# Empirical Evaluation of the Impact of Mobile Development Frameworks on the Application Source Code Quality

by

Parsa Karami

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN SOFTWARE ENGINEERING
M.A.Sc.

MONTREAL, FEBRUARY 22ND, 2023

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

## ACKNOWLEDGEMENTS

I would like to sincerely thank my supervisor, Mrs. Ghizlane El Boussaidi, for the research opportunity and for her kind and great support throughout accomplishing my master's. It was a great experience to work with her, and every meeting we had was an opportunity for me to learn from her. This is an opportunity I will never forget.

I would like to thank the members of the jury to accept to judge my work.

I would like to thank my family, especially my lovely mom, my dad, my brother, and my dear sister for their endless support and unconditional kindness throughout my life. I would also like to thank my amazing cousin Sara, her husband Farhad, and her son Pakan for their welcoming support when I came to Canada. Also, I would like to thank my friends Ehsan, Vahid, and Javad for their support during the fulfillment of my master's.

Finally, I would like to thank all my teachers and every person who supported me and encouraged me in this journey.

**Évaluation empirique de l'impact des frameworks de développement mobile sur la qualité du code source**

Parsa Karami

## RÉSUMÉ

Cette thèse a examiné expérimentalement si les frameworks de développement mobile affectent la qualité du code source. Dans un premier temps, une revue systématique de la littérature a été menée dans cette étude pour trouver les frameworks, les perspectives d'évaluation, les paramètres, les méthodes et les lacunes de recherche les plus étudiés dans les études existantes. Nous avons observé qu'aucune des études survolées n'utilisait un grand nombre d'applications mobiles dans leur évaluation et que les critères d'évaluation n'incluaient pas de critères sur la qualité du code source. En conséquence, nous avons conçu une étude empirique qui utilise un grand nombre d'applications mobiles pour évaluer l'impact des frameworks sur la qualité des applications. En particulier, nous avons comparé un certain nombre de métriques d'applications mobiles développées à l'aide du framework Android à celles d'applications mobiles développées à l'aide du framework React Native. Notre enquête empirique indique que le choix d'un framework mobile peut avoir un impact sur la qualité du code source.

**Mots-clés:** Développement mobile, Analyse d'applications mobiles, Framework mobile, Développement natif, Framework multiplateforme, Qualité du code source

# Empirical Evaluation of the Impact of Mobile Development Frameworks on the Application Source Code Quality

Parsa Karami

## ABSTRACT

This thesis examined experimentally whether mobile development frameworks affect the source code quality. As a first step, a systematic literature review was conducted in this study to find the most studied frameworks, evaluation perspectives, metrics, methods, and research gaps in existing studies. We observed that none of the surveyed studies used a large set of mobile applications in their evaluation and that the evaluation criteria did not include criteria about source code quality. As a result, we designed an empirical study that uses a large set of mobile applications to evaluate the impact of mobile frameworks on the quality of the applications. In particular, we compared a number of metrics of mobile applications developed using the Android Native framework to metrics of mobile applications developed using the React Native framework. Our empirical investigation indicates that choosing a mobile framework can impact the application source code quality.

**Keywords:** Mobile Development, Mobile application analysis, Mobile Framework, Native development, Cross-platform framework, Source code quality

## TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| API | Application Programming Interface |
| CPDT | Cross-Platform Development Tool |
| ETS | École de Technologie Supérieure |
| IDE | Integrated Development Environment |
| JSON | JavaScript Object Notation |
| LOC | Lines of Code |
| POV | Point of View |
| PWA | Progressive Web Application |
| RS | Research Question |
| SAAS | Software As A Service |
| SDK | Software Development Kit |
| SLOC | Source Lines of Code |
| SLR | Systematic Literature Review |

# INTRODUCTION

## 0.1 Research Context

Mobile applications have become very popular in recent years, and smartphone users continue to increase yearly (Bankmycell (2022)). Based on the information from the Statista website, the number of smartphones users reached 6.56 billion in 2022 (Statista (2022b)). This number indicates a high potential market for developing mobile applications. A total of 230 billion mobile applications have been downloaded by 2021 (Statista (2022a)). The smartphone's market has had several operating systems in the last decade, like BlackBerry, Symbian, Windows Mobile, Windows Phone, Android, and iOS. Some of those, including Symbian and Microsoft Windows Mobile, are discontinued. Apple's iOS and Google's Android dominate the mobile operating systems market share, with over 99 percent of the global market in the last decade (Statista (2022c)). According to the StatCounter website, the market share is split into two operating systems. Android, with 71.85% of the market share, is first, and iOS, with 27.5%, is the second most used smartphone platform (StatCounter (2022)). Thus, developers need to develop their mobile applications targeting both platforms to reach a maximum number of users.

In addition to the mobile platform fragmentation[1], we face more fragmentation in mobile application development even if developers target only one mobile platform to develop an application, fragmentation is still a problem. (Joorabchi, Mesbah & Kruchten (2013)) identified two types of fragmentation in the mobile development field, fragmentation within the same platform and fragmentation across platforms. Although developers target a single platform to develop mobile applications, they face various device types, including smartphones, tablets, smart watches, smart TVs, cars, and more. Each of these device types has its own set of hardware,

[1] Mobile platform fragmentation refers to when users are running different mobile operating systems (or different versions of an OS).

screen size, and support of operating system functionality. Therefore, this fragmentation is a challenge for mobile developers aiming to reach all potential users in the market.

### 0.1.1 Mobile development framework

Mobile frameworks are software frameworks that provide fundamental structures for developing mobile applications (Wikipedia (2022)). Mobile frameworks enable developers to overcome fragmentation and access platform features. Operating systems vendors and other software companies have introduced a variety of mobile frameworks since the advent of mobile development. Some frameworks are more used in practice and popular among developers, and some are used only in academia to produce experimental prototypes. In addition, the frameworks differ in terms of performance, access to hardware and software features on platforms, and the ability to develop cross-platform applications (Humayoun, Ehrhart & Ebert (2013)). The number of mobile frameworks has increased significantly in recent years. Each framework is based on a set of technologies (e.g., programming languages) and has specific advantages. For example, the Android Native framework delivers the best overall performance (Willocx, Vossaert & Naessens (2015)), while ReactNative reduces development time. To create mobile applications, mobile developers need to choose the appropriate mobile framework, and finding the proper framework to meet their needs can be challenging.

Typically, there are two approaches to developing mobile applications. The first approach is called Native. It uses programming languages, development tools, and environments specific to a mobile operating system (Biørn-Hansen, Grønli & Ghinea (2018)). For instance, to create a native application for the iOS platform, developers use Swift or Objective-C programming languages inside XCode IDE. The second technique is Cross-Platform development. These frameworks let you develop an application that can be installed and executed on more than one mobile platform. These frameworks usually use general-purpose programming languages like JavaScript or Dart. As opposed to native frameworks, programming languages, and development tools for the cross-platform approach are not typically provided by mobile operating system vendors. The following sections will explain the characteristics of the different categories of development frameworks that we identified.

### 0.1.2 Native development

A native framework uses a software development kit (SDK) and specific development tools to create a mobile application. The operating system vendor provides the SDK. Native applications have full access to platform functionalities. These applications deliver the highest performance and user experience (Jia, Ebone & Tan (2018)). However, native frameworks do not support code reuse because the developed code targets a unique platform. Consequently, it increases the development costs and time compared to a cross-platform framework (Jia *et al.* (2018)).

### 0.1.3 Cross-platform development

Cross-platform frameworks enable developers to build mobile applications that target multiple mobile platforms with the same or at least similar structured code. Using a shared codebase increases reusability and decreases the time and cost of development(Jia *et al.* (2018)). Nevertheless, the cross-platform frameworks have noteworthy shortcomings. From the user's point of view, cross-platform applications generally deliver lower performance in comparison to native applications (Willocx *et al.* (2015)). In addition, cross-platform mobile frameworks affect energy consumption. Regardless of how a cross-platform framework generates an application, the application leads to consuming more energy (Ciman & Gaggi (2017)). Existing cross-platform approaches can be classified into different categories depending on the techniques they use ((Xanthopoulos & Xinogalos (2013)), (Majchrzak, Biørn-Hansen & Grønli (2017)), (Biørn-Hansen *et al.* (2018)), and (El-Kassas et al. (2015))). We briefly discuss here the major cross-platform approaches.

#### 0.1.3.1 Hybrid

The Hybrid approach uses standard Web technologies to produce cross-platform mobile applications. The Hybrid strategy allows Web developers to create a mobile application using their knowledge in Web development. This approach internally uses a browser engine that renders Web contents inside a native Web container (UIWebView in iOS and WebView in Android) on the device (Latif, Lakhrissi, Nfaoui & Es-Sbai (2016)). This, developers are able to package developers package a project written using HTML5, CSS, and JavaScript into a mobile application installation file. The Hybrid technique allows developers to access devices' sensors and platform APIs. For instance, functionalities like GPS, contact lists, and Bluetooth are accessible. A drawback of this approach is the user interface behavior. Since it renders the user interface from the HTML elements inside a WebView component, the user interface and controls are not similar to native ones. The most popular Hybrid framework is PhoneGap[2]. The Hybrid approach is also referred to as a Web-To-Native wrapper. A native Web container is used to run Web application files and access device features (Willocx, Vossaert & Naessens (2016)).

#### 0.1.3.2 Interpreted

The Interpreted approach utilizes the JavaScript programming language to let developers produce applications. The Interpreted applications render platform-native user interfaces via an on-device JavaScript interpreter (Majchrzak *et al.* (2017)). The JavaScriptCore[3] in iOS and the V8[4] in the Android operating system are the two most used engines for interpreting JavaScript code. The main difference between the Hybrid and Interpreted approaches is the execution environment. The Hybrid approach uses the WebView component to render the user interface element, and

[2] https://experienceleague.adobe.com/docs/experience-manager-64/mobile/developphonegap/developing-in-phonegap.html

[3] https://developer.apple.com/documentation/javascriptcore

[4] https://v8.dev

the user interface elements are HTML and CSS. The Interpreted approach uses an on-device JavaScript interpreter engine to render user interface elements. In addition, the JavaScript interpreter is an abstraction layer that allows the application to communicate with the device and platform's native APIs. One of the most famous framework in this approach is React Native[5].

#### 0.1.3.3 Cross-compiled

The Cross-Compiled is more similar to the native approach in terms of performance and user interface (Willocx *et al.* (2015)). In this approach, the project source code is compiled into native byte code. Hence, the compiled file executes on the target platform as a native application. Unlike the interpreted and hybrid approaches, there is no abstraction layer to communicate with the device and platform APIs (Biørn-Hansen *et al.* (2018)). Thus, the performance of this approach seems to be higher than the aforementioned cross-platform approaches, and the approach is preferred for developing CPU-intensive applications (Willocx *et al.* (2016)). The Xamarin[6] is one of the most used frameworks among cross-compiled frameworks. It compiles project source code (C#) to native byte code instead of a common language. This compilation technique lets developers use all platform and device features like native frameworks. Moreover, the native compilation method generates user interfaces that are rendered as the native interface components (Willocx *et al.* (2016)). However, the size of the application installation file developed using frameworks of this approach is larger than the application developed using native frameworks. (Willocx *et al.* (2016)).

#### 0.1.3.4 Progressive Web Apps (PWA)

The progressive Web application (PWA) is a type of mobile Web application. It uses the same technologies and programming languages used to develop Web applications. PWAs are

[5] https://reactnative.dev

[6] https://dotnet.microsoft.com/en-us/apps/xamarin

executing in the platform's Web browser. There are some differences compared to a simple Web application that you run using a mobile browser. For instance, users can run it from the home screen and the address bar at the top is hidden when running a PWA (Biørn-Hansen *et al.* (2018)). In addition, the application is available offline. The PWA development frameworks also support APIs like device storage, handling push notifications, and location. These features improve the application's user experience. However, a PWA implementation cannot access all native hardware and platform APIs. Angular[7] is one of the most used frameworks to build PWAs.

## 0.2 Statement of the Problem

Developers use different mobile frameworks to develop mobile applications. Each framework has pros and cons that can impact the application design quality. For instance, one of the problems of mobile developers, designers, and engineers encounter is selecting the appropriate framework to develop applications (Biørn-Hansen *et al.* (2018)). This choice might impact different aspects of application quality. For example, using cross-platform frameworks over native frameworks increases energy consumption (Ciman & Gaggi (2017)) and negatively impacts the overall performance (Biørn-Hansen, Rieger, Grønli, Majchrzak & Ghinea (2020)). Using a cross-platform framework also limits using the full capabilities of the platform's feature (Joorabchi *et al.* (2013)).

Some studies compared the impact of choosing mobile frameworks on application quality (e.g., (Jia *et al.* (2018)), (Ciman, Gaggi & Gonzo (2014)), (Humayoun *et al.* (2013)) , (Corbalan *et al.* (2018)), (Sommer & Krusche (2013))). The purpose of most of these studies is to evaluate this impact from the end user perspective. None of these studies has analyzed the impact of the frameworks on the quality of the application's source code. Moreover, all experimental studies used prototype applications, and none of these used a large number of real mobile applications

[7] https://angular.io

to evaluate the studied frameworks. Thus, to fill these gaps, this thesis aims to examine the impact of mobile frameworks on application source code quality through the analysis of a large set of real mobile applications.

## 0.3 Goal of the thesis

The main goal of this thesis is to investigate the impact of choosing a mobile development framework on application source code quality. We divided our main goal into two subgoals.

The first sub-goal is reviewing how other researchers compared mobile frameworks and what are their evaluation results. The motivation of this sub-goal is to find out what are the most studied frameworks and the evaluation perspectives, comparison metrics, and methods used by researchers.

The second sub-goal is to design and execute an empirical investigation to evaluate the impact of choosing mobile development frameworks on application source code quality. In particular, we are interested in the evaluation of the impact of mobile frameworks on the bugs and the code smells found in mobile applications. By performing the empirical study, we aim to address the thesis's main goal.

## 0.4 Research Methodology

To achieve the thesis goal, we follow the research methodology depicted in the figure 0.1.

### Phase-1: Understanding the Research Context

In the first phase, we studied relevant papers on mobile development to gain a better understanding of the research context. During this stage, we learned about mobile development frameworks in general. Afterward, we studied both approaches for developing mobile applications, the native approach and cross-platform approach to understand the differences between approaches. Then we examined the different types of cross-platform mobile frameworks to understand the pros and cons of each type.

**Phase-2: Systematic Literature Review**

In the second phase we carried out a systematic literature review (SLR) to find the research gaps in the comparison studies of mobile development frameworks. The findings from the SLR will help us to identify the gaps in existing work.

**Phase-3: Empirical study**

In this phase, we carried out an empirical study on a large set of real mobile applications to achieve the thesis main goal. We conducted the empirical study considering the research gaps identified by SLR.

