Development Frameworks for Mobile Devices: A Comparative Study about Energy Consumption

Leonardo Corbalan
Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática de la UNLP
La Plata, Buenos Aires, Argentina
corbalan@lidi.info.unlp.edu.ar

Juan Fernandez
Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática de la UNLP
La Plata, Buenos Aires, Argentina
jfernandez@lidi.info.unlp.edu.ar

Alfonso Cuitiño
Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática de la UNLP
La Plata, Buenos Aires, Argentina
acuitino@lidi.info.unlp.edu.ar

Lisandro Delia
Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática de la UNLP
La Plata, Buenos Aires, Argentina
ldelia@lidi.info.unlp.edu.ar

Germán Cáseres
Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática de la UNLP
La Plata, Buenos Aires, Argentina
gcaseres@lidi.info.unlp.edu.ar

Pablo Thomas
Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática de la UNLP
La Plata, Buenos Aires, Argentina
pthomas@lidi.info.unlp.edu.ar

Patricia Pesado
Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática de la UNLP
La Plata, Buenos Aires, Argentina
ppesado@lidi.info.unlp.edu.ar

ABSTRACT

The development frameworks and the power efficiency in mobile devices have been studied separately. This paper deals with both topics together to determine how the selection of a development framework can impact on energy consumption of a mobile application. The focus is on applications with high processing load, audio and video playback. The results were analyzed and conclusions were reached.

CCS CONCEPTS

• **Hardware~Hardware-software codesign** • *Software and its engineering~Virtual worlds software* • *Software and its engineering~Software performance* • *Software and its engineering~Application specific development environments*

KEYWORDS

Mobile devices, multiplatform development approaches, development frameworks, native mobile applications, energy consumption, energy efficiency.

ACM Reference format:

Leonardo Corbalan, Juan Fernandez, Alfonso Cuitiño, Lisandro Delia, Germán Cáseres, Pablo Thomas and Patricia Pesado. 2018. Development Frameworks for Mobile Devices: A Comparative Study about Energy Consumption. In *Proceedings of 5th IEEE/ACM International Conference on Mobile Software Engineering and Systems 2018, Gothenburg, May 2018 (MOBILESoft 2018), 11 pages.*
https://doi.org/10.1145/3197231.3197242

MOBILESoft '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5712-8/18/05...$15.00
https://doi.org/10.1145/3197231.3197242

1 INTRODUCTION

The mobile application development suggests a series of challenges, typical of this activity, that Software Engineers must deal with. One of the most important is to place the mobile application (app) conveniently in a market characterized by the presence of several platforms.

According to Joorabchi et al. [1] the difficulties related to multiplatform development are numerous and considerable, so much as to constitute a specific analysis category in the study they presented. According to data analyzed in this study, the tendency in this area is to move to fragmentation more than to unification.

Other issues that impact on software production for mobile devices are related to market aspects rather than technology aspects. When facing a development project for a mobile application, time-to-market demands must be taken into account. Therefore, execution speed and the necessity to include several

platforms tend to be two crucial factors in this activity.

To face these demands and disadvantages originated by the external fragmentation, several software engineers have decided to use multiplatform development frameworks, some of which have become very popular, such as Cordova [2] Xamarin [3] and Titanium [4], among others. With these frameworks, it is possible to generate applications for different platforms from a unique development project, instead of launching several specific projects at the same time for each platform (native development). However, to choose the most adequate multiplatform development framework as regards the needs of the case is an important decision to make. A balance must be found in order to maximize the number of reachable users without an excessive cost having in mind quality and time measurement as regards its design and development.

Despite the kind of development (native or multiplatform), there exists other difficulties related to limitations attached to the mobile nature of devices, such as the need to use batteries, the reduced size of the screens and the lack of physical keyboards. However, among all these difficulties, battery life is one of the most restrictive.

Battery development and energy-saving technology have been put back regarding the quick evolution of other aspects of mobile devices. The current trend is to integrate more applications in a unique mobile device with general use, which often turns out to a higher energy consumption and, therefore, a lower battery life. This high consumption lays out problems for the evolution of the mobile information technology, since developers cannot use all the potential of the current technology.

According to the concept above mentioned, it is not surprising that the efficient use of energy is of top priority for smartphone users. A study carried out over 170000 users' reports from the GooglePlay application store showed that the users are more prone to uninstall an application if it shows an inefficient behavior regarding energy, comparing this to other kind of inefficient behaviors [5].

It is more important to achieve an efficient use of energy than to satisfy the demands of the mobile applications users. This is related to a broader duty related to the environmental care and the general health of the planet. The highest energy consumption is against the current tendency of green computing, which tries to get more energy saving and eco-friendly computer systems. Extending the battery charge lifetime has a positive effect on the environment, since it reduces pollution from energy generation processes and well as the production or waste of the battery material.

In [6] the creation of techniques that optimize the energy consumption of mobile devices is discussed, and a certification programme is suggested to validate whether a device can be considered "Green" or not. According to the authors of this article, the creation of the green mobile certification programme will make developers consider the saving energy characteristic as a nonfunctional requirement of their systems, following the current trends of the creation of green products.

In this direction, the power-aware computing methods and principles apply in general. However, power efficiency and energy efficiency cannot be mistaken as two different goals. Power is the speed used to consume, use, transfer or transform energy (Power=Energy/time). To reduce the power does not necessarily mean to reduce the energy consumption. For example, to reduce the processor power by decreasing the clock speed could be the opposite to energy efficiency, leading to a longer execution time that could increase the total consumption of energy. Low power devices that work for a long time could consume more energy than high power devices that work during a shorter time [7].

