorized as a native-like app. The developers also wanted to use a built-in barcode scanner, so the app could scan bottled waters, juices, etc. With these additional requirements, the score table has changed considerably.

Embarcadero is still at the forefront, but Xamarin surpassed it together with Tabris.js, which is now in the 4th place. Both Monaca and NativeScript seem to be insufficient for this app. The

Table 28: Project W - Secondary questions results

Tool:	Embarcadero	Ionic	Monaca	NativeScript	Fabris.js	Xamarin
Score:	86	72	57	52	66	94

developers were curious, whether the last set of questions will have similarly dramatic effect on the score of individual frameworks.

Table 29: Project W - Tertiary questions

Question	Answer
Which programming languages do you have experience with?	C#, JavaScript
Do you require MBaaS features or cloud connection?	Yes
Do you need embedded web browser?	No
Would you like to access the contact list, call or send SMS?	No
Is a testing environment for multiple devices required?	No
Do you need continuous integration features?	Nice to have
Would you like to test in closed groups?	Yes
Do you need OpenGL support?	No
Would you like to develop games?	No
Is the app targeted for augmented or virtual reality?	No

Table 30: Project W - Tertiary questions results

Tool:	Embarcadero	Ionic	Monaca	NativeScript	Fabris.js	Xamarin
Score:	80	72	60	52	63	100

As we can see, Embarcadero has lost 6 points, while Xamarin has only strengthened its position at exactly 100 points. It is clear now that Xamarin is the most suitable framework for Project W. The developers have admitted they heard of Xamarin a lot and considered it even before passing through this methodology. With Xamarin, they can develop not only for mobile, but also Windows desktop (with Universal Windows Platform) and Mac (with Xamarin.Mac). The only platform Xamarin does not target is Linux.

6.3 Sensus

7 Conclusion

References

[1] WARGO, John M. *Apache Cordova 3 Programming*. First release. New Jersey 07458: Pearson Education, 2013. ISBN 978-0-321-95736-8.

[2] RAJ, R a SB TOLETY. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. *2012 Annual IEEE India Conference (INDICON)*. 2012, 625–9.

[3] *AWS Device Farm* [online]. [cit. 2017-03-11]. Available at: https://aws.amazon.com/device-farm/

[4] *Bitbar*. [online]. [cit. 2017-03-11]. Available at: http://bitbar.com/testing/

[5] BlackBerry World. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-10]. Available at: https://en.wikipedia.org/wiki/BlackBerry_World#Milestones

[6] Bring your Windows Phone Silverlight apps to Windows Runtime XAML; prepare for universal app development in Windows 10. *Windows blogs* [online]. [cit. 2017-03-10]. Available at: https://blogs.windows.com/buildingapps/2014/12/17/bring-your-windows-phone-silverlight-apps-to-windows-runtime-xaml-prepare-for-universal-app-development-in-windows-10/

[7] COMPARING PHONEGAP/CORDOVA AND CODENAME ONE. *Codename One* [online]. [cit. 2017-02-26]. Available at: https://www.codenameone.com/blog/comparing-phonegap-cordova-and-codename-one.html

[8] PALMIERI, Manuel, Inderjeet SINGH a Antonio CINCCHETTI. *Comparison of Cross-Platform Mobile Development Tools* [online]. [cit. 2017-03-18]. Available at: https://pdfs.semanticscholar.org/be08/83eab3d3d6eb10c4ff1189163f6453254da1.pdf

[9] Cross-platform. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-10]. Available at: https://en.wikipedia.org/wiki/Cross-platform

[10] DE ANDRADE, Paulo R. M. a Adriano B. ALBUQUERQUE. *Cross Platform App: A comparative study* [online]. Postgraduate program in applied information University of Fortaleza - UNIFOR Fortaleza - CE, Brazil, 2015 [cit. 2017-02-19].

[11] FURUSKOG, Martin a Stuart WEMYSS. *Cross-platform development of smartphone applications: An evaluation of React Native* [online]. Uppsala Universitet, 2016 [cit. 2017-02-19]. In: https://uu.diva-portal.org/smash/get/diva2:948617/FULLTEXT01.pdf

[12] NIELSEN, Bobby. *Cross Platform Mobile Development* [online]. Department of Computer Science, University of Aarhus, 2015 [cit. 2017-02-19]. In: https://pdfs.semanticscholar.org/de11/7b518d31ebcad7a864f9b16d1bcd53365d6a.pdf

[13] CROSS PLATFORM MOBILE STILL BETTER THAN NAIVE IN AGE OF FLAT DESIGN. *Codename One* [online]. [cit. 2017-03-10]. Available at: https://www.codenameone.com/blog/cross-platform-mobile-still-better-than-native-in-age-of-flat-design.html

[14] *Cross-Platform Tool Benchmarking 2014* [online]. [cit. 2017-02-19]. In: http://www.research2guidance.com/r2g/Cross-Platform-Tool-Benchmarking-Report-2014.pdf

[15] Defining a New Breed of Cross-Platform Mobile Apps. *Telerik* [online]. 2015 [cit. 2017-02-19]. In: http://developer.telerik.com/featured/defining-a-new-breed-of-cross-platform-mobile-apps/

