Approaches to Mobile Application Development: Comparative Performance Analysis

Lisandro Delía
Institute of Research in Computer Science III-LIDI. School of Computer Science.
National University of La Plata
50 y 120. 2nd Floor
La Plata, Argentina
+54 221 4227707
ldelia@lidi.info.unlp.edu.ar

Nicolás Galdamez
Institute of Research in Computer Science III-LIDI. School of Computer Science.
National University of La Plata
50 y 120. 2nd Floor
La Plata, Argentina
+54 221 4227707
ngaldamez@lidi.info.unlp.edu.ar

Leonardo Corbalan
Institute of Research in Computer Science III-LIDI. School of Computer Science.
National University of La Plata
50 y 120. 2nd Floor
La Plata, Argentina
+54 221 4227707
corbalan@lidi.info.unlp.edu.ar

Patricia Pesado
Institute of Research in Computer Science III-LIDI. School of Computer Science. National University of La Plata
50 y 120. 2nd Floor
La Plata, Argentina
+54 221 4227707
ppesado@lidi.info.unlp.edu.ar

Pablo Thomas
Institute of Research in Computer Science III-LIDI. School of Computer Science. National University of La Plata
50 y 120. 2nd Floor
La Plata, Argentina
+54 221 4227707
pthomas@lidi.info.unlp.edu.ar

***Abstract*—The purpose of software development is meeting both functional and non-functional requirements. In mobile device applications, non-functional requirements are more relevant due to the restrictions inherent to these devices. The performance of a mobile application affects user preference for use. In this article, we present a performance study of the approaches used to develop software for mobile devices for the two currently more commonly used operating systems: iOS and Android. The results obtained are analyzed, and conclusions supported by the tests carried out are drawn.**

Keywords—mobile devices; multi-platform mobile applications; native mobile applications; performance

I. Introduction

Barely a few decades ago, the use of software systems was limited to a reduced group of specialized users. Those times are in high contrast with the current situation, with smartphones, small general purpose mobile computers, have become everyday and ubiquitous products. These devices can be used to carry out complex and critical tasks, which results in a requirement for the continuous improvement of computational capacity, availability, efficient performance, and so forth. The fast evolution of this technology puts a strain on Software Engineering.

In relation to the development for mobile devices, a number of features specific to this activity that were not present in traditional software development have to be taken into account [1]. The type of device on which the application to be developed will be run should be considered. The diversity of platforms, programming languages, development tools, standards, protocols and network technologies, limited device capacity in some cases, and time-to-market demands, to mention but a few, are some of the issues to be dealt with.

In most cases, the success of a software product for mobile devices will be conditioned by the popularity it achieves. To maximize market presence, it should be possible to run the application on as many existing mobile platforms as possible, especially the two most popular ones: Android and iOS [2]. To achieve this goal, there are two alternatives:

1) Developing specific applications for each platform, with several parallel development projects, using the tools and languages specific to each platform. These applications are known as native applications.

2) Developing applications that can be run directly on more than one operating system platform using a single development project. These applications are known as multi-platform applications.

In recent years, the interest of the Software Engineering community for the development of multi-platform applications for mobile devices has increased.

In [3], the authors present a comparative analysis of development approaches for mobile device multi-platform application, and the following taxonomy is proposed: mobile web applications, hybrid applications, interpreted applications and cross-compilation applications.

In [4] and [5], the general aspects of multi-platform development frameworks for mobile devices are discussed.

In [6], non-functional aspects are compared for the different multi-platform application development approaches for mobile devices.

In [7], the authors of this paper have analyzed the

978-1-5090-5443-5/17/$31.00 ©2017 IEEE

advantages and disadvantages of the multi-platform development methods mentioned above, from the point of view of the Software Engineer.

The method of choice for the development of applications for mobile devices depends on several factors. One of these, oftentimes essential, is execution time. The desire for optimizing the execution time of any software application is inherent to the Computer Science field. This is evident, for example, by the evolution of processor computation power. Additionally, efficient performance is one of the attributes that software applications must meet based on several quality standards, including ISO/IEC 9126 and ISO/IEC 25010 [8].

