
# ESCUELA SUPERIOR DE INGENIERÍA

## GRADO EN INGENIERÍA INFORMÁTICA

Cross-platform development frameworks for the development of hybrid mobile applications: Implementations and comparative analysis

Manuel Rodríguez-Sánchez Guerra

September 10, 2018


ESCUELA SUPERIOR DE INGENIERÍA

GRADO EN INGENIERÍA EN INFORMÁTICA

Cross-platform development frameworks for the development of hybrid mobile applications: Implementations and comparative analysis


- Department: Lenguajes y Sistemas informáticos
- Director of the project: Juan Manuel Dodero Beardo
- Co-Director of the project: David Díez Cebollero
- Author of the project: Manuel Rodríguez-Sánchez Guerra


Cádiz, September 10, 2018

Fdo: Manuel Rodríguez-Sánchez Guerra

## *Acknowledgements*

*I'm extremely grateful to Juan Manuel Dodero for teaching me so much and helping me during the course of this study, to José Perez for giving me the opportunity to face this wonderful study and believe in me during the course of it.*

*Many thanks to Manuel Palomo for teaching me what was the way to go and helping me to decide.*

*Special thanks to my parents, for supporting me in long work sessions and being a fundamental pillar in my life, to my friends Antonio, Emi, Javi, Jaime, Juanma and Pablo for believing in me even when I did not.*

*Special thanks to Intelygenz for making this study possible.*

# License information

# Abstract

As the number of mobile applications grows exponentially, new alternatives to the traditional native developments are presented. These are the cross-platform developments, developments that allow having a common code base to deploy the application on multiple platforms.

This study aims to dispel doubts when choosing a cross-platform development. For this, the most used cross-platform frameworks in the software development industry (React Native, Ionic, Flutter and Weex) have been compared in terms of execution times and code quality, offering an objective comparison between them. This comparison has been made through the development of four benchmark applications, one in each of the frameworks and the measurement of metrics in them for the subsequent contrast of the data obtained.

After this, it has been concluded that, despite the difference in performance, cross-platform development is an alternative to bear in mind when making a mobile development due to the great advantages it offers in versatility and costs .

## Keywords

Cross-Platform, Hybrid, React Native, Ionic, Flutter, Weex, Mobile, Execution times, Code Quality, Javascript, Dart.

# Contents

# List of Figures

# Part I

# Prolegomenon

# Chapter 1

# Introduction

## 1.1 Motivation

Along the last years the inclination in the world of software development towards mobile applications has become increasingly evident. After all, most users access internet through their mobile devices. This fact, combined with the growing popularity of web applications and web programming languages such as JavaScript, it has meant that more software development teams are increasingly considering the cross-platform development alternative. That is, instead of segmenting the development in the two main mobile operating systems (iOS and Android), why not deploying the same code on both platforms and save the cost of carrying out two separate developments?

Based on this idea, several libraries and cross-platform frameworks have been emerging. These tools support us to deploy our application in both platforms, however, many development teams still use the native programming languages of each platform for carrying out their projects. This is because the performance of the application in a platform is greater if we use the native programming language of that platform.

The aim of this document is carrying out an study of the most popular Hybrid and cross-platform frameworks in each platform and compare them according to the obtained data.

This document is the outcome of a collaboration project between University of Cádiz and Intelygenz, a software consultancy specialized in agile development. This project came up in October 2017 when Intelygenz was interested in a study with these characteristics.

## 1.2 Goals of the Thesis and Scope

The main target of this thesis is to compare different frameworks used for mobile app development by applying software metrics and measuring performance in products that have used those same frameworks.

The required steps to accomplish this goal are as follows:

1. Development of Benchmark applications making use of each one of the frameworks

   Benchmark applications with identical functionalities will be developed, in order that we can compare the result of applying different metrics to the applications.

2. Deploy these applications using quality inspection tools and use its default metrics

   Code quality analysis tools will be used to take metrics about the size, complexity and other aspects of each application.

3. Use of profiling tools to take measures about the operation of the application.

   Profiling tools will be used to analyze the performance of each application. We will apply different metrics including CPU and memory use.

4. Gather and analyze the results of the measurements.

   Once the application have been analyzed, we will gather the results of the measurement and metrics and we will conduct an study based on that data with a view of selecting the most efficient framework for mobile development.

5. Summarize the result of the study.

## 1.3 Methodology

For this thesis the following approach was adopted. First, we made a selection of the most used hybrid/cross-platform frameworks in the business. Several companies were asked in order to get information about which frameworks do they use to build mobile software products. With the same purpose a review of several articles was made in order to have an insight about which are the most used hybrid/cross-platform frameworks. In addition, We have inspected the GitHub topics related with those frameworks and compared the number of repositories. Applying this metric gave us the following output and in consequence the frameworks that are going to be studied.

1. **React Native**

2. **Ionic**
3. **Weex**
4. **Flutter**

Then, having selected the frameworks to analyze we will develop benchmark application using each one of these tools. The measures will be carried out over these applications.

The aim of the study is to compare the most commonly used cross-platform development tools in the software development industry. In order to carry out this comparison, metrics regarding quality of code, such as the number of lines; performance and battery life among others will be taken.

To measure code quality metrics output, the applications will be deployed on code quality inspection tools such as SonarQube or Codacy.

For measuring the rest of the metrics output, We will use Appium.io automation library and various profiling tools. Appium.io allow us to automate functional test on the developed applications. While the functional tests are automated, the profiling tool will be running to record calls to functions, number of objects created and execution time among other data.

Once we have collected data for each one of the applications, we will carry out the comparison of the obtained data. After this, we will draw the final conclusions about each one of the frameworks.

The study has been planned iteratively, that is, we will start developing, testing and measuring metric outputs on one framework, once we have finished with it, we will start the same process with the rest of the frameworks one by one. Thus we will obtain the result of the metrics progressively, so we will only draw final conclusions once all the applications are finished.

# Chapter 2

# State of the art

Having discussed the motivation involved in carrying out this study, we shall now discuss the current state of mobile applications development and hybrid and cross-platform development technologies.

## 2.1 Mobile Applications

Since the launch of the App Store by Apple and the subsequent Google Play Store by Google, we have seen how software development turned towards mobile application (apps) development. For instance, in the last 9 years the number of published applications in the App Store has increased from 800 to 2 200 000 published applications as can be seen in figure 2.1.

### 2.1.1 Disadvantages of native development

Throughout this document we will reference the two main mobile development platforms, Android and iOS. Each one has its own development environment and language for the development of native applications. These native applications are developed directly atop the services provided by the platform. These services are exposed through an API [19].

The fact that the code written to develop an application in one platform can not be reused in another platform raises the problem of splitting the development. In other words, a development team that wants to develop on mobile platforms must either focus on a specific platform and develop exclusively for this one, leaving out a high percentage of users, or carry out two simultaneous developments with their corresponding development cost. This implies (potentially) more development time, higher testing and maintenance costs and lower portability [19].

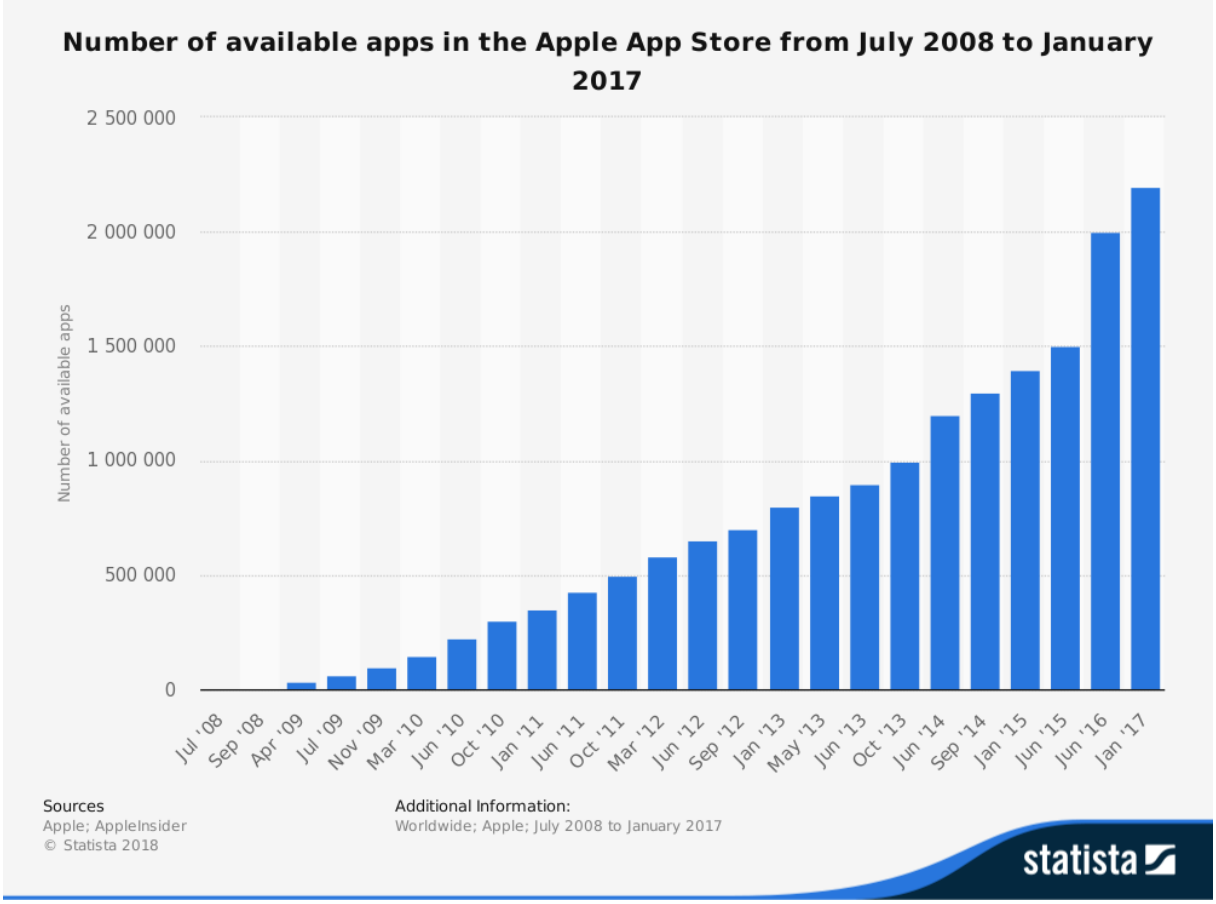


Figure 2.1: Number of available apps in the Apple App Store. [2]

## 2.2 Cross-platform Applications

Cross-platform applications are proposed as a solution for the previously presented problematic. These applications are developed using web development technologies such as HTML, CSS, JavaScript or Dart instead of being developed in the native language of each platform.

### 2.2.1 Advantages

This development approach has many advantages, the main of them is that it ends the problem of splitting the development. Using the same code we can deploy the application in any of the main mobile development platforms, therefore, there is no need to maintain two parallel developments, saving development, testing and maintenance time and increasing portability.

A study carried out by [20] after analyzing 11,917 applications published in the Google Play Store concluded that in those applications where a cross-platform development approach was used, developers made great use of code reuse. This is due to the fact that in contrast with the native applications, using a cross-platform development approach the base code of the application is shared between the different development platforms, allowing us to reuse code from one platform to another.

These applications, being web-based allow us to use third party libraries that are not specifically designed for mobile development. In the previously mentioned study [20] it was concluded that 16 of the 20 Third Party libraries used in the development of the applications that had adopted a cross-platform approach in the total of the analyzed ones, were not specifically designed to be used in mobile but in desktop browsers. The most used were jQuery, jQuery Mobile and Json2.

### 2.2.2 Disadvantages

We will now proceed to document the drawbacks presented by the cross-platform approach.

One of the biggest disadvantages of cross-platform applications is their lack of performance compared to native ones. Some studies [21] suggest that the cause of this may be due to the fact that languages such as JavaScript are interpreted and not compiled as in the case of the native languages of each platform. However, a study carried out by [20] shows that although in 7 out of 8 cases the cross-platform implementation was slower than the native one for general purpose applications this was not seen as a major drawback.