[16] Detachable Tablets Set To Grow From 8% of the Tablet Market in 2015 to 30% in 2020, According to IDC. *IDC* [online]. [cit. 2017-03-10]. Available at: http://www.idc.com/getdoc.jsp?containerId=prUS41072516

[17] Developers earn more on Windows Phone than Android or iOS. *Betanews* [online]. [cit. 2017-03-10]. Available at: http://betanews.com/2016/02/29/windows-phone-developer-revenue/

[18] FRIBERG, Joy. *Evaluation of cross-platform development for mobile devices* [online]. Department of Computer and Information Science , Linköpings universitet, 2014 [cit. 2017-02-19]. In: https://www.diva-portal.org/smash/get/diva2:691708/FULLTEXT01.pdf

[19] *Firebase Test Lab for Android* [online]. [cit. 2017-03-11]. Available at: https://firebase.google.com/docs/test-lab/

[20] *Gartner* [online]. [cit. 2017-03-11]. Available at: http://www.gartner.com/technology/home.jsp

[21] Get started with Ionic Framework. *Ionic* [online]. [cit. 2017-02-26]. Available at: http://ionicframework.com/getting-started/

[22] How Much Do Mobile Developers Make Per App? *Lifehacker* [online]. [cit. 2017-03-10]. Available at: http://www.lifehacker.com.au/2016/03/how-much-do-mobile-developers-make-per-app/

[23] IOS Programming Tutorial: Creating a Universal App. *Appcoda* [online]. [cit. 2017-03-10]. Available at: http://www.appcoda.com/ios-univeral-app-tutorial/

[24] IOS vs Android Market Share & Revenue: One Win for Each App Store in 2015. *Latin Post* [online]. [cit. 2017-03-10]. Available at: http://www.latinpost.com/articles/110519/20160121/ios-vs-android-market-share-revenue-one-win-for-each-app-store-in-2015.htm

[25] Mobile cellular subscriptions (per 100 people). *The World Bank* [online]. [cit. 2017-03-10]. Available at: http://data.worldbank.org/indicator/IT.CEL.SETS.P2?view=map&year=2000

[26] SMUTNÝ, P. *Mobile development tools and cross-platform solutions.* 13th International Carpathian Control Conference (ICCC). 2012, 653–6.

[27] SHACKLES, Greg. *Mobile development with C#.* Sebastopol, CA: O'Reilly, c2012. ISBN 978-1-449-32023-2.

[28] Mobile operating system. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-11]. Available at: https://en.wikipedia.org/wiki/Mobile_operating_system#By_operating_system

[29] Mobile/Tablet Operating System Market Share. *NetMarketShare* [online]. [cit. 2017-03-10]. Available at: https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1

[30] NativeScript Runtime Preview for Windows 10. *NativeScript* [online]. [cit. 2017-03-10]. Available at: https://www.nativescript.org/blog/nativescript-runtime-preview-for-windows-10

[31] New Report Shows iOS Users Spend Money, Like to Check Weather. *The Mac Observer* [online]. [cit. 2017-03-10]. Available at: https://www.macobserver.com/tmo/article/new_report_shows_ios_users_spend_money_like_to_c

[32] Number of apps available in leading app stores as of June 2016. *Statista* [online]. [cit. 2017-03-10]. Available at: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[33] Number of smartphone users worldwide from 2014 to 2020 (in billions). *Statista* [online]. [cit. 2017-03-10]. Available at: http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[34] One chart shows why BlackBerry 10 has struggled to attract developers. *BGR* [online]. [cit. 2017-03-10]. Available at: http://bgr.com/2013/11/26/blackberry-10-developer-revenues/

[35] Plugins for Xamarin and Windows Projects. *GitHub* [online]. [cit. 2017-03-11]. Available at: https://github.com/jamesmontemagno/Xamarin.Plugins

[36] React-native-camera. *GitHub* [online]. [cit. 2017-03-11]. Available at: https://github.com/lwansbrough/react-native-camera

[37] React-native-windows. *Npm* [online]. [cit. 2017-02-26]. Available at: https://www.npmjs.com/package/react-native-windows

[38] React Native on the Universal Windows Platform. *Windows blogs* [online]. [cit. 2017-03-10]. Available at: https://blogs.windows.com/buildingapps/2016/04/13/react-native-on-the-universal-windows-platform/#cFzRYQ06Y3lx1Yo8.97

[39] Reports. *Developer Economics* [online]. [cit. 2017-03-10]. Available at: https://www.developereconomics.com/reports/

[40] Smartphone OS sales market share. *Kantar Worldpanel* [online]. [cit. 2017-03-11]. Available at: https://www.kantarworldpanel.com/global/smartphone-os-market-share/

[41] Supporting Tablets and Handsets. *Android Developer* [online]. [cit. 2017-03-10]. Available at: https://developer.android.com/guide/practices/tablets-and-handsets.html

[42] DA SILVA, Ribeiro A. *Survey on cross-platforms and languages for mobile apps.* 2012 eighth international conference on the Quality of Information and Communications Technology (QUATIC). **2012**, 255-60.

