Scenarios to use React Native, native code or integrate both technologies

Escenarios para utilizar React Native, código nativo o integrar las dos tecnologías

C.F. Valarezo¹  R. Triviño²

¹ThoughtWorks, Quito, Ecuador
E-mail: cvalarezo@thoughtworks.com, carlos.valarezo@gmail.com
²Universidad de las Fuerzas Armadas, Sangolquí, Ecuador
E-mail: rdtrivino@espe.edu.ec

Abstract
Over the last years, different techniques and frameworks have risen to provide a solution for creating a cross-platform application. The goal is to develop only one source code that can be deployed to different operating systems and provide a native experience. Since 2015, React Native has been promoted as an option to build mobile applications to write the code once and deploy it to any platform seamlessly. In the current time, there is not other library that promotes native mobile development using JavaScript. This work makes an experimental analysis with an unbiased vision, based on the application development experience using React Native and native code of several mobile applications. It analyses different scenarios to consider a technology or framework to build a mobile application. Conclusions state that the election of native code, React Native, or a mix of both relay on time, budget, the experience of the development team, and the features of the application.

Index terms-- React Native; iOS; Android; hybrid mobile apps; mobile applications.

Resumen
En los últimos años, diferentes técnicas y frameworks han surgido con el fin de proveer una solución para crear aplicaciones cross-platform. El objetivo es desarrollar un solo código que puede ser desplegado en diferentes sistemas operativos y proveer una experiencia completamente nativa. Desde 2015, React Native ha sido promocionado como una opción para desarrollar aplicaciones móviles con el objetivo de escribir el código una sola vez y desplegarlo en cualquier plataforma perfectamente. Al momento no hay otra librería JavaScript que permita el desarrollo de aplicaciones móviles nativas. Este trabajo realiza un análisis experimental con una visión imparcial, basada en la experiencia del desarrollo de aplicaciones móviles utilizando React Native y código nativo. Analiza varios escenarios para elegir analíticamente código nativo, React Native o una combinación de ambos para desarrollar una aplicación móvil. Algunas conclusiones encontradas sostienen que la elección de la tecnología depende de la naturaleza del proyecto, tiempo, presupuesto, experiencia del equipo de desarrollo y los requerimientos de la aplicación.

Palabras clave-- React Native; iOS; Android; hybrid mobile apps; aplicaciones móviles.
1. INTRODUCTION

Creating a mobile application often requires the developers to create one for Android and one for iOS, the two leading operating systems for mobile devices. The two applications may have the same layout and logic but several components of the user interface (UI) differ and the applications themselves need to be developed in two different languages. This development process is gruesome since it is time consuming to create two applications and it requires two different sets of knowledge. There have been attempts to create techniques, services or frameworks in order to solve this problem, but sometimes these hybrids have not been able to provide a native feeling of the resulting applications [1].

In the last years, different techniques and frameworks have risen in order to provide a solution for this. The goal is to only develop one service which can either be deployed to different operating systems or provide a homogeneous and native feeling. However, even though the development cost of hybrid applications are significantly lower than natives, the hybrids cannot provide the same native user experience and therefore they have not been successful in replacing native development. [2]

React, sometimes referred to as React.js, is a JavaScript framework developed by Facebook and released as open source in 2013 in order to aid the development community to build interfaces. Facebook were in need of a framework that could solve their problem with complex user interfaces which had data that changed over time. Tom Occhino, who is an engineer at Facebook, said “React wraps an imperative API with a declarative one. React’s real power lies in how it makes you to write code” [3]. A declarative programming style describes what to do but not how it should be done, resulting in less code whilst the imperative programming style however describes how to do it. [4]

At the React.js conference in 2015 Facebook introduced their new framework React Native, a framework they thought would revolutionise the way mobile applications are created. When React Native was released, there was only support for iOS but since then the support for Android has been added and is still expanding. Facebook have started to become more open source and is the approach they have chosen for React Native. Even though the source is not completely open yet, Facebook attempt to achieve this and contemplates that the community will contribute to improve the framework [6].

The main purpose of React Native is simple, a developer should not require the knowledge or need to spend superfluous time in order to create a mobile application since at least two applications need to be developed in order to support both iOS and Android. Since different platforms have different looks, feels, and capabilities, there cannot be an application which is homogeneous on all operating systems. However, since it is the graphical interface that differs, the development could base on the same language but have the graphics be rendered differently depending on the targeted platform and be real native components.

Facebook call this approach "learn once, write anywhere" which describes what React Native is all about. According to Facebook official documentation, developing a React Native application consists on building a real mobile application but writing its code with JavaScript rather than Swift or Java [6]. Additionally, NPM Inc [7] offers a wealth of libraries to do different tasks that even include interactions with hardware. Likewise, in case the project demands it, there is official documentation that affirms that it is possible to include native modules in it [8].