Due to cross-platform applications need an API that bridges the service requests from the web code on which they are based to the corresponding platform on which they are being executed, they can not interact with the system at low level, as can be done with the use of native languages. A study carried out by [20] indicates that the categories of the Google Play Store (more specifically photography, Music and audio, Tools, Games and Personalization) that have less cross-platform applications were those that require a deeper interaction with the system or with

the hardware on which they were running.

The user experience provided by an hybrid app is similar in all the platforms, these can be seen as an advantage, but as it is said in [20] the developers must consider how the application is integrated into the platform on which it runs.

## 2.3 Industry Impact

Nowadays, native development applications continue to dominate the mobile software development market. This is because, as we mentioned previously, the use of web technologies negatively affects the usability and performance of the applications in some cases.

This study is very aware of the current state of the industry, so, before starting it, we asked representatives of the technological area of four different companies about the impact that mobile applications have on their businesses and whether they would consider implementing cross-platform alternatives. The companies consulted were Inditex, Prisa, Bankinter and Sngular.

We have mentioned the exponential growth that the software market is having in mobile devices in recent years. To make sure of this, we ask what is the impact of mobile applications in each of the businesses.

All agree that mobile applications are one of the most relevant fields of the business today. Without going any further, Carlos Tauroni of Bankinter, states that currently over 60% of Bankinter's access to online banking is done through mobile terminals. David García Rafols from Prisa responds that access to content through the mobile application constitutes 1% of its global traffic of users, but more than 10% in consumer content traffic, therefore they are their most loyal and valuable users .

We asked if they saw cross-platform technologies as a relevant alternative for their businesses.

They all agreed that, given their current relevance, a native approach seemed more appropriate than a cross-platform one. But in the future they could become relevant if they managed to equal the native ones in user experience.

Emilio Calvo, from Sngular, answered that although the client usually likes the idea of developing only once and that it is valid for several platforms, the reality is that due to issues such as testing duplication, the gain of time with respect to the native alternatives is not much.

From Inditex, Marcos González Castro, responds that given their current relevance they do not consider using cross-platform technologies in the short term.

When we asked them if they had studied any cross-platform technology, they all pointed to those that are based on JavaScript, like Ionic or React Native.

## 2.4 Cross-Platform Frameworks

To talk about multi-platform technologies first we have to differentiate between hybrid and cross-platform framework concepts. Each one represents a different approach to multiplatform development.

On the one hand, cross-platform frameworks are those that to deploy the application on the mobile platform use an embedded web browser. That is, a webview is being rendered, there are no native components. These frameworks are those derived from Apache Cordova, such as Ionic.

On the other hand, hybrid frameworks are those that display native components and views. These have a bridge or an API that make the conversion between the code with which we developed the application and the native code of the platform. In this class of frameworks we can find React Native or Flutter, although, being both cross-platform frameworks that use a different approach.

### 2.4.1 Framework Selection

In order to find out which of the selected frameworks were the most relevant for the software developer community, we carried out a study during 6 months in which we measured the number of repositories in GitHub associated with each framework.

Two different metrics were taken:

- The number of repositories associated with the topic of the framework.
- The number of repositories associated with a search of the framework.

With this we will obtain two results, on the one hand, which of the frameworks studied were the most used by the development community. On the other hand, what is the deviation between one measure and the other.

As we can see, the frameworks with more repositories are Ionic and React Native since, as we mentioned previously, they are currently the most popular cross-platform development alternatives.

We can also observe the growth in the number of repositories of each of the frameworks. While Weex has maintained practically the same number of repositories during these 6 months, others, such as Flutter, have quadrupled the initial number of repositories from March to August.

From the deviation between the two measures we can deduce that those frameworks whose measurement in search is far from its measure in topic are more used to test them than complete developments.

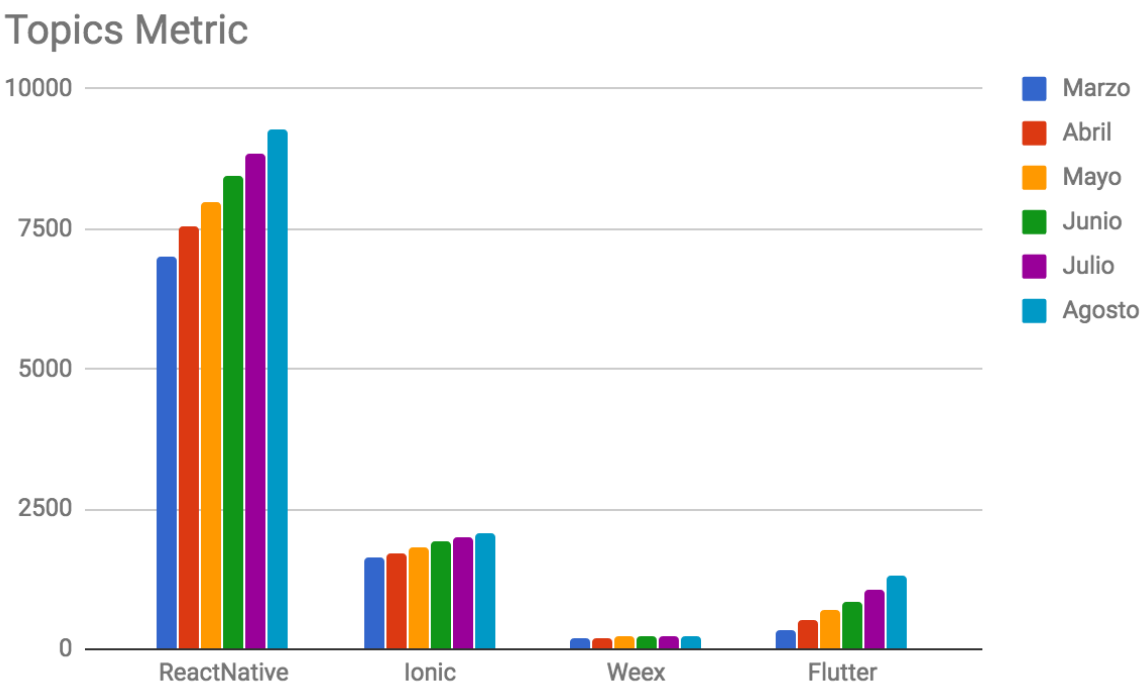


Figure 2.2: GitHub Topics Metric

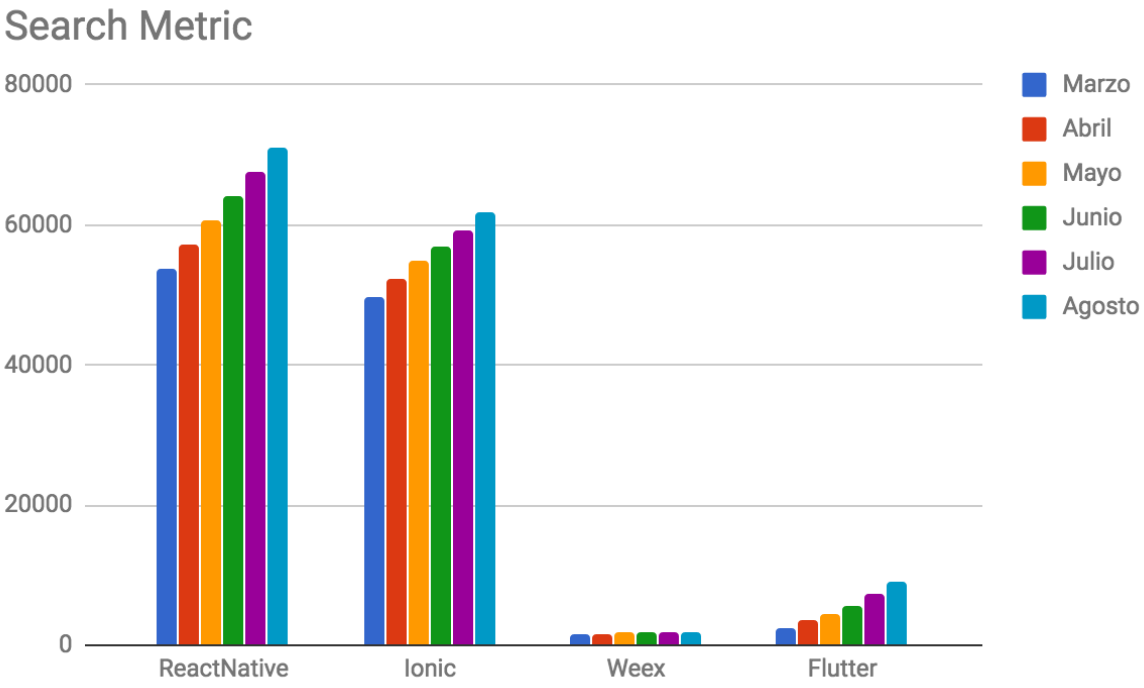


Figure 2.3: GitHub Search Metric

### 2.4.2 React Native

Initially developed by Facebook, React native has became one of the most used alternatives for cross-platform app development.

It is based in the JavaScript framework React.JS. One of its main features is that it does not rely on HTML5 for the view construction, but everything is built making use of JavaScript.

Unlike frameworks derived from Apache Cordova, React Native does not run using a WebView. It has direct access to the native API's and the views offered by the Mobile OS. Thus, the "look & feel" applications developed using React Native is very similar to a native application, which is why it is one of the most used cross-platform alternatives.

As we can see in 2.4, to discuss the architecture of React Native is necessary to distinguish between three main parts:

1. Native code of the platform in which the application is running, Objective-C or Java.

2. JavaScript VM, virtual machine that is in charge of running all the JS code.

3. React Native Bridge, that is responsible of communicating the JavaScript VM with the native code of the platform.

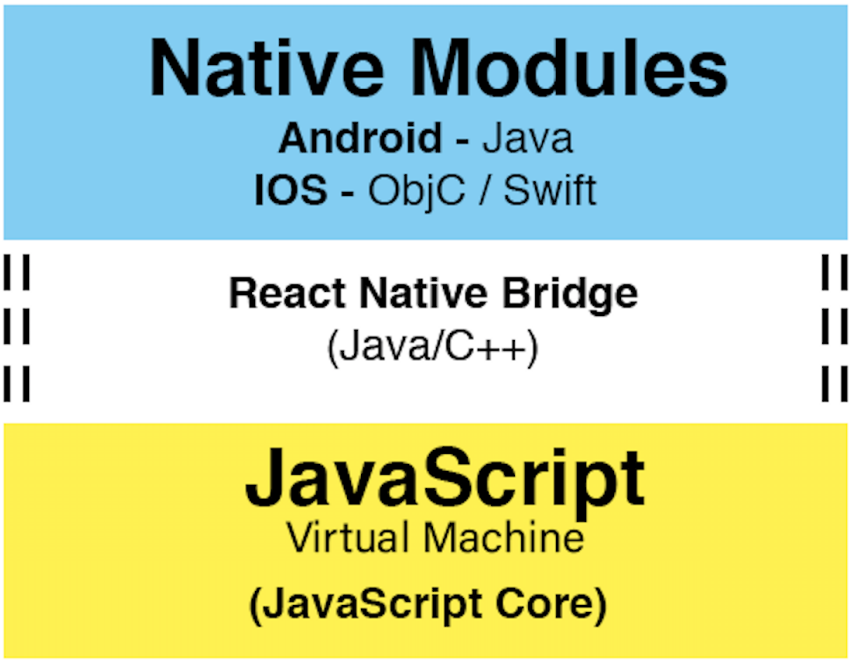


Figure 2.4: React Native architecture diagram

When rendering our components, React Native, instead of rendering them over the browser DOM, calls the corresponding API's of the OS to render the components. As [7] indicates , this is thanks to the bridge provided by React Native, which provides it with an interface to Native UI components.

### 2.4.3 Ionic & Weex

On the one hand, Ionic is an open-source SDK, along with React Native is other of the most used options for cross-platform development. It is built on top of Angular.

On the other hand, Weex, maintained by the giant Alibaba group, is an alternative to Ionic built on top of Vue.JS.

Both share the same architecture, this is due to they are both built on top of Apache Cordova (Previously called PhoneGap). Cordova allows the developer to use web development technologies to build mobile applications. It builds a wrapper around the application that targets the platform in which it runs. Cordova, relies on an API to communicate this wrapper with the native platform, allowing the application to access native functionality of the platform such as the camera, GPS location or the Accelerometer. [5]

Both frameworks provide the developer with a wide variety of standard components, typographies and themes that can be used to give shape to the application.