This article focuses on the study of energy consumption of mobile applications with high processing load, audio and video playback. Even though the interest for this topic is being considered central, as it is shown by the great amount of existing papers, the articles that tackle the topic in relation to development frameworks are particularly limited. It is important, therefore, to increase the scientific production in this direction. It is crystal clear, due to the ideas shaped in this introduction, that to add knowledge about the way in which development framework affect the energy consumption of applications is enormously useful. This article makes a major contribution in this sense and it is organized as follows:

A list of related works is presented in the next section. In section 3, different approaches of mobile applications development are described and in section 4 the consumption energy problem in mobile devices area is presented. Then, in sections 5 and 6, the scope of this study and the relevant experimentation are described. In section 7, the obtained results are analyzed and showed. The rest of the article presents conclusions and further analysis.

2 RELATED WORK

The efficient use of energy has been well analyzed during the recent years. Only taking into account the IEEE *Xplore* Digital Library site, the search of articles using the expression "energy efficient", in December 2017, gave more than 21,300 results, of which more than 1,900 were published that same year.

In the field of mobile devices, many researchers have presented solutions to better the energy use. Some of them have focused on good practices of programming to decrease the energy consumption in applications. As regards [8] energy demand of different algorithms it was studied and proved that the fastest algorithms are not always those which consume less energy. In [9] and [10] recommendations for the development of applications were presented, with the goal of reducing the energy consumption. In [11] the authors of this article concluded that most of the best programming practices (5 out of 8) published by Google to optimize the performance of Android applications also impact positively on energy consumption.

Other researchers drew attention to different aspects of mobile applications and studied the way they affect the energy demand. In [12] the authors focused on the energy consumption derived from web use through mobile devices. They concluded that downloading and analyzing CSS and Javascript is a process that consumes great part of the total energy necessary to process the page, and they offered key recommendations about design-

ing web pages with the goal of minimizing the necessary energy during page processing.

In the same direction, researchers from Texas University, Austin, presented *GreenWeb,* a group of language extension to facilitate the creation of more efficient websites and it provides more flexibility and control over energy consumption to site developers [13]. According to the published results, the tests carried out with *GreenWeb* extensions observed energy saving from 29.2% to 66% with minimal QoS (Quality of Service) violations.

Other researches revealed new software tools to carry out studies about mobile devices consumption. In general, it is hard for developers to measure the energy demand that their applications produce and the way in which it varies due to conditions that are out of the developer's control, such as network congestion mobile operator choice and the user's settings for the screen brightness. Because of this necessity, in [14] an emulation tool was presented that allows mobile applications developers estimate the energy used in their applications. This tool allows developers explore multiple operation conditions that cannot be easily reproduced in their laboratories. In [15] Pathak et al. presented *eprof* tool, a fine-grained energy profiler for mobile applications to answer the question: Where is the energy spent inside my app? The authors concluded that the applications in smartphones waste the greatest amount of energy in I/O components, such as 3G, WiFi and GPS.

Opposite to Bayer's work et al. that states that the fastest algorithms are not necessarily those which consume the least [8], Corral et al. presented in [16] a series of experiments showing that in most of the proven cases, the consumed energy by a part of a code of a mobile application can be calculated due to the time that such code requires to complete its realization. Nevertheless, the authors concluded that a deeper analysis is required about the statistical reliance between these two studies parameters to generalize this idea and take the realization as a reliable sign of energy consumption.

While great part of the current research work focuses on the energetic models whose primary is energy consumption of each application, in [17] a different strategy, based on data prediction, was proposed to reduce the energy demanded by the applications,. The authors of this article concluded that saving energy is possible through a predictive model that anticipates data based on what the user wants to get in the next step and reducing, this way, time and frequency of communications.

Other possibility that has been considered, in order to reduce energy consumption in mobile devices, consists of an evolving technology called Mobile Cloud Computing. This group of techniques integrates the cloud computing concept within the mobile devices scenery. Certain limitations of these devices can be overcome with this technology, such as the need to save energy, with the processing executed in the cloud. In [18] it was shown that this technology is highly useful to save energy. In [19] a revision about several energy efficient frameworks available to offload the computation from the mobile to the cloud environment was presented.

In recent years, the Software Engineers community has showed special interest about multiplatform applications development for mobile devices, an issue also related to the current investigation work. This accounts for the great amount of scientific articles published about this matter.

In [20] Holzinger et al. the advantages of the HTML5 web applications about native applications related to the design of the user's interface for a mobile application were compared. Moreover, a group of factors that must be considered to determine which is the most appropriate development focus — web or native— was defined. These factors also included application characteristics, users target and skills of the development team.

In [21] non-functional aspects between different approaches of multiplatform applications development for mobile devices were compared. The authors set out a list of 14 criteria for the evaluation of these solutions, some from the infrastructure perspective and others from the development perspective.

In [22], Raj and Tolety classified the approaches of the multiplatform mobile applications development into four types: (1) mobile web approach, (2) hybrid approach, (3) interpreted approach and (4) crossed-compilation approach. The described each of these types along with their advantages and challenges. They also classified the applications into server data-driven, sensor/IO based, standalone and client–server, categories that must be taken into account to decide the most appropriate development approach.

In this article multiplatform development approach presented in [22] was adopted, which is also used by Xanthopoulos et al. in [23], where a detailed and comparative analysis of the advantages and disadvantages of each kind was also added. The comparison criteria included deployment, technologies to develop mobile applications, hardware and data access, user interface and look and feel perceived by the user (for example, loading time and execution speed).

In [24] the authors of this article started a line of investigation that continued in [25] and its main goal is to study the effects produced by the multiplatform software development approaches about different aspects of the mobile applications already developed. Specifically, in [24] the effects about issues from the Software Engineer's point of view were considered, and in [25] the main focus was processing time.

