Mobile Application Development Approaches: A Comparative Analysis on the Use of Storage Space

Juan Fernández Sosa[1], Pablo Thomas[1], Lisandro Delía[1], Germán Cáseres[1], Leonardo Corbalán[1], Fernando Tesone[1], Alfonso Cuitiño[1], Patricia Pesado[1]

[1] Computer Science Research Institute LIDI (III-LIDI)*
School of Computer Science, National University of La Plata,
La Plata, Buenos Aires, Argentina
*Partner Center of the Scientific Research Agency of the Province of Buenos Aires (CICPBA)
{ jfernandez, pthomas, ldelia, gcaseres, corbalan, ftesone,
acuitino,ppesado}@lidi.info.unlp.edu.ar

Summary. 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. One of these restrictions is the availability of limited storage space. Therefore, the size of a mobile application affects user preference for use. In this article, we assess how the choice of a mobile application development approach affects the final size of the application; we focus our analysis on text-, audio- and video-based applications.

Keywords: Mobile devices, multi-platform mobile applications, native mobile applications, application size.

1 Introduction

Application development for mobile devices poses a number of challenges specific to this activity that were not present in traditional software development [1]. The diversity of platforms, programming languages, and development tools, as well as device heterogeneity as regards computation power, storage and battery life, are just some of the aspects that Software Engineers have to consider.

In many cases, the success of an application for mobile devices depends on its popularity. To maximize its presence in the market, it should be possible to run it on different platforms [2]. Currently, the universe of operating systems for mobile devices is led by the Android (74.24%) and iOS (20.83%) platforms [3].

In recent years, various methodologies for developing mobile applications have been proposed and studied – the native approach and several multi-platform development approaches (web, hybrid, interpreted and cross-compilation). The former requires the development of specific applications for each platform, with parallel development projects, using specific programming languages and tools for each platform. On the other hand, multi-platform approaches allow generating applications that can be run in more than one platform and produced within a single development project.

In addition to the choice of development approach, there are other difficulties to be considered in relation to the limitations posed by mobile devices. Battery life, computation power and storage space are non-functional requirements that significantly affect the decision of the end users to install or keep a mobile application in their smartphones.

In [4], the authors of this paper have analyzed the advantages and disadvantages of the multi-platform development approaches mentioned above, from the point of view of the Software Engineer. Similarly, the authors of [2] and [5] have carried out, respectively, a comparative analysis of performance and battery consumption for native developments and various multi-platform approaches.

Available storage space varies significantly among mobile devices. The operating system and pre-installed applications (also known as bloatware) take up a large portion of this space. This limits the possibilities for use [6] and, as a result, users are usually reluctant to install new apps, or they even stop using those that require a lot of space.

The size of the apps for mobile devices has increased with time. In Android, it quintupled between 2012 and 2017 [7]. In iOS, the 10 most-installed applications in the US increased their size 12 times (1,100%) between 2012 and 2017 [8]. This increase is largely due to market maturity – as time goes by, users require new functionalities for their apps.

For all these reasons, it is essential that developers consider a limited use of storage space to attract more users.

In this paper, a comparative analysis of the storage space used by mobile applications developed using the native approach and the different multi-platform approaches described in [9] is presented.

In Section 2, the different approaches for developing mobile apps are discussed and described; Section 3 details the experiments carried out to compare the storage space used by applications developed with these approaches. In Section 4, the results obtained are presented and discussed. Finally, the conclusions and future lines of work are presented.

2. Types of Applications for Mobile Devices

In recent years, the mobile device market, especially that of smartphones, has seen a remarkable growth [10]. As regards operating systems, Android and iOS are the strongest in the market. Each has its own development infrastructure. The main challenge for mobile device application developers is being able to offer solutions for all market platforms; in these cases, development costs are so high that sometimes are hard to afford [11].

An appropriate solution to this problem is creating and maintaining a single application that is compatible with all platforms. The goal of multi-platform development is maintaining a single source code for several platforms. This results in a significant reduction of effort and costs.

In the following sections, different approaches are presented for the development of applications for mobile devices:

2.1 Native Applications

Native applications are developed to be run on a specific platform, considering the type of device and the operating system and its version. The source code is compiled to obtain executable code, similar to the process used for traditional desktop applications. When the application is ready for distribution, it is published in the app store specific for each platform. These stores have an audit process to check if the application meets the requirements of the platform on which it is going to be run. Finally, the application becomes available for download by end users.

One of the characteristics of a native application is that it allows unlimited interaction with all the functions and features offered by the device (GPS, camera, accelerometer, calendar, etc). Additionally, Internet access may not be a requirement to run this type of applications. They are fast and can be run in the background, and they issue an alert when there is an event that requires user intervention.