### 2.4.4 Flutter

Developed by Google, Flutter is its alternative for cross-platform technologies. Flutter, in contrast with the previous ones, is developed using Dart, a language developed by the same company as a modern alternative to JavaScript.

As can be seen in 2.5 Flutter works through three layers, the first contains a shell (specific to each platform) with the Dart VM. This is the one that gives access to the native APIs of the platform in which it is executed. The next layer contains the engine, which provides the Dart Runtime, Skia, ... etc. And the third and last layer contains the Embedder, which is platform specific.

Unlike hybrid frameworks like Ionic or Weex, Flutter does not rely on web wrappers to display the application. In Flutter every object is a Widget that is part of a view, once the views are ready to be displayed, Flutter send them to a graphic engine like Skia in order to render them.

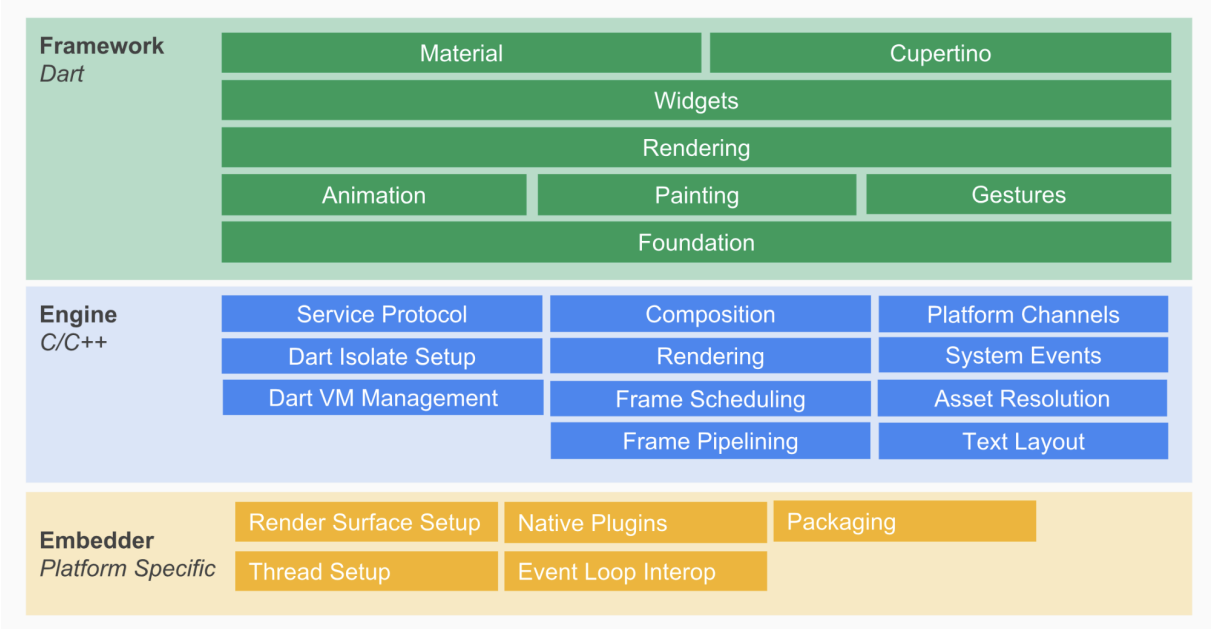


Figure 2.5: Flutter architecture diagram [9]

# Chapter 3

# Project Planning

Once the foundations of the current state of the art have been laid, let us discuss which is the methodology used in the development of the project and how it has been planified.

## 3.1 Life cicle model

The approach selected for the development of the project was an incremental life cycle model. SWEBOK defines incremental life cycle model as *"a model that produces successive increments of working, deliverable software based on partitioning of the software requirements to be implemented in each of the increments"* [4].

## 3.2 Stages

Due to the complexity of it, the project development was carried out in two differentiated phases:

### 3.2.1 Development

During this phase, the four developments corresponding to each of the applications developed were carried out.

For each one of the applications we carried out the following tasks:

- Research of the framework used during the development
- Development of the application on which the metrics will be taken
- Implementation of functional tests
- Measurement of execution times with different sets of tasks and in different platforms
- Measurement of code quality metrics

### 3.2.2 Analysis

Once the developments of the applications were finished, the project entered a second phase in which the results obtained were analyzed in order to draw our final conclusions about the study.

During the analysis we carried out the following tasks:

- Times analysis by application
- Generation of time graphs for each framework
- Time comparison
- Comparison of code quality metrics

## 3.3 Time Management

As previously mentioned, in order to carry out with the study it was splitted in two different phases.

We estimated a duration of four months for the development phase and two months for the analysis phase. To illustrate the time estimations we developed a Gantt chart.

A Gantt chart is a type of bar chart that illustrates the schedule of the project. The vertical axis of this graph shows the different tasks that have led to the development of the project, while the horizontal axis illustrates how much time should be spent on the development of each tasks and the dependencies between them, allowing us to illustrate the critical path. [10]

## 3.4 Technologies used

Through the development of this study the following technologies have been used.

### 3.4.1 Programming languages

#### Javascript

Javascript is one of the most used programming languages for web development and during this project it was used to develop the React Native, Weex and Ionic apps.

#### Dart

Developed by Google as a modern alternative to Javascript for web development, Dart is mainly used as the language to program apps using Flutter. It also supports web and server-side development. [6]

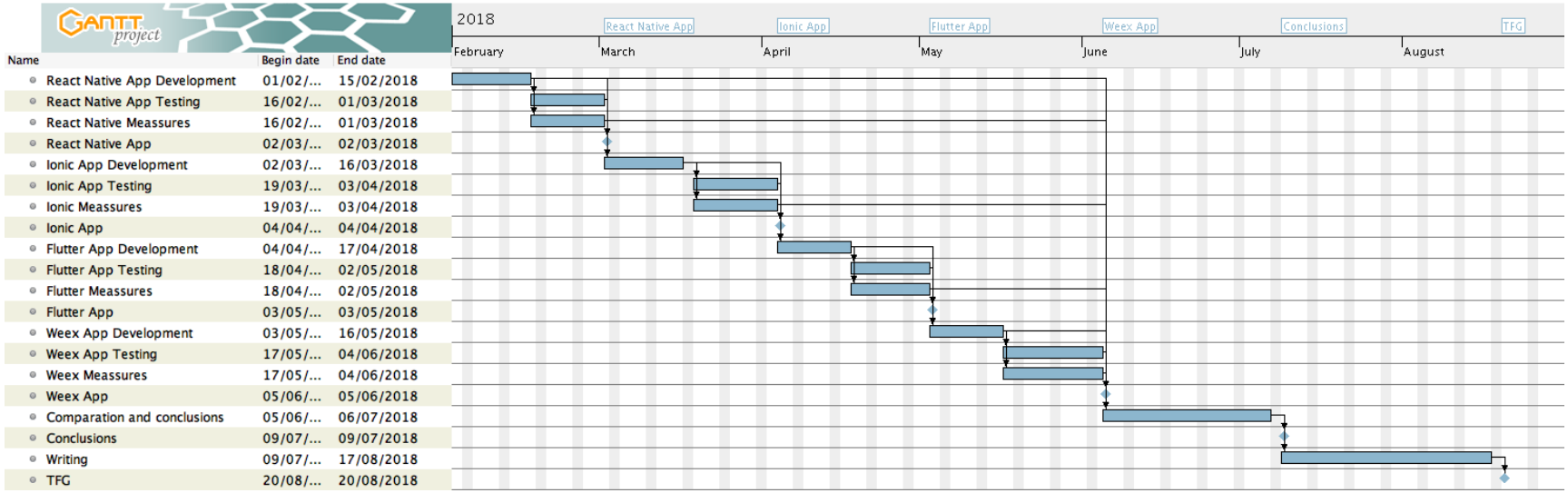


Figure 3.1: Gantt Chart of the project

### 3.4.2 Version control

Version control systems are indispensable for the developers. They allow us to record all the changes in a project so we can recall to a previous version if necessary. [12]

Since its birth in 2005 [11] Git has become the most popular version control system of the software community. On top of it has grown some gigantic development communities like GitHub, with more than 24 million users and 64 million repositories [13]

Git was the version control system of our choice. Four different repositories have been maintained in GitHub.com, one per application. Each repository contains the source code of the application as well as the necessary documentation to be able to deploy it.

Repositories are as follows:

- React Native [16]
- Ionic [15]
- Flutter [14]
- Weex [17]

### 3.4.3 Frameworks

This study was built on top of four multi platform frameworks. They are the main technologies used during the development of this project. As previously mentioned these frameworks are Ionic, Weex, React Native and Flutter.

Aside from those four frameworks, ExpressJS have been used for programming the server API for receiving the time measures.

### 3.4.4 Testing

In order to carry out with testing, we used the JavaScript library Jest in React Native, Ionic and Weex. Jest is a testing library maintained by Facebook that integrates well with TypeScript and vanilla JavaScript. [18]

When it comes to automating these tests, the automation tool Appium.io has been used. Appium is a tool derived from Selenium.io that automates cross-platform apps through a client [1]

### 3.4.5 Documentation

GanttProject has been used as a Gantt chart generation tool. In order to generate the mockups used for the design of the app interface it has been used Balsamiq Mockups.

### 3.4.6 Metrics

For measuring the execution times of the applications developed using JavaScript the library performance-now [22] has been used. When it comes to Flutter, due to it is developed in Dart, it has been used the Stopwatch class included in the framework.

For measuring code quality metrics, SonarQube has been used. It has been configured with default metrics. (Sonar-Way). SonarQube is quality inspection tool that integrates with git repositories. It uses a wide variety of metrics in other to inspect every aspect of the project in which it is running. Some of its measures are technical debt, vulnerabilities or code smells [23]

# Part II

# Development

# Chapter 4

# Application

In this chapter we will discuss the applications design as well as their main functionalities.

## 4.1 Design

During the realization of this study four functionalities were developed in one application. This development was in order to carry out the framework comparison. Each one of them is representative of the cross-platform or hybrid framework in which it is developed.

It was established that the functionality of the application had to be as representative as possible of a standard mobile application. So it was decided to implement a task management application.

Previous to the implementation of the applications, a Mockup was made using Balsamiq Mockup Tool [3].

As can be seen in 4.1 the elements of interface have been indicated. This interface is shared by the four applications. Let us now discuss the functionality of each of the elements indicated in 4.1

- **Top Bar:** Top bar of the application, the name of the app is indicated in it. When the user clicks on the bar, the application will scroll automatically to the beginning of the task list.

- **Refresh Spinner:** This icon appears when we perform the *"Pull & Refresh"* action on the task list. This action will reload the task list

- **Task number and description:** This field indicates the number of the task in the list and the description or name given by the user at the time of introducing the task.