While the published articles about energy consumption are numerous, just like the studies about multiplatform development, the idea of considering both topics together to analyze possible relations has not been explored in depth. Some authors have approached these issues in a tangential way. Such is the case in Metri et al. that in [26] a generic study about the energetic efficiency of the mobile applications was presented. Although the study is not focused on multiplatform development, conclusions about web applications, a kind of multiplatform application according to the classification considered in the present work, were presented among the results. The authors of the article concluded that, in general, applications consume less energy than versions based on the web. They stated that these results were due to the fact that the native applications tend to use less amount of memory than their counterparts web-based.

Willocx et al. presented in [27] an extensive comparative

study about the performance of applications developed with different multiplatform development frameworks. Although important evaluation criteria were taken into account, such as CPU use, memory consumption and disk space, the analysis of battery consumption was omitted on purpose. According to these authors, the multiplatform development frameworks cause an additional battery use but the impact is insignificant compared to the overloading caused by the design/nature of the application.

In [28] general aspects of multiplatform development frameworks were analyzed and, particularly, the energy consumption was compared among different technologies to generate the user's interface by using PhoneGap (an Apache Cordova distribution). The conclusion was that PhoneGap with HTML and CSS consume less energy than PhoneGap with Sencha Touch 2.0, which, at the same time, consume less than PhoneGap + HTML + Jquery. The test cases are very specific and, therefore, it is difficult to obtain general conclusions.

The search of studies related to the main subject of the present work, carried out over the current corpus of scientific articles published, has resulted in only one case with significant coincidences. Ciman et al. analyzed in [29] the way in which the use of multiplatform development frameworks can impact on the amount of energy required for the execution of the same subject. The study was focused on two frameworks: Titanium and PhoneGap over Android devices. The same authors extended their investigation in [30] adding the analysis about the web applications, and in [31] also taking into account MoSync, a crossed-compilation framework. In all the cases, it was found that the multiplatform development impacts negatively over the energy consumption, even when the final result is a native application developed with the crossed-compilation approach.

In the present article the line of investigation initiated by the same authors is carried on in [24] and [25]. Here, the goal is to study how development frameworks for mobile devices impact on energy demand. This goal is similar to that set out by Ciman et al. In the works mentioned in the previous paragraphs; however, other aspects of the applications are analyzed here, as well other development frameworks not analyzed by that group of investigation.

3 APPROACHES TO MOBILE APPLICATION DEVELOPMENT

In recent years, the mobile device market, especially that of smartphones, has seen a remarkable growth. In particular, the operating systems that have grown the most are Android and iOS [32].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 [33].

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.

3.1 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.

3.2 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 [34].

3.3 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 [24].

Hybrid applications have two disadvantages in relation to native applications: (1) user experience suffers from not using the native components in the interface and (2) execution could be slower due to the additional load associated to the web container.

One of the most popular frameworks is Apache Cordova [2]; 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.

3.4 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. Some of the most popular interpreted development environments are Appcelerator Titanium [4] and NativeScript [35].

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

3.5 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 [3] and Corona [36].

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 logic code, but user interfaces must be programmed separately for each target platform. Thus, code reuse is affected by the characteristics of the application being developed. Statistical studies carried out by Xamarin report that code reuse 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.

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.

4 ENERGY CONSUMPTION

In recent years, mobile devices technology has experienced a fundamental progress by adding new components and advanced functionalities. These devices have increased the screen size, pixel density and computing power, matching or even outstripping desktop computers from not so long ago, but they have also increased the energy demand by reducing battery runtime.

Unfortunately, the evolution of the technology used in batteries has not matched the evolution of other components of mobile devices. The energetic efficiency has become a matter of interest as much in hardware manufacturing as in software development.

The introduction of big.LITTLE of ARM technology [37] to tackle the need of a better performance and energetic efficiency in mobile devices constitutes an example of the effort that hardware manufacturers make [5]. A further evidence for this is mobile devices equipped with an assisted GPS (A-GPS) that improves precision, performance and energetic efficiency compared to the conventional GPS. Fast charging technology is another way that many manufacturers have adopted to deal with the high demand of energy their devices require. This aims to reduce the inconvenient that entails charging the device in the middle of the day using the shortest time possible.

However, the energetic efficiency of hardware components is not enough to improve the battery charge duration. It is of upmost importance to optimize the way in which the applications use these components. Therefore, several procedures of efficient

use of energy have been developed and used at the operative system level, transparent for the developer and designed to work without damaging the performance of the applications. Despite all this, it has been proved that, in many cases, energetic efficiency of a particular application can improve substantially through changes in its source code, even in presence of such procedures of efficient use of energy.

As a result, it is crucial for application developers to be aware of the energy consumption of mobile devices when performing certain operations. It is essential to measure or estimate the energy consumed in real time by different components of a device, while they are used by an application in use. However, this still presents some difficulties given the lack of appropriate tools and technologies.

Most of mobile devices do not inform by default the total amount of energy consumed in a certain moment. To tackle this problem and measure the energetic demand, some engineers resort to external instruments, such as multimeters connected directly to the battery interface or internal circuits of the mobile device. It is then required to open the device in order to connect the measurement instrument, which is quite unadvisable and, sometimes, very difficult to perform, since, in some cases, the battery is not easily accessible. This method to measure the energy consumption, used in a laboratory space, has been used in several published research papers, such as [6] [12, 30, 38], but it is not practical for most of the developers.

It must also be considered that the results obtained with external measurement devices, connected to the battery interface, indicate the total energy consumption of the device. It is not easy to identify which are the hardware resources used by the monitored application that are contributing to the total energy consumption. Although it is possible to remove the contributions of some individual subcomponents through an extensive analysis of the measurement data, the effort requires specific knowledge of the domain, which is not always possible [39].

An alternative to measurements with multimeters connected to devices, more user friendly for developers, consists of the use of specialized software to estimate and analyze the energy consumption. The analysis about energy consumption is called energy profiling and the applications used to perform this analysis are called energy profilers. A basic function of these energy profilers is to provide information about the use of certain batteries such as the energy consumed by the entire device or discriminated by the hardware component and/or application. This methodology does not require physical connection of any measurement tool and, therefore, it is simpler the methods such as external millimeters. The estimations about the energy consumption tend to be based in power models.