As regards the development of applications for mobile devices, application execution time is relevant and should be considered for various reasons.

An application's execution time is strongly linked to energy consumption [9], which is limited by battery life.

Also, application performance is mainly reflected by user ratings in on-line application stores. Low-performing applications can result in non-satisfied users, resulting in negative publicity [2]. Andre Charland and Brian Leroux identify execution time as one of the main issues to solve when developing multi-platform applications, and they state that end users care about software quality and user experience [10].

Corral, Sillitti y Succi carried out a comparative analysis between the performance of native and hybrid applications using the Phonegap framework for one version of the Android OS [11].

It should be noted that no articles assessing and comparing the performance for the various multi-platform development methods following the taxonomy proposed in [3], namely mobile web applications, hybrid applications, interpreted applications and cross-compilation applications, have been found. This is the taxonomy used as reference in this article.

In this paper, a comparative analysis of the performance achieved by mobile device applications developed using the native approach and the different multi-platform approaches described in [3] is presented.

Section 2 introduces the different types of mobile applications, Section 3 describes the experiment used to determine execution times for native applications and the various multi-platform mobile application development methods, Section 4 presents and discusses the results obtained, and subsequent sections summarize our conclusions and future work.

II. Types of Applications for Mobile Devices

In recent years, the mobile device market, especially that of smart phones, has seen a remarkable growth. In particular, the operating systems that have grown the most are Android and iOS [2].

Each of these operating systems has its own development infrastructure. The main challenge application providers face is offering solutions for all platforms in the market; however, achieving this goal usually involves high development costs that are often hard to afford [12].

The ideal solution to this problem from a developer perspective is creating and maintaining a single application for all platforms. The purpose of multi-platform development is maintaining the same code base for various platforms. Thus, the development effort and cost is significantly reduced.

In the following sections, we present the different approaches used for developing applications for mobile devices:

A. Native Applications

Native applications are developed to be run on a specific platform, considering the type of device, the operating system and the version to be used.

The source code is compiled to obtain the executable code, similar to the process used for traditional desktop applications.

When the application is ready for distribution, it is transferred to the specific App Stores (application stores) of each operating system. These stores have an audit process in place to assess if the application meets the requirements of the platform on which it is to be run. Finally, the application becomes available to the end users.

An important characteristic of native applications is the possibility of interacting with all the capabilities offered by the device (camera, GPS, accelerometer, calendar, and so forth). Additionally, Internet access is not required to run these applications. Their execution is fast and they can be run in the background and alert the user when an event requiring their attention occurs.

This development approach involves higher costs, since a different programming language has to be used for each platform. Therefore, if the goal is to span over several platforms, an application for each of them has to be produced. This involves carrying out the codification, testing, maintenance, and new version distribution processes more than once.

B. Web Applications

Web applications for mobiles are designed to be executed in the browser of the device. They are developed using HTML, CSS and JavaScript, the same technologies used for creating web sites.

One of the advantages of this approach is that no specific component has to be installed in the device, and no approval from the manufacturer is required for the applications to be published and used. Only Internet access is required. Also, updates appear directly on the device, since changes are applied on the server and available immediately to the users. In brief, it is fast and easy to implement.

However, the greatest advantage of web applications is unquestionably their independence from the platform. There is no need to adapt to any specific operating system. Only a browser is required. On the other hand, this could reduce execution speed and result in a poorer user experience with interfaces that are more limited than those offered by native applications. Also, performance can be affected by connectivity issues.

Finally, the security restrictions imposed by the execution of the code through a browser result in a more difficult access for the applications to all the features offered by the device [13].

C. Hybrid Applications

Hybrid applications use web technologies (HTML, JavaScript and CSS), but are not run by a browser. Instead, they are run on a web container of the device that has access to device-specific features through an API.

Hybrid applications offer great advantages because they allow code reuse for the various platforms, access to device hardware, and distribution through application stores [7].

Hybrid applications have two disadvantages in relation to native applications:

1) User experience suffers from not using the native components in the interface.