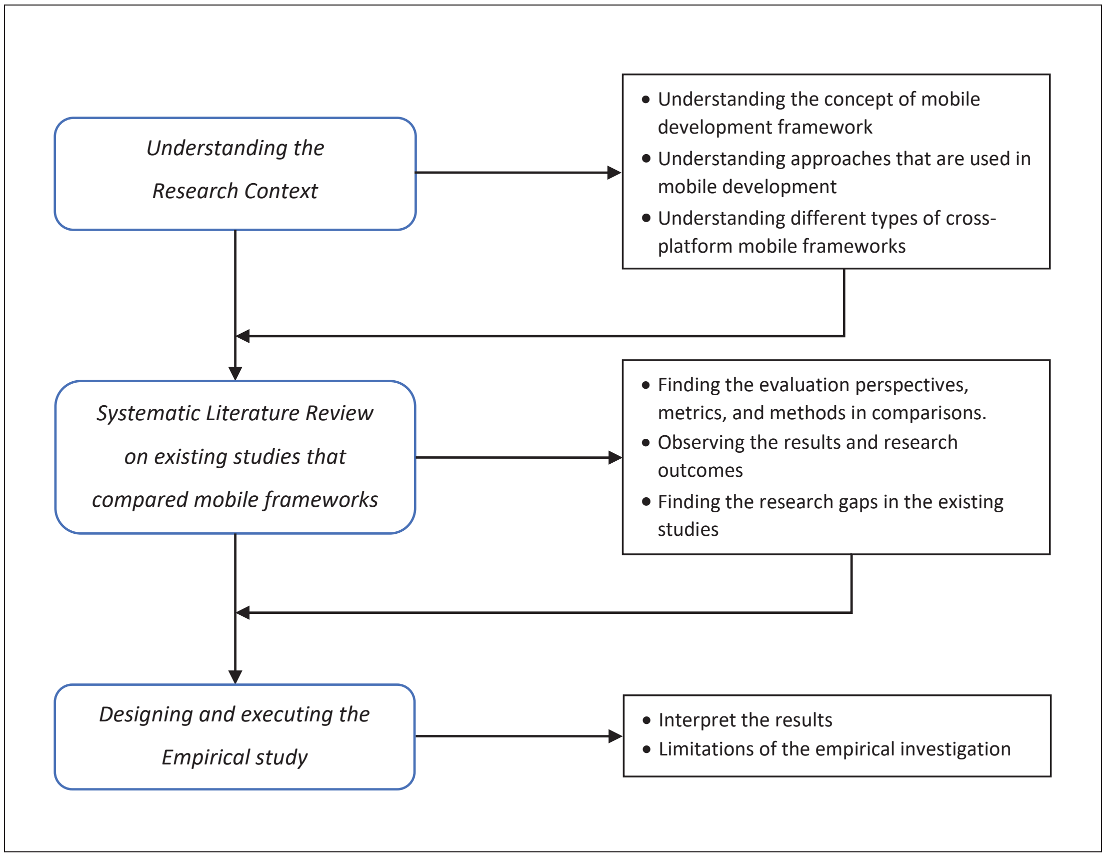

Understanding the Research Context
Understanding the concept of mobile development framework
Understanding approaches that are used in mobile development
Understanding different types of cross-platform mobile frameworks
Systematic Literature Review on existing studies that compared mobile frameworks
Finding the evaluation perspectives, metrics, and methods in comparisons.
Observing the results and research outcomes
Finding the research gaps in the existing studies
Designing and executing the Empirical study
Interpret the results
Limitations of the empirical investigation

Figure 0.1 Research methodology

## 0.5 Organization of the Thesis

This thesis is organized as follows. Chapter 1 presents the SLR on studies that compared mobile development frameworks, and discusses its results. Chapter II presents the empirical study we carried out, as well as its results. Finally, we conclude and discuss the limitations of the research, and recommendations for future research.

# CHAPTER 1

# A SYSTEMATIC LITERATURE REVIEW OF WORK ON THE COMPARISON OF MOBILE DEVELOPMENT FRAMEWORKS

To conduct our systematic literature review, we adopted the approach proposed by (Kitchenham (2004)). Our SLR's methodology included the following steps:

1. Identifying the research questions
2. Defining the search strategy for primary studies
3. Selecting the studies
4. Extracting data from the selected studies
5. Synthesizing and interpreting the results

We present each step in the following subsections.

## 1.1 Research questions

The goal of our systematic literature review is to collect information and characterize the existing works on the comparison of mobile frameworks. Our SLR aims at answering the following research questions:

- RQ1. What are the most studied mobile application development frameworks?
- RQ2. What are the goals of existing studies on mobile frameworks?
  - RQ2.1. What are the criteria used to evaluate the studied frameworks?
  - RQ2.2. Which of the end-user or developer's perspectives are targeted by existing studies?
- RQ3. How did existing studies carry out the evaluation of the frameworks?
- RQ4. What is the gap between the literature on mobile frameworks and the practice?
  - RQ4.1. What are the frameworks that are currently used by mobile developers?
  - RQ4.2. Do existing studies target the frameworks used in practice?

To answer research questions 1, 2, and 3, we will rely on information extracted from existing studies on mobile development frameworks. To answer research question 4, we hypothesized that the frameworks used in practice are those for which developers frequently ask questions on StackOverflow. We decided to use *StackOverflow*[1] data since it is the largest question-based community for developers. Thus, we compare information extracted from Stackoverflow to the information extracted from the reviewed literature.

## 1.2 Search strategy

To identify and select the primary studies for our SLR, we used our research questions to define a series of search strings. Afterward, we executed the search string on selected digital libraries. In particular, we used Engineering Village[2] for finding the relevant papers. Engineering Village indexes papers from prevalent digital libraries that provide peer-reviewed software engineering articles (e.g., ACM, IEEE, Scopus, etc.). For the search string, we extracted keywords from the research questions. We built multiple search strings with different combinations of these keywords and tried these search strings in order to reach a comprehensive collection of papers in this context. We used the search string that yielded the most relevant results:

```
(mobile) AND (application) AND (development) AND (framework OR frameworks OR
approach OR approaches) AND ("cross-platform" OR "crossplatform" OR "cross platform"
OR "multiplatform" OR "multi-platform" OR "multi platform")
```

The result of the search included a large number of papers. Thus, we defined inclusion and exclusion criteria to narrow the search to relevant studies. Table 1.1 lists the inclusion and exclusion criteria that we used.

[1] https://stackoverflow.com

[2] https://www.engineeringvillage.com

Table 1.1 Inclusion and exclusion criteria to filter out the studies

| Inclusion criteria | Exclusion criteria |
|---|---|
| Papers that provide analysis and comparison of mobile frameworks | The studies that did not compare mobile frameworks |
| Publication year must be between 2001 and 2021 | Papers that were not published in the English language |

## 1.3 Study selection

The study selection procedure was carried out according to the search strategy. The following items show the study selection steps:

- Searching using the search string on the engineering village digital libraries. We performed our search query between 5th April 2021 and 20th April 2021.
- Filtering the papers that were published in the english language.
- Selecting the articles that were published between 2001 and 2021.
- Excluding duplicate records
- Filtering the articles based on the title and abstract by reading manually. We identified forty nine (49) relevant papers at the end of the manual screening.
- Carrying out a snowballing process which allowed us to identify twelve additional studies.
-

The final number of primary studies for our SLR was sixty one (61). The primary studies are listed in Appendix I. Figure 1.1 shows the process and the resulting number of studies for each step.

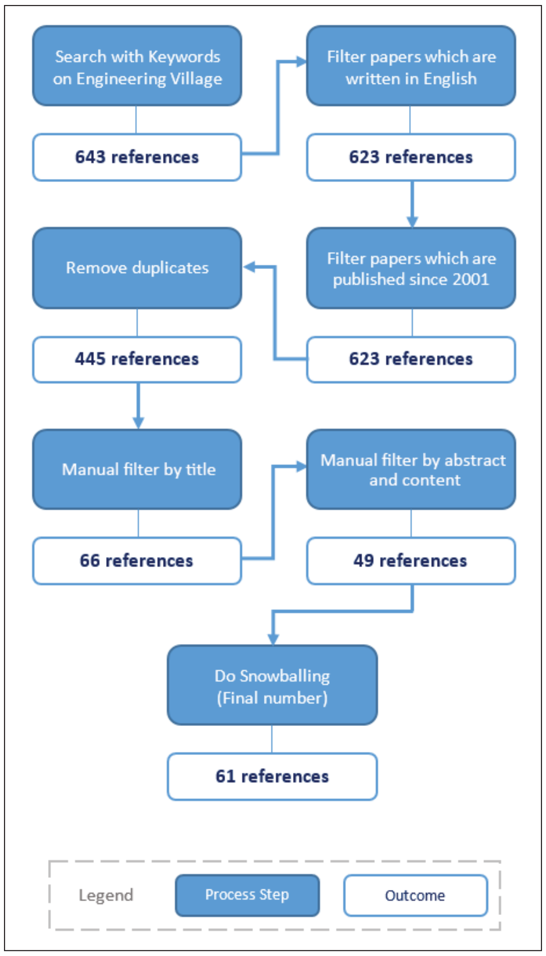

Search with Keywords on Engineering Village
643 references
Filter papers which are written in English
623 references
Filter papers which are published since 2001
623 references
Remove duplicates
445 references
Manual filter by title
66 references
Manual filter by abstract and content
49 references
Do Snowballing (Final number)
61 references
Legend
Process Step
Outcome

Figure 1.1 Overview of the selection process and its results

## 1.4 Data Extraction

The selected studies have gone through a full-text review. Relevant information was extracted and recorded in a predefined form to get a general perception of the studies. We created a guideline document defining each of the classification terms and a spreadsheet to collect the classification information. For each study, the spreadsheet contained the following data fields:

- Criteria that were used in the evaluation such as performance, usability, etc
- The definition for each criterion
- Source of the criteria, for example some studies extracted the criteria from some quality model (e.g. FURPS).

- The points of view or perspectives from which the criteria were analysed. The point of view could be the end user or developer's point of view for instance
- The mobile operating systems that are targeted
- The evaluation method used by the study to compare the studied frameworks. The method can be based on some practical development experience or just on documentation of the frameworks
- The features of the prototype application used for evaluation if the evaluation method is experimental.
- The targeted mobile application development frameworks
- The results of comparison for each criterion

### 1.4.1 StackOverflow Data

To answer our fourth research question, we need to find out the popular mobile frameworks in practice. To do so, we hypothesized that the frameworks used in practice are those for which developers frequently ask questions on *StackOverflow*. Hence, the number of questions should indicate which frameworks are used in practice. Thus, we extracted questions asked on *StackOverflow* that had a tag referring to a mobile development framework. Each question has tags that identified the category and the related programming language or software framework to that question. We recovered the number of questions attached to those tags from 2016 to 2020 for the mobile frameworks. For instance, we found the total number of questions attached to the "xamarin" tag for the Xamarin framework.

## 1.5 Results

### 1.5.1 RQ1 Results. What are the most studied mobile application development frameworks?

Figure 1.2 shows the top ten most analyzed mobile application development frameworks in the surveyed studies. Based on the results the PhoneGap is the most studied mobile application development framework. Android Native framework is in the second position. According to the data we extracted from the surveyed studies, thirty one from all sixty one papers (more than 50%) studied at least a native framework or approach in their comparative analysis.

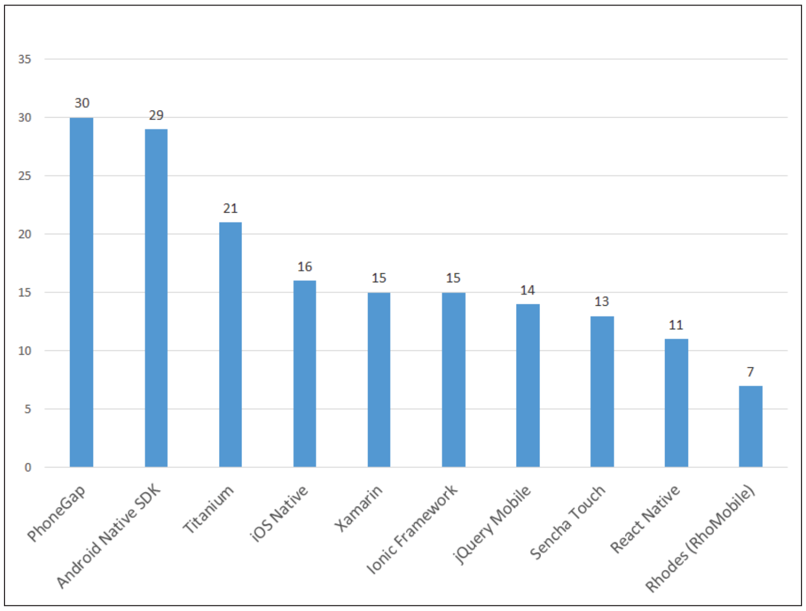


Figure 1.2    Top ten most studied mobile frameworks

Since the native frameworks can deliver applications with the highest compliance with platform vendors, the native frameworks were used as a baseline in some comparisons like (Ciman & Gaggi (2017)) and (Fahlesson & Johansson (2013)). If we only want to focus on cross-platform frameworks, we need to look at targeted cross-platform approaches by surveyed papers. The hybrid approach with 49 occurrences in papers has the highest focus among cross-platform approaches. 80.3% of papers use at least one hybrid framework in their studies. After that, The Interpreted approach, with 36 occurrences and 59% is in second place. Table 1.2 indicates the percentage and number of occurrences of each cross-platform approach among all the reviewed papers.

Table 1.2 Occurrences of studied cross-platform approaches in papers

| **Cross-Platform approach** | **Number of occurrences** | **Percentage** |
|---|---|---|
| Hybrid | 49 | 80.3% |
| Interpreted | 36 | 59% |
| Cross-Compiled | 33 | 54% |
| Web | 24 | 39.3% |
| PWAs | 5 | 8.1% |
| Model Driven | 5 | 8.1% |

When we reviewed studies, we also analyzed the evolution of the focus of the surveyed studies over the years. Analyzing this evolution also helps to compare our findings in using the mobile frameworks by developers. Hence, we performed a trend analysis to find out if a pattern exists. The number of published papers in a specific year can be higher than in other years, which might affect the trend analysis result. Therefore, we divided studies into two groups: the first group contains papers published from 2011 to 2015, and the second group covers studies published from 2016 to 2020. Figure 1.3 shows each framework the number of studies for the two periods (from 2011 to 2015, and from 2016 to 2020). The number of studies in the first group is 32, and in the second group we have 29 papers.