A power model is a mathematical function that quantifies the factors that impact on the energy consumption, such as the use of a determined hardware subcomponent. An example of a simple power model let us think of one that estimates the consumption of the system as a function of only one variable: brightness level [39]. The Android operative system has its own energy models and profilers that estimate the energy consumption of different components and applications.

Another alternative for the analysis of energy consumption in mobile devices combines the software practicality of consumption estimation and real measurement precision. This method depends on certain services that some hardware manufacturers provide, which incorporate sensors in the internal circuits and their devices. This way, it is possible to control the energy consumption in real time of each subcomponent through an adequate profiler. An example of these can be Snapdragon from Qualcomm Technologies, Inc. processors, along with *Trepn Profiler* application [40] developed by the same company. The use of these tools have gained popularity among application developers and it has also been used in published research papers such as in [26].

5 DESCRIPTION AND STUDY SCOPE

The ambition to carry out a study worldwide in the mobile devices field is seriously hampered by the high level of internal and external fragmentation. It is necessary to narrow down the amount of possibilities to make the investigation factual. Here, the parameters of interest selection considered in this work is detailed. These decisions determined the cases of analysis, the tests design and, consequently, the scope of the study. The excluded variables here will be examined in further research studies.

5.1 Platform Selection and Test Device

The study presented in this article is focused on the analysis of the impact that different development approaches and frameworks have over the energy consumption of the Android applications. The target platform selection was due to the fact that Android is the most present operative system in the market of mobile devices. However, it is also important for further studies to focus on other platforms, such as iOS.

The differences observed in a group of preliminary tests on mobile devices of different trademarks and models were not very significant. It was decided, then, to base all the experimentation of this investigation in a medium range Motorola, Moto-G2 model, 1.2 Quad-core processor 1.2 GHz Qualcomm Snapdragon 400, GPU Adreno 305, 1GB of RAM, S.O. Android 6.0 smartphone present in the market at the time the study was carried out. This device came up as an acceptable average example of the rest that were examined in the preliminary test phase.

5.2 Selection of Approaches and Development Frameworks

Native, hybrid, interpreted and cross-compilation approaches were selected for this study. The web approach was excluded and it will be analyzed in detail in further studies.

The development frameworks selected for the tests are well known in the area. Six different scenery analyses were defined using the latest stable version of each framework: (1) Android SDK API 23 (native), (2) Apache Cordova version 7 (hybrid), (3) Appcelerator Titanium version 5 (interpreted), (4) NativeScript version 3 (interpreted), (5) Xamarin version 6 (cross-compilation) and (6) Corona version 2016 (cross-compilation). Additionally,

Android NDK revision 15c (Native Development Kit) was used [41] for one of the types of the applications studied.

5.3 Selection of the Type of Applications Studied

Three different types of applications have been selected in order to measure the energy consumption caused by the approach and the framework used for its development. These types, that identify characteristics frequently present in the applications, are: (1) intensive processing applications, (2) video playback applications and (3) audio playback applications.

6 EXPERIMENTATION

This section deals with the general details of the implementation of the tests and those particular details for each of the types of application considered.

6.1 Measurement Tools and Details of the Tests Implementation

Android is an open platform, so the measurement process can be more controlled and detailed compared to other platforms, such as iOS and Windows Phone. More control and precision is possible if profilers, developed by hardware manufacturers, use the internal sensors placed in their own components. Therefore, to measure the energy consumption in the present research study the *Trepn Profiler* from Qualcomm tool was chosen, the same company that developed the smartphone processor used in all the tests.

Trepn Profiler has been designed to measure and analyze the energy consumption of Android applications. When it is used with Mobile Development Platform (MDP) devices, based on the Snapdragon from Qualcomm processor, *Trepn Profiler* is able to show the energy consumption of the individual hardware components. [42] (See Fig. 1).

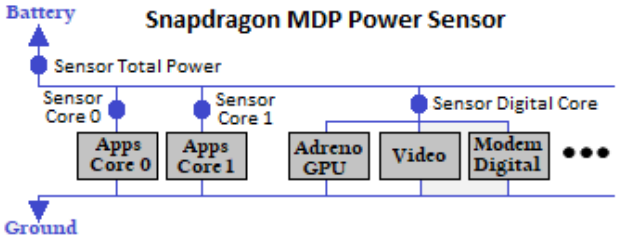

Figure 1: How Trepn Profiler Measures Power

Trepn Profiler works with three levels of granularity: system, subcomponent and application. Due to the fact that the Moto G2 device, used in the tests of this experimentation, has a Qualcomm Snapdragon 400 processor, it is possible to carry out measurements of the energy consumption at a complex level. However, as the *Trepn Profiler* granularity related to the energy consumption is due to subcomponents and not to application, this forces the researchers to be prepared to minimize the external interferences to the applications during the measurements. Therefore, some preconditions have been observed, they are:(1) set-in airplane mode, (2) screen brightness configured at a minimum level, (3) sound volume configured at 20%, (4) battery charge between 80% and 100%, (5) device not connected to battery charger, (6) application used with black and white background and (7) application running in foreground in screen during the test.

Three different applications were carried out to test the intensive use of the CPU, the video playback and the audio playback. Six versions of each of them were set up, each of them using the six development frameworks already mentioned. It was used a seventh version using the Android NDK native framework only for the tests of intensive use of CPU. Therefore, 19 cases of tests were defined. The reason why two ways of native development for Android (SDK and NDK) were considered only in the case of intensive data processing is due to the fact that both alternatives present great differences in this aspect; however, they are quite similar in the other two types of applications studied.