This development approach has a high cost, since each platform requires the use of a specific programming language. Therefore, if the goal of a project is to encompass several platforms, a different application must be generated for each of them. This means that the coding, testing, maintenance and going live processes must be carried out more than once.

2.2 Web Applications

Web applications for mobiles are designed to be executed in the web browser of the device. They are developed using standard technologies such as HTML, CSS and JavaScript.

One of the advantages of this approach is that no specific component needs to be installed in the device, and no third-party approval is required before publication and distribution. Only Internet access is required. Additionally, updates are pushed directly to the device, since changes are applied on a server and enabled for immediate access by the users. In summary, they are easy and quick to implement. However, the greatest advantage of mobile web applications is that they are fully platform-independent. There is no need to adapt to a specific operating system, only a web browser is needed.

On the other hand, this approach can reduce execution speed, which can result in a somewhat less satisfactory user experience, and interfaces are more limited than those offered by native applications. Performance can also be affected due to connectivity issues, among others. Finally, some limitations could also be observed in relation to access to specific features offered by the device [12].

2.3 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 provides access to device-specific features through an API.

Hybrid applications offer great advantages, such as code reuse for the different platforms, access to device hardware, and distribution through application stores [13].

Hybrid applications have two disadvantages when compared to native applications. The first of these is that user experience suffers from not using the native

components in the interfaces. The second disadvantage is that these apps may be slower due to the additional load associated to the web container where they are run.

One of the most popular frameworks used in this approach is Apache Cordova [14].

2.4 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 non-platform dependent 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 for these applications are Appcelerator Titanium [15] and NativeScript [16].

2.5 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 [17] and Corona [18].

Xamarin allows compiling fully native applications for iOS and Android by sharing the same base code written in C#. Integrated with Microsoft Visual Studio, it also allows generating applications for Windows Phone.

Xamarin allows sharing the entire business logics code, but user interfaces must be programmed separately for each target platform.

Corona is a multi-platform framework that allows developers build both general-purpose applications and games for the main platforms. 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.

3. Experiment

The different approaches for mobile development, native and multi-platform, use different techniques to build the applications and the code, which can affect the final size of the application.

The experiments presented here are aimed at assessing the impact that choosing one development approach has on the final size of the applications.

3.1 Designing the Experiment

To carry out the experiment, development frameworks were selected based on the different approaches discussed in previous sections. Tests were designed to assess application size using the latest stable versions of these frameworks at the time of the experiments.

The frameworks used, and their versions, are:

1. Android SDK API 25 (Java, native)
2. Apache Cordova, version 7 (multi-platform, hybrid)
3. Appcelerator Titanium, version 5 (multi-platform, interpreted)
4. NativeScript, version 3 (multi-platform, interpreted)
5. Xamarin, version 6 (multi-platform, cross-compilation)
6. Corona, version 2016 (multi-platform, cross-compilation)

When assessing the use of storage space, the various functionalities offered should be analyzed since, depending on the framework development strategy, there is a chance that more libraries, modules or plug-ins are added to the generated APK. An APK, or file with .apk extension, contains an application for the Android operating system. Its name is an abbreviation of the term **A**ndroid **P**ac**K**age.

To carry out the task, three applications with different functionalities were built.

The first experiment consisted in showing the text "Hello, World!" on the screen until the user closes the application. This trivial functionality allows calculating how many bytes are added by each framework.

Experiments #2 and #3 were designed to play multimedia files (audio and video, respectively), since there is a chance that other tools or libraries are added to that end, increasing the final size of the application.

The app that plays audio uses a 1.32 MB audio file, while the app that plays video uses an 89.2 MB video file. In both cases, the file is played automatically when the app is started.

Thus, there are 18 test cases: 3 applications (text, audio and video) for each of the 6 frameworks.

In all cases, when building the application special care was taken to check that no additional files were added, since the frameworks could potentially add them.

All tests were carried out for the Android operating system compiled for version 7.1, since Android currently represents the lion's share of the global market [3].

The experiments are independent from the mobile device used, since the results considered the size of the APK generated by each framework.

The source code for all developments is available for public access [19].

4 Results obtained

For all tests, the applications were generated following the standard procedure recommended by the documentation for each framework.

Below, the results obtained for each experiment are discussed.

4.1.1 Use of Storage Space in Experiment #1

The text-based application developed using the native approach turned out to be the smallest one. Cordova, hybrid approach, produced an APK only 18% larger than that of the native approach.