Therefore, the objective of this work is to make an empirical analysis of how React native or Native code should be used in key functional requirements for any project like budget, time, knowledge, user experience, hardware involved, testing, documentation and others, versions and releases of SDKs, operating systems and libraries. There were not considered other JavaScript libraries since React Native is the only library to develop native mobile applications and integrate with native languages. The rest of the paper is organized as follows. Section II describes when to use React native. Section III presents guidelines of when to use native code. Section IV presents aspect when to use React Native plus native code and Section V concludes the paper with a summary of the advances, challenges and problems.

2. WHEN TO USE REACT NATIVE

One of the most remarkable mobile projects already developed and delivered by ThoughtWorks Quito included a bridge between React Native and native code. The breaking feature on that hybrid application included: to display a video on the screen of the mobile device whilst the camera was recording audio and video from the person who was using the application. After some spikes with JavaScript libraries that record audio and video, the bridge between React Native and native code became the only option to continue the project, since none of those JavaScript libraries allowed that particular behaviour. That moment was crucial for the team as there were two options, either start over the project with native code, after three months of development, or bridging React Native with native code to avoid start the project again from the beginning. Eventually, the bridge worked on both platforms (iOS and Android) with a seamless integration of JavaScript libraries, Swift & Java classes, events of the respective operating system and the involved hardware (camera, storage and microphone).

The languages Swift and Java were considered respectively for each platform due to the support and documentation for mobile development. After that particular experience the analysis, based on practical work, to choose carefully on native code, React Native or the bridge between, relies on the following criterion:
2.1 Budget
In every project, budget is a primarily point to analyze [9]. Thus, consider React Native if the budget is limited, and the application has to be deployed to different platforms.

2.2 Time
Take into consideration React Native if the application has to be quickly developed and if its requirements do not change drastically.

2.3 Knowledge and experience
Consider this option if the development team knows about or has had previous experience with React [10], JavaScript and CSS. Having knowledge on JavaScript might be useful if the application demands a backend. In fact, it might be developed using the same language. This is a feature that most programming languages do not offer [6].

2.4 User experience
When it comes to mobile applications, user experience is usually important. However, if the experience is not an important feature for the user or the project, think of React Native considering the following: each mobile platform (iOS and Android) render components with particular styles. For instance, datePicker [8] in iOS is some sort of carousel, and in Android, it is a popup that shows a calendar.

Despite the fact that React Native allows to check the platform where the application is running, not every native component has a pair in JavaScript. The rendered components might have different behaviours and colours than the native component. In this scenario, the same component with the same behaviour will be on both platforms giving a clue to the user that it is not a real native experience. For example, the iOS datePicker rendered in Android or vice versa.

2.5 Freedom on writing code
Even though Javascript is a language used to develop any type of application, the limitation with React Native appears when there is no JavaScript library to perform a particular activity [7]. Also, limitations could occur when the current JavaScript libraries implement functionalities partially or not as the way the project requires. Moreover, it could even be hard to implement that particular feature using interaction between React Native plus Native code.

2.6 Features on the project
Consider React Native if the project consists of interactions with APIs where requests and responses are mostly about text and images.

2.7 Hardware
An important point to consider before deciding on React Native is if the application may or not interact with specialized hardware like a camera, microphone, gyroscope, or other piece of hardware. Moreover, in spite of the fact that there are JavaScript libraries that implement hardware interactions [7], usually people do not consider thoroughly what it takes to accomplish a complete interaction between the software and the hardware. In fact, there is a likelihood to have side effects at some point in the project. An example of a complex interaction with hardware is, overlapping the mirroring camera with a video whilst the device is recording audio and video. There is no JavaScript library to accomplish this task [7]. Therefore, in this scenario, native code becomes the only option.

2.8 Testing
Since its beginning, React Native has presented difficulties to test components. In order to solve this inconvenience, create stateless components that call functions in a different file. These functions could be developed using the Test-Driven Development (TDD) technique whereas the tests for components would rely on functional tests.

2.9 Documentation
JavaScript is by far one of the top programming languages chosen by the IT community. Therefore, the documentation available is wide.

2.10 Interface Development Environments (IDEs) and hot reloading
Since the language used will eventually be JavaScript, any IDE with JavaScript and JSX support will be appropriate. Additionally, hot reloading offers a quick way to check the changes made on the application’s code immediately. However, when this tool behaves strangely, everything must start all over again.

2.11 Lifecycle of the application
Go to React Native if the application under development is going to be alive for a short period of time, when new features will not be added, and maintenance will be limited. Similarly, consider React Native if the application under development is a demo. Nonetheless, if the scenario involves continuous maintenance, take into account that JavaScript libraries and frameworks are continuously updated. Hence, the application’s code might be affected by some of these changes. Furthermore, JavaScript libraries are normally subject to vulnerabilities.

2.12 Prototypes
React Native is an attractive option if the application is a prototype.

2.13 Physics
JavaScript manages physics fashionably in some web applications. In fact, there are specialized libraries to achieve the desired results. However, for mobile projects, choose React Native as long as the physics do not involve high processor activity, or require a lot of memory to render heavy and sudden changes on graphics. Nevertheless, on scenarios that demand high memory and processor activity, native applications have specialized SDKs to manage them. Again, on this scenario, consider preferably native code.