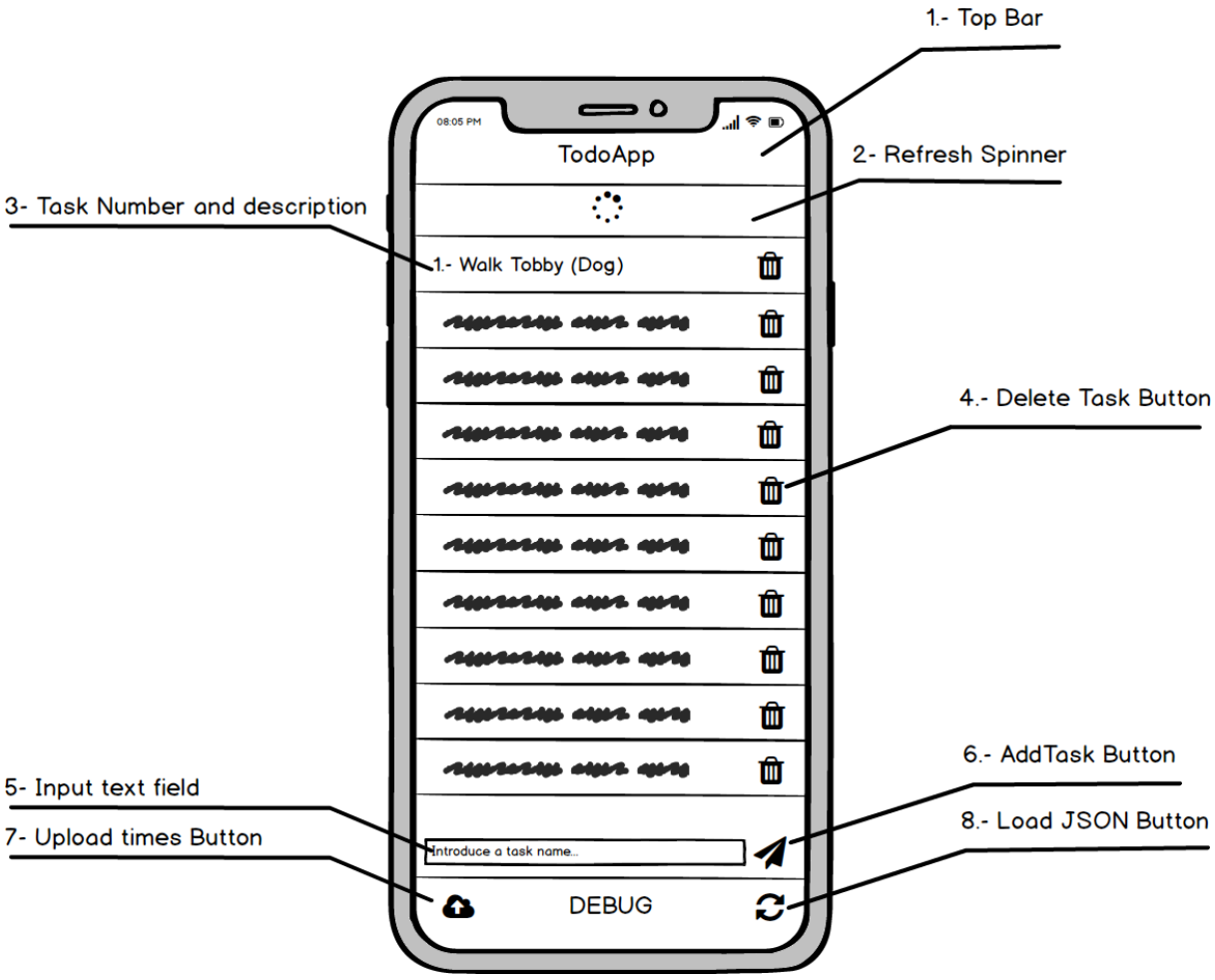


Figure 4.1: Mockup

- **Delete task button:** This button, represented by a trash can icon, will delete the selected task from the list when it is clicked by the user.

- **Input text field:** In this text field, the user will enter the name or description of the task they wish to add.

- **Add task button:** After entering the name or description of the task the user will press this button, represented by a paper airplane icon. This will add the task to the end of the list, after this the application will scroll automatically to the bottom of it.

- **Upload times button:** This button, visible only in test versions, will send the results of the times measurements to the server when it is pressed.

- **Load JSON button:** This button, visible only in test versions, will load the JSON file with the taks into the application when it is pressed.

This interface, specified in 4.1, has been implemented equally in the all applications developed. For each framework UI components have been used, so the visual style can vary from one to another.

The following figures show the final result in each one of the applications.

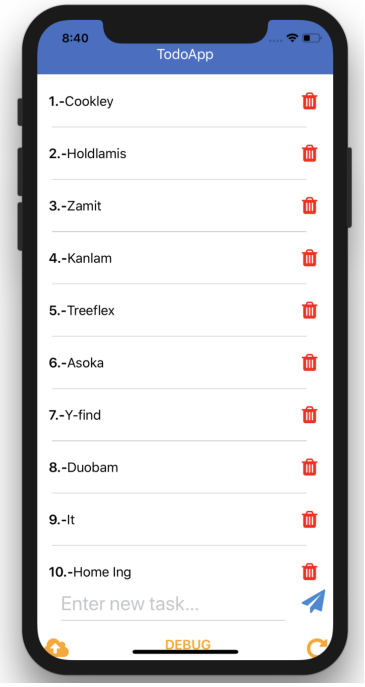


Figure 4.2: React Native App Screenshot

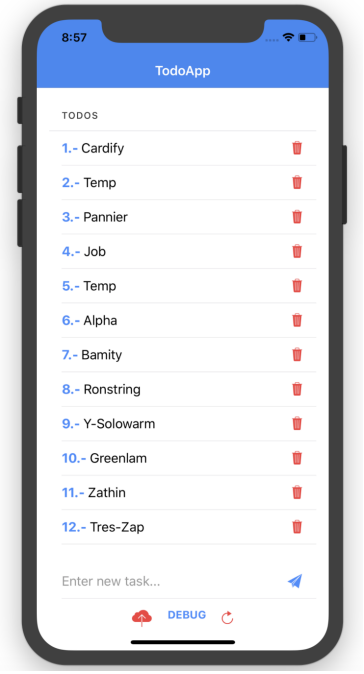


Figure 4.3: Ionic App Screenshot

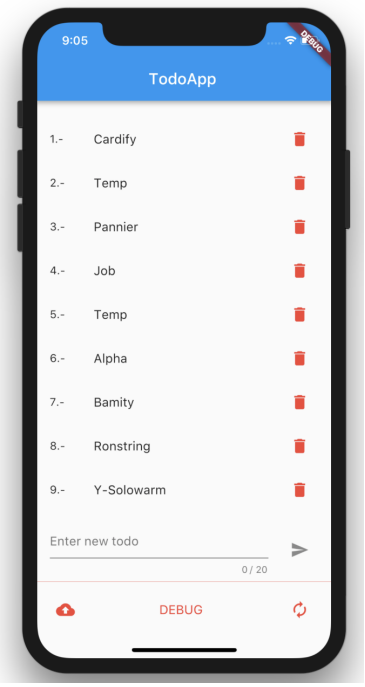


Figure 4.4: Flutter App Screenshot

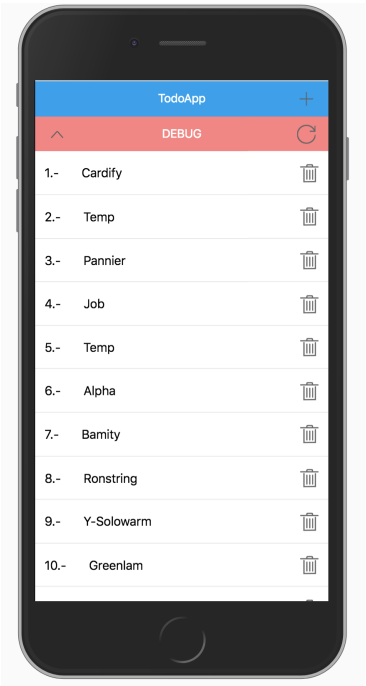


Figure 4.5: Weex App Screenshot

## 4.2 Functionality

As we have previously mentioned we sought to make the functionality of the application as representative as possible to a standard mobile application. This is why it was decided to implement a list of tasks.

The four main functionalities of the application correspond to the four basic functions of persistent storage or CRUD [24] In this case it has been decided not to implement the update operation since it is equivalent to Read (getTask) + Create (addTask) + Delete (removeTask).

This operations have been used to obtain the measures of execution times:

- **Add Tasks:** You can add tasks to the list, once the task is added, the application will scroll automatically until the end of the list.

- **Delete Tasks:** You can delete tasks from the list by clicking on the icon associated with the task.

- **Update the task list:** You can update the task list by means of a pull & refresh. This will load in the app the list stored in the local database.

- **Load a list of tasks:** Lists of tasks previously defined in a JSON file can be loaded. In this case the sets of candidates for the tests are loaded. These are 100, 500 and 1000 candidates.

Another functionality to comment is the possibility of sending temporary measures to a server through an HTTP call, but this has not been taken into account when performing the measurements.

The first time we enter the application a set of candidates (task list) is loaded from a JSON file and then stored in the local database, this is given by the Storage component of the framework.

Subsequently, each time we add, delete a task or load a set of tasks, the storage of the framework will be updated.

# Chapter 5

# Metrics

In this chapter we will analyze how the temporal and quality code metrics have been taken and what the results are for each one of the frameworks.

## 5.1 Process

When taking the execution times of each of the frameworks, the four main operations performed by the application have been taken into account. These have been previously described.

We have obtained sets of measurements in milliseconds with 100 elements, on which the arithmetic mean was subsequently made.

The process of measuring execution times is as follows. Each time one of the four operations is launched, the application records the execution time of the operation using an internal timer. Once the measurements have been made, they are sent to the server through an HTTP request. Subsequently, the server processes and stores the measurements in a JSON file named *"frameworkname_platform_candidate#.json"*.

This server is an small droplet in DigitalOcean.com. The API of the server have been done making use of NodeJS and ExpressJS.

When it comes to code quality metrics, we used SonarQube. It is launched through a small script defined in the package.json configuration file.

```
// Import dependencies
const express = require('express');
const router = express.Router();
var fs = require('fs');


/* GET api listing. */
router.get('/', (req, res) => {
   res.send('api works');
});

/* Create a user. */
router.post('/test', (req, res) => {
   // console.log(req.body);
   res.send('post works')

   console.log(req.body)
   var json = require('../../ionic.json');
   json.push(req.body)

   var data = fs.writeFileSync("../ionic.json", JSON.stringify(json));

});


module.exports = router;
```

Listing 1: Server Backend Example

```
{
    "name": "IonicApp",
    "version": "0.0.1",
    "author": "Ionic Framework",
    "homepage": "http://ionicframework.com/",
    "private": true,
    "scripts": {
      "clean": "ionic-app-scripts clean",
      "build": "ionic-app-scripts build",
      "lint": "ionic-app-scripts lint",
      "ionic:build": "ionic-app-scripts build",
      "ionic:serve": "ionic-app-scripts serve",
      "test": "jest",
      "sonar": "sonar-scanner \
               -Dsonar.projectKey=tfg-ionic \
               -Dsonar.organization=manuelrdsg-github \
               -Dsonar.sources=. \
               -Dsonar.host.url=https://sonarcloud.io \
               -Dsonar.login=`` \
               -Dsonar.java.binaries=./node_modules//* \
               -Dsonar.cfamily.build-wrapper-output.bypass=true"
},
```

Listing 2: Ionic package.json SonarQube script

## 5.2 Execution times

Let's now discuss the results of the execution time measurements in each of the frameworks.

### 5.2.1 React Native

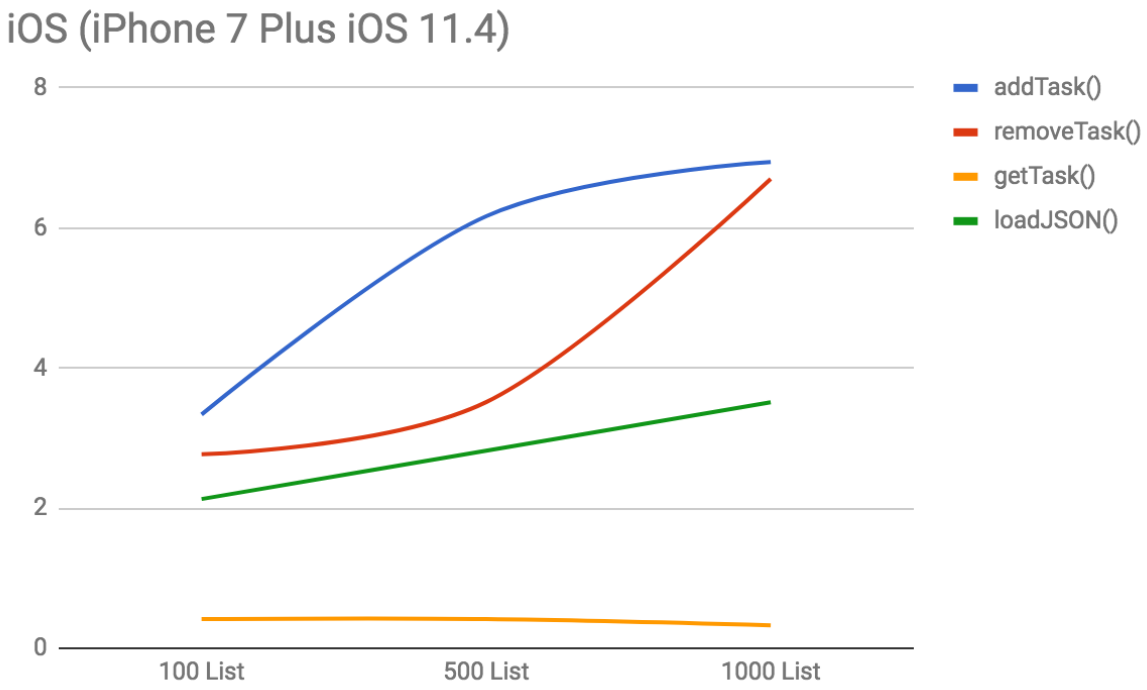


Figure 5.1: React Native iOS Execution times

As can be seen in the figure 5.1 , the most expensive operations in the case of React Native were addTask(), removeTask(), and loadJSON(). These three operations increased their execution time as the set of candidates grew, while getTask () remained at the same times.