It is necessary to consider the random nature of the experimentation to carry out a reliable measurement of energy consumption of an application. For each of the 19 tests defined, 30 independents performances of the designed experiment were carried out, obtaining in each case an $X = X_1, X_2 \ldots X_{30}$ sample, with X_i = measure valued of the variable in the i-th execution of the given experiment.

To characterize each of the obtained samples, it has been calculated $\bar{X} = (1/n)\sum_{i=1}^{n} X_i$ and $S_X = \sqrt{(\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2)}$. They correspond to the sample average and sample standard deviation respectively.

In most of the cases of practical interest, 30 samples are enough to name $\bar{X}$ as a good approximation to the real measure of distribution. A great amount of experimental works published in the area use this number of measurements in their phase of data recovery; among them, some were quoted previously in this article, such as [10] and [11].

The variables studied were energy consumption, execution time and CPU use percentage. The sample average was calculated and the sample standard deviation of the consumed energy in mWh (milliwatt per hour), the CPU charge percentage and the execution time of the tests measured by seconds.

6.2 Intensive Data Processing Application

In order to assess the energy consumed by an application with high processing data load, a simple calculation was suggested, that included several iterations, mathematical and floating point arithmetic that is summarized in the following series:

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

The suggested experiment allows the researchers to measure accurately the variable analyzed; in this case, the energy consumption that the intensive use of the CPU involves.

This type of mathematical calculation is frequent in several applications that are carried out in mobile devices; for example, games, applications with augmented reality, application for

image treatment, among others. It is not always possible to use the capability of the Graphic processing unit (GPU) for the calculation.

Seven versions of the same application were implemented, using six selected development frameworks plus other version using Android NDK. In this last development the programming language C++ was used. Android NDK includes C and C++ modules in Android applications, and its aim is to maximize performance and reuse libraries used in those languages. Android NDK is often used in applications where a maximum performance in the images processing, graphic production, physical simulation, etc. is to be obtained [43].

The introduced algorithms NDK are binary runtime code specific for each of the architectures where Android is performed [44]. The programmer must choose which architectures can stand. This makes the code carry out directly in the processor, unlike Java code, that is compiled to bytecode to be executed by the Android RunTime (ART) virtual machine.

The source code used for the experiments carried out are found in [45].

6.3 Video Playback Application

For the experimentation with video playback, six versions of the same application were used, one for each developed framework chosen, with the functionality to reproduce a 1 minute video in full screen.

The video used was encoded in a 89.2 Mb file, with standard high definition, 1280 by 720 pixels. The simplest codec in the market was used, H.264 with bits rate of 5585 Kbps. For the codification of the audio tracks, the AAC format was used with bits rate of 128 Kbps.

6.4 Audio Playback Application

For the experimentation with audio playback, six versions of the same application were used, one for each developed framework chosen, with the functionality to reproduce a 1 minute audio file.

The audio used was encoded in a 1.32 Mb file. The most popular standard code in the market was used, MP3 AC3, with bits rate of 128 Kbps.

7 RESULTS

7.1 Applications with intensive processing

Table 1 summarizes the obtained results during the tests carried out to analyze Android applications with high computing. Each data file refers to one of the developed frameworks analyzed, organized by energy consumption, from the most efficient: Cordova, to the least efficient: Corona. The column that says "Energy" is the most relevant, since it shows the standard average and deviation of energy consumption obtained during the test performance. The other columns, "CPU load" and "Duration" complete the description of the results and refer to the CPU percentage used by the application and to the execution time that the development of the task lasted respectively.

Table 1: Application with Intensive Processing

Framework	Energy (mWh.) $\overline{E}$	S_E	CPU load (%) $\overline{C}$	S_C	Duration (s.) $\overline{T}$	S_T
Cordova	1.597	0.136	35.924	2.571	8.467	0.679
Titanium	1.692	0.096	37.480	2.395	8.355	0.643
Android NDK	1.789	0.092	32.434	1.876	9.745	0.366
NativeScript	1.792	0.176	33.357	2.217	9.109	1.789
Xamarin	3.036	0.185	32.072	1.768	17.891	0.973
Android SDK	3.463	0.149	32.468	1.332	18.568	2.938
Corona	7.304	0.189	34.315	1.102	38.877	1.492

The histograms in Figure 2 (section A) describe with more details the obtained measurements of energy consumption. Obtained frequency distributions are plotted for each of the seven test cases in the following order: best results at the top, worst results at the bottom.

From the analysis of Table 1 together with Figure 2 (section A), three groups of frameworks emerge clearly. The first, and the one with the most energy efficiency, consists of Cordova, Titanium, Native Android NDK and NativeScript. The second, with regular efficiency, consists of Xamarin and Native Android SDK. Finally, the least energy efficient group consists of Corona. Corona was also the framework that produced the highest CPU load and execution time (twice more than Android SDK, the most inefficient, except for Corona), this is due to its highest energy consumption.

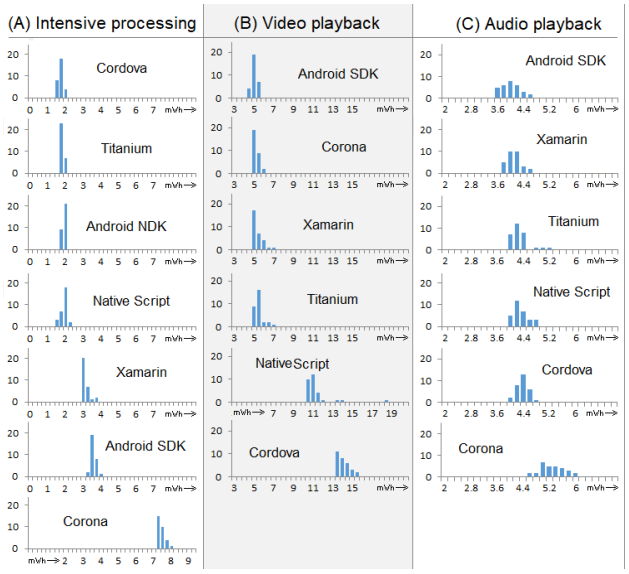

Figure 2: Histograms of samples obtained during experimentation

It is important to note that the native development in Android using SDK (Java language) cannot be found in the group with the highest energy efficiency, that consists mainly of frameworks that use JavaScript code (Cordova, Titanium and NativeScript). The native version developed in Java consumes approximately twice the energy of the versions that use JavaScript. This is due, mainly, to the use of Java mathematical func-