The time frames show a meaningful change in targeting mobile frameworks in the studies published between the two-time frames. Newly released frameworks are coming in front of the old ones. It also shows, the targeting of the Ionic framework and React Native in studies significantly increased in recent years. Since the initial release of these frameworks was in recent years, no paper selected them as targeted frameworks. The PhoneGap was the most studied framework in all surveyed studies, targeted more from 2011 to 2015. This decremented trend also applies to the additional older frameworks like Titanium, Sencha Touch, jQuery Mobile, standard Web languages (HTML, CSS, and JavaScript) which are used to develop mobile Web

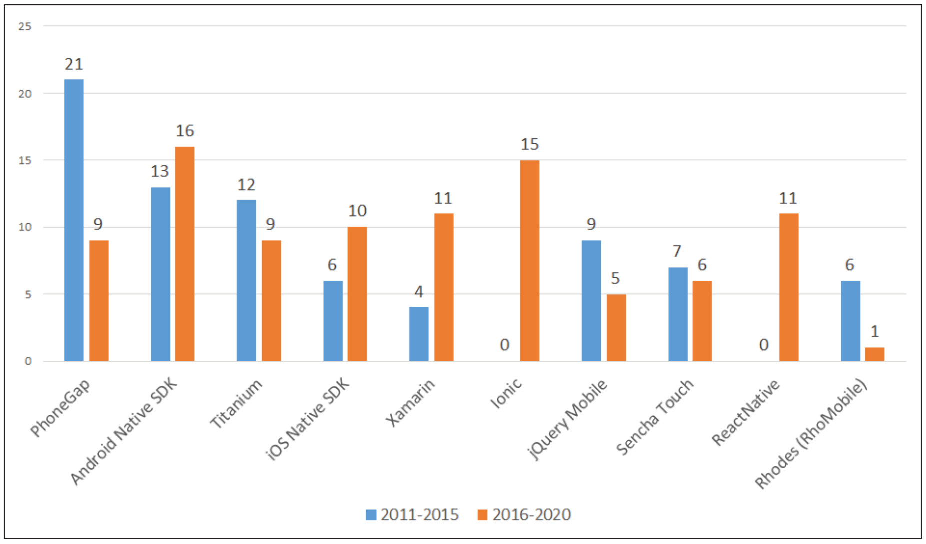


Figure 1.3 Comparison of the selected frameworks in studies over a period of five years

applications, and the Rhodes frameworks. The main reason for this notable drop in Rhodes is a large margin in the release cycle for the framework. It leads to losing the developers' tendency to use the framework. On the other hand, we think rising of Progressive Web applications (PWAs) which are web apps with enhanced capabilities (Biørn-Hansen *et al.* (2018)), causes less use of the traditional mobile website.

Figure 1.4 shows the number of studies per targeted operating system (OS). The trend is consistent with the fact Android and iOS are the dominant OS in the market.

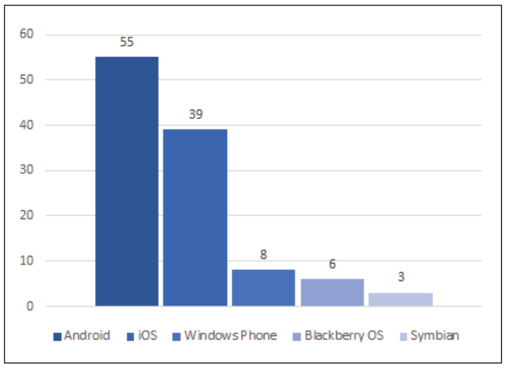


Figure 1.4 Distribution of the targeted OS in the studies

### 1.5.2 RQ2 Results. What are the goals of existing studies on mobile framework?

The main motivation of this research question is to find out what criteria were used in the reviewed studies and from which perspective (user, developer).

**RQ2.1** The first sub-question is about the criteria used to analyze frameworks. Figure 1.5 indicates the number of criteria occurrences in the reviewed studies. According to this figure, the performance criterion with 37 occurrences (60%) is the most considered in all reviewed studies. After the performance, the platform's APIs accessibility and the hardware and sensors accessibility with 36 occurrences (59%) are in the second place of most analyzed criteria. The criteria definition are given in Appendix II.

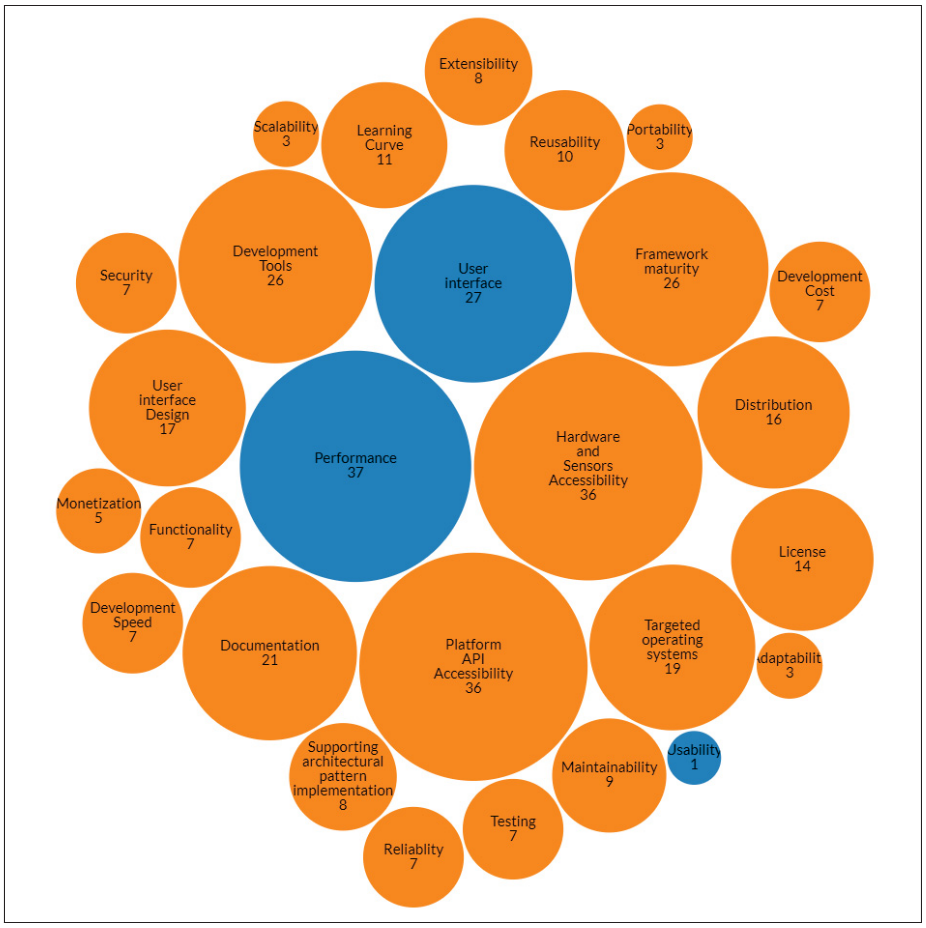


Figure 1.5 Distribution of criteria in studies

**RQ2.2** The motivation of the second sub-question is to find out which perspectives (end-user or developer) are targeted in the comparison studies. We divided the criteria into two groups. The criteria in the first group are used to analyze the mobile frameworks from the end users' perspective. The criteria in the second group are used in order to evaluate frameworks from the developer's perspective. The blue bobbles in Figure 1.5 show the criteria from the user's perspective, and the orange bobbles indicate the criteria from the developer's perspective. As shown by Figure 1.5, most of studies focused on criteria related to the developer's perspective.

Figure 1.6 indicates the studies' points of view distribution by papers. 51%(31 papers from a total of 61) have used criteria from users' and developers' points of view. Then, 26% have used only criteria from developers' points of view in their comparison. And Finally, 23% have analyzed frameworks using criteria from users' points of view.

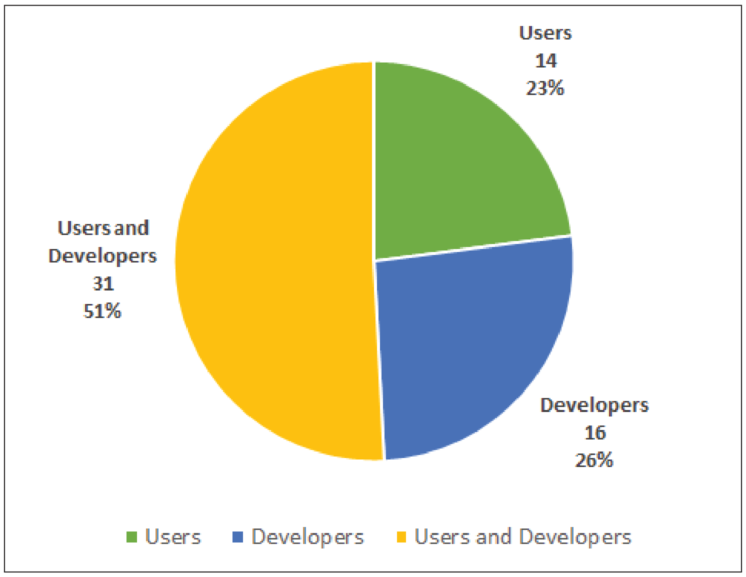


Figure 1.6 Distribution of studies POVs by papers

Most of the time to compare mobile frameworks, the reviewed papers used more than one criterion. Figure 1.7 indicates the distribution of number of the criteria used in studies. The primary study P6 with the 24 criteria used the largest set of criteria in its analysis. These 24 criteria contain two criteria from end user's perspectives and 22 criteria from developer's point of view. Another insight that the result indicates is that most of the studies used a small set of criteria in their comparison. It indicates fifty papers from the total(81%) used less than ten criteria for comparing mobile frameworks.

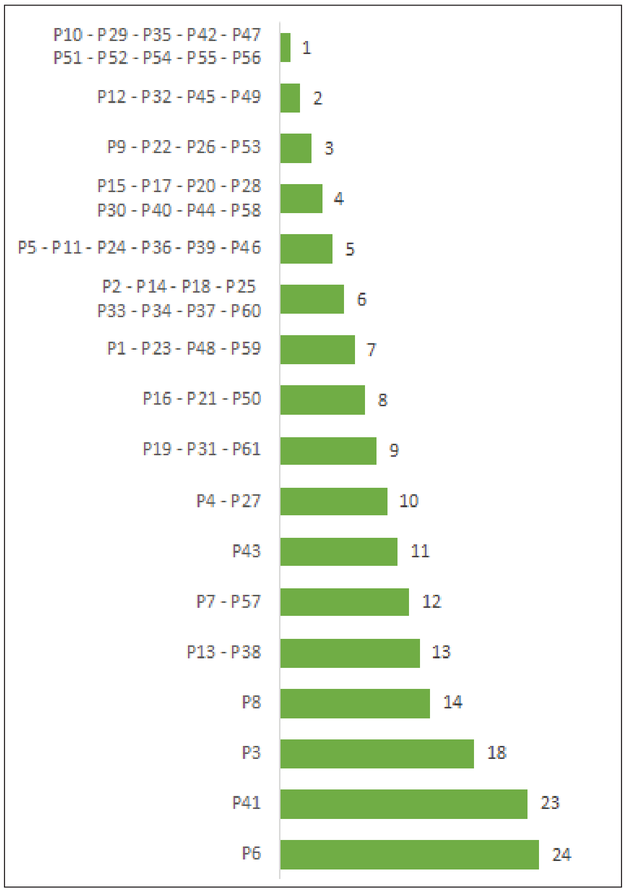


Figure 1.7 Distribution of the number of used criteria in reviewed papers

Figure 1.8 shows the number of occurrences of criteria from end user's points of view. In Appendix II, Table III-1 provides a list of the papers that use these criteria of this group in their comparison.

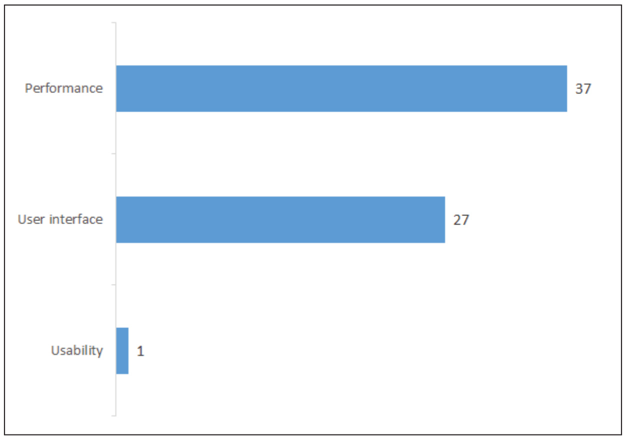


Figure 1.8  Number of criteria occurrences from end user's point of view

The performance criterion with 37 is the most analyzed criteria in comparison. The performance in most cases refers to the application speed, application installation size, and resource usage. Table 1.3 shows the sub-criteria of the performance criterion and the primary studies that used them.

Table 1.3  The performance criterion sub-items

| **Sub-criterion** | **Paper IDs** |
|---|---|
| Application speed | P3 - P5 - P6 - P7 - P8 - P9 - P12 - P13 - P35 - P41 - P42 - P43 - P44 - P49 - P51 - P52 - P54 - P55 - P56 - P61 |
| Overall performance | P16 - P23 - P30 - P32 - P37 - P40 - P46 - P48 - P57 - P60 - P61 |
| Application size | P5 - P7 - P9 - P13 - P14 - P28 - P51 - P52 - P55 - P56 |
| RAM usage | P5 - P7 - P9 - P10 - P27 - P35 - P45 - P51 - P52 |
| Energy consumption | P10 - P27 - P29 - P42 - P47 - P49 |
| CPU usage | P10 - P27 - P35 - P42 - P45 - P51 |
| Resources consumption | P28 - P49 - P60 |
| Smoothness of animation running | P8 - P45 |
| GPU usage | P45 |

Figure 1.9 shows the number of occurrences of criteria from developer's point of view. The platform API accessibility and hardware and sensors accessibility with the same number of occurrences are the most analyzed criteria in this group. In Appendix IV, Table A provides a list of the papers that used developers' criteria in their evaluation.

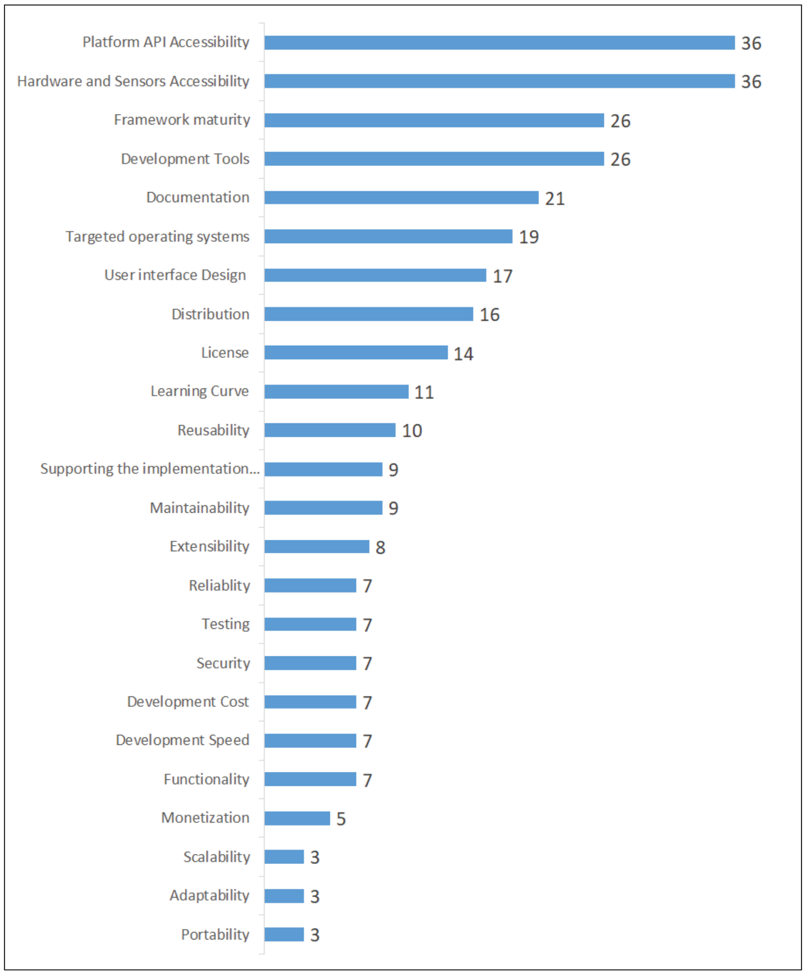


Figure 1.9 Number of criteria occurrences from developer's point of view

The functionality criterion regroups different sub-criteria in the developer's perspective. Table 1.4 lists the functionality sub-criterion and the primary studies that used them.