The application generated by cross-compilation with Xamarin had a size of 4.08 MB (176% larger than the native approach), and the one generated with Corona was 6.51 MB (340% larger than the native one).

Finally, the applications created with interpreted approach frameworks were the largest of the lot.

Table 1 shows the end sizes of the developed applications in ascending order of storage size used by each.

Framework	Development approach	Size in MB
Android with Java	Native	1.48
Cordova	Hybrid	1.74
Xamarin	Generated by cross-compilation	4.08
Corona	Generated by cross-compilation	6.51
Titanium	Interpreted	8.54
NativeScript	Interpreted	12.49

Table 1. Application sizes; Experiment #1

4.1.2 Use of Storage Space in Experiment #2

Audio-playing applications were generated following the standard procedure recommended by the documentation for each framework.

Table 2 displays the results obtained. The third column on the table details the total size of each of the generated applications. The fourth column shows the difference between total size and the size of the audio file, since this file has to be packed together with the code for each application. This allows seeing more clearly the size added by each framework.

For this experiment, the native approach was the best in terms of final application size. The APK file generated with the hybrid approach is just 32% larger than that generated with the native approach.

This difference increases with cross-compilation frameworks. The application generated with Xamarin is 196% larger than the natively generated application. The application produced by Corona was 388% larger than the one created using the native approach.

As in the previous experiment, the largest applications in MB are those produced with the frameworks used for generating interpreted applications. In this case, Titanium and NativeScript generated applications that were 564% and 1430% larger than those produced with the native approach.

Framework	Development approach	Size in MB	Size excluding audio file
Android with Java	Native	2.7	1.38
Cordova	Hybrid	3.14	1.82
Xamarin	Generated by cross-compilation	5.4	4.08
Corona	Generated by cross-compilation	8.05	6.73
Titanium	Interpreted	10.48	9.16
NativeScript	Interpreted	22.43	21.11

Table 2. Application sizes; Experiment #2

4.1.3 Use of Storage Space in Experiment #3

In the case of the applications that played video, the procedures recommended by their corresponding documentation were also followed.

Table 3 displays the results obtained. The same as in Table 2, the third column shows the total size of each application, and the fourth column displays the difference between total size and the size of the video file used. For result analysis, the last column was used.

In this experiment, application sizes followed the same order as that observed in experiments #1 and #2.

The native approach was once again the option that resulted in the smallest APK. The second place was taken by the hybrid approach, with Cordova. It was followed by cross-compilation frameworks, with Corona and Xamarin, respectively.

Lastly, the frameworks that resulted in the largest applications were those based on the interpreted approach.

Framework	Development approach	Size in MB	Size excluding video file
Android with Java	Native	90.24	1.04
Cordova	Hybrid	91.97	2.77
Xamarin	Generated by cross-compilation	94.21	5.01
Corona	Generated by cross-compilation	95.78	6.58
Titanium	Interpreted	98.43	9.23
NativeScript	Interpreted	101.67	12.47

Table 3. Application sizes; Experiment #3

4.1.4 General Result Analysis

Figure 1 displays the results for each of the 18 applications built, sorted by application final size.

As it can be seen, the order of the results obtained is the same in all three experiments.

The difference between the best and the worst framework is percentually significant in experiments #1 and #2, not so much in experiment #3. This is due, possibly, to the fact that the greatest impact of the framework on application size occurs when packing the basic components of the app. That is, the libraries, interpreters and other resources that the framework needs to include within the generated application for it to work.

A native application will be run directly by the underlying operating system, which means that it does not need to pack an interpreter or other tools within the application.

A hybrid application developed with web tools can be designed to be run on the web browser provided by the operating system, or it could include its own browser within the application, which would affect application size. A similar situation occurs with interpreted applications, which are run partly by the operating

system and partly by an interpreter. This interpreter can be either internal or provided by the platform.

Thus, the applications created with the native and hybrid approaches ended up being the smallest ones. The second place was taken by cross-compilation frameworks. Finally, interpreted frameworks generated the largest applications; this is due to the fact that both NativeScript and Titanium embed a JavaScript interpreter in the application.