tions, used in an ART virtual machine and to the fact that they show great deficiencies regarding the execution time and, therefore, regarding the energy consumption. On the other hand, JavaScript interpreter is much more efficient for the computing of those mathematical functions. It can be concluded that the adequate option to develop native applications in Android with high computing is NDK that, unlike SDK, it has shown an efficient use of consumed energy.

If $\bar{T} \times \bar{C}$ (time used to carry out multiplied by the CPU percentage use) and the tested frameworks are organized by this result in an increasing way, the following result is obtained: 1st. NativeScript, 2nd. Cordova, 3rd. Titanium, 4th. Android NDK, 5th. Xamarin, 6th. Android SDK and 7th.Corona. It can be observed that, except for the position of NativeScript, the organization according to the energy consumption that is shown in Table 1 is the same. This proposes the multiplication of the time product used by the CPU loading as a good estimator for energy consumption, which turns out to be convenient, due to the fact that both parameters are simpler to obtain than the real consumption of the application.

7.2 Application of Video Playback

Table 2 summarizes the results obtained during the video playback tests. The development frameworks analyzed are organized according to energy consumption, from the most efficient, Android SDK, to the least efficient: Cordova.

In Figure 2 (section B) the histograms of the samples from the experimentation are shown. Along with Table 2 an acceptable characterization of the results is obtained. It is possible to identify two groups clearly defined. The first of them, with higher energy efficiency, consists of Android SDK, Corona, Xamarin and Titanium. The other group consists of NativeScript and Cordova. These two frameworks showed a really low energetic efficiency level, consuming twice the amount than the other tested frameworks.

Table 2: Application of video playback

Framework	Energy (mWh.)		CPU load (%)		Duration (s.)	
	$\bar{E}$	S_E	$\bar{C}$	S_C	$\bar{T}$	S_T
Android SDK	4.776	0.287	14.540	0.862	61.600	0.814
Corona	4.992	0.235	14.704	0.711	62.733	0.907
Xamarin	5.119	0.473	15.465	1.608	62.333	0.959
Titanium	5.262	0.502	15.204	1.643	63.633	1.033
NativeScript	11.112	1.590	17.839	2.210	63.333	1.295
Cordova	13.866	0.536	22.358	0.903	62.833	0.834

Although the duration of the video playback was exactly 60 seconds in all the tests, the execution time of each implementation was different (see table2). This is due to the fact that, according to the development framework used, the startup time of the application is different. The native approach shows advantages in this aspect; however, it must be considered that this difference is significant only if the time of use of the application is short, otherwise, it loses relevance. The organization of the frameworks according to result of the product $\bar{T} \times \bar{C}$ is exactly the same than the obtained using the result of the energy consumed. Therefore, the same as in the case of the application with intensive processing, the time product used by the CPU loading is a good estimator of the energy consumption of the application.

Cordova hybrid development framework stands for its inefficiency since it consumes almost three times the energy than the best option corresponding to the Android SDK native development. Probably, the sharp difference is due to Cordova use of HTML video playback that requires a higher CPU loading than the given by the operative system

7.3 Application of Audio Playback

Table 3 summarizes the results obtained during the audio playback tests. The development frameworks analyzed are organized according to energy consumption, from the most efficient —Android SDK—, to the least efficient —Corona.

Table 3 analysis reveals that there are no considerable differences regarding energy consumption among the different development approaches, except for Corona, that is the least efficient of all. The histograms of the samples collected that are shown in figure 2 (section C) support this conclusion as well. It is observed that in the case of the audio playback, the selection of the development framework has less effect compared to the other two types of the applications already analyzed.

It can also be observed here, just like with the video playback application, differences in the startup time of the application depending on the development framework used. The native approach shows, one more time, a slight advantage in this aspect.

Table 3: Application of audio playback

Framework	Energy (mWh.)		CPU load (%)		Duration (s.)	
	$\bar{E}$	S_E	$\bar{C}$	S_C	$\bar{T}$	S_T
Android SDK	3.920	0.291	10.497	0.882	64.033	0.999
Xamarin	4.010	0.201	10.592	0.613	64.967	1.098
Titanium	4.189	0.277	11.865	0.835	64.767	1.104
NativeScript	4.224	0.229	11.233	0.644	65.867	1.042
Cordova	4.288	0.191	11.473	0.487	65.733	1.388
Corona	5.194	0.387	14.680	1.080	64.800	1.031

The organization of the frameworks regarding the results of the multiplication of $\bar{T} \times \bar{C}$ is very similar to the obtained using the result of the consumed energy, being the same in their ends and pointing out the biggest difference between Corona and the rest of the frameworks. Therefore, in this case, the time used by the CPU loading is a good estimator of the energy consumption of the application as well.

7.4 Global Analysis

In figure 3, the suggested $\bar{E}$ results obtained during the realisation of the 19 test cases are shown. From a quick view, it can be seen that out of the three types of applications analyzed, only one or two development frameworks stand out due to their inefficiency (high energy consumption). Such is the case of Corona for intensive processing, Cordova and NativeScript for

video playback and Corona for audio playback. Opposite to this, the most efficient framework does not differ considerably from the other efficient frameworks. This indicates that knowing the frameworks to avoid, in each case, is extremely useful, even though the most efficient option is still to be discovered.