[43] Swift Embedded Runtime Library Increases App Size. *Doing things the hard way. . . : But only once* [online]. [cit. 2017-03-11]. Available at: http://blog.diogot.com/blog/2014/09/29/swift-embedded-runtime-library-increases-app-size/

[44] Tablet operating systems' market share worldwide from 2013 to 2020. *Statista* [online]. [cit. 2017-03-10]. Available at: https://www.statista.com/statistics/272446/global-market-share-held-by-tablet-operating-systems/

[45] EL-KASSAS, Wafaa S., Bassem A. ABDULLAH, Ahmed H. YOUSEF a Ayman M. WAHBA. *Taxonomy of Cross-Platform Mobile Applications Development Approaches* [online]. Department of Computer and Systems Engineering, Faculty of Engineering, Ain Shams University, Egypt, 2015 [cit. 2017-02-19]. In: http://www.sciencedirect.com/science/article/pii/S2090447915001276

[46] *TestObject* [online]. [cit. 2017-03-11]. Available at: https://testobject.com/

[47] Welcome to Continuum for phone. *Microsoft Support* [online]. [cit. 2017-03-10]. Available at: https://support.microsoft.com/en-us/help/17280/windows-10-mobile-continuum

[48] Windows. *Windows Dev Center* [online]. [cit. 2017-03-10]. Available at: https://developer.microsoft.com/en-us/windows

[49] Windows 10 Mobile. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-10]. Available at: https://en.wikipedia.org/wiki/Windows_10_Mobile

[50] WINDOWS PHONE 8.1 & UWP SUPPORT. *Codename One* [online]. [cit. 2017-03-10]. Available at: https://www.codenameone.com/blog/windows-phone-8-1-uwp-support.html

[51] Worldwide Tablet Market Expected to Rebound in 2018 as Windows Opens Doors for Growth and iPads Come Out of a Slump, According to IDC. *IDC* [online]. [cit. 2017-03-10]. Available at: https://www.idc.com/getdoc.jsp?containerId=prUS41699516

A Overview of studied frameworks

A.1 Alpha Anywhere

Table 31: Alpha Anywhere

Framework	**Alpha Anywhere**	https://www.alphasoftware.com/mobi app-development/
Approach	Hybrid	
Most suitable for	Small simple apps, prototyping	
Popularity	Average	
Available APIs	Audio, video, device status, push notifications, file system, app properties	
Supported assistants	None	
Augmented or virtual reality	No	
Background services or processes	No	
Databases	SQLite	
Can be developed in	Web environmnet	Does not require Mac for iOS builds.
Game development	No	
Invoke native libraries	No	
Programming languages	Code-free, JavaScript	
Can build apps for	Android, iOS	
MBaaS	Yes	
Multithreading	Partially	Available with script thread processes.
OpenGL	No	
Platform-specific overrides	No	
Approximate prices (per developer per year)	Indie or up to 5 employees - $ 1499. Up to 25 employees - $2499.	Enterprise prices on demand.
Supported sensors	Camera, microphone, geolocation, accelerometer, gyroscope, fingerprint and barcode scanners	

Table 32: Alpha Anywhere ctd

Framework	**Alpha Anywhere**	https://www.alphasoftware.com/mobi app-development/
Crash analysis	Yes	Custom Alpha Anywhere crash analytics.
App profiling	No	
Continuous integration	No	
Debugging	Yes	
Closed group shipping	Default	Available via native app stores.
Device farm testing	No	
Automated UI testing	No	
Unit testing	No	
Designer or preview tool	Yes	
Native look and feel	No	
Adaptive layout for mobile and tablet	Yes	
UI layers	Single platform-agnostic UI code-base.	

A.2 Appcelerator

Table 33: Appcelerator

Framework	**Appcelerator**	https://www.appcelerator.com/
Approach	Interpreted	
Most suitable for	Small to medium apps, business-grade	
Popularity	Very high	
Available APIs	Audio, video, device status, push notifications, file system, app properties, embedded browser, address book, calls and messages	
Supported assistants	Alexa, Cortana, Google Assistant, Siri	
Augmented or virtual reality	Yes	
Background services or processes	Yes	
Databases	SQLite, Couchbase	
Can be developed in	Linux, macOS, Windows	
Game development	No	
Invoke native libraries	Yes	
Programming languages	JavaScript	
Can build apps for	Android, iOS, Windows	
MBaaS	Yes	
Multithreading	Partially	Available via web-workers.
OpenGL	Yes	
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Indie - $432, company up to 25 employees - $1188.	Enterprise prices on demand.
Supported sensors	Camera, microphone, geolocation, accelerometer, gyroscope, NFC, fingerprint scanner	Barcode scanner available as 3rd-party library. Bluetooth available for limited use-cases.

Table 34: Appcelerator ctd

Framework	**Appcelerator**	https://www.appcelerator.com/
Crash analysis	Yes	Custom Appcelerator Live Stats analytics.
App profiling	Yes	
Continuous integration	Yes	
Debugging	Yes	
Closed group shipping	Default	Available via native app stores.
Device farm testing	Default	Available via common device farms.
Automated UI testing	Yes	
Unit testing	Yes	
Designer or preview tool	Yes	
Native look and feel	Yes	
Adaptive layout for mobile and tablet	Yes	
UI layers	Multiple platform-specific UI code-bases.	