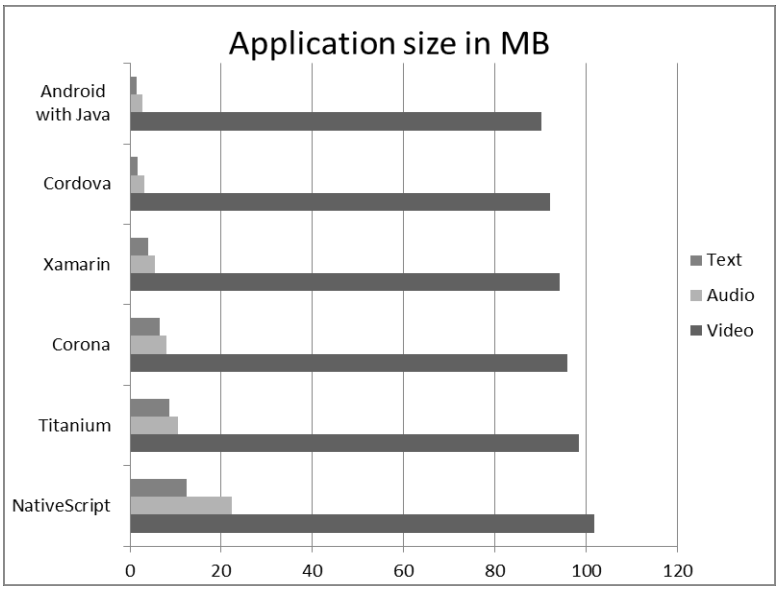

Figure 1. Summary of storage space measurements

5 Conclusions

In mobile application development, non-functional requirements are more relevant due to the restrictions imposed by devices: battery life, connectivity, limited development times, and significant fragmentation among the different platforms.

On the other hand, several mobile application development approaches are available. In this article, the native, interpreted, hybrid and cross-compilation approaches were analyzed.

A comparative study of the storage space used by applications for mobile devices, generated using different development approaches, was presented.

Three experiments were carried out – the first one consisted in displaying a text message on the screen, while the other two were designed to play multimedia files (audio and video, respectively).

It should be noted that the results presented in this paper are linked to the development framework versions used for the experiments and, therefore, could change in the future as these frameworks evolve.

The Android SDK framework corresponding to the native approach and the Apache Cordova framework corresponding to the hybrid approach yielded the best results as regards storage space usage in mobile devices.

Cross-compilation frameworks came in second.

The worst results were yielded by the frameworks that used the interpreted approach, since they generated the largest applications. This is mainly due to the fact that this development approach translates part of the code to native code and interprets the rest at runtime, which means that an interpreter has to be embedded within the application package.

In short, the experiments carried out help find out which framework would be more convenient, if size is a priority, for developing mobile applications similar to those presented in this article.

6 Future Work

In the future, we plan on expanding the number of frameworks tested, as well as considering any new versions of the frameworks used here.
On the other hand, iOS application size will also be studied, since iOS is the second most widely used operating system in the market.

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. L. Delía, N. Galdamez, L. Corbalan, P. Pesado and P. Thomas, "Approaches to mobile application development: Comparative performance analysis," *2017 Computing Conference*, London, 2017.
3. http://gs.statcounter.com/os-market-share/mobile/worldwide [Last access: May 2018]
4. Delia, L.; Galdamez, N.; Thomas, P.; Corbalan, L.; Pesado, P., Multiplatform mobile application development analysis, Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on, Athens, Greece, 2015.
5. Corbalan L.; Fernandez Sosa J.; Cuitiño A.; Delia L.; Caseres G.; Thomas P.; Pesado P., Development Frameworks for Mobile Devices: A Comparative Study about Energy Consumption (ICSE), MobileSoft 2018 5th IEEE/ACM International Conference on Mobile Software Engineering and Systems on, Gothenburg Sweden, 2018.
6. K. Vandenbroucke, D. Ferreira, J. Goncalves, V. Kostakos y K. D. Moor, «Mobile cloud storage: a contextual experience,» Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services (MobileHCI '14), pp. 101-110, 2014.
7. S. Tolomei, «Shrinking APKs, growing installs,» November 20, 2017. [Online]. Available: https://medium.com/googleplaydev/shrinking-apks-growing-installs-5d3fcba23ce2. [Last access: May 2018].

8. https://sensortower.com/blog/ios-app-size-growth [Last access: May 2018].
9. Spyros Xanthopoulos, Stelios Xinogalos, A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications, BCI' 2013, Greece, 2013.
10. http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-201403-201803 [Last access: May 2018].
11. 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
12. Tracy, K. W. (2012). Mobile application development experiences on Apple's iOS and Android OS. *Ieee Potentials*, 31(4), 30-34.
13. 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.
14. http://cordova.apache.org [Last access: May 2018].
15. http://www.appcelerator.com [Last access: May 2018].
16. https://www.nativescript.org/ [Last access: May 2018].
17. https://xamarin.com [Last access: May 2018].
18. https://coronalabs.com/ [Last access: May 2018].
19. https://gitlab.com/iii-lidi/papers/apps-size.git