Table 1.4 Functionality sub-criterion

| **Sub-criterion** | **Paper IDs** |
|---|---|
| Intra/Inter application communication | P6 - P13 - P41 |
| Internationalization | P6 - P13 |
| Input/Output device connectivity | P6 - P41 |
| Supporting multi threading | P6 - P41 |
| Offline running | P39 - P57 |
| Biometric or voice authentication | P6 |
| Support instant-on experience | P6 |
| Support in-app browser | P38 |
| Support the implementation of PWA | P43 |

### 1.5.3 RQ3 Results. How did existing studies carry out the evaluation of the frameworks?

The goal of this research question is to analyze the evaluation methods used in reviewed papers. We identified four evaluation methods in the primary studies. The first evaluation method is *experiment based*. In this type of evaluation, the authors of the paper developed prototype applications with the frameworks. Then, they analyzed the frameworks according to some metrics using the prototype applications. The second type of evaluation is *documentation based*. In this type, the comparison of development frameworks is done using frameworks' documentation. The third evaluation method is a combination of the first and second methods. In this case, authors take advantage of using both prototype applications and frameworks' documentation. The last evaluation method is based on *questionnaire and user reviews*. In this case, the comparison of frameworks and developed applications is usually done based on the users' or developers' thoughts, opinions, and preferences. Questionnaires and users' comments in application stores are the two primary data sources in this evaluation method.

Table 1.5 indicates the number and percentage of evaluation methods in the studies, and figure 1.10 shows the distribution of the evaluation methods by the reviewed papers. The most used evaluation method is the Experiment based. Thirty-seven (37) out of 61 surveyed studies use this method. The least used evaluation method is the questionnaire and user reviews. Applying the experimental evaluation method needs developing prototype applications to compare mobile frameworks. However, in most cases, the prototype applications are simple. Since this method compares mobile development frameworks in practice, it is the most reliable evaluation method.

Table 1.5 Distribution of evaluation methods in papers

| **Evaluation Method** | **Number of occurrences** | **Percentage** |
|---|---|---|
| Experiment based | 37 | 61% |
| Documentation based | 17 | 28% |
| Combination of Experiment and Documentation | 4 | 6% |
| Questionnaire and User Review based | 3 | 5% |
| Total | 61 | 100% |

Moreover, Figure 1.10 shows the distribution use of evaluation methods by the reviewed papers.

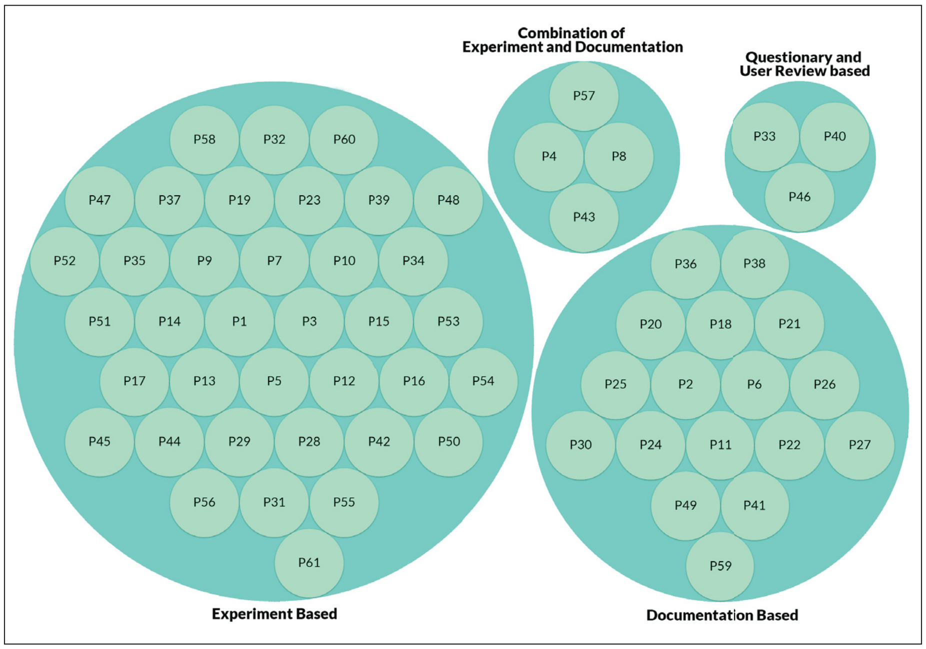


Figure 1.10 Distribution of evaluation methods in reviewed papers

We also observed that studies with a fewer number of criteria use experimental methods like P10 and P42. Moreover, in most cases, these studies use *performance* as the criterion for comparison. Figure 1.11 shows the distribution of the evaluation methods per criteria. Except for performance, there isn't a correlation between the criterion and the evaluation method.

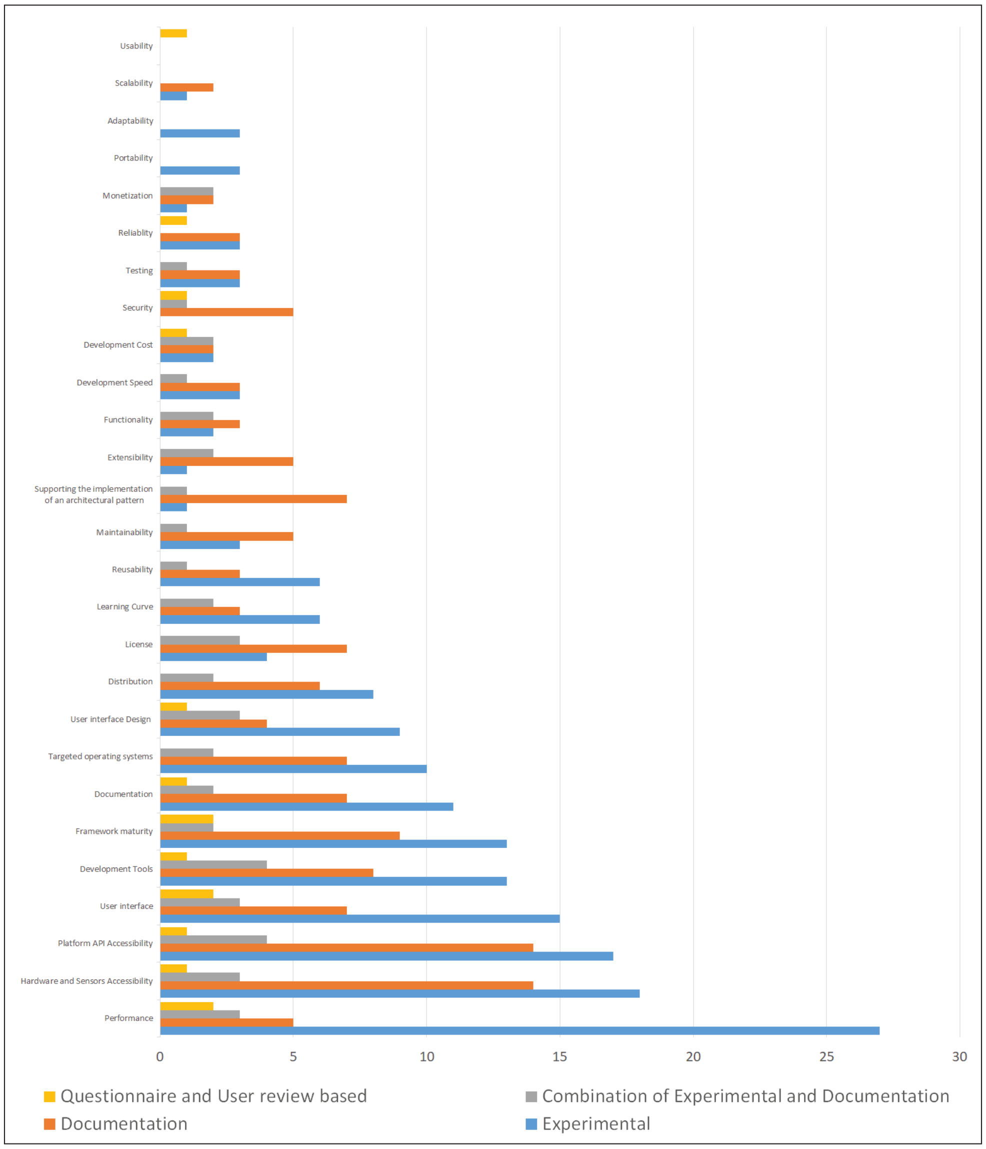


Figure 1.11 Occurrences of criteria by evaluation methods

### 1.5.4 RQ4 Results. What is the gap between the literature on mobile frameworks and the practice?

As stated in section 1.1, this question was further refined into two sub-questions:

- **RQ4.1.** What are the frameworks that are currently used by mobile developers?
- **RQ4.2.** Do existing studies target the frameworks used in practice?

To answer these questions, we used data extracted from StackOverflow. We extracted questions asked on StackOverflow that had a tag referring to a mobile development framework. We recovered the number of questions attached to those tags from 2016 to 2020.

**RQ4.1.** The first sub-research question addresses the most used mobile frameworks in practice. Figure 1.12 denotes the number of questions for the most used cross-platform mobile frameworks from 2016 to 2020. According to the figure, based on the number of questions for each framework, Flutter with the 41384 questions is the most used cross-platform mobile framework in practice in 2020. The React Native with 28462 questions placed in the second rank. Following these two frameworks, the Ionic with 6428 and the Xamarin with 5395 questions are the third and fourth frameworks respectively in the ranking. There is a meaningful difference between the question numbers of React Native and Ionic in terms of number of questions. The difference indicates that Flutter and React Native are the major cross-platform mobile frameworks in practice today. According to the figure, React Native was the most used framework in 2019. In 2020, the Flutter passed the React Native and became the first framework. On the other hand, The Ionic frameworks and the Xamarin were very close in these two years and still had a large margin when comparing to the first two frameworks.

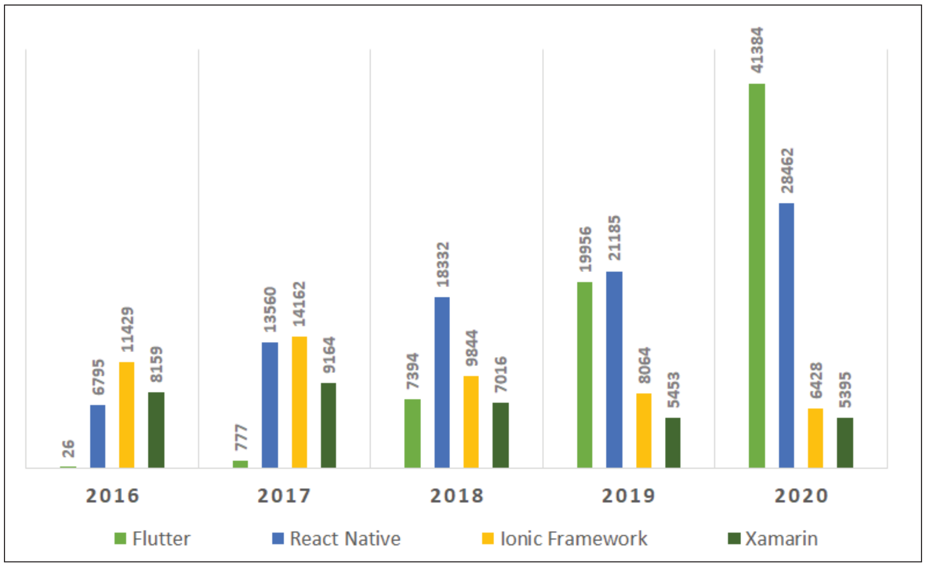


Figure 1.12 Most used mobile frameworks in practice, based on the number of questions in the stack overflow

The number of questions for each framework in Stack Overflow has changed over the years. This evolution over the years can show the tendency of developers to use the frameworks in real-world projects.

Figure 1.13 depicts the trend of asking questions for cross-platform frameworks. The figure indicates the number of asked questions for Flutter increased exponentially from 2016 to 2020. In 2017 which was the initial alpha release year of the framework, the stack overflow users asked 777 questions. After four years, in 2020, the number of questions boosted to 41384. This considerable jump shows the high popularity of the framework in practice. A similar rising pattern with a lower acceleration applies to the React Native framework. In 2016 the number of questions for the framework was 6795. After five years, in 2020, the number increased to 28462. The question numbers for Xamarin and the Ionic frameworks decreased over these five years similarly. This might indicate that these two frameworks are used less in practice.

As a result, we can conclude that Flutter, followed by React Native, Ionic and Xamarin are the most used cross-platform mobile frameworks.

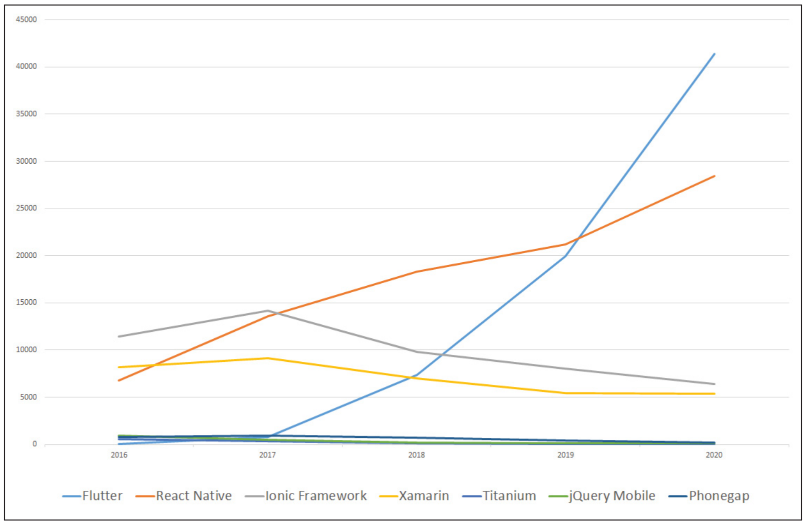


Figure 1.13 Numbers of questions for cross-platform frameworks in the Stack Overflow over the last five years

**RQ4.2.** This question aims at identifying the gap between the surveyed studies and the practice. In particular, we wanted to check if the frameworks analyzed by academia are actually those used in practice. Previously in our first research question, we identified the most studied frameworks in the primary studies of our SLR. Figure 1.14 shows the top five most studied cross-platform mobile frameworks in the last five years in the literature, which are Ionic, Xamarin, React Native, Appcelerator Titanium, and PhoneGap.