This, due to getTask() is the only one of the three operations in which the list of tasks is not updated, indicates that the greater the number of changes to be made during rendering, the greater the execution time.

When it comes to removeTask(), React Native, completely reloads the FlatList component when an item from the list is deleted, so it is rendered again.

On the other hand, in the case of addTask(), the application scrolls to the end of the list when we add a task, so that, the larger the list will be the larger the scroll to perform.

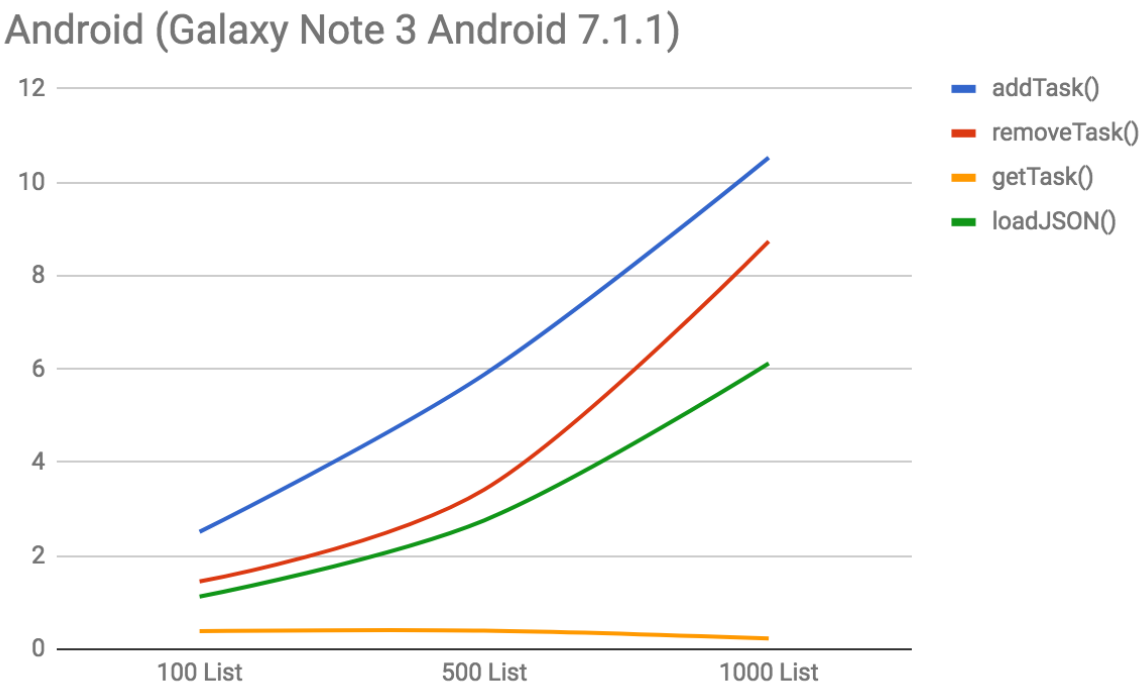


Figure 5.2: React Native Android Execution times

If we compare the performance of React Native in devices with Android operating system (5.2), we see that the times are significantly longer. This may be because the Android components to which React Native translates its own components are more expensive to render than its counterpart in iOS.

The only operation that stays in the same times is getTasks (). The only one of the four operations analyzed that does not involve rendering components on the screen.

### 5.2.2 Ionic

As we can see in 5.3, in this case the most expensive operation in temporal terms turns out to be addTask(). While removeTask(), getTask() are kept around the same times and loadJSON () even decreases, addTask() increases to almost 15 ms in the case of the list with 1000 elements.

While the results obtained in both removeTask() and loadJSON() seem to indicate that Ionic does not render the complete list when updating it, addTask() matches the previously mentioned, the greater the number of candidates the greater the scroll and therefore, the execution time will be longer.

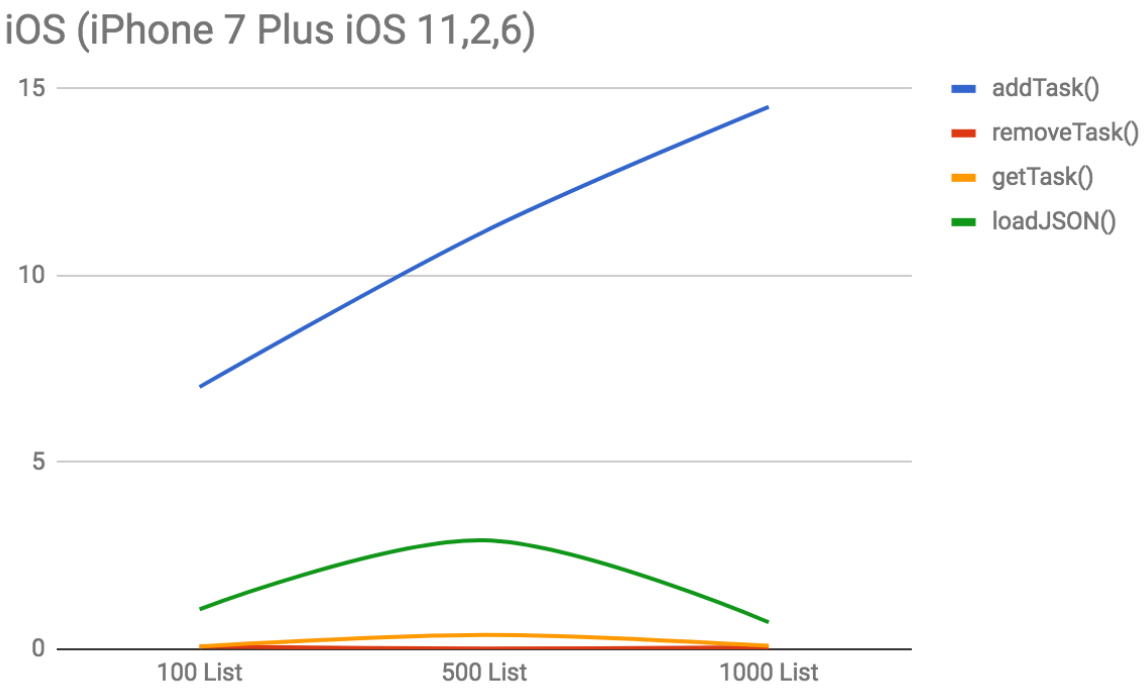


Figure 5.3: Ionic iOS Execution times

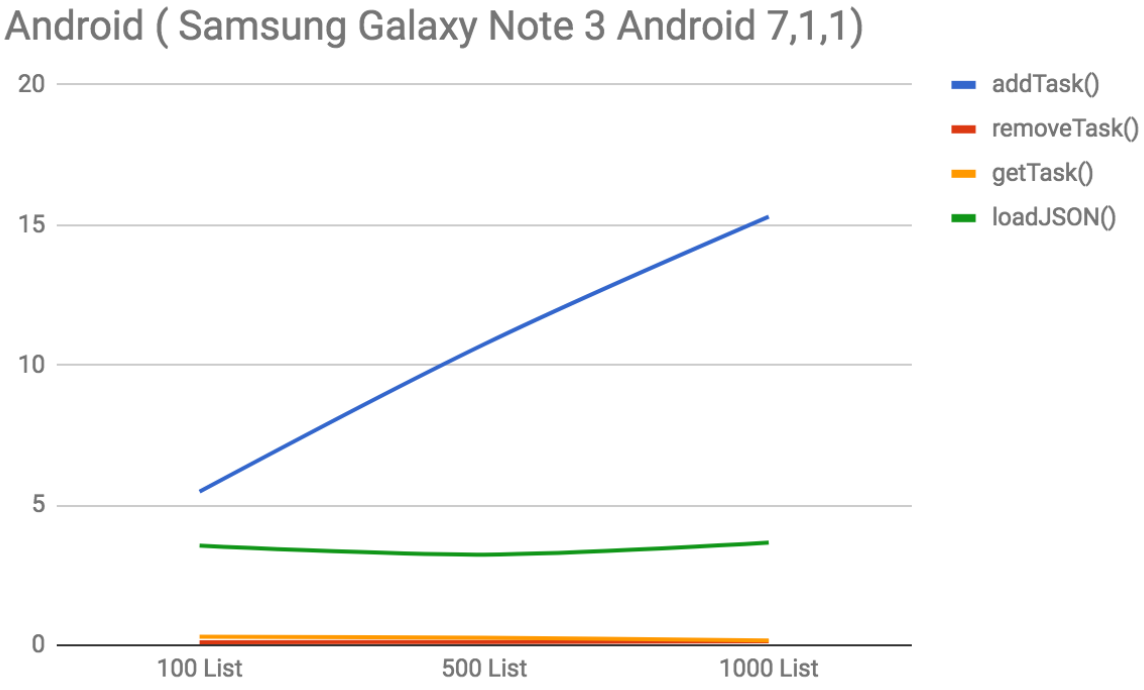


Figure 5.4: Ionic Android Execution times

If we compare the previous graph with its equivalent in Android in 5.4, we see again that the execution times are somewhat higher. While in the case of iOS the operation of addTask() barely reached 15 ms in its worst case, in the case of Android exceeds 15 ms.

In the case of Ionic, this time difference may be indicated by the difference between the WebViews in iOS and Android, while in iOS these are given by Safari, in Android the WebViews are given by Chrome.

### 5.2.3 Flutter

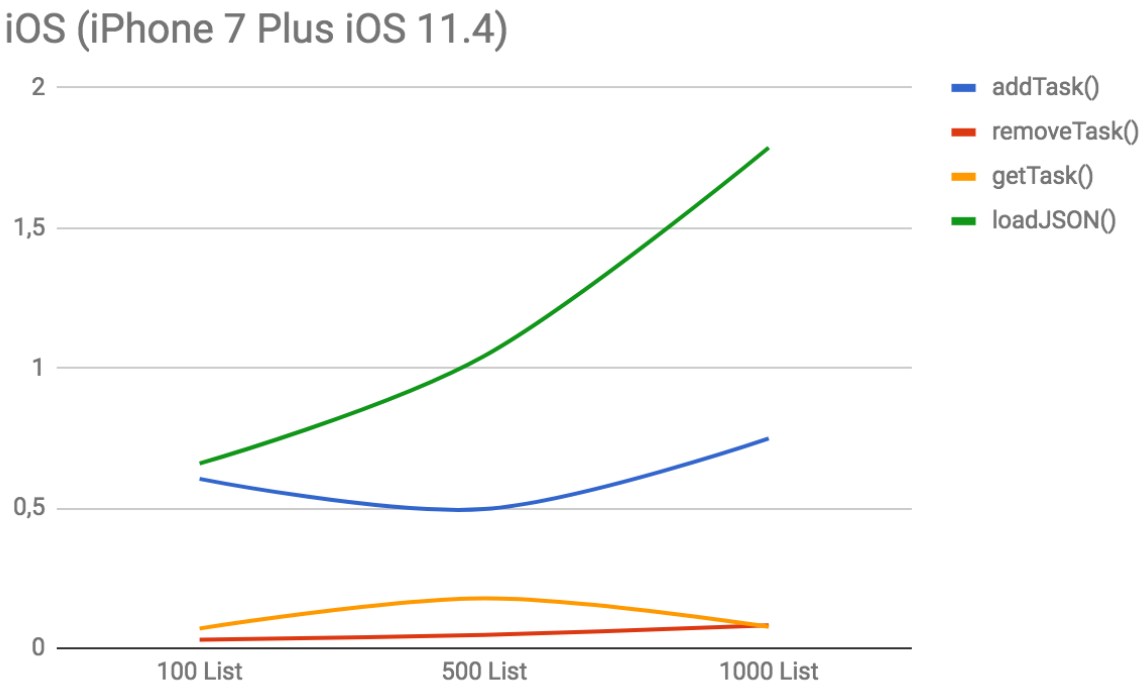


Figure 5.5: Flutter iOS Execution times

In the case of Flutter we see that the two operations with the highest cost are loadJSON() and addTask(). While removeTask() and getTask() remain in the same numbers, loadJSON increases significantly as the number of items in the list grows.