2) Execution could be slower due to the additional load associated to the web container.

One of the most popular frameworks is Apache Cordova [14]; it uses HTML, JavaScript and CSS technologies, run on a specific web container, plus an API to access the functionalities of the mobile device itself. The architecture of an Apache Cordova application is represented in Fig. 1.

D. Interpreted Applications

Interpreted applications are built from a single project that is mostly translated to native code, with the rest being interpreted at runtime. Their implementation is platform-independent and uses several technologies and languages, such as Java, Ruby, XML, and so forth.

Unlike the web and hybrid multi-platform development approaches, with the interpreted applications approach native interfaces are obtained, which is one of the main advantages of this type of applications.

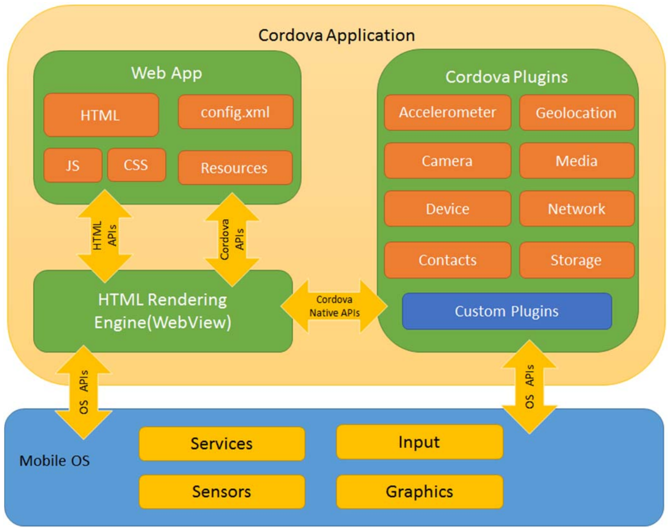

Fig. 1. Architecture of an Apache Cordova application [14]

Some of the most popular interpreted development environments are Appcelerator Titanium [15] and NativeScript [16].

Appcelerator Titanium is an open source framework that allows creating mobile applications for iOS and Android platforms. This framework includes Titanium Studio, a free-code development environment for the codification of multi-platform mobile applications, and SDK Titanium, a number of tools for developing, testing, analyzing, debugging and compiling applications.

The applications developed with Appcelerator Titanium are coded using JavaScript, which is interpreted at runtime by means of a JavaScript engine that is run on the operating system of the device. Using Titanium's API, each element of the JavaScript code is mapped to its corresponding native element. Thus, Titanium's API acts as a bridge, providing user interfaces built with native controls.

NativeScript is a recent open source project that allows generating native applications using JavaScript. Additionally, the application can be developed using TypeScript, which is a free, open source language developed by Microsoft, that extends to JavaScript, essentially adding static typing and class-based objects. In this sense, when the application is compiled, the TypeScript code is translated to JavaScript code.

NativeScript provides a multi-platform module that allows obtaining native applications from JavaScript code. This module allows accessing the functionalities offered by the device and its underlying platform consistently from the JavaScript code. Similarly, user interfaces can be defined by means of JavaScript code, HTML documents and CSS files, independently from the real native components. When the application is compiled, part of the multi-platform code is translated to native code, while the remaining code is interpreted at runtime. Fig. 2 shows a representation of the internal architecture of NativeScript [16].

For the time being, NativeScript allows generating applications for Android and iOS, but Windows Phone support is projected.