Based on the result of the previous sub-question, Flutter is the most used mobile framework practically. On the other hand, in academia, Flutter is not among the top five targeted cross-platform frameworks in the surveyed studies in the last five years. This might be explained by the fact that Flutter has been the most recently introduced framework compared to the others. According to Figure 1.14, the React Native framework was targeted by only 11 papers in the last five years. However, it was the most-used cross-platform mobile framework in 2019 and the second one in 2020.

In conclusion, we observe a meaningful discrepancy between the surveyed studies and the actual mobile frameworks used in practice.

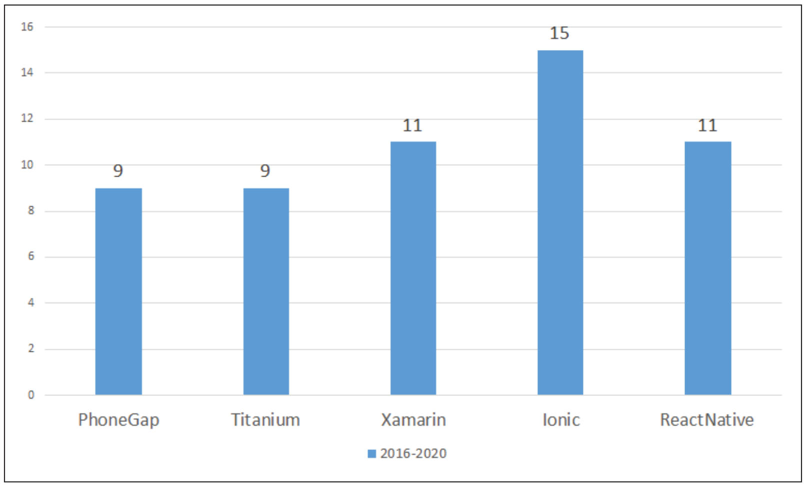


Figure 1.14 Most studied cross-platform mobile frameworks from 2016 to 2020 in surveyed studies

## 1.6 Limitations of the SLR

In this section, I discuss the limitations of my systematic literature review. The first limitation is in the search strategy step. We limited our search strategy to papers published after 2001. In fact, most of the mobile frameworks were introduced in the last decade. The search strategy may also be biased by the search string. Nevertheless, we tried to use a large combination of keywords to mitigate the risk of missing relevant studies. We also carried out a snowballing process.

The second limitation is the study selection process. Selection bias is one of the common risks in the systematic literature review. I selected studies based on the title, abstract, and content. To reduce the risk of selection bias, the selection process was carried out by another researcher (my supervisor in this case) and we cross-validated the results. Moreover, the data extraction from studies might affect the results of the research questions. In this step, two researchers counter-checked the information I extracted to mitigate the risks.

Finally, to answer the fourth research question, we use the number of asked questions for a mobile framework on the Stackoverflow website. We hypothesized this number shows the usage and popularity of the framework in practice. Our hypothesis might affect the results of this research question.

## 1.7 Discussion and Conclusion

In this SLR, we surveyed sixty one primary studies that analyze mobile development frameworks. We went through each research question and answered them according to our findings. In the first research question, we wanted to find the most studied mobile frameworks in the literature. We saw that PhoneGap is the most studied framework in the literature. However, the trend in literature considerably changed over the last five years, and fewer studies considered PhoneGap in their comparisons. In this research question, we also observed that Ionic was the most studied cross-platform mobile framework from 2015 to 2020 in the surveyed studies.

The second research question looked at the goals of the surveyed studies. We reviewed the studies to find out the evaluation perspectives, comparison metrics, and methods that they considered. The results show studies considered more criteria from the developer's perspective than the end-user perspective in the comparison. These studies usually compared mobile frameworks in terms of framework features. These criteria include platform API accessibility, hardware/sensor accessibility, framework maturity, development tools, and framework documentation. Only a few studies used software quality metrics, including maintainability, code reusability, reliability, and testing. But, none of those compare the impact of a mobile framework on the application source code's quality, such as the increasing number of bugs or code smell in the source code by using a mobile framework.

Through the third research question, we gained a detailed understanding of the evaluation methods used in the studies. The result indicates most studies compared mobile frameworks based on experiments or frameworks' documentation. However, few studies combined both methods, and a small group of studies only used the questionnaire and user opinions to compare mobile frameworks. Among the papers that used the experimental method, all studies used one or more prototype applications to compare their targeted mobile frameworks using their defined criteria. Prototype applications, in all cases, are simple applications developed merely for the comparison study. None of the studies compared mobile frameworks using real mobile applications.

The fourth research question provided valuable insights into the gaps between the most studied mobile frameworks in academia and the most used in practice. To answer this research question, we utilized the results of the first research question and the data obtained from the Stackoverflow website. We observed a meaningful discrepancy between the frameworks studied by academia and those currently used in practice. Flutter's popularity has grown among developers over the past couple of years. Aside from Flutter, ReactNative is gaining popularity among developers, however, Ionic and Xamarin are losing ground. This trend analysis might be helpful, especially for new developers who aim to learn a cross-platform mobile framework.

An essential reason for conducting the SLR before designing an empirical study is to find research gaps in the existing literature that analyzes mobile development frameworks. We observed that: 1) all the studies that used experiments to analyze the frameworks, used a prototype application to do so; 2) none of these studies used a large set of mobile applications in their analysis; and 3) the evaluation criteria didn't include criteria about the impact of a mobile framework on the application source code's quality (e.g., bugs or code smell, etc.). As a result, we designed an empirical study that uses a large set of mobile applications to evaluate the impact of mobile frameworks on the application source code quality.

# CHAPTER 2

# AN EMPIRICAL STUDY ON THE IMPACT OF A MOBILE DEVELOPMENT FRAMEWORK ON THE APPLICATION SOURCE CODE QUALITY

This chapter presents an empirical study that we carried out to investigate the impact of a mobile development framework on the application source code quality. As stated earlier in the SLR chapter, we could not find any evaluation study in mobile development frameworks executed on a large set of mobile applications. Therefore, we decided to select a large number of mobile applications for our empirical investigation and analyze the impact of a mobile framework on application source code quality. In this chapter, we first discuss the methodology and the study scope. Then, we present the tool we developed to automate the execution phase. Afterward, we detail the data collection and execution phase. Before concluding, we discuss the results of the study. Finally, we conclude our investigation and summarize the insights.

## 2.1 The methodology and the scope of the study

We designed and executed an empirical study to answer the thesis's question, which is what is the impact of the mobile framework on the application source code quality. This empirical study was performed on a large set of open-source mobile applications' source code. Figure 2.1 shows an overview of our study design and the steps we followed to perform our empirical study.

To conduct our empirical study, we followed these three steps. First, we set the study scope and the tools to be used. The targeted mobile development frameworks, source code analyzer, and the criteria on how to find mobile applications' source code will be identified in this step. Second, we proceed with the study execution and data collection phase. We want to collect 3000 mobile applications' source codes per targeted mobile framework in our study. Finally, based on the results we will interpret the results and conclusions.

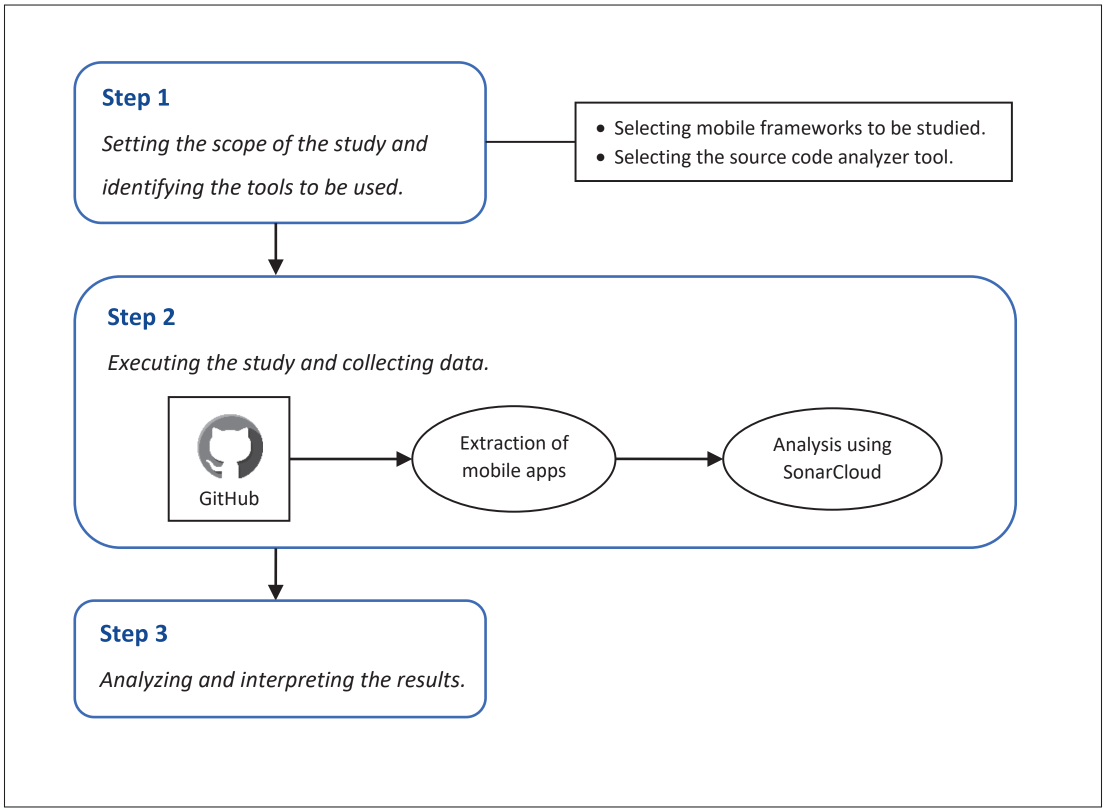

Step 1
Setting the scope of the study and identifying the tools to be used.
• Selecting mobile frameworks to be studied.
• Selecting the source code analyzer tool.
Step 2
Executing the study and collecting data.
GitHub
Extraction of mobile apps
Analysis using SonarCloud
Step 3
Analyzing and interpreting the results.

Figure 2.1 Overview of the steps of the empirical study

In the empirical study, our focus is to analyze mobile application source codes and find the framework impact on the source code quality. Hence, we analyze mobile application source codes from developers' points of view. Table 2.1 represents the source code quality metrics we used in our study and their definition.

Table 2.1 Source code quality metrics definition

| Metric | Definition |
|---|---|
| Bugs | The total number of bugs in source code. A bug is a coding error that will break your code and needs to be fixed immediately. |
| Code Smells | The total count of code smells in source code. A code smell is a maintainability issue that makes your code confusing and difficult to maintain. |

### 2.1.1 The selection of the mobile frameworks

We chose the frameworks to cover two approaches in mobile development: Native and Cross-Platform in our study. Developers can select between a large set of mobile development frameworks to develop mobile applications. Due to time and resource limitations, targeting all mobile frameworks in our investigation is impossible. Therefore we defined two criteria to select mobile frameworks to be targeted in our study:

- The programming languages used in the mobile framework must be supported by the source code analyzer tool to be used.
- The mobile frameworks must be among the most popular frameworks in practice and must not be introduced in recent years. As long as a mobile framework is more popular in practice and is mature enough, we are more likely to find open-source projects.

To narrow the scope of our study and make it feasible to perform, we targeted two mobile development frameworks: Android Native framework and React Native. Since Android has more open-source projects than iOS, we targeted *Android Native framework* as our native mobile framework. The second framework should be cross-platform. Earlier in the SLR, we examined the current popularity of mobile frameworks in practice. We identified the top five most popular cross-platform mobile frameworks based on StackOverflow questions. According to the results, Flutter is the most popular cross-platform mobile framework among other mobile frameworks. But, Flutter is a new cross-platform framework and is not supported by some of the current source code analyzers. Since it is a new mobile development framework, finding mobile applications' source codes would be challenging. As a result, we decided to choose the second most popular cross-platform mobile framework after Flutter which is *React Native*. React Native is a cross-platform mobile application development framework created by Facebook. It uses JavaScript and TypeScript as programming languages to develop mobile applications.

### 2.1.2 The selection of a source code analyzer

Our main goal is to compare these quality metrics in projects that were developed using different mobile frameworks to answer the thesis's question. During the evaluation process, it is crucial to use the same quality metrics to reduce the threat to the validity of our study's results. Hence, We must select and use a single source code analyzer tool. In the next step, we must choose a static source code analyzer to evaluate the quality of the mobile application source codes. There are some criteria we need to consider when selecting the tool for our study. First, the analyzer must evaluate the projects using the same source code quality metrics.

Second, due to this high number of applications' source codes, it is impossible to evaluate each application's source code manually. Therefore, the analysis tool must let us work with it programmatically and by using APIs. Moreover, It must let us read and extract the analysis result after generating reports by APIs.

Third, the source code analyzer must let us narrow the scope of analysis. One of the differences between mobile applications and traditional desktop applications is that mobile software heavily depends on third-party libraries and APIs (Nagappan & Shihab (2016)). And there is no need to analyze this specific part of the source code in our investigation. Hence, the tool must give the ability to define the scope of the analysis to avoid including unnecessary components.

Fourth, we chose GitHub to collect our mobile applications' source codes. GitHub is the most popular version control system and hosts many open-source projects. We chose GitHub and expect the source code analyzer tool to work closely with GitHub. The source code analyzer tool must be capable of importing mobile applications' source codes directly from GitHub. In other words, it should provide GitHub integration to make this process automatic and systematic.

Finally, and in addition to all the mentioned features, it would be better if the analyzer tool is well documented and widely used in practice.

Based on our criteria for the tool selection, we chose SonarCloud[1] source code analyzer. SonarCloud is a cloud-based static code analysis service. It is a product of the SonarSource[2] company which owns the well-known SonarQube source code analyzer that can be used for . SonarSource has more than 6,000 customers, including eBay, Bank of America, and BMW (Wikipedia contributors (2022)), and its products are widely used in practice. The tool supports more than 20 programming languages. Moreover, lets you define inclusion and exclusion rules to narrow the analysis scope of the source code to have more precise results. Integrating with GitHub is made easy in SonarCloud by connecting a GitHub account and importing projects. It also can get the latest changes to repositories automatically. There are no fees associated with using SonarCloud to analyze public open-source projects. The free version of the tool lets us do infinite analysis for unlimited lines and does not limit the access to tool features. Moreover, its cloud-based service does not oblige you to keep the source code on a local machine. By accessing the SonarCloud APIs, we can collect the analysis results pragmatically (API (2022b)).

The integration with GitHub allows users to log in with their GitHub account and import their accounts or organizations and their repositories directly to SonarCloud. SonarCloud automatically analyzes all imported repositories as soon as they are imported. Additionally, SonarCloud re-analyses the source code immediately after each change is made in repositories.

Depending on the size of the project, the analysis process would be different. It can take up to 30 minutes. Upon analysis completion, the user will receive the analysis results of the repository for the first and the five most recent active pull requests. Moreover, each new commit and push on

[1] https://www.sonarcloud.io

[2] https://www.sonarsource.com

the default branch or a pull request will trigger an analysis automatically. Figure 2.2 represents the results for a sample mobile application source code.