A.3 Appery.io

Table 35: Appery.io

Framework	**Appery.io**	https://appery.io/
Approach	Hybrid, Web apps	
Most suitable for	Small simple apps, prototyping	
Popularity	Average	
Available APIs	Audio, video, device status, push notifications, file system, app properties, embedded browser, address book, calls and messages	
Supported assistants	None	
Augmented or virtual reality	No	
Background services or processes	Partial	Available via 3rd-party libraries or custom implementation.
Databases	SQLite	
Can be developed in	Web environment	Does not require Mac for iOS builds or Windows PC for Windows apps builds.
Game development	No	
Invoke native libraries	Yes	
Programming languages	Code-free, JavaScript	
Can build apps for	Android, iOS, Windows	
MBaaS	Yes	
Multithreading	No	
OpenGL	Partially	WebGL
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Indie - $720, company up to 5 employees - $270.	Prices for companies with more developers on demand.
Supported sensors	Camera, microphone, geolocation, accelerometer, gyroscope, bluetooth, NFC, barcode scanner	Fingerprint scanner available only for Android and iOS.

Table 36: Appery.io ctd

Framework	**Appery.io**	https://appery.io/
Crash analysis	No	
App profiling	No	
Continuous integration	Yes	
Debugging	Yes	
Closed group shipping	Default	Available via native app stores.
Device farm testing	No	
Automated UI testing	No	
Unit testing	Yes	
Designer or preview tool	Yes	
Native look and feel	No	
Adaptive layout for mobile and tablet	Yes	
UI layers	Single platform-agnostic UI code-base.	

A.4 Aquro

Table 37: Aquro

Framework	**Aquro**	http://www.aquro.com/
Approach	Hybrid	
Most suitable for	Small simple apps, prototyping	
Popularity	Very low	
Available APIs	Audio, video, device status, push notifications, file system, app properties, embedded browser, address book, calls and messages	
Supported assistants	None	
Augmented or virtual reality	No	
Background services or processes	Partially	Available via 3rd-party library or custom implementation.
Databases	None	
Can be developed in	Windows, Web environment	Does not require Mac for iOS builds.
Game development	No	
Invoke native libraries	Unknown	
Programming languages	JavaScript	
Can build apps for	Android, iOS	
MBaaS	Yes	
Multithreading	No	
OpenGL	Partially	WebGL
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Indie - free, company up to 25 employees - $108, company up to 25 employees - $588.	Enterprise prices on demand.
Supported sensors	Camera, microphone, geolocation, accelerometer, bluetooth, NFC, barcode scanner	Gyroscope available via 3rd-party libraries or custom implementation. Fingerprint scanner supported only on iOS.

Table 38: Aquro ctd

Framework	**Aquro**	http://www.aquro.com/
Crash analysis	No	
App profiling	No	
Continuous integration	No	
Debugging	Yes	
Closed group shipping	Default	Available via native app stores.
Device farm testing	No	
Automated UI testing	No	
Unit testing	No	
Designer or preview tool	Yes	
Native look and feel	No	
Adaptive layout for mobile and tablet	No	
UI layers	Single platform-agnostic UI code-base.	

A.5 Codename One

Table 39: Codename One

Framework	**Codename One**	https://www.codenameone.com/
Approach	Interpreted	
Most suitable for	Small to medium apps, business-grade	
Popularity	Average	
Available APIs	Audio, video, push notifications, file system, app properties, embedded browser, address book, calls and messages	Device status available via 3rd-party library or custom implementation.
Supported assistants	None	
Augmented or virtual reality	No	
Background services or processes	Yes	
Databases	SQLite	
Can be developed in	Linux, macOS, Windows	Does not require Mac for iOS builds or Windows PC for Windows builds.
Game development	Yes	
Invoke native libraries	Yes	
Programming languages	Java	
Can build apps for	Android, BlackBerry, iOS, Windows	
MBaaS	No	
Multithreading	Yes	
OpenGL	Yes	
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Indie - Free, company up to 5 employees - $228, up to 25 employees - $849. Enterprise prices range from $3792 to $4788.	
Supported sensors	Camera, microphone, geolocation, accelerometer.	Gyroscope, bluetooth, NFC, fingerprint and barcode scanners are available either as 3rd-party libraries or by custom implementation.

Table 40: Codename One ctd

Framework	**Codename One**	https://www.codenameone.com/
Crash analysis	Yes	Crash reports are sent as an email.
App profiling	Yes	
Continuous integration	Yes	
Debugging	Yes	
Closed group shipping	Default	Available via native app stores.
Device farm testing	Default	Available via common device farms.
Automated UI testing	Yes	
Unit testing	Yes	
Designer or preview tool	Yes	
Native look and feel	No	
Adaptive layout for mobile and tablet	No	
UI layers	Single platform-agnostic UI code-base.	