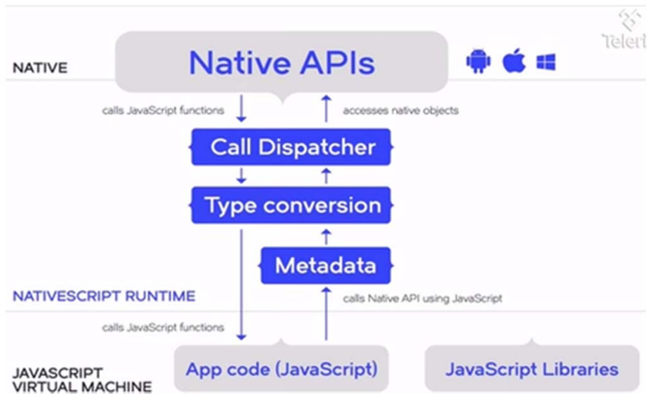

Fig. 2. Interpretation process with NativeScript [16]

E. Applications Generated by Cross-Compilation

These applications are compiled natively by creating a specific version for each target platform. Some examples of development environments used to generate applications by cross-compilation are Xamarin [17] and Corona [18].

Xamarin allows compiling fully native applications for iOS, Android and OS X sharing the same base code written in C#. Integrated to Microsoft Visual Studio, it also allows generating applications for Windows, including Windows RT for tablets and Windows Phone for mobiles.

Xamarin allows sharing the entire business logics code, but user interfaces must be programmed separately for each target platform (see Fig. 3). Thus, code reutilization is affected by the characteristics of the application being developed. Statistical studies carried out by Xamarin report that code reutilization is close to 85%.

Corona is a multi-platform framework that allows developers build general-purpose applications and games for major platforms, including OS X, Windows, iOS, Android, Kindle, Windows Phone 8, Apple TV and Android TV. A single base code is used, which is then published for the different platforms. Unlike Xamarin, no specialized rewriting or projects are required. Programming is done with Lua, which is a simple scripting language.

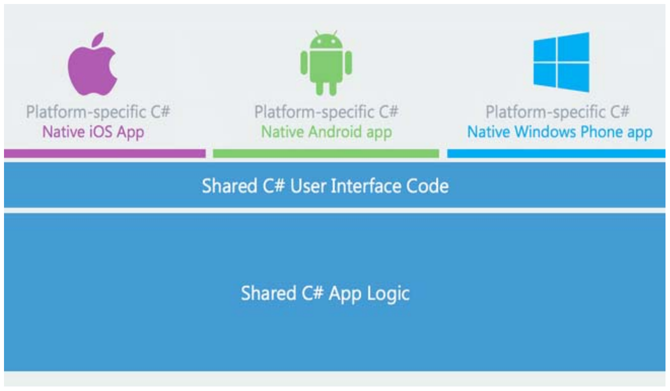

Fig. 3. Xamarin's unique development approach [17]

Basically, Corona focuses on helping the developer build applications in a fast and simple manner. Corona provides a large number of APIs and plugins that add specific functionalities and help speed up and simplify application development.

III. Experiments

A. Test Design

All tests whose results are presented here were carried out on mobile devices with Android and iOS operating systems, since currently both OS combined represent the lion's share of the global market, as indicated in [2].

To carry out experiments, multi-platform development frameworks were selected based on the alternatives presented in [7]. Tests were designed to assess the performance of Apache Cordova [14], Appcelerator Titanium [15] and Xamarin [17], corresponding to the hybrid, interpreted and cross-compilation development methods, respectively. Other multi-platform development frameworks that have been gaining popularity recently were also tested: NativeScript (interpreted development) and Corona (cross-compilation). Finally, web applications and the native development approach for Android and iOS were also included in the set of tests. This resulted in a fairly representative sample of the various currently available options.

To carry out the tests, six different mobile devices were used – three of them, identified as D_{A1}, D_{A2} and D_{A3} (two smartphones and one tablet) had the Android operating system, while the other three, identified as D_{I1}, D_{I2} and D_{I3} (two smartphones and one tablet), had the iOS operating system (see Table 1).