2.14 Speed of development
This also depends on the skills and capabilities of the developers. However, the React Native boilerplate to create a basic application is easy to follow and understand.
2.15 Devices and Operating Systems (OS) versions
Some JavaScript libraries might not work properly on the newest devices and OS releases. Therefore, choose a specific range of devices to support the application on iOS and Android.

3. WHEN TO USE NATIVE CODE

3.1 Budget
Developing a mobile application normally demands at least two teams, one for each platform. In some scenarios, development may require two or more teams, which might result in more money investment. Only if budget is not an issue for the project, consider native code.

3.2 Time
If budget is already on the list of checked items, be sure that time is not critical for the project. Having a mobile application on two platforms means developing the same product twice.

3.3 Knowledge of tools
In the scenario where the developers are fully stacked and have knowledge of at least one frontend and one backend language, the project will progress at a good pace. If the scenario is the opposite, the developers should do several spikes, and learn those new languages and tools.

3.4 User experience
Consider native code if the application demands a strong user experience. Take Apple and Google as an example of this criterion. Both have developed their own tools considering that developers need to build applications that provide the best experience possible on the respective platform.

3.5 Freedom
Since developers have at their hands the whole ecosystem of iOS and Android, they could build any feature the project demands.

3.6 Nature of the project
Consider a native approach if the application is going to deal with specialized hardware, specific SDKs, implementations like augmented reality, or machine learning.
Native tools provide a complete set of features to develop a real augmented reality and other advanced mobile experiences.

3.7 Testing
In either mobile platform there is a specialized framework already developed to write code using TDD. Likewise, there is a variety of tools available to perform any other type of testing, for instance” integration test. Therefore, developing a code with coverage could not be a problem.

3.8 Documentation
Since both platforms belong to software commercial houses, they provide official documentation and resources to deal with common problems. Also, there is an extensive community to provide support for both platforms.

3.9 Lifecycle of the application
Consider writing a native code if the application might be alive for a long time, or if it is the core of a particular business. In other words, when the application has to be maintainable, extensible, and new features could be added or improved periodically. Regarding languages and tools updates, the same official IDEs provide support to upgrade the project’s code.

3.10 Prototypes
Unless the project demands to prototype some special requirements, do not spend too much time on it if the same result can be achieved with React Native.

3.11 Physics
It involves games and simulations. Additionally, it is tied with the user experience. Consider native code for the physics on mobile platforms, especially on projects that demand high processor and memory activity.

3.12 Speed of development
As explained in the previous section, it depends on the skills and capabilities of the development team.

3.13 Devices and OS versions
An application developed with native code will run on most (or even all) of the devices already fabricated. Additionally, new OS or SDKs releases (or patches) will not affect radically the application’s performance or compatibility on the newest devices.

4. WHEN TO USE REACT NATIVE BRIDGED WITH NATIVE CODE

4.1 Unexpected requirements
It could happen that at first, based on the requirements, React Native is the best option. However, new features involve native implementations. In these scenarios bridge both technologies.

4.2 Dealing with hardware
JavaScript libraries do not always implement interactions with hardware correctly.

4.3 JavaScript libraries
If the JavaScript libraries do not provide a native feature implementation, bridge both technologies.

4.4 Dealing with specialised SDKs
It is tied to the lack of JavaScript libraries. However, even if the JavaScript library exists, it does not always implement the feature as the application demands. For example, Siri for iOS, machine learning, and others.

5. RESULTS
In general terms, using either React Native, native code or a bridge in between of both depends of the criterion already described and even other criterion that might appear over the project like new operating systems. The following table shows a summary of the already described criterion and the recommend scenario to be used. The criteria were chosen based on the experience developing several mobile projects.
6. CONCLUSIONS

This work has analysed the scenarios to use React Native, native code or a bridge between both. After developing applications with maps, hardware, microphone, specialized SDKs and other resources of mobile devices, as conclusion the use of one of opinions depends of the type of project and their criterions. Think over the inception of the project or discussing with the stakeholders, what are the possible future directions that the application under development will take in the future, before deciding the technology or the tools to be used. Therefore, it is recommended not to decide on using React Native based only on the fact that some JavaScript libraries state they accomplish the desired outcome.

Develop spikes before the project starts with the current features and the possible changes using React Native, native code and the bridge in between. This exercise will alleviate any rush and save time in the future to decide the correct technology. On the other hand, as a recommendation, list the key features that might involve hardware, SDKs to start the spikes first with.

7. ACKNOWLEDGES

Thanks to ThoughtWorks Ecuador, to provide all the support on writing this work. Likewise, an acknowledge to SENESCYT, the institution that provided a scholarship to study my Masters degree at The University of Manchester (Carlos Fernando Valarezo Loaiza).

8. REFERENCES