A.6 Corona Labs

Table 41: Corona Labs

Framework	**Corona Labs**	https://coronalabs.com/
Approach	Interpreted	
Most suitable for	Small to medium apps, business-grade	
Popularity	High	
Available APIs	Audio, video, push notifications, app properties, embedded browser, address book, calls and messages	File system access and device status have limited capabilities.
Supported assistants	Alexa, Cortana, Siri	
Augmented or virtual reality	Yes	
Background services or processes	No	
Databases	SQLite, Couchbase	
Can be developed in	macOS, Windows	
Game development	Yes	
Invoke native libraries	Yes	
Programming languages	Lua	
Can build apps for	Android, iOS, Windows	
MBaaS	No	
Multithreading	Partially	Can be partially substituted by Lua co-routines.
OpenGL	Yes	
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Indie - Free, company up to 10 employees - $948, companies with more employees and enterprise - $2388.	
Supported sensors	Camera, microphone, geolocation, accelerometer, gyroscope, fingerprint scanner	NFC, bluetooth and barcode scanner available as 3rd-party libraries or by custom implementation.

Table 42: Corona Labs ctd

Framework	**Corona Labs**	https://coronalabs.com/
Crash analysis	Default	Provided by native app stores or specialized 3rd-party tools.
App profiling	Yes	
Continuous integration	Yes	
Debugging	Yes	
Closed group shipping	Yes	Available via native app stores or HockeyApp.
Device farm testing	No	
Automated UI testing	No	
Unit testing	Yes	
Designer or preview tool	No	
Native look and feel	No	
Adaptive layout for mobile and tablet	Partially	Partially available with runtime platform detection.
UI layers	Single platform-agnostic UI code-base.	

A.7 Embarcadero

Table 43: Embarcadero

Framework	**Embarcadero**	https://www.embarcadero.com/
Approach	Cross-compilation, Web apps	
Most suitable for	Medium to large apps, complex and native-like	
Popularity	Low	
Available APIs	Audio, video, device status, push notifications, file system, embedded browser, address book, calls and messages	App properties access available via a 3rd-party library or custom implementation.
Supported assistants	None	
Augmented or virtual reality	Partially	Work-around possible by invoking native libraries.
Background services or processes	Yes	
Databases	SQLite, Couchbase	
Can be developed in	macOS, Windows	Linux support is planned.
Game development	Yes	
Invoke native libraries	Yes	
Programming languages	C++, Delphi	
Can build apps for	Android, iOS, Windows	BlackBerry development available via mobile web apps.
MBaaS	No	
Multithreading	Yes	
OpenGL	Yes	
Platform-specific overrides	Partially	Possible with conditional compilation.
Approximate prices (per developer per year)	Indie - Free, company up to 25 employees - $3155. Enterprise prices range from $4947 to $6984.	
Supported sensors	Camera, microphone, geolocation, accelerometer, gyroscope, bluetooth, NFC, barcode scanner	Fingerprint scanner available via 3rd-party libraries or custom implementation.

Table 44: Embarcadero ctd

Framework	**Embarcadero**	https://www.embarcadero.com/
Crash analysis	Yes	Custom Embarcadero AppAnalytics service.
App profiling	Yes	
Continuous integration	Yes	
Debugging	Yes	
Closed group shipping	Default	Available via native app stores.
Device farm testing	No	
Automated UI testing	No	
Unit testing	Yes	
Designer or preview tool	Yes	
Native look and feel	Yes	
Adaptive layout for mobile and tablet	Yes	
UI layers	Single platform-agnostic UI code-base.	

A.8 Fuse

Table 45: Fuse

Framework	**Fuse**	https://www.fusetools.com/
Approach	Interpreted	
Most suitable for	Small to medium apps, business-grade	
Popularity	Average	
Available APIs	Video, push notifications, file system, address book	
Supported assistants	None	
Augmented or virtual reality	No	
Background services or processes	No	
Databases	SQLite	
Can be developed in	macOS, Windows	
Game development	No	
Invoke native libraries	Yes	
Programming languages	JavaScript	
Can build apps for	Android, iOS	
MBaaS	No	
Multithreading	Partially	Available via web-workers.
OpenGL	Partially	WebGL
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Indie and companies up to 5 employees - Free.	Prices for larger companies and enterprise are not publicly disclosed.
Supported sensors	Camera, microphone, geolocation	Accelerometer, gyroscope, bluetooth and NFC available as 3rd-party libraries or by custom implementation.

Table 46: Fuse ctd

Framework	**Fuse**	https://www.fusetools.com/
Crash analysis	Default	Provided by native app stores or specialized 3rd-party tools.
App profiling	No	
Continuous integration	Partially	Possible, depending on selected IDE.
Debugging	Partially	Possible, depending on selected IDE.
Closed group shipping	Default	Available via native app stores.
Device farm testing	Default	Available via common device farms.
Automated UI testing	Partially	Possible, depending on selected IDE.
Unit testing	Yes	
Designer or preview tool	No	
Native look and feel	Yes	
Adaptive layout for mobile and tablet	Yes	
UI layers	Single platform-agnostic UI code-base.	