TABLE I. Mobile Devices Used for Testing

ID	OS	Characteristics
D_{A1}	Android 4.4.	Smartphone, brand: Motorola, model: Moto-G2, processor: Quad-core 1.2 GHz Cortex-A7, RAM 1GB Snapdragon 400
D_{A2}	Android 5.0.2.	Smartphone, brand: Samsung, model: S6, processor: Octa-core (4x2.1 GHz Cortex-A57 & 4x1.5 GHz Cortex-A53), RAM 3GB Exynos 7420 Octa
D_{A3}	Android 4.2.2	Tablet, brand: Samsung, model: Tab 2, processor: Dual-core 1.0 GHz, RAM 1GB TI OMAP 4430
D_{I1}	iOS 9.2	Smartphone, brand: Apple, model: 5S, processor: Dual-core 1.3 GHz Cyclone (ARM v8-based), RAM 1GB Apple A7
D_{I2}	iOS 9.1	Smartphone, brand: Apple, model: 6 plus, processor: Dual-core 1.4 GHz Typhoon (ARM v8-based), RAM 1GB Apple A8
D_{I3}	iOS 9.1	Tablet, brand: Apple, model: Ipad Air, processor: Dual-core 1.3 GHz Cyclone (ARM v8-based), RAM 1GB Apple A7

Seven different analysis scenarios were defined, one for each development strategy used:

1) Native for Android and native for iOS
2) Web applications (multi-platform)
3) Apache Cordova (multi-platform, hybrid)
4) Appcelerator Titanium (multi-platform, interpreted)
5) NativeScript (multi-platform, interpreted)
6) Xamarin (multi-platform, cross-compilation)
7) Corona (multi-platform, cross-compilation)

Tests for each of the seven scenarios listed above were carried out in all six devices, for a total of 42 test cases.

To assess processing speed, a simple calculation including several iterations, mathematical operations and floating point arithmetic was proposed, which is summarized in the following series:

$$serie = \sum_{j=1}^{5} \sum_{k=1}^{100000} \left(\log_2(k) + \frac{3k}{2j} + \sqrt{k} + k^{j-1}\right) \qquad (1)$$

As way of example, Fig. 4 shows the multi-platform code developed in Apache Cordova for calculating this series.

The experiment proposed allows accurately measuring the variable that is being analyzed, in this case, the execution time required to carry out intensive mathematical calculations.

This type of mathematical calculation is frequent in various applications that are run on mobile devices, such as games, augmented reality applications, image processing applications, and so forth, in which using the processing power of the Graphics Processing Unit (GPU) to carry out the calculation is not always possible.

The source code used for the experiments carried out can be found in [19].

In the following sections, we describe the experiments and discuss the results obtained.

B. Data Collection

For each of the 42 test cases defined, 30 separate runs of the experiment were carried out, obtaining in each case a sample T, where T = T_1, T_2, … T_{30}, and T_i = time required for calculating the series on the *n*th run of the experiment. Time T_i is expressed in milliseconds.

To characterize each of the samples obtained, statistic variables $\bar{T}$ and *S*, corresponding to the mean (or sample average) and sample standard deviation (see Table 2) were calculated.

TABLE II. STATISTICAL VARIABLES USED FOR DATA ANALYSIS

Given sample T=T_1, T_2, …, T_n	
Mean or sample average	$\bar{T} = \left(\frac{1}{n}\right)\sum_{i=1}^{n} T_i$
Sample standard deviation	$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(T_i - \bar{T})^2}$

IV. RESULTS OBTAINED

Table 3 shows a summary of the results obtained. It includes the values for $\bar{T}$ and *S* calculated for each of the test cases. These values allow comparing the performance of the different applications generated with the different development approaches, assessed on each of the six devices used.

The values presented in Table 3 and represented as bar charts in Fig. 5 suggest that Android operating system and iOS operating system cases should be analyzed separately. Clearly, the shapes of the bar charts shown in Figure 2 are repeated in a similar manner in the scenarios with the same operating system, but there are marked differences between different operating systems.

```
var startTime = new Date().getTime();
var serie = 0;
for ( var j=1; j <= 5; j++ )
{
   for ( var k=1; k <= 100000; k++ )
   {
      series = series + (Math.log(k)/Math.LN2) + (3*k/2*j) +
              Math.sqrt(k) + Math.pow(k, j-1);
   }
}
var finalTime = new Date().getTime();
var duration = finalTime - startTime;
document.getElementById('result').innerHTML = duration + ' -> ' + series;
```