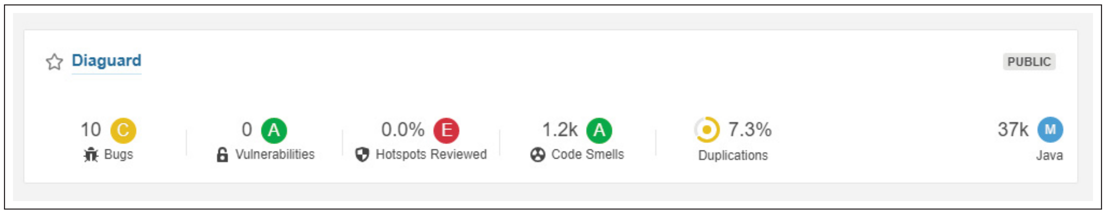


Figure 2.2 The brief analysis results of a source code in SonarCloud

By selecting the project and clicking on it, SonarCloud shows an overview of the analysis results. Figure 2.3 represents this overview for a sample mobile application source code. The overview section in SonarCloud also shows the historical analysis data of a repository. Moreover, it shows the number of findings issues, code duplication, and total lines of code.

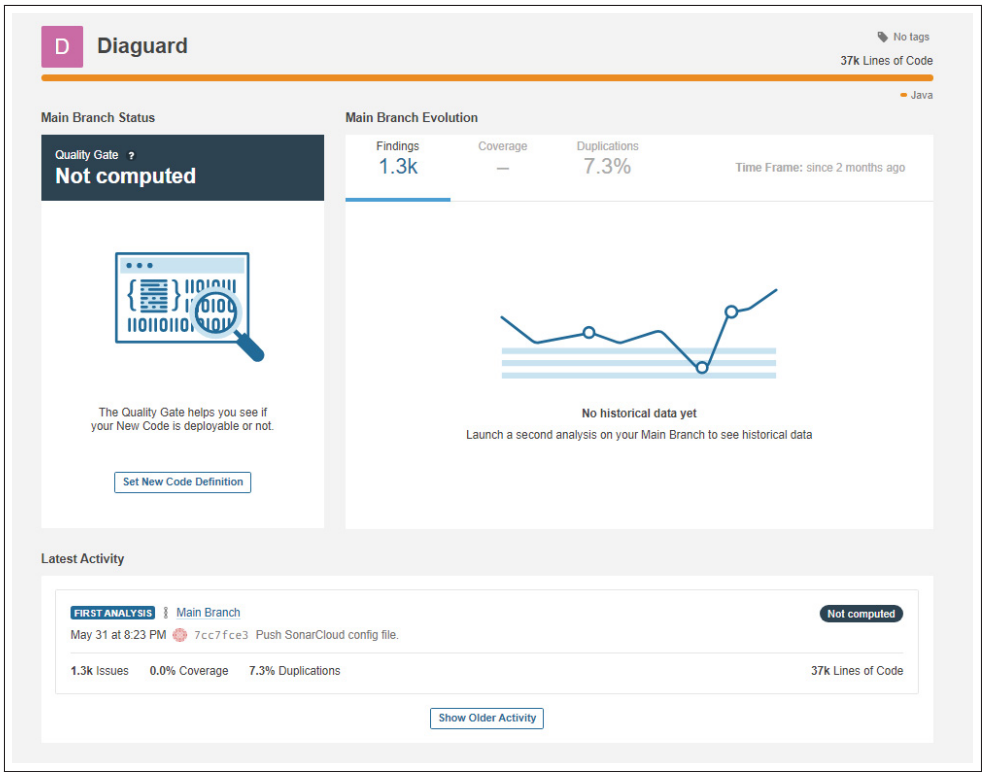


Figure 2.3 The overview analysis results of a source code in SonarCloud

By clicking on the Main Branch item in the left menu, users will see the summary of the analysis result. Figure 2.4 shows the summary view of results.

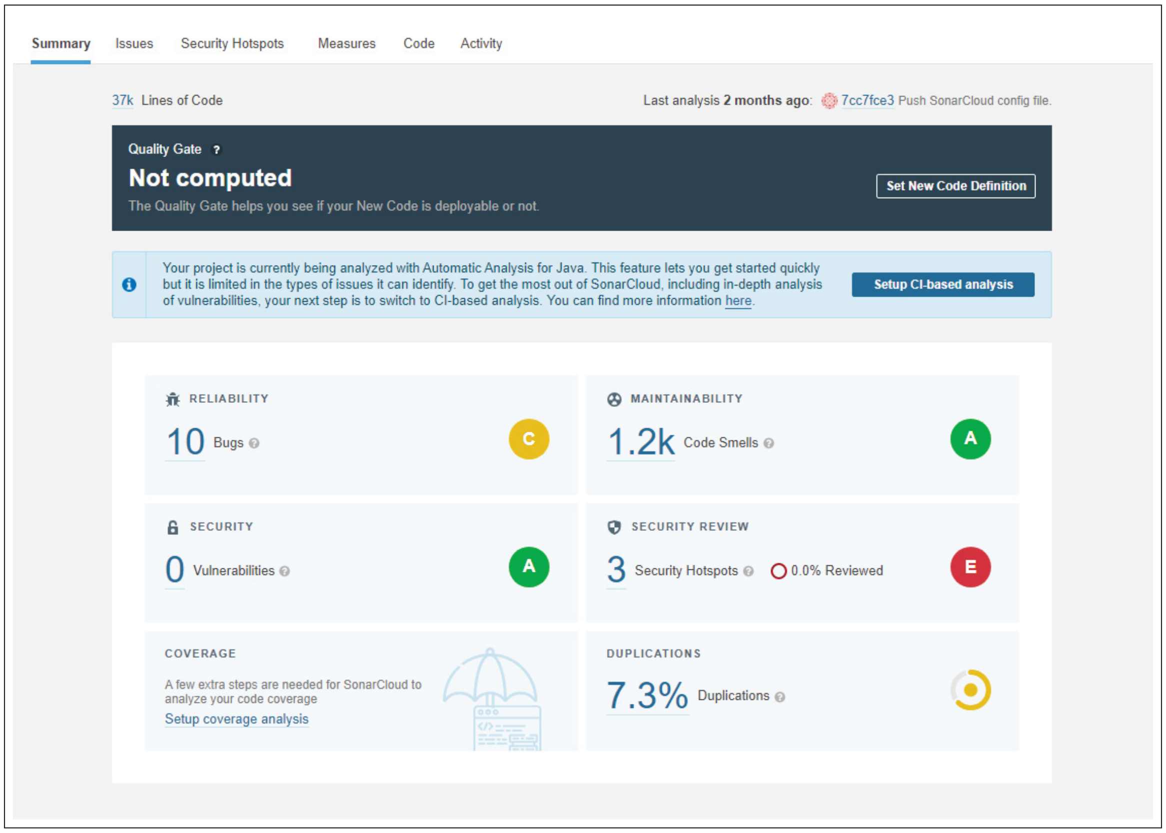

Summary
Issues
Security Hotspots
Measures
Code
Activity
37k Lines of Code
Last analysis 2 months ago: 7cc7fce3 Push SonarCloud config file.
Quality Gate ?
Not computed
The Quality Gate helps you see if your New Code is deployable or not.
Set New Code Definition
Your project is currently being analyzed with Automatic Analysis for Java. This feature lets you get started quickly but it is limited in the types of issues it can identify. To get the most out of SonarCloud, including in-depth analysis of vulnerabilities, your next step is to switch to CI-based analysis. You can find more information here.
Setup CI-based analysis
RELIABILITY
10 Bugs
C
MAINTAINABILITY
1.2k Code Smells
A
SECURITY
0 Vulnerabilities
A
SECURITY REVIEW
3 Security Hotspots
0.0% Reviewed
E
COVERAGE
A few extra steps are needed for SonarCloud to analyze your code coverage
Setup coverage analysis
DUPLICATIONS
7.3% Duplications

Figure 2.4 The summary of analysis results of a source code in SonarCloud

Another view to see the analysis results appears at the top of this window. Users can see the analysis results in detail by selecting the tabs from the top section. Figure 2.5 shows the details of finding issues in a project. According to SonarCloud documentation, the tool can identify three issue types in source code. The following list shows the issue types in SonarCloud:

- **Bug** – A coding mistake that can lead to an error or unexpected behaviour at runtime.
- **Vulnerability** – A point in the code that's open to attack.
- **Code Smell** – A maintainability issue that makes the code confusing and difficult to maintain.

In addition to issue types, SonarCloud also classifies five severity levels for issues in its platform. The following list shows these five severity levels for issues in SonarCloud:

- BLOCKER

Bug with a high probability to impact the behaviour of the application in production like a memory leak or unclosed JDBC connection.

- CRITICAL

Either a bug with a low probability to impact the behaviour of the application in production or an issue that represents a security flaw, like an empty catch block or a SQL injection.

- MAJOR

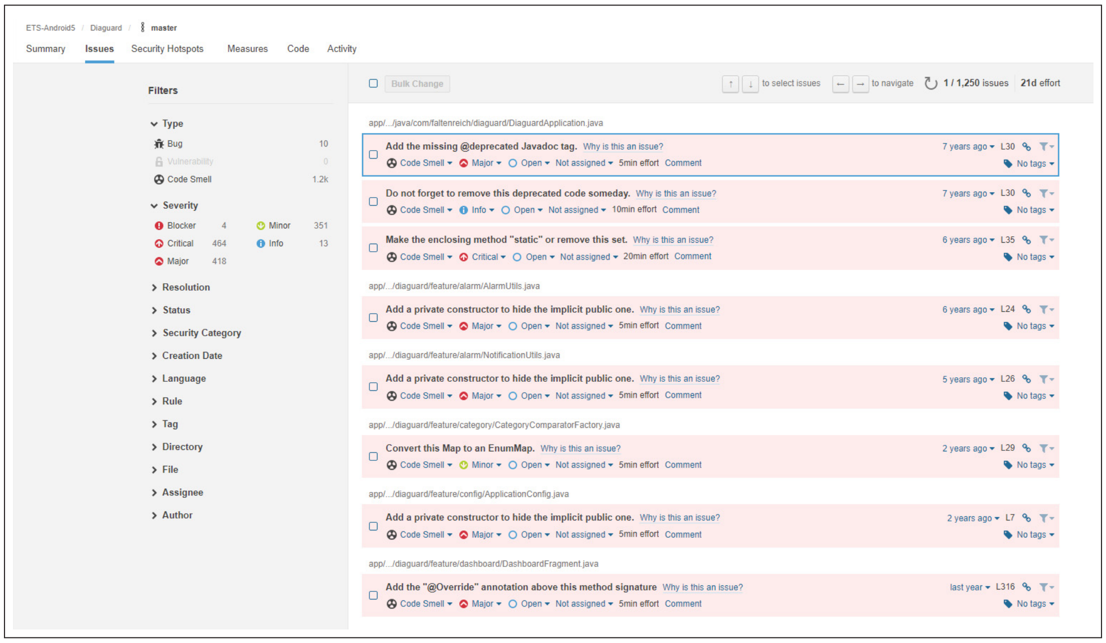
The quality flaw that can highly impact the developer's productivity, like an uncovered piece of code, duplicated blocks, or unused parameters.

- MINOR

The quality flaw that can slightly impact the developer productivity, like lines should not be too long or "switch" statements should have at least 3 cases.

- INFO

Neither a bug nor a quality flaw, just a finding issue.



app/.../java/com/faltenreich/diaguard/DiaguardApplication.java

Add the missing @deprecated Javadoc tag. Why is this an issue? 7 years ago L30
Code Smell Major Open Not assigned 5min effort Comment No tags

Do not forget to remove this deprecated code someday. Why is this an issue? 7 years ago L30
Code Smell Info Open Not assigned 10min effort Comment No tags

Make the enclosing method "static" or remove this set. Why is this an issue? 6 years ago L35
Code Smell Critical Open Not assigned 20min effort Comment No tags

app/.../diaguard/feature/alarm/AlarmUtils.java

Add a private constructor to hide the implicit public one. Why is this an issue? 6 years ago L24
Code Smell Major Open Not assigned 5min effort Comment No tags

app/.../diaguard/feature/alarm/NotificationUtils.java

Add a private constructor to hide the implicit public one. Why is this an issue? 5 years ago L26
Code Smell Major Open Not assigned 5min effort Comment No tags

app/.../diaguard/feature/category/CategoryComparatorFactory.java

Convert this Map to an EnumMap. Why is this an issue? 2 years ago L29
Code Smell Minor Open Not assigned 5min effort Comment No tags

app/.../diaguard/feature/config/ApplicationConfig.java

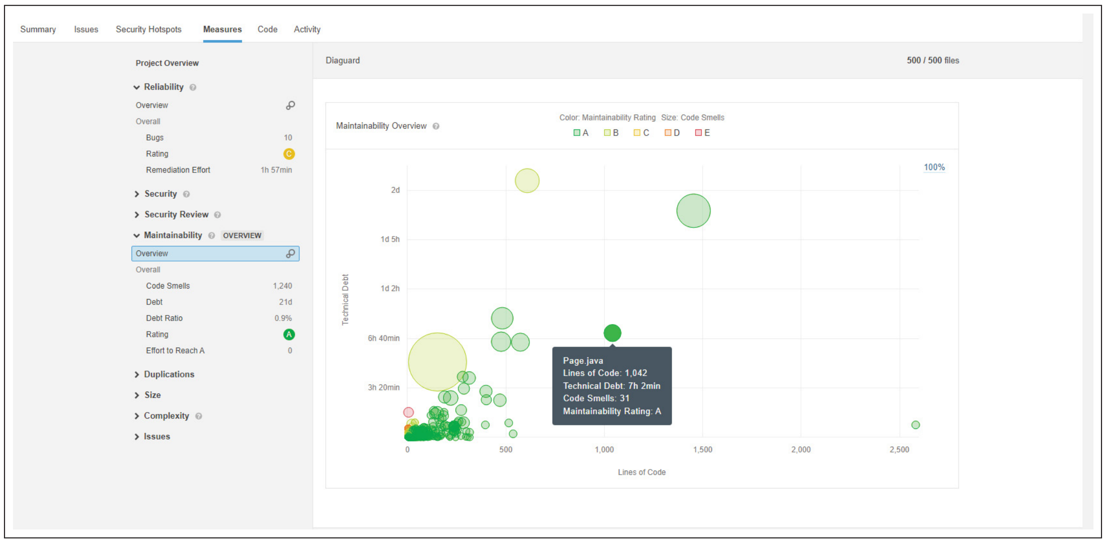
Add a private constructor to hide the implicit public one. Why is this an issue? 2 years ago L7
Code Smell Major Open Not assigned 5min effort Comment No tags

app/.../diaguard/feature/dashboard/DashboardFragment.java

Add the "@Override" annotation above this method signature Why is this an issue? last year L316
Code Smell Major Open Not assigned 5min effort Comment No tags

Figure 2.5  The details of finding issues of a source code in SonarCloud

The Measures tab in the SonarCloud tool provides the results of a set of usable quality metrics after analysis. In this tab, SonarCloud represents the values of source code quality metrics graphically in detail. Figure 2.6 shows the analysis results of the Reliability and Maintainability of a project in SonarCloud. In addition to numeric reports, SonarCloud also provides bobble graphs. The graphs show the quality metrics value of each file. Moreover, Figure 2.7 represents the bobble graph with analysis results for Duplication, Size, and Complexity metrics.