A.9 Instant Developer

Table 47: Instant Developer

Framework	**Instant Developer**	http://www.instantdeveloper.com/
Approach	Hybrid	
Most suitable for	Small simple apps, prototyping	
Popularity	Very low	
Available APIs	Audio, device status, push notifications, file system, address book, calls and messages	
Supported assistants	None	
Augmented or virtual reality	No	
Background services or processes	No	
Databases	SQLite	
Can be developed in	Windows	
Game development	No	
Invoke native libraries	No	
Programming languages	Code-free, C#, Java	
Can build apps for	Android, iOS, Windows	
MBaaS	Yes	
Multithreading	No	
OpenGL	No	
Platform-specific overrides	No	
Approximate prices (per developer per year)	Indie - Free, companies up to 5 employees - $3588, companies up to 25 employees - $4788. Enterprise prices range from $5988 to $11880.	
Supported sensors	Camera, microphone, geolocation, accelerometer, gyroscope, barcode scanner	

Table 48: Appcelerator ctd

Framework	**Instant Developer**	http://www.instantdeveloper.com/
Crash analysis	No	
App profiling	No	
Continuous integration	No	
Debugging	Yes	
Closed group shipping	Default	Available via native app stores.
Device farm testing	Default	Available via common device farms.
Automated UI testing	Yes	
Unit testing	No	
Designer or preview tool	Yes	
Native look and feel	No	
Adaptive layout for mobile and tablet	No	
UI layers	Single platform-agnostic UI code-base.	

A.10 Ionic

Table 49: Ionic

Framework	**Ionic**	http://ionicframework.com/
Approach	Hybrid	
Most suitable for	Small to medium apps, business-grade	
Popularity	Very high	
Available APIs	Audio, video, device status, push notifications, file system, app properties, address book, calls and messages	Embedded browser available as 3rd-party libraries or by custom implementation.
Supported assistants	Cortana, Google Assistant, Siri	
Augmented or virtual reality	Partially	Possible with 3rd-party libraries or custom implementation.
Background services or processes	Partially	Only custom implementation, not achievable on all mobile operating systems.
Databases	SQLite, Realm, Couchbase	
Can be developed in	Linux, macOS, Windows	
Game development	No	
Invoke native libraries	Yes	
Programming languages	AngularJS, JavaScript	
Can build apps for	Android, iOS, Windows	Limited unofficial support for BlackBerry.
MBaaS	No	
Multithreading	Partially	Available via web-workers.
OpenGL	Partially	WebGL
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Free	
Supported sensors	Camera, microphone, geolocation, accelerometer, gyroscope, bluetooth, NFC, fingerprint and barcode scanner	

Table 50: Ionic ctd

Framework	**Ionic**	http://ionicframework.com/
Crash analysis	Default	Provided by native app stores or specialized 3rd-party tools.
App profiling	Partially	Possible, depending on selected IDE.
Continuous integration	Partially	Possible, depending on selected IDE.
Debugging	Partially	Possible, depending on selected IDE.
Closed group shipping	Yes	Available via native app stores and HockeyApp.
Device farm testing	Default	Available via common device farms.
Automated UI testing	Yes	
Unit testing	Yes	
Designer or preview tool	Yes	
Native look and feel	Yes	
Adaptive layout for mobile and tablet	Yes	
UI layers	Single platform-agnostic UI code-base.	

A.11 Kivy

Table 51: Kivy

Framework	**Kivy**	https://kivy.org/#home
Approach	Cross-compiled, Hybrid	
Most suitable for	Small to medium apps, business-grade	
Popularity	Average	
Available APIs	Video, file system, app properties	Audio, device status and embedded browser are available as 3rd-party libraries or by custom implementation. Push notifications are supported only on Android and Windows, custom implementation is required. Address book, calls and messaging are available only on Android and iOS, custom implementation is required.
Supported assistants	Alexa	
Augmented or virtual reality	No	
Background services or processes	Partially	Possible with 3rd-party library or custom implementation.
Databases	SQLite	
Can be developed in	Linux, macOS, Windows	
Game development	Yes	
Invoke native libraries	Yes	
Programming languages	Python	
Can build apps for	Android, iOS, Windows	
MBaaS	No	
Multithreading	Yes	
OpenGL	Yes	
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Free	
Supported sensors	Camera	Bluetooth, NFC and barcode scanner are available as 3rd-party libraries or by custom implementation. Geolocation, accelerometer and gyroscope are available only on Android and iOS, custom implementation is required. Microphone is available only on Android, custom implementation is required.

Table 52: Appcelerator ctd

Framework	**Kivy**	https://kivy.org/#home
Crash analysis	No	.
App profiling	Partially	Possible only in code via custom libraries.
Continuous integration	Partially	Possible, depending on selected IDE.
Debugging	Partially	Possible, depending on selected IDE.
Closed group shipping	Default	Available via native app stores.
Device farm testing	No	
Automated UI testing	No	
Unit testing	Yes	
Designer or preview tool	Partially	3rd-party solutions available.
Native look and feel	No	
Adaptive layout for mobile and tablet	No	
UI layers	Single platform-agnostic UI code-base.	