Fig. 4. Series calculation, multi-platform code developed in Apache Cordova

TABLE III. SUMMARY OF THE RESULTS OBTAINED

		Native	**WebApp**	**Apache Cordova**	**Titanium**	**NativeScript**	**Xamarin**	**Corona**
D_{A1}	$\bar{T}$	532.93	186.27	230.33	211.67	187.30	395.17	1401.73
	S	16.14	6.32	14.22	24.95	9.39	8.95	12.60
D_{A2}	$\bar{T}$	211.80	90.67	85.77	95.63	89.67	211.00	600.53
	S	19.97	12.48	8.83	7.64	9.16	6.69	5.95
D_{A3}	$\bar{T}$	763.80	172.73	190.60	192.70	183.50	379.33	1344.30
	S	28.98	15.51	9.36	16.80	3.04	8.31	23.39
D_{I1}	$\bar{T}$	4.13	57.10	323.73	299.77	252.03	125.43	39.63
	S	0.78	16.55	16.62	4.01	7.28	11.03	0.85
D_{I2}	$\bar{T}$	4.13	41.90	263.97	241.13	223.43	103.03	98.63
	S	0.73	5.38	15.44	4.95	8.61	4.91	6.13
D_{I3}	$\bar{T}$	2.53	41.67	292.23	272.67	225.97	110.53	109.67
	S	0.57	4.44	10.82	5.50	2.77	3.90	2.75

V. Result Analysis

Due to hardware differences in the devices used, comparing the performance obtained with native Android applications to that obtained with native iOS applications would not be prudent. However, the results obtained allow concluding that the native approach used in iOS is much more efficient than the one used in Android. There are several factors that can justify this, for example, the inherent difference in running Objective C code in iOS, compared to running Java code in Android, which requires Android Runtime (ART) to operate, which slows it down.

As regards multi-platform web applications, performance results stood out positively compared to the rest, both in Android and iOS. Thus, the multi-platform web development approach would be a convenient option to achieve good performance in all mobile devices, regardless of their operating system. However, this choice could be affected by the limitations these applications have to fully access the specific features of each device.

As regards the hybrid and interpreted approaches —analyzed through Córdova, Titanium and NativeScript technologies—, it should be noted that, even if these approaches operate in different ways, they do have something in common: they run JavaScript code. In this sense, the role of the JavaScript engine responsible for converting the JavaScript code into optimized code, which will in turn be interpreted by a WebView, is essential. The tests carried out with these approaches in Android —which uses the JavaScript V8 engine— had a similar behavior to that of the web approach, and better than the native and the cross-compilation approaches. On the contrary, the results of the tests carried out in the mobile devices that run iOS —which uses the JavaScriptCore engine— were worse than the native, web and cross-compilation approaches.

In iOS mobile devices, the cross-compilation cases analyzed with Xamarin and Corona technologies obtained better results than hybrid and interpreted cases, but performed worse than native and web approaches. By contrast, in Android mobile devices, the results obtained when testing the applications built using the cross-compilation approach were the worst. The need to run Java code through Android Runtime, among other factors, explains this result.

VI. Conclusions

A comparative study of the processing time of software applications for mobile devices, generated using different development approaches, was presented.

The test scenarios used included the two dominant operating systems in the mobile device market, Android and iOS. Each OS was run on two smartphones (considered to be medium-end and high-end at the time of writing this article) and one tablet.

These six devices were used to compare the performance of applications built using both native and multi-platform development approaches. To this end, a set of ad-hoc applications was used:

1) Native application for Android and native application for iOS
2) Web application (multi-platform)
3) Apache Cordova application (multi-platform, hybrid)
4) Appcelerator Titanium application (multi-platform, interpreted)
5) NativeScript application (multi-platform, interpreted)
6) Xamarin application (multi-platform, cross-compilation)
7) Corona application (multi-platform, cross-compilation)