Reliability

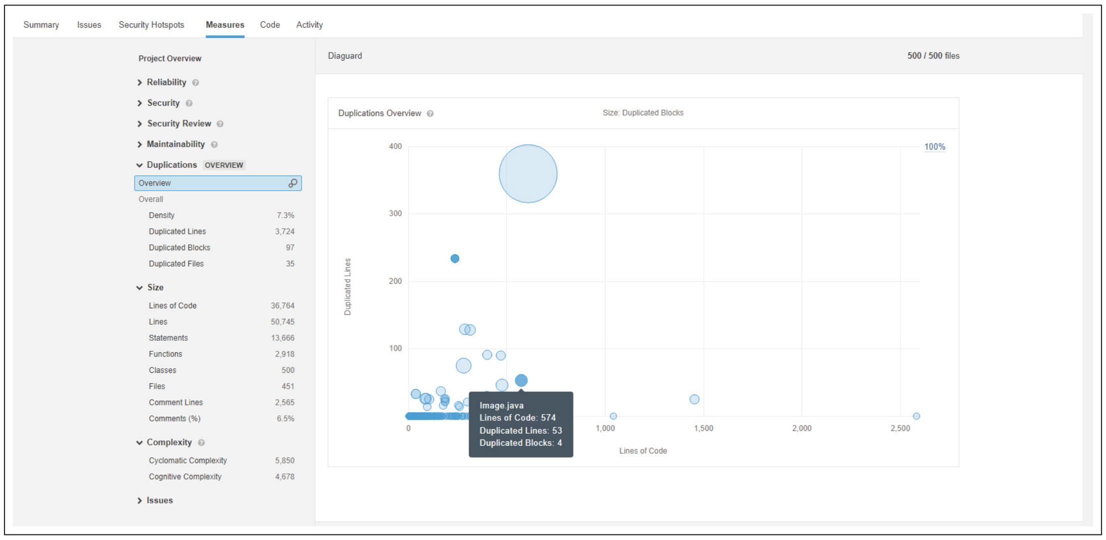
| | |
|---|---|
| Overview | |
| Overall | |
| Bugs | 10 |
| Rating | C |
| Remediation Effort | 1h 57min |

Maintainability OVERVIEW

| | |
|---|---|
| Overview | |
| Overall | |
| Code Smells | 1,240 |
| Debt | 21d |
| Debt Ratio | 0.9% |
| Rating | A |
| Effort to Reach A | 0 |

Page.java
Lines of Code: 1,042
Technical Debt: 7h 2min
Code Smells: 31
Maintainability Rating: A

Figure 2.6 The detail of Reliability and Maintainability metrics in SonarCloud



Duplications OVERVIEW

| | |
|---|---|
| Overview | |
| Overall | |
| Density | 7.3% |
| Duplicated Lines | 3,724 |
| Duplicated Blocks | 97 |
| Duplicated Files | 35 |

Size

| | |
|---|---|
| Lines of Code | 36,764 |
| Lines | 50,745 |
| Statements | 13,666 |
| Functions | 2,918 |
| Classes | 500 |
| Files | 451 |
| Comment Lines | 2,565 |
| Comments (%) | 6.5% |

Complexity

| | |
|---|---|
| Cyclomatic Complexity | 5,850 |
| Cognitive Complexity | 4,678 |

Image.java
Lines of Code: 574
Duplicated Lines: 53
Duplicated Blocks: 4

Figure 2.7 The detail of Duplication, Size and Complexity metrics in SonarCloud

### 2.1.3 The selection of mobile apps

To perform our study, we must extract and analyze a collection of mobile applications developed using the two targeted frameworks: Android native framework and ReactNative. Here, we explain where and how we select mobile applications for our empirical study.

Selecting a large number of mobile applications is challenging. We must choose similar mobile application source codes regarding targeted frameworks to analyze them equitably. To overcome this challenge, we must search and find mobile application repositories with the same characteristics on GitHub. For instance, the size of the mobile application should be in the same range. Due to the fact that the size of the project might affect the quality metrics values and the results, we need to take this into account. Hence, we define a size classification for repositories in order to have similar mobile applications by source code size.

For the input size, we targeted to analyze 3000 mobile applications' source codes in each framework. Based on our size calcification, we will select 600 applications from each size class to collect our target number for each framework. Table 2.2 represents the size definition of each size class.

Table 2.2 Size classes of mobile application source code

| **Size class** | **Size(kilobytes)** |
|---|---|
| Tiny | 5000 - 8000 |
| Small | 8001 - 12000 |
| Medium | 12001 - 20000 |
| Large | 20000 - 35000 |
| Extra large | 35001 - Max |

As we stated earlier, we want to execute our investigation on a large number of mobile applications. But, it is impossible to extract and analyze all applications manually. Hence, we needed a tool that automates the execution phase of the study. This tool must be responsible for finding and extracting mobile applications on Github regarding our targeted frameworks. After that, it must filter the findings to ensure that only real mobile applications will be extracted. Then, it should help us in the process of exporting the applications to the source code analyzer. It must enable us to collect the analysis results from the analyzer tool. Accordingly, we developed this tool, called GitToSonar, to automate the execution phase of our empirical study.

## 2.2 The GitToSonar tool

In this section, we talk about the GitToSonar tool and explain its features. Additionally, we display some screenshots of the GitToSonar and explain which information is needed to use the tool. The following list shows the main features of the GitToSonar tool.

1. Managing operations in repositories include Fork, retrieve, search, and remove
2. Searching and finding repositories in GitHub using search criteria
3. Searching and finding files within repositories
4. Creating exclusion/inclusion configuration files
5. Pushing files into repositories
6. Collecting generated reports
7. Creating report files

The GitToSonar tool helps us search GitHub repositories to find mobile applications' source codes. It enables us to validate the mobile application repositories by searching framework-specific files in repositories. It also helps us to fork the finding results to our GitHub account connected and used by the SonarCloud tool. SonarCloud provides integration with GitHub. It enables SonarCloud to import repositories directly from GitHub and analyze them initially and after each change (commit) automatically. GitToSonar also helps us create the configuration file required by SonarCloud, for each mobile application source code to narrow the analysis scope and exclude third-party libraries and plugins. Then it pushes the file to the GitHub account again. Finally, after finishing the analysis, the tool fetches the results from SonarCloud and generates the report.

The GitToSonar tool uses the GitHub and SonarCloud APIs. We developed it using Microsoft .Net framework and C# programming language. We published the application source code on GitHub[3], so other developers can add features based on their needs. Figure 2.8 shows a screenshot of the application's main page.

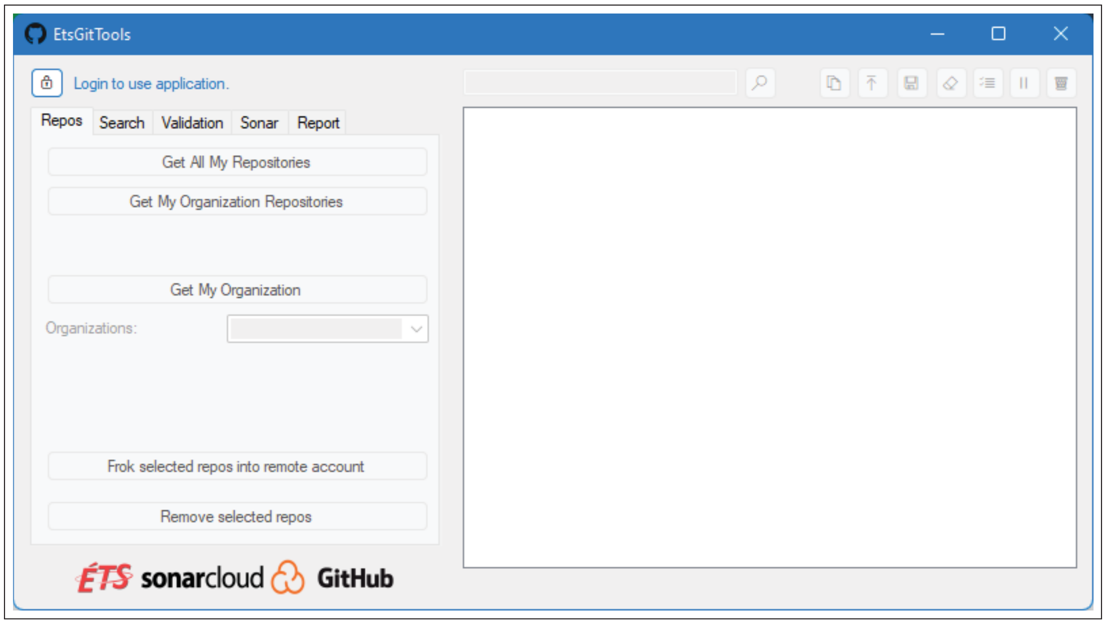


Figure 2.8 The GitToSonar tool main page

In order to use GitToSonar, we need to log in using the GitHub account information. This information includes the GitHub username, password, and personal access token[4]. The requested information is used to access the GitHub APIs (GitHubAPI (2022)). After logging in, we can access the application's features. Figure 2.9 shows a screenshot of the application's login page and the required information to login into the application.

[3] https://github.com/Parsakarami/GithubTool

[4] https://docs.github.com/en/enterprise-server@3.4/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

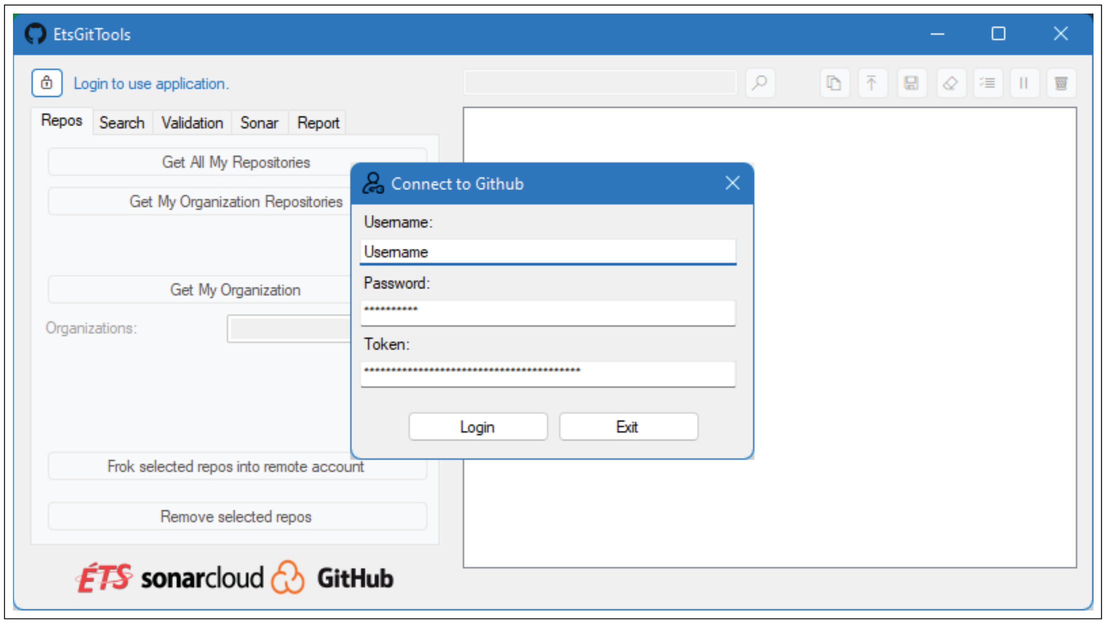


Figure 2.9 The GitToSonar tool login page

After successfully logging in, the user will have access to the application's main features. They are visible on the left-hand side of the application. The first feature of the application is organization and repository management. They can retrieve repositories from their account in the right-hand list and apply the management operation to the selected repositories. Users can use those operations in their organizations as well. They can do it by loading the organizations and selecting from the list. Figure 2.10 shows a screenshot of the application's main page after login. The green label at the top left hand of the application indicates the username of the current user.

The "Get My Organization" button fetches the current user's organizations, and the user can select the target organization from the box behind it. After choosing the target organization, it will be added inside the parenthesis in front of the username. Figure 2.11 shows a screenshot of the application after selecting the organization.