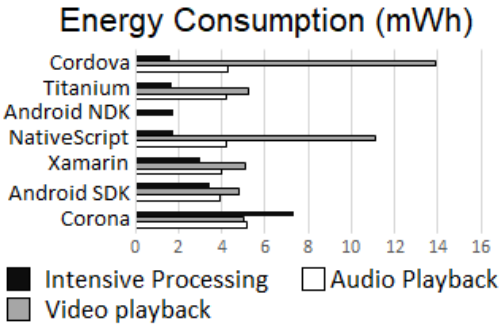

Figure 3: Energy consumption according to the development framework used for the three types of application studied

From tables 1, 2 and 3, it becomes apparent that the impact that the development framework used has on energy consumption is higher in the applications of intensive processing (the consumption of the application developed by Corona is 4.7 times the consumption of the application developed with Cordova). This is also important as regards the applications of video playback (the consumption of the application developed with Cordova is 2.9 times the consumption of the application developed with Android SDK). Finally, the least impact of the selection of the development framework is on the audio applications; in this case, the consumption of the development with Corona (the least efficient framework) is just 1.33 times the consumption of the development with Android SDK (the most efficient).

It can also be noted that, as regards these three types of applications studied, the development framework Titanium, which did not obtain the best position, always belonged to the group of the most efficient frameworks.

8 CONCLUSIONS AND FURTHER STUDY

The special features of the mobile devices field, among them a high level of fragmentation and a group of limiting characteristics, such as the need to use batteries, caused an increasing number of scientific works about multiplatform development frameworks and energy efficiency. However, few studies have addressed the relation between these two topics. This article presents a research about the impact that the native and multiplatform development approaches have on the energy consumption of the Android applications developed. This way a contribution is made to promote the development of systems that aim to reduce the software environmental impact and the information systems.

This research focused on three types of applications and seven different development frameworks. The comparative analysis determined that Titanium (interpreted multiplatform development approach) is the only development framework that results energy efficient can be obtained from the three types of applications tested. Therefore, its use is appropriate in all the cases studied in this paper.

The performance of other development frameworks was very variable regarding the type of application used, for example, Cordova proved to be the best as regards energy efficiency in an intensive processing application, but was the worst option in a video playback application.

The data analysis suggests that discovering which is the most efficient development framework to use is not as important as determining which are the least efficient in order to avoid them. This is due to the fact that in the three tested situations, the difference between the best and the followings did not turn out significant; however, great differences were found in the performance, between the worst one (or the worst frameworks) and the rest of the frameworks.

The consequences of a bad selection of the development framework are reinforced with the applications that carry out the intensive processing. In this scenario, in the development with Corona, the energy consumption is five times the one obtained when developed with Cordova. The other type of applications which is very sensitive as regards development framework is the video playback; in this case, an application developed with Cordova consumes almost three times the energy than a native Android application developed with SDK. Finally, it must be point out that in the audio playback applications the observed differences were not of upmost importance.

It is necessary to consider that the results presented here are tied to the state of the art of the development frameworks examined at the time of the production of this article and, therefore, can vary in the future according to the evolution of these frameworks.

As an indirect effect of this study, it should be pointed out that, as regards the three types of applications analysed, the result of the execution time by the CPU loading constitutes a evidence of the energy consumption of an application. This result can be used with comparative purposes, which turns out to be convenient due to the fact that the CPU loading and the execution time are easier to obtain than the actual results of energy consumption.

For further research, we suggest the incorporation of tests on IO access high loading applications, interaction with local databases, sensing, among others. Additionally it is suggested to include other development frameworks. Particularly, it is intended to address the attention to the energy consumption of web applications. This further analysis must consider the different types of nets (WiFi and mobile telephony), the distribution of the workload (form the server and/or the client) and the use of the most popular browsers of the market.

The experimentation and the analysis of the results to the iOS platform may also be of relevance for a future work.

REFERENCES

[1] Joorabchi, M. E., Mesbah, A., & Kruchten, P. (2013, October). Real challenges in