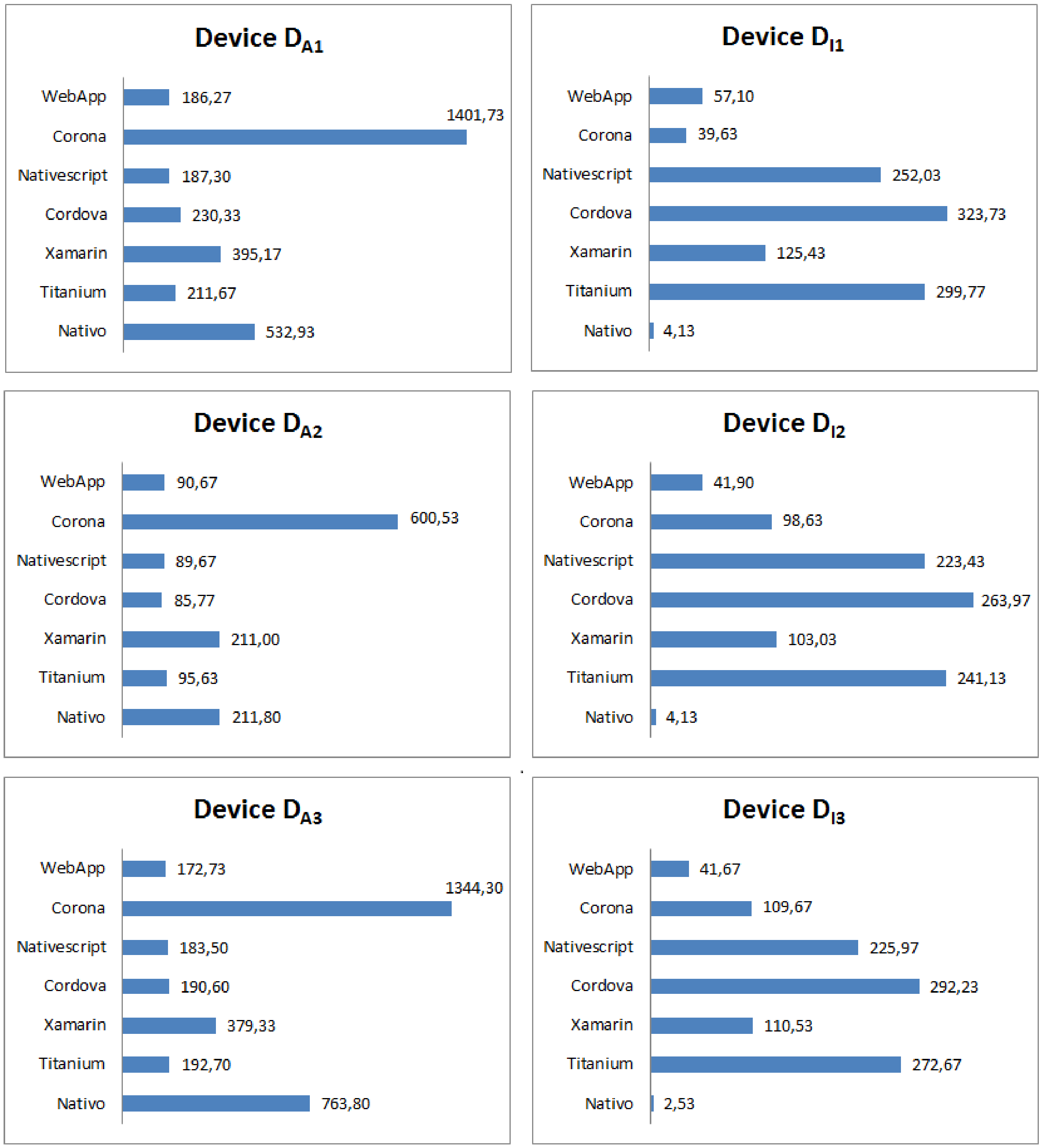

Fig. 5. Bar chart of the samples collected (bars show average time expressed in milliseconds)

In all devices with Android operating system, the worst performer was Corona, followed by the native approach and Xamarin. This is the complete opposite of the results obtained with iOS, where the development technologies mentioned above were always in the top four. In particular, the dominance of the native approach compared to all others in the case of iOS devices is remarkable.