A.12 Kony

Table 53: Kony

Framework	**Kony**	http://www.kony.com/
Approach	Interpreted, Web apps	
Most suitable for	Small to medium apps, business-grade	
Popularity	Low	
Available APIs	Audio, video, push notifications, file system, embedded browser, address book, calls and messages	App properties available as a 3rd-party library or by custom implementation.
Supported assistants	Alexa, Siri	
Augmented or virtual reality	Partially	Experimental apps exist, full support not guaranteed.
Background services or processes	Partially	Available only on Windows.
Databases	File database	
Can be developed in	macOS, Windows	
Game development	No	
Invoke native libraries	Yes	
Programming languages	Code-free, JavaScript	
Can build apps for	Android, iOS, Windows	BlackBerry development available as mobile web apps.
MBaaS	Yes	
Multithreading	Yes	
OpenGL	Partially	Possible with custom implementation.
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Indie - Free.	Only starter prices are publically disclosed.
Supported sensors	Camera, geolocation, accelerometer, gyroscope	Microphone, bluetooth, NFC, fingerprint and barcode scanner available as 3rd-party libraries or by custom implementation.

Table 54: Kony ctd

Framework	**Kony**	http://www.kony.com/
Crash analysis	Default	Provided by native app stores or specialized 3rd-party tools.
App profiling	Yes	
Continuous integration	Yes	
Debugging	Yes	
Closed group shipping	Yes	Available via native app stores and HockeyApp.
Device farm testing	Default	Available via common device farms.
Automated UI testing	Yes	
Unit testing	Yes	
Designer or preview tool	Yes	
Native look and feel	No	
Adaptive layout for mobile and tablet	Yes	
UI layers	Single platform-agnostic UI code-base.	

A.13 Monaca (Onsen UI)

Table 55: Monaca (Onsen UI)

Framework	**Monaca (Onsen UI)**	https://monaca.io/
Approach	Hybrid	
Most suitable for	Small simple apps, prototyping	
Popularity	Average	
Available APIs	Audio, video, device status, push notifications, file system	App properties available as 3rd-party library or by custom implementation.
Supported assistants	None	
Augmented or virtual reality	No	
Background services or processes	No	
Databases	SQLite	
Can be developed in	macOS, Windows	CLI tool available for Linux.
Game development	No	
Invoke native libraries	Yes	
Programming languages	JavaScript	
Can build apps for	Android, iOS, Windows	
MBaaS	Yes	
Multithreading	Partially	Available via web-workers.
OpenGL	Yes	
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Indie and companies up to 5 employees - Free, up to 10 employees - $190, companies up to 25 employees - $490. Enterprise features available from $780.	
Supported sensors	Camera, geolocation, accelerometer, bluetooth, barcode scanner	Microphone, gyroscope, NFC and fingerprint scanner available as 3rd-party libraries or by custom implementation.

Table 56: Monaca (Onsen UI) ctd

Framework	**Monaca (Onsen UI)**	https://monaca.io/
Crash analysis	Default	Provided by native app stores or specialized 3rd-party tools.
App profiling	Partially	Possible, depending on selected IDE.
Continuous integration	Partially	Possible, depending on selected IDE.
Debugging	Partially	Possible, depending on selected IDE.
Closed group shipping	Default	Available via native app stores.
Device farm testing	Default	Available via common device farms.
Automated UI testing	Yes	
Unit testing	Yes	
Designer or preview tool	No	
Native look and feel	Yes	
Adaptive layout for mobile and tablet	Yes	
UI layers	Single platform-agnostic UI code-base.	

A.14 NativeScript

Table 57: NativeScript

Framework	**NativeScript**	https://www.nativescript.org/
Approach	Interpreted	
Most suitable for	Small to medium apps, business-grade	
Popularity	Very high	
Available APIs	Audio, device status, push notifications, file system, app properties, address book, calls and messages	Video and embedded browser available as 3rd-party libraries or by custom implementation
Supported assistants	Alexa, Cortana, Google Assistant, Siri	
Augmented or virtual reality	Yes	
Background services or processes	Partially	Possible with 3rd-party libraries or custom implementation.
Databases	SQLite, Couchbase	
Can be developed in	Linux, macOS, Windows	
Game development	No	
Invoke native libraries	Yes	
Programming languages	AngularJS, JavaScript	
Can build apps for	Android, iOS	Preview for Windows available.
MBaaS	No	
Multithreading	Partially	Available via web-workers.
OpenGL	Partially	WebGL
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Free	
Supported sensors	Camera, geolocation	Microphone, accelerometer, gyroscope, bluetooth, NFC, fingerprint and barcode scanner available as 3rd-party libraries or by custom implementation.

Table 58: NativeScript ctd

Framework	**NativeScript**	https://www.nativescript.org/
Crash analysis	Yes	Custom Telerik Analytics.
App profiling	Yes	
Continuous integration	Partially	Possible, depending on selected IDE.
Debugging	Partially	Possible, depending on selected IDE.
Closed group shipping	Yes	Available via native app stores and HockeyApp.
Device farm testing	Yes	Telerik Test Studio.
Automated UI testing	Yes	
Unit testing	Yes	
Designer or preview tool	No	
Native look and feel	Yes	
Adaptive layout for mobile and tablet	Yes	
PUI layers	Both platform-specific and platform-agnostic UI code-base available.	