This may be due to the way in which Flutter loads the tasks, whereas, in React Native as well as in Ionic, the file is already parsed when is loaded. In Flutter we perform the operation of parsing the file and then casting to list the result and assign it to the widget in charge of rendering it.

```
JsonDecoder decoder = new JsonDecoder();
String json = await _fetchJSON();
List dec = decoder.convert(json);

setState(() {
      widget.items = dec.cast<String>().toList();
});

_updateStorage();
```

Listing 3: Flutter loadJSON process

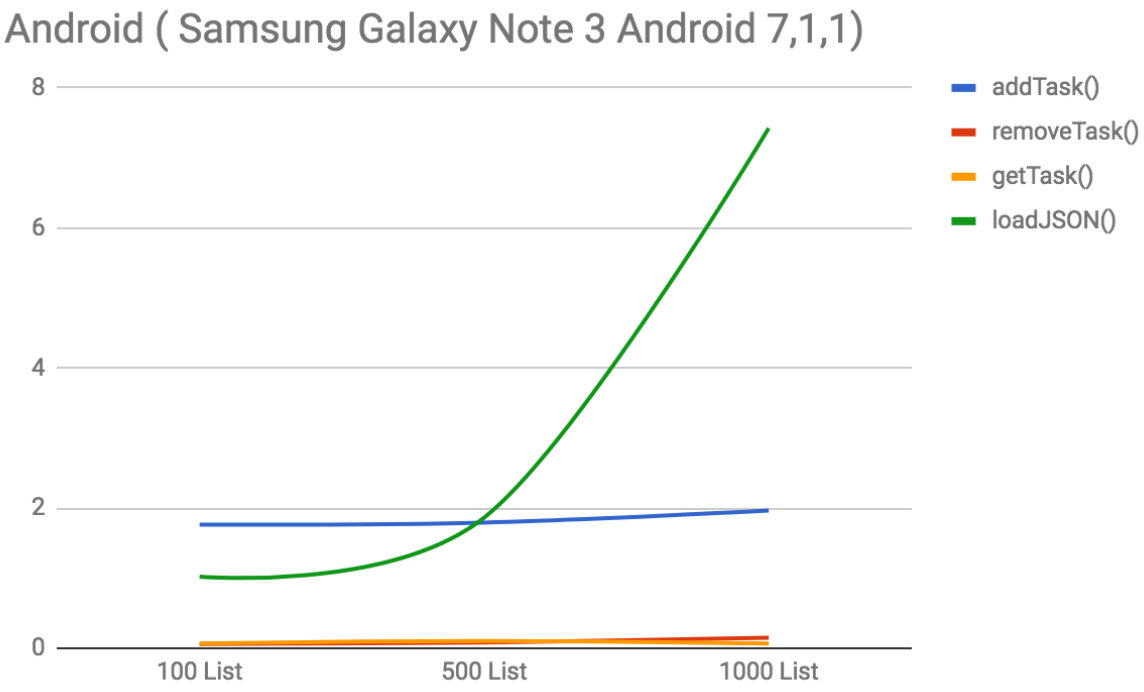


Figure 5.6: Flutter Android Execution times

If we compare the times of the iOS version with its counterpart in Android, we see how the times increase significantly again. In this case the loadJSON operation practically quadruples its time of the version in iOS in the worst case.

While addTask increases its time in this version, but remains more stable against the number of tasks. RemoveTask() and getTask() are kept at the same times in both versions.

### 5.2.4 Weex

Due to the precarious current state of Weex we do not have data of temporary measurements for this framework. This is so because despite the fact that the development of the benchmark application has been carried out, it has not been possible to deploy the application in native language.

If it has been possible to deploy the application in web browser, and the measurement of execution times in that environment. So, although not measuring times in the same environments we can not make a direct comparison with the rest of frameworks, then we will comment on how the four main operations in Weex have been implemented and where we believe that their weaknesses are.

In the case of Weex, because the framework did not have components for text fields, we implemented the data entry through a modal, launched by an event at the press of a button. When we send the text, as in the rest of the frameworks, it is processed, added to the task list and an automatic scroll is made at the end of the list.

```
addTodo() {
    var t1 = now();
    modal.prompt(
      {
        message: "Enter new task..."
      },
      value => {
        if (value.result === "OK" && value.data.length > 0) {
          this.tasks.push(value.data);
        }
      }
    );
    var t2 = now();
    this.Times.addTasks.push(t2 - t1);
}
```

Listing 4: Weex addTask process

The weakest point of Weex currently is its precarious state, it does not include components like the rest of the frameworks, so it is necessary to use external libraries like weex-ui (even in beta). Due to is developed by the AliBaba group, the majority of the documentation is only available in simplified Chinese, as well as being quite incomplete.

## 5.3 Code Quality

Let's now discuss the result of the code quality metrics taken for each one of the frameworks.

When it comes to analyzing the code quality metrics we must take into account that the complete projects have been analyzed, this includes, dependency directories like node_modules and the compiled projects for each one of the platforms (iOS and Android), so that the majority of duplications, vulnerabilities, code smells etc.. correspond to generated code by each one of the frameworks.

This can give us another point of view. How do the projects generate each one of the frameworks? What are the minimum packages necessary for the project to work?

### 5.3.1 React Native

Let's now analyze the quality of React Native code. In the figure shown below the results of the SonarQube scan can be seen.

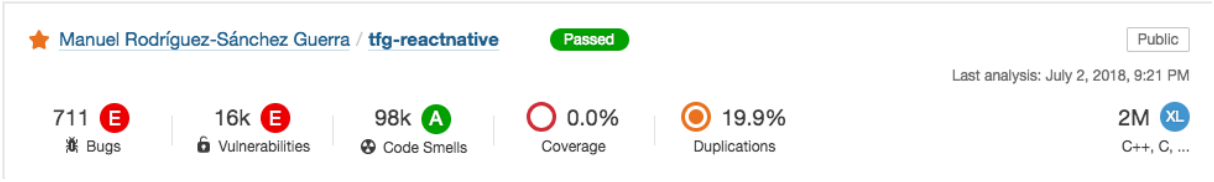


Figure 5.7: React Native code quality

As can be seen in the figure 5.7 the project consists of 2 million lines of code, of which 462 have been written and the rest generated by the framework itself.

The code has 16k vulnerabilities, most of them relating to the visibility of variables or functions within classes. Fixing the vulnerabilities and code smells carries a technical debt of 2809 days.

### 5.3.2 Ionic

If we look now at the results obtained for Ionic in the scan made by SonarQube, we find that the number of lines of code is drastically reduced until 911k

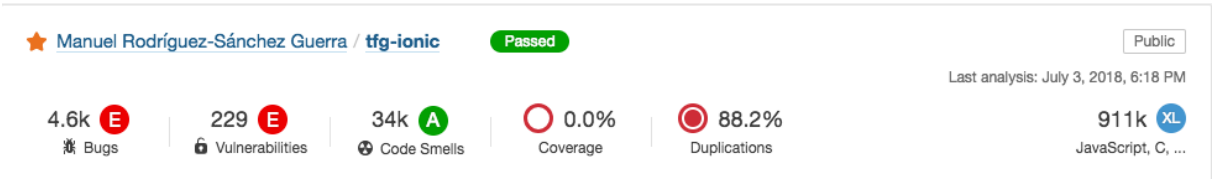


Figure 5.8: Ionic code quality

In this case, being a smaller project (as far as lines of code are concerned) we find a lower number of vulnerabilities, 229 to be exact. In this case related to the control of exceptions and the conversion to static of certain variables that are not updated throughout the code.

The 34k code smells and vulnerabilities carry a technical debt of 654 days.

Due to the generation of Cordova libraries for the compiled native projects, the project has a worrying 88.2% of duplications with a total of 1.3M of duplicate lines.

### 5.3.3 Flutter

Let's now discuss the results of the code quality metrics for Flutter. In this case, due to is written in Dart and this is a minority language (used in the case of flutter and in some cases for web development) SonarQube does not have a scan plugin for that language, so we will limit ourselves to comment the quality of the project code itself (generated files, libraries and projects in native language).


Figure 5.9: Flutter code quality

In this case we find a minimum number of bugs and code smells compared to previous frameworks. With a total of 1.2k lines of code in question of libraries and native projects, the project has a technical debt amounting to 1 day.

Most of the code smells correspond to header files generated for the native project corresponding to iOS.

We find a percentage of duplication of code corresponding to 69.7%, with a total of 2476 lines duplicated in the project. Corresponding once more to the native projects generated for iOS and Android.

### 5.3.4 Weex

Let's now discuss the results of the analysis for Weex. This framework, as previously discussed in the document, is similar to Ionic due to both make use of Apache Cordova to deploy its components in the form of WebView.

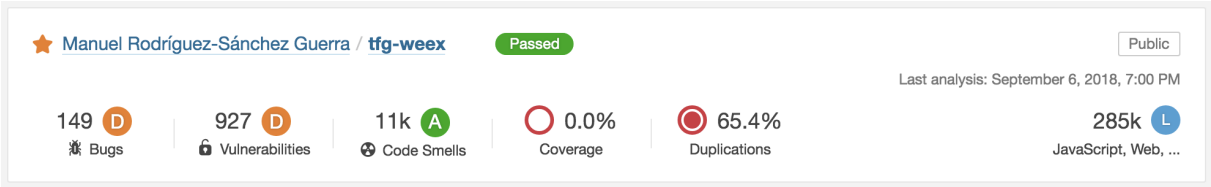


Figure 5.10: Weex code quality

We find a low number of bugs and vulnerabilities compared to other frameworks. The project has a total of 285k lines of code, with a technical debt amounting to a total of 138 days.

Most code smells correspond to refactoring and elimination of unused variables.

With a total of 257k duplicate lines we find a percentage of duplication of 65.4%. This percentage of duplication corresponds in its great majority with packages installed in the folder node_modules.

# Chapter 6

# Testing

Software testing is defined in [4] as *"the dynamic verification that a program provides expected behaviors on a finite set of test cases, suitably selected from the usually infinite execution domain"*

The process of testing the software developed is not necessary but essential. It is not only focused in discovering issued produced during the implementation phase, but also on the fact that the final product meets the requirements specified by the client in the specification.

In this chapter we will discuss which has been the software testing technique applied to this project.

## 6.1 Functional testing

Functional testing is defined as *"testing technique that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions and conducted to evaluate the compliance of a system or component with specified functional requirements"* in [8]

These tests are used to prove that every functionality implemented is working as requested by the client, therefore functional testing is one of the most important testing techniques.

In this project, functional tests have been implemented for the four main use cases.

- Add task
- Remove task
- Load JSON
- Reload task list

These tests have been automated making use of Appium, a test automation tool that uses the same webDriver as the automation tool Selenium, widely used in web development testing.

```
        test('Adding Task', async () => {
            let el5 = await driver.
                elementByXPath("(//XCUIElementTypeOther
                    [@name=\"Enter new task...\"])[1]");
            await el5.click();
            await el5.sendKeys("Test");
            let el6 = await driver
                .elementByXPath("//XCUIElementTypeStaticText[@name=\"\"]");
            await el6.click();
           });
```

Listing 5: 'addTask' functional test for React native

The process of automation goes by searching the elements by XPath and automate their use by calling functions such as *"click()"* or *"sendKeys(string)"*.

# Part III

# Conclusions

# Chapter 7

# Comparison

After obtaining the metrics of each one of the frameworks and analyzing them, in this chapter, we will compare the frameworks among them and we will draw our final conclusions.

## 7.1 Execution times

As far as execution times are concerned, we can observe a great disparity between the results of each framework for each of the operations.

As we mentioned at the beginning of this document, we have to differentiate between hybrid frameworks (those that use a webview to render the content on the screen, therefore, they are webApps using a wrapper) and cross-platform frameworks (those that do render native components)

Bearing this in mind, if we look at the three operations that require on screen rendering (these are addTask(), removeTask(), LoadJSON() ) we see how Flutter achieves better times than the rest of the frameworks.