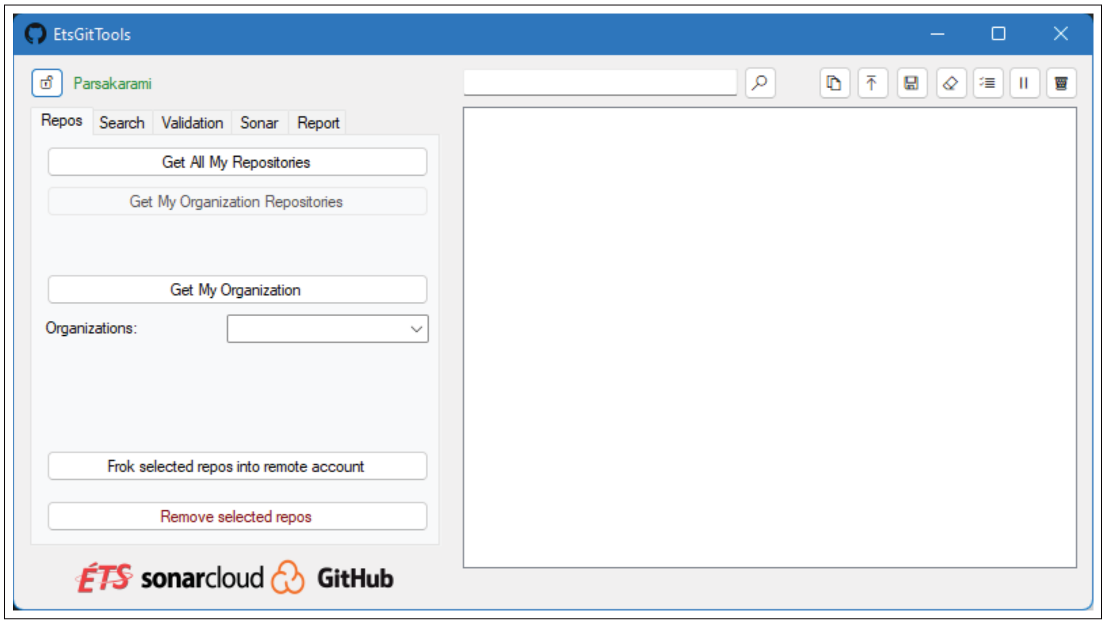


Figure 2.10 The GitToSonar tool main page after successful login

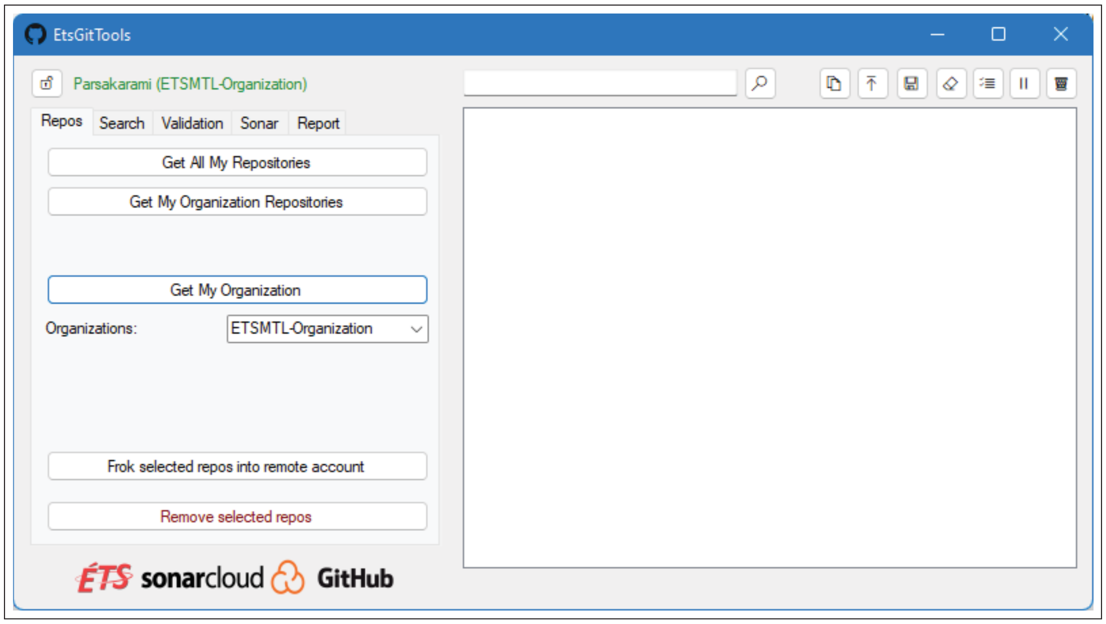
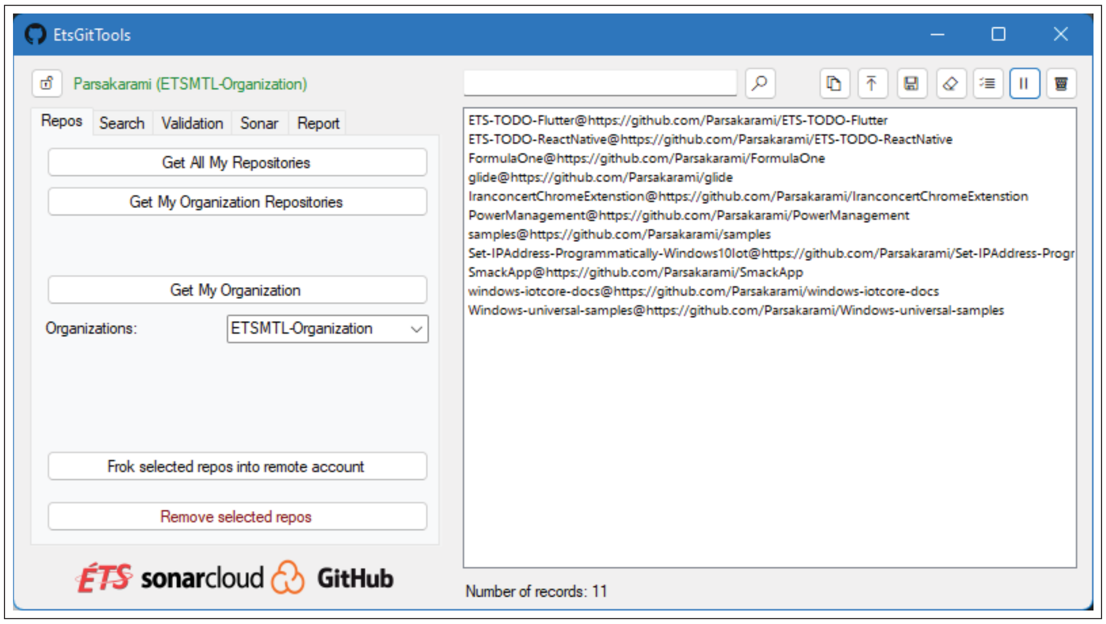


Figure 2.11 The GitToSonar tool main page after selecting the organization

The ”Get All My Repositories” button fetches all of the current user’s repositories from GitHub. Following the retrieval of repositories from GitHub, they will be added to the list on the right side of the application's main page. The user can also fetch the organization's repositories by pressing the "Get My Organization Repositories" button. Figure 2.12 shows a screenshot of the application's main page after fetching the user's repositories.



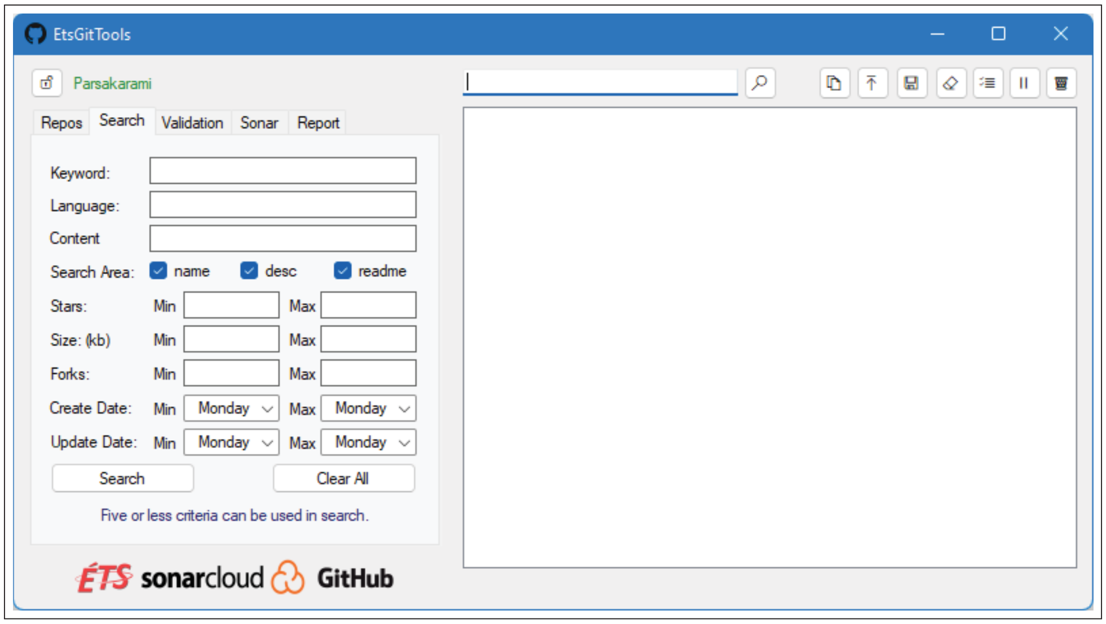
```
ETS-TODO-Flutter@https://github.com/Parsakarami/ETS-TODO-Flutter
ETS-TODO-ReactNative@https://github.com/Parsakarami/ETS-TODO-ReactNative
FormulaOne@https://github.com/Parsakarami/FormulaOne
glide@https://github.com/Parsakarami/glide
IranconcertChromeExtenstion@https://github.com/Parsakarami/IranconcertChromeExtenstion
PowerManagement@https://github.com/Parsakarami/PowerManagement
samples@https://github.com/Parsakarami/samples
Set-IPAddress-Programmatically-Windows10Iot@https://github.com/Parsakarami/Set-IPAddress-Progr
SmackApp@https://github.com/Parsakarami/SmackApp
windows-iotcore-docs@https://github.com/Parsakarami/windows-iotcore-docs
Windows-universal-samples@https://github.com/Parsakarami/Windows-universal-samples
```

Number of records: 11

Figure 2.12 The GitToSonar tool main page after fetching current user's repositories

The user can select among the repositories in the list and remove their selections from the account or account's organization permanently. The user also can export and import repositories from or into the list. Moreover, the tool provides the ability to search within the list and find the repositories inside the list.

GitHub enables us to search and manage repositories by providing an API. Developers can use GitHub's API using REST or GraphQL standards. To search and manage repositories, we employ the REST API in GitToSonar. Figure 2.13 shows a screenshot of the search feature in the GitToSonar tool. Users can create the search query employing the search properties in UI to optimize their search results over GitHub repositories.



Search Area: [✓ name] [✓ desc] [✓ readme]

Create Date: Min Monday Max Monday

Update Date: Min Monday Max Monday

Five or less criteria can be used in search.

Figure 2.13 The GitToSonar tool search page

The third tab in GitToSonar helps us to search for a specific file within the selected repositories. Figure 2.14 shows this feature in the tool. By choosing the "other" option in the drop-down menu, we can determine whether the searched file is present in the selected repositories. The feature can be used to find framework-specific files within mobile application repositories.

Moreover, we developed another feature to collect and generate reports from SonarCloud in our GitToSonar tool. We can collect analysis results from SonarCloud and generate report files. In addition, this feature enables users to obtain and filter project issues based on the issue types. Figure 2.15 shows the report tab in our tool.

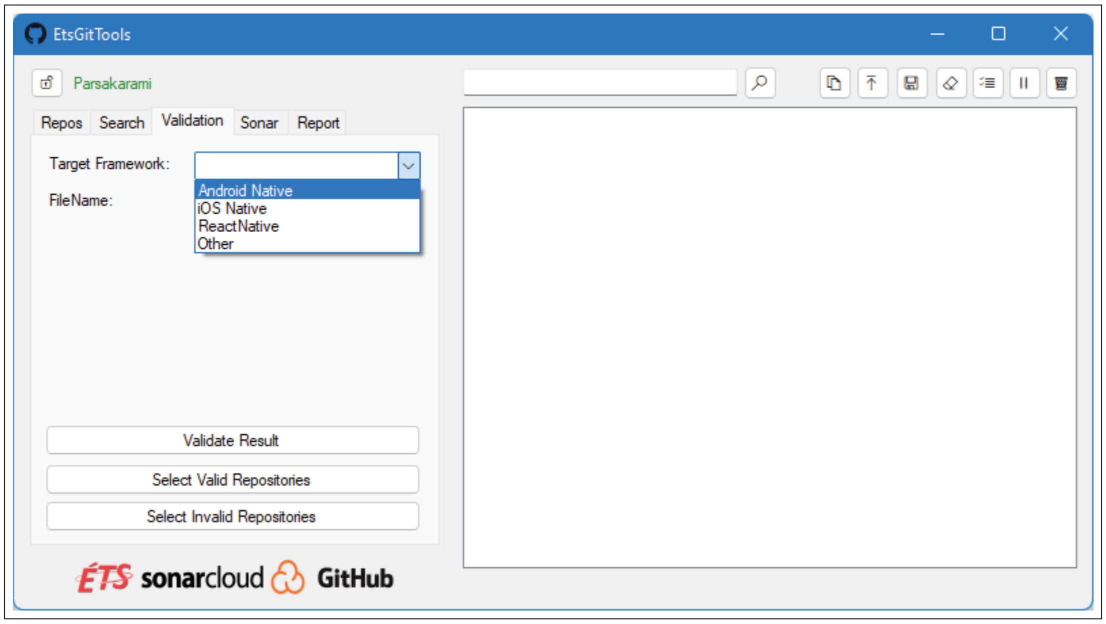


Figure 2.14 The GitToSonar tool feature to search files within a repository

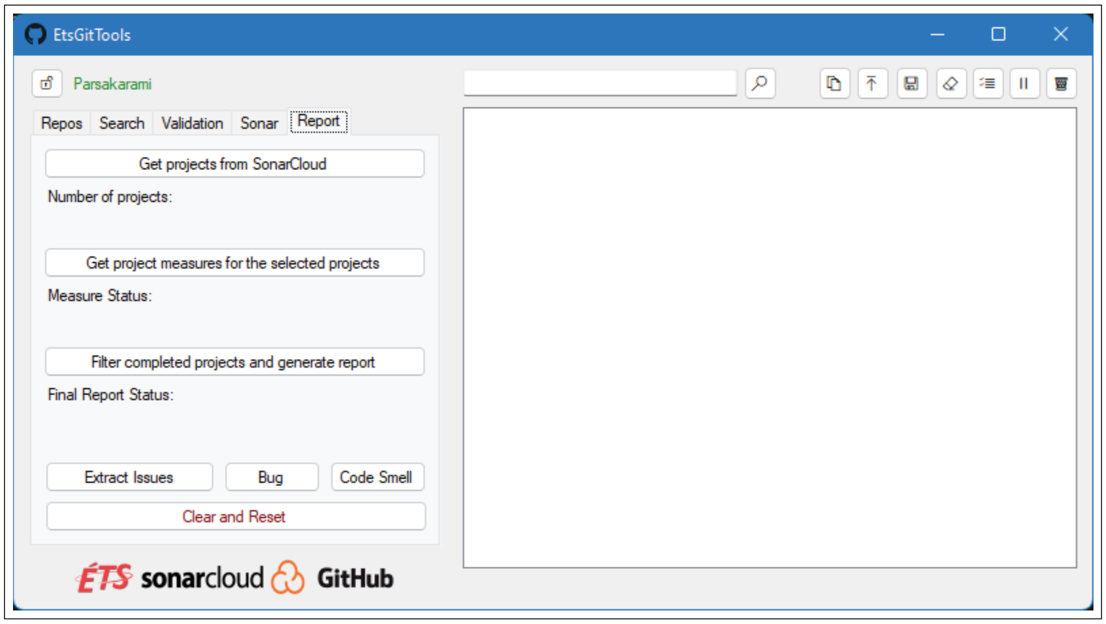


Figure 2.15 The GitToSonar tool report feature

In addition to the systematic literature review and the empirical study, the GitToSonar tool is one of the main contributions of this thesis. It provides a set of features to manage GitHub's repositories systematically. Moreover, it helps us collect the analysis results from SonarCloud. We specifically developed the GitToSonar tool for this empirical study. The tool's features help us answer the thesis's question. However, we believe GitToSonar will be helpful for other researchers, especially those who do similar empirical studies in this context. In order to achieve this, they must be able to extend the tool's features based on their needs. Because of this, we decided to publish GitToSonar on GitHub as an open-source project. In the next section, we discuss how GitToSonar helps us in the study execution phase.

## 2.3 The study execution

In this section, we explain the steps of the execution phase of our empirical study and obtain the results. The following list shows the execution steps.

1. Searching and finding mobile application repositories in GitHub.
2. Filtering repositories to verify extracting only real mobile applications regarding our targeted mobile framework.
3. Forking repositories to keep them in a single account to export them into SonarCloud.
4. Creating an exclusion/inclusion configuration file for SonarCloud to narrow the analysis scope.
5. Pushing the created configuration file into collected repositories.
6. Import the repositories into SonarCloud using the integration feature of SonarCloud.
7. Collecting analysis results from SonarCloud and creating report files.

In addition, we present these steps in Figure 2.16. Moreover, the figure highlights the steps supported by GitToSonar. In the figure, arrows indicate the execution steps and the color of the arrow represents which tool performed that step.