A.15 NeoMAD

Table 59: NeoMAD

Framework	**NeoMAD**	http://neomades.com/
Approach	Cross-compiled	
Most suitable for	Medium to large apps, complex and native-like	
Popularity	Very low	
Available APIs	Audio, video, device status, push notifications, file system, app properties	Embedded browser, address book, calls and messaging available as 3rd-party libraries or by custom implementation.
Supported assistants	None	
Augmented or virtual reality	No	
Background services or processes	Yes	
Databases	SQLite	
Can be developed in	macOS, Windows	
Game development	Yes	
Invoke native libraries	Yes	
Programming languages	Java	
Can build apps for	Android, BlackBerry, iOS, Windows	
MBaaS	No	
Multithreading	Yes	
OpenGL	Yes	
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Indie - Free, companies up to 10 employees - $999, larger companies and enterprise - $4999.	
Supported sensors	Camera, microphone, geolocation, accelerometer, gyroscope, NFC, barcode scanner	Bluetooth and fingerprint scanner available as 3rd-party libraries or by custom implementation.

Table 60: NeoMAD ctd

Framework	**NeoMAD**	http://neomades.com/
Crash analysis	No	
App profiling	Yes	
Continuous integration	Yes	
Debugging	Yes	
Closed group shipping	Default	Available via native app stores.
Device farm testing	Default	Available via common device farms.
Automated UI testing	Partially	Possible with 3rd-party tools.
Unit testing	Yes	
Designer or preview tool	No	
Native look and feel	No	
Adaptive layout for mobile and tablet	Yes	
UI layers	Single platform-agnostic UI code-base.	

A.16 NS Basic

Table 61: NS Basic

Framework	**NS Basic**	https://www.nsbasic.com/
Approach	Hybrid, Web apps	
Most suitable for	Small simple apps, prototyping	
Popularity	Very low	
Available APIs	Audio, video, device status, push notifications, file system, app properties, embedded browser, address book, calls and messages	
Supported assistants	Google Assistant, Siri	
Augmented or virtual reality	No	
Background services or processes	Partially	Available only on Android, custom implementation needed.
Databases	SQLite	
Can be developed in	Linux, macOS, Windows	
Game development	No	
Invoke native libraries	No	
Programming languages	BASIC, JavaScript	
Can build apps for	Android, iOS	Windows development available as mobile web apps.
MBaaS	No	
Multithreading	No	
OpenGL	Partially	WebGL
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Indie or companies up to 25 employees - $150. Enterprise features available from $900.	
Supported sensors	Camera, microphone, geolocation, accelerometer, gyroscope, bluetooth, NFC, fingerprint and barcode scanners.	

Table 62: NS Basic ctd

Framework	**NS Basic**	https://www.nsbasic.com/
Crash analysis	No	
App profiling	Partially	Possible, depending on selected IDE.
Continuous integration	Partially	Possible, depending on selected IDE.
Debugging	Partially	Possible, depending on selected IDE.
Closed group shipping	Default	Available via native app stores.
Device farm testing	Default	Available via common device farms.
Automated UI testing	Yes	
Unit testing	Yes	
Designer or preview tool	Yes	
Native look and feel	No	
Adaptive layout for mobile and tablet	No	
UI layers	Single platform-agnostic UI code-base.	

A.17 PhoneGap

Table 63: PhoneGap

Framework	**PhoneGap**	http://phonegap.com/
Approach	Hybrid	
Most suitable for	Small simple apps, prototyping	
Popularity	Very high	
Available APIs	Audio, video, device status, push notifications, file system, app properties, address book, calls and messages	Embedded browser available only on Android, iOS and Windows.
Supported assistants	Cortana, Google Assistant, Siri	
Augmented or virtual reality	Yes	
Background services or processes	Partially	Limited availability, custom implementation is needed.
Databases	SQLite	
Can be developed in	Linux, macOS, Windows	Does not require Mac for iOS builds.
Game development	No	
Invoke native libraries	Yes	
Programming languages	JavaScript	
Can build apps for	Android, BlackBerry iOS, Windows	
MBaaS	No	
Multithreading	Partially	Available via web-workers.
OpenGL	Partially	WebGL
Platform-specific overrides	Yes	
Approximate prices (per developer per year)	Free	
Supported sensors	Camera, microphone, geolocation, accelerometer, gyroscope, NFC, barcode scanner.	Bluetooth available only on Android, iOS and Windows. Fingerprint scanner available only on Android and iOS.

Table 64: PhoneGap ctd

Framework	**PhoneGap**	http://phonegap.com/
Crash analysis	Default	Provided by native app stores or specialized 3rd-party tools.
App profiling	Partially	Possible, depending on selected IDE.
Continuous integration	Partially	Possible, depending on selected IDE.
Debugging	Partially	Possible, depending on selected IDE.
Closed group shipping	Yes	Available via native app stores and HockeyApp.
Device farm testing	Default	Available via common device farms.
Automated UI testing	Yes	
Unit testing	Yes	
Designer or preview tool	No	
Native look and feel	Yes	
Adaptive layout for mobile and tablet	Yes	
UI layers	Single platform-agnostic UI code-base.	