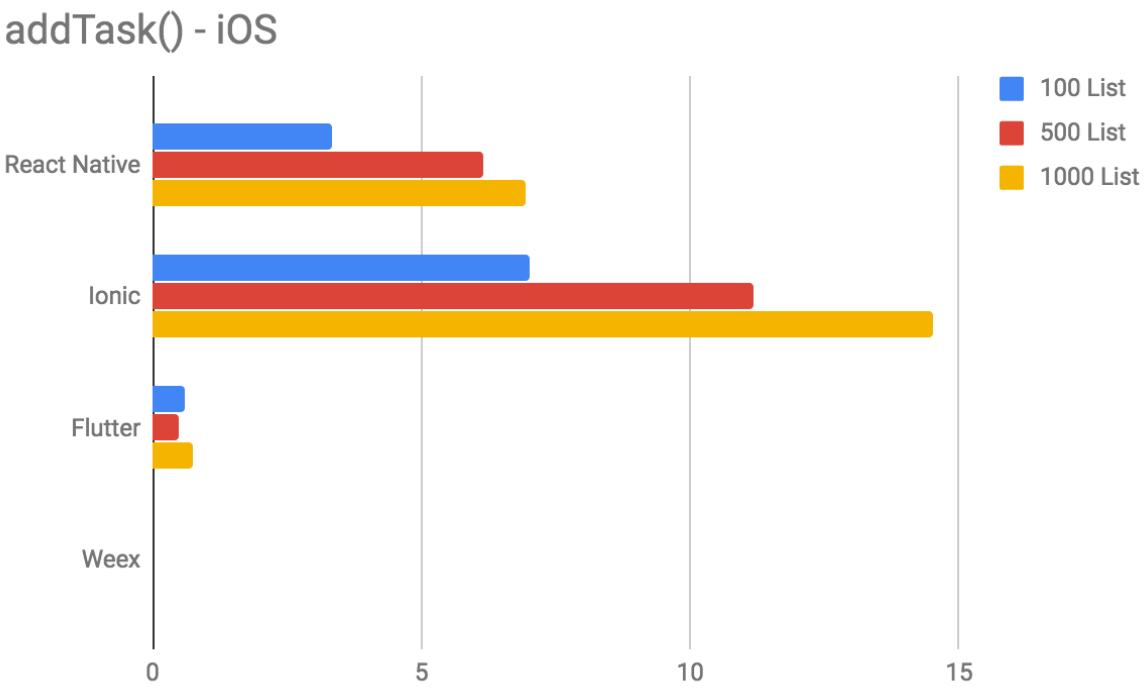


Figure 7.1: addTask() time comparison

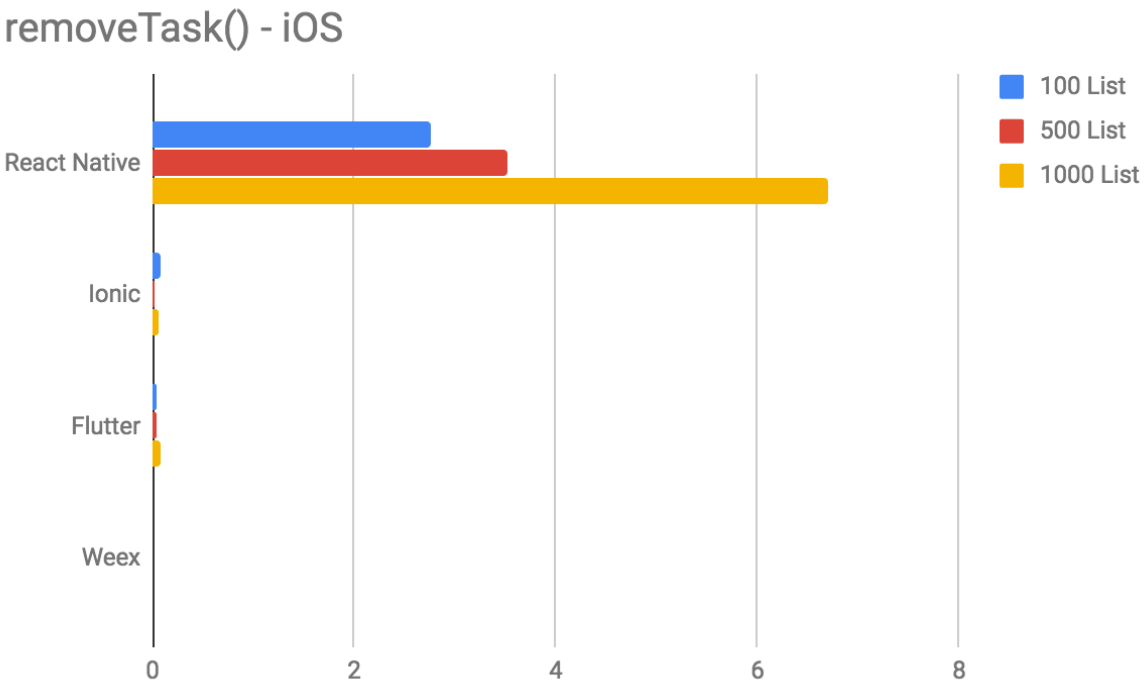


Figure 7.2: removeTask() time comparison

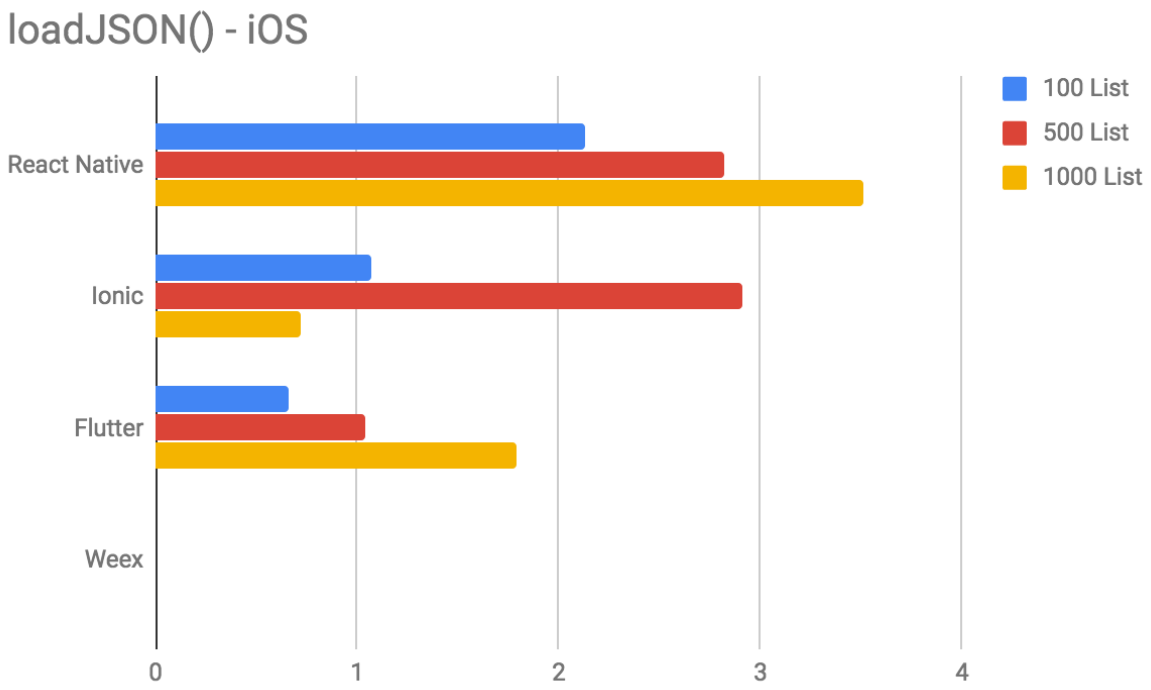


Figure 7.3: loadJSON() time comparison

This is because its own architecture, unlike React native, Flutter does not bridge its components and those that are native to the system. And unlike Ionic, it does not render its components using a webview.

It prepares a view in which it includes its widgets (or components) and renders them using an API for rendering graphics such as OpenGL or Skia. Thanks to this, it is saving the step of translating its components into the native language of the system to later render them, so the performance is much better than in the rest of the frameworks.

In addition to this, we can appreciate that, React Native does its worst times in the removeTask() and loadJSON() functions, which are precisely those that require rendering the complete or practically complete list.

In the case of removeTask () all the tasks in the list that are below the deleted task must be updated and in the case of loadJSON we reload the list completely.

This can be seen if we compare the times in removeTask and loadJSON with those obtained for addTask, which are much better, since we do not have to render the entire list but only the last component of the list (the one that is added).

In the case of hybrid frameworks, Ionic and Weex, we see how the operation in which they get their worst time is addTask(). This, being the scroll one of the most expensive operations carried out, leaves evidence of how factors such as the scrolling of applications that render using webViews is not comparable to the native scroll, provided in this case by React Native or Flutter.

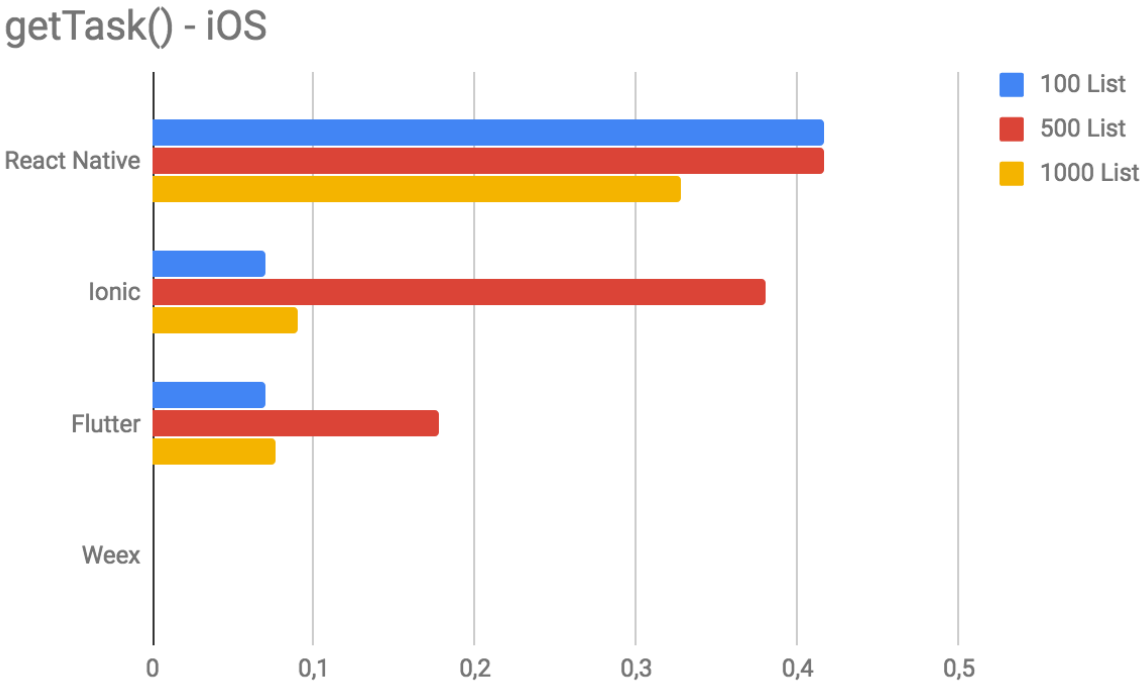


Figure 7.4: getTask() time comparison

If we look now at the operation getTask(), we see that although the times are the least for all operations (because the operation does not involve the rendering of elements) all frameworks get their worst time in the case of the list with 500 elements.

## 7.2 Code quality

In terms of code quality we can also observe a great disparity between the measures taken.

As we discussed in the chapter corresponding to the metrics, the code quality measures related to the complete projects have been taken, including libraries and native projects generated by the framework, so we will compare them at the framework level.