mobile app development. In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium* on (pp. 15-24). IEEE.
[2] http://cordova.apache.org [Accessed December 2017].
[3] https://xamarin.com[Accessed December 2017].
[4] http://www.appcelerator.com [Accessed December 2017].
[5] Banerjee, A., & Roychoudhury, A. (2017, May). Future of mobile software for smartphones and drones: Energy and performance. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems* (pp. 1-12). IEEE Press.
[6] Siebra, C., Costa, P., Marques, R., Santos, A. L., & Silva, F. Q. (2011, October). Towards a green mobile development and certification. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on* (pp. 288-294). IEEE.
[7] Hassan, H., & Moussa, A. S. (2014). Power Aware Computing Survey. *International Journal of Computer Applications*, 90(3).
[8] Bayer, H., & Nebel, M. (2009). Evaluating Algorithms according to their Energy Consumption. *Mathematical Theory and Computational Practice*, 48.
[9] Larsson, P. (2011). Energy-efficient software guidelines. *Intel Software Solutions Group, Tech. Rep.*
[10] Siebra, C., Costa, P., Miranda, R., Silva, F. Q., & Santos, A. (2012, December). The software perspective for energy-efficient mobile applications development. In *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia* (pp. 143-150). ACM.
[11] Cruz, L., & Abreu, R. (2017, May). Performance-based guidelines for energy efficient mobile applications. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems* (pp. 46-57). IEEE Press.
[12] Thiagarajan, N., Aggarwal, G., Nicoara, A., Boneh, D., & Singh, J. P. (2012, April). Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web* (pp. 41-50). ACM.
[13] Zhu, Y., & Reddi, V. J. (2016). GreenWeb: language extensions for energy-efficient mobile web computing. *ACM SIGPLAN Notices*, 51(6), 145-160.
[14] Mittal, R., Kansal, A., & Chandra, R. (2012, August). Empowering developers to estimate app energy consumption. In *Proceedings of the 18th annual international conference on Mobile computing and networking* (pp. 317-328). ACM.
[15] Pathak, A., Hu, Y. C., & Zhang, M. (2012, April). Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems* (pp. 29-42). ACM.
[16] Corral, L., Georgiev, A. B., Sillitti, A., & Succi, G. (2014, June). Can execution time describe accurately the energy consumption of mobile apps? an experiment in android. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software* (pp. 31-37). ACM.
[17] Xu, J., Yang, D., Wang, J., Guan, C., Reiff-Marganiec, S., & Shen, H. (2016, June). Increasing Energy Efficiency on *Smartphones through Data forecashing. In Mobile Services (MS), 2016 IEEE International Conference on* (pp. 150-155). IEEE.
[18] Kumar, K., & Lu, Y. H. (2010). Cloud computing for mobile users: Can offloading computation save energy?. *Computer*, 43(4), 51-56.
[19] Gill, Q. K., & Kaur, K. (2016). A Review on Energy Efficient Computation Offloading Frameworks for Mobile Cloud Computing.
[20] Holzinger, A., Treitler, P., & Slany, W. (2012). Making apps useable on multiple different mobile platforms: On interoperability for business application development on smartphones. Multidisciplinary research and practice for information systems, 176-189.
[21] Heitkötter, H., Hanschke, S., & Majchrzak, T. A. (2012, April). Evaluating cross-platform development approaches for mobile applications. In *International Conference on Web Information Systems and Technologies* (pp. 120-138). Springer, Berlin, Heidelberg.
[22] Raj, C. R., & Tolety, S. B. (2012, December). A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In *India Conference (INDICON)*, 2012 Annual IEEE (pp. 625-629). IEEE.
[23] Xanthopoulos, S., & Xinogalos, S. (2013, September). A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics* (pp. 213-220). ACM.
[24] Delia, L., Galdamez, N., Thomas, P., Corbalan, L., & Pesado, P. (2015, May). Multi-platform mobile application development analysis. In *Research Challenges in Information Science (RCIS)*, 2015 IEEE 9th International Conference on (pp. 181-186). IEEE.
[25] Delia, L.; Galdamez, N.; Corbalan, L.; Pesado, P.; Thomas, P.; Approaches to Mobile Application Development: Comparative Performance Analysis. *SAI Computing Conference* (SAI), 2017. IEEE, 2017. p. 652 – 659.
[26] Metri, G., Shi, W., & Brockmeyer, M. (2015, February). Energy-efficiency comparison of mobile platforms and applications: A quantitative approach. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications* (pp. 39-44). ACM.
[27] Willocx, M., Vossaert, J., & Naessens, V. (2016, May). Comparing performance parameters of mobile app development strategies. In *Mobile Software Engineering and Systems (MOBILESoft), 2016 IEEE/ACM International Conference on* (pp. 38-47). IEEE.
[28] Dalmasso, I., Datta, S. K., Bonnet, C., & Nikaein, N. (2013, July). Survey, comparison and evaluation of cross platform mobile application development tools. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International* (pp. 323-328). IEEE.
[29] Ciman, M., & Gaggi, O. (2014). Evaluating Impact of Cross-platform Frameworks in Energy Consumption of Mobile Applications. In *WEBIST* (1) (pp. 423-431).
[30] Ciman, M., & Gaggi, O. (2014, April). Measuring energy consumption of cross-platform frameworks for mobile applications. In *International Conference on Web Information Systems and Technologies* (pp. 331-346). Springer, Cham.
[31] Ciman, M., & Gaggi, O. (2016). An empirical analysis of energy consumption of cross-platform frameworks for mobile development. *Pervasive and Mobile Computing* (pp.214-230).
[32] Rösler, F., Nitze, A., & Schmietendorf, A. (2014). Towards a mobile application performance benchmark. In *International Conference on Internet and Web Applications and Services* (Vol. 9, pp. 55-59).
[33] Raj, C. R., & Tolety, S. B. (2012, December). A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In *India Conference (INDICON), 2012 Annual IEEE* (pp. 625-629). IEEE.
[34] Tracy, K. W. (2012). Mobile application development experiences on Apple's iOS and Android OS. *Ieee Potentials*, 31(4), 30-34.
[35] https://www.nativescript.org/[Accessed December 2017].
[36] https://coronalabs.com/[Accessed December 2017].
[37] big.LITTLE technology, https://developer.arm.com/technologies/big-little [Accessed December 2017].
[38] Carroll, A., & Heiser, G. (2013, July). The systems hacker's guide to the galaxy energy usage in a modern smartphone. In *Proceedings of the 4th Asia-Pacific workshop on systems* (p. 5). ACM.
[39] Hoque, M. A., Siekkinen, M., Khan, K. N., Xiao, Y., & Tarkoma, S. (2016). Modeling, profiling, and debugging the energy consumption of mobile devices. *ACM Computing Surveys (CSUR), 48* (3), 39.
[40] Qualcomm Technologies. Trepn Profiler https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler. [Accessed December 2017]
[41] Android NDK. https://developer.android.com/ndk/index.html. [Accessed December 2017].
[42] Qualcomm Technologies Technical Document, "Trepn Profiler Starter Edition User Guide", May 22, 2015.
[43] Android NDK, Getting Started with the NDK, https://developer.android.com/ndk/guides. [Accessed December 2017].
[44] Android NDK, CPUs and Architectures, https://developer.android.com/ndk/guides/arch.html. [Accessed December 2017].
[45] https://gitlab.com/iii-lidi/energy-consumption [Accessed 14 March 2018]