As regards the best performers in Android devices, no clear winner could be identified. Both web applications as well as Córdova, NativeScript and Titanium achieved good results compared to the rest. Again, this situation is the opposite to that obtained with iOS devices, where Córdova, Titanium and NativeScript obtained the worst measurements among the technologies used. That was not the case with web applications in iOS, which also produced good results compared to the rest.

In general terms, operating system aside, web applications showed good performance when considering all of the cases analyzed.

Nowadays, when a software system is developed, it is possible to generate the corresponding version for mobile devices. Oftentimes, this task is not simple, and one of the most important decisions to be made is choosing the development method.

Based on the work presented here, a performance indicator is available that can be useful for Software Engineers who need to select a software development approach for mobile devices.

On the other hand, no articles by other authors have been found analyzing the performance of all software development approaches for mobile devices. This issue, as indicated in the introduction, is important for the Software Engineering community, and the articles surveyed so far have focused only on native and/or hybrid approaches.

VII. Future Work

As a future line of work, we are planning to expand this performance assessment to include other development aspects for mobile devices, such as disk access, battery consumption, and other functionalities.

References

[1] Mona Erfani Joorabchi, Ali Mesbah, Philippe Kruchten. Real Challenges in Mobile App Development, ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, Maryland, US, October 2013.

[2] Florian Rösler, André Nitze, Andreas Schmietendorf. Towards a Mobile Application Performance Benchmark. ICIW 2014: The Ninth International Conference on Internet and Web Applications and Services, At Paris, France.

[3] Spyros Xanthopoulos, Stelios Xinogalos, A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications, BCI' 2013, Greece, 2013.

[4] Yonathan Aklilu Redda, Cross platform Mobile Applications Development, Norwegian University of Science and Technology , Master in Information Systems, June 2012.

[5] Dalmasso I., Datta S.K., Bonnet C. Nikaein N., Survey, comparison and evaluation of cross platform mobile application development tools, Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International.

[6] Henning Heitkötter, Sebastian Hanschke and Tim A. Majchrzak, Comparing Cross Platform Development approaches For Mobile Applications, 8th International Conference on Web Information Systems and Technologies (WEBIST), Porto, Portugal, 2012.

[7] Delia, L.; Galdamez, N.; Thomas, P.; Corbalan, L.; Pesado, P., Multi-platform mobile application development analysis, Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on, Athens, Greece, 2015.

[8] Jung, H.W, Kim, S.G., Chung, C.S. Measuring Software Quality: A Survey of ISO/IEC 9126. IEEE Software, September/October 2004. pp. 88 – 92. 2004.

[9] Luis Corral, Anton B. Georgiev, Alberto Sillitti, Giancarlo Succi, Can execution time describe accurately the energy consumption of mobile apps? An experiment in Android. GREENS 2014 Proceedings of the 3rd International Workshop on Green and Sustainable Software. Pages 31-37

[10] Andre Charland, Brian Leroux, Mobile application development: web vs. native. Magazine Communications of the ACM CACM Homepage archive Volume 54 Issue 5, May 2011 Pages 49-53 ACM New York, NY, USA

[11] Luis Corral, Alberto Sillitti, Giancarlo Succi, Mobile multiplatform development: An experiment for performance analysis, The 9th International Conference on Mobile Web Information Systems (MobiWIS), Ontario, Canada, 2012.

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

[13] Tracy, K.W., Mobile Application Development Experiences on Apple's iOS and Android OS, Potentials, IEEE, 2012

[14] http://cordova.apache.org [Accessed 14 Sep 2016]

[15] http://www.appcelerator.com [Accessed 14 Sep 2016]

[16] https://www.nativescript.org/ [Accessed 14 Sep 2016]

[17] http://xamarin.com [Accessed 14 Sep 2016]

[18] https://coronalabs.com/ [Accessed 14 Sep 2016]

[19] https://gitlab.com/iii-lidi/performance-assessment-multiplatform-mobile-applications/tree/master [Accessed 14 Sep 2016]