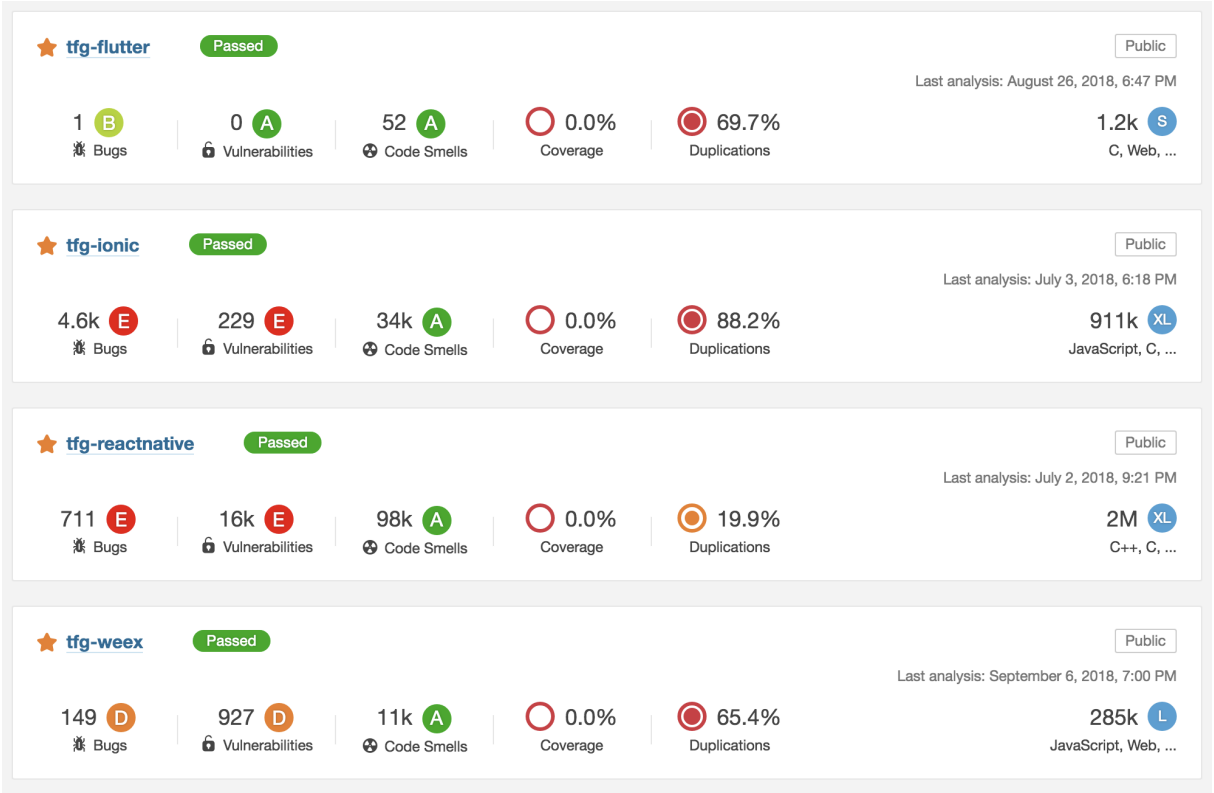


Figure 7.5: Code Quality comparison

With this in mind, we can see that the number of lines in each project differs greatly, from the 2M corresponding to React Native to the 1.2k corresponding to Flutter.

This measure can tell us the amount of code that is generated by the framework and / or corresponding to libraries. All the projects have been developed without using frameworks or libraries additional to those included by the framework itself except for [22] the performance library used to measure execution times.

As far as vulnerabilities, code smells, bugs and technical debt are concerned, React Native loses. Being a project with many more lines of code than the rest is more likely to find us with a greater number of bugs, vulnerabilities and code smells ... resulting in a technical debt for several weeks.

Another metric that can be useful is the percentage of duplication because it can give us an idea of how each framework generates the project for the native platform.

The highest percentage of duplication is found in Ionic with 88.2%, corresponding in this case in its vast majority to files generated by cordova in both platforms.

In the case of Weex, even making use of cordova its percentage of duplication is around 20% lower and in its great majority corresponds to libraries installed in node_modules.

## 7.3 Conclusions

As we have advanced to the beginning of this document, nowadays when choosing which technology to develop a mobile project we find ourselves with the dilemma of choosing between 100% native development or trying a cross-platform alternative.

This study aimed to dispel these doubts by making a comparison between the cross-platform development frameworks most used in the software development industry. Throughout it we have compared four frameworks: React Native, Ionic, Flutter and Weex. Three of them have been compared in terms of execution times and all in terms of code quality.

Once the comparisons are made, we can find a great difference between those frameworks that use directly native components and those that make use of tools such as Apache Cordova deploy their components in the form of a webview. This difference can be observed both in terms of execution times and code quality. The times obtained in Flutter are superior to Ionic in the operations that required more screen rendering. In code quality terms, the Ionic code duplication rate is one of the highest due to the libraries used by Apache Cordova.

In the case of React Native, it obtains its worst times in the operations that require more screen rendering, so in terms of scalability it is not the most optimal. In this case tests have been carried out with lists of maximum size of 1000 elements, but there may be cases in which the implementation or management of many more elements is needed, in which case, React Native would not be an advisable choice.

Having established the difference between purely cross-platform applications and hybrid applications, it is important to mention that in order to deploy 100% of the potential of a purely cross-platform framework it is necessary to have much higher knowledge of native programming for both platforms. On the contrary, in the case of hybrid applications, it is not necessary to have as much knowledge in native languages as in cross-platform, because by making full use of Javascript, good results can be achieved.

This distinction is important to make in order to lean towards a framework of one kind or another since, currently there are many programmers with extensive knowledge in Javascript but there are few programmers who have extensive knowledge of Javascript and the native languages of the two main platforms (Java / Kotlin in the case of Android and Objective-C / Swift in the case of iOS).

The same criteria can be applied in the case of Flutter, despite getting the best execution times with it. Its development language although powerful is too premature, so there are few programmers that dominate it or companies that dare to carry out a development due to the lack of tools.

In spite of all this and answering the dilemma enunciated at the beginning of this section, is a cross-platform development in front of a native one recommended?

Yes, the cross-platform development frameworks have evolved a lot since the launch of Apache Cordova so the performance differences between cross-platform and native although existing are blurring more and more. Given the advantages in development, costs and temporary estimation of development cross-platform development frameworks are an alternative to bear in mind when making a development for mobile devices.

In case of opting for a cross-platform development, the most recommendable option is a purely cross-platform framework, despite the clear failures of React Native in terms of scalability, the possibility of optimizing the code at a much lower level thanks to the native components is a clear advantage over those like Ionic that make use of WebViews.

# Chapter 8

# Conclusions and future work

En este último capítulo se detallan las lecciones aprendidas tras el desarrollo del presente proyecto y se identifican las posibles oportunidades de mejora sobre el software desarrollado.

## 8.1 Conclusions

Throughout this thesis we have compared the most used cross-platform frameworks in the software development industry. Given the current state of these technologies and their application in the industry, it was a necessary comparison.

The results of this study show that although it is currently a technology in development. Cross-platform development frameworks given its great number of advantages and more than competent performance are a very important alternative to take into account when choosing a technology to carry out the development of a project.

We hope that the results of this study will clear up doubts about what is the technological state of cross-platform frameworks and what is the current performance of the most used in the software development industry up until now.

## 8.2 Future work

In this section we will give guidelines and recommendations for the expansion of this study or future work on it.

### 8.2.1 Comparison with native applications

Throughout this study we have compared cross-platform frameworks between them, but if there is one thing that we may have left it is a comparison of the result of this study with applications developed in native languages.

That is, add to the four applications developed two in the two main native languages iOS and Android. Carry out the same measures taken during this study (execution times and code quality) and compare the result obtained with the results of this study. Obtaining an objective comparison between the main cross-platform development frameworks and the main native development languages.

### 8.2.2 Obtaining new metrics by expanding the device catalog

Given the limitations of this study, measurements could only be made on one device per system. That is, one for iOS system (iPhone 7 Plus) and one for Android system (Samsung Galaxy Note 3).

In the future, more devices would be added to the measurement catalog. In order to polish the results of the study, deviation of execution times between them could be checked.

# Bibliography


[1] Appium.io. *About Appium*. URL: http://appium.io/docs/en/about-appium/intro/ (visited on 08/25/2018).

[2] Apple and AppleInsider. *Apple App Store: number of available apps 2017 | Statistic*. URL: https://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/ (visited on 02/18/2018).

[3] *Balsamiq. Rapid and effective wireframing software. | Balsamiq*. URL: https://balsamiq.com/ (visited on 08/25/2018).

[4] Pierre Bourque and Richard E. Fairley. *Guide to the Software Engineering - Body of Knowledge*. 2014, p. 346. ISBN: 0-7695-2330-7. DOI: 10.1234/12345678. arXiv: arXiv:1210.1833v2. URL: www.swebok.org.

[5] Cordova.apache.org. *Architectural overview of Cordova platform - Apache Cordova*. URL: https://cordova.apache.org/docs/en/latest/guide/overview/ (visited on 08/25/2018).

[6] Dartlang.org. *Dart programming language*. URL: https://www.dartlang.org/ (visited on 08/25/2018).

[7] Bonnie Eisenman. *Learning React Native : building mobile applications with JavaScript*. ISBN: 9781491929070. URL: https://books.google.es/books?hl=es%7B%5C&%7Dlr=%7B%5C&%7Did=274fCwAAQBAJ%7B%5C&%7Doi=fnd%7B%5C&%7Dpg=PR2%7B%5C&%7Ddq=React+Native%7B%5C&%7Dots=tFsk9Je4r3%7B%5C&%7Dsig=ymZnmqNr1V5RTQNIOlWbAhVDDZ0%7B%5C#%7Dv=onepage%7B%5C&%7Dq=React%20Native%7B%5C&%7Df=false.

[8] Systems Engineering Standards Committee of the IEEE Computer Society. *ISO/IEC/IEEE 24765-2010(E), Systems and software engineering — Vocabulary*. Tech. rep. 2010. URL: www.iso.org.

[9] Flutter. *Architecture Diagram*. URL: https://github.com/flutter/engine/wiki.

[10] *Gantt Chart*. URL: https://en.wikipedia.org/wiki/Gantt%7B%5C_%7Dchart.

[11] Git-scm.com. *Git - A Short History of Git*. URL: https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git (visited on 08/25/2018).

[12] Git-scm.com. *Git - About Version Control*. URL: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (visited on 08/25/2018).

[13] Github.com. *GitHub Octoverse 2017 | Statitics*. URL: https://octoverse.github.com/ (visited on 08/25/2018).

[14] Manuel Rodríguez-Sánchez Guerra. *Flutter TodoApp Git Repository*. URL: https://github.com/manuelrdsg/Flutter-Todo.

[15] Manuel Rodríguez-Sánchez Guerra. *Ionic TodoApp Git Repository*. URL: https://github.com/manuelrdsg/Ionic-Todo.

[16] Manuel Rodríguez-Sánchez Guerra. *React Native TodoApp Git Repository*. URL: https://github.com/manuelrdsg/ReactNative-Todo.

[17] Manuel Rodríguez-Sánchez Guerra. *Weex TodoApp Git Repository*. URL: https://github.com/manuelrdsg/Weex-Todo.

[18] Jestjs.io. *Jest · JavaScript Testing*. URL: https://jestjs.io/en/ (visited on 08/25/2018).

[19] Ivano Malavolta. "Beyond native apps: web technologies to the rescue!" In: *Proceedings of the 1st International Workshop on Mobile Development - Mobile! 2016*. 2016. ISBN: 9781450346436. DOI: 10.1145/3001854.3001863.

[20] Ivano Malavolta et al. *Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation*. 2015.

[21] *Mobile Application Development- Web vs. Native.pdf*.

[22] Performance-now. *performance-now | npm package*. URL: https://www.npmjs.com/package/performance-now.

[23] Sonarqube.org. *Clean Code | SonarQube*. URL: https://www.sonarqube.org/features/clean-code/ (visited on 08/25/2018).

[24] Wikipedia.com. *CRUD - Create, read, update and delete*. URL: https://en.wikipedia.org/wiki/Create,%7B%5C_%7Dread,%7B%5C_%7Dupdate%7B%5C_%7Dand%7B%5C_%7Ddelete.

# Appendices

# Appendix A

# GNU Free Documentation License

Version 1.3, 3 November 2008



<http://fsf.org/>



**Preamble**

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of

the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

## 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

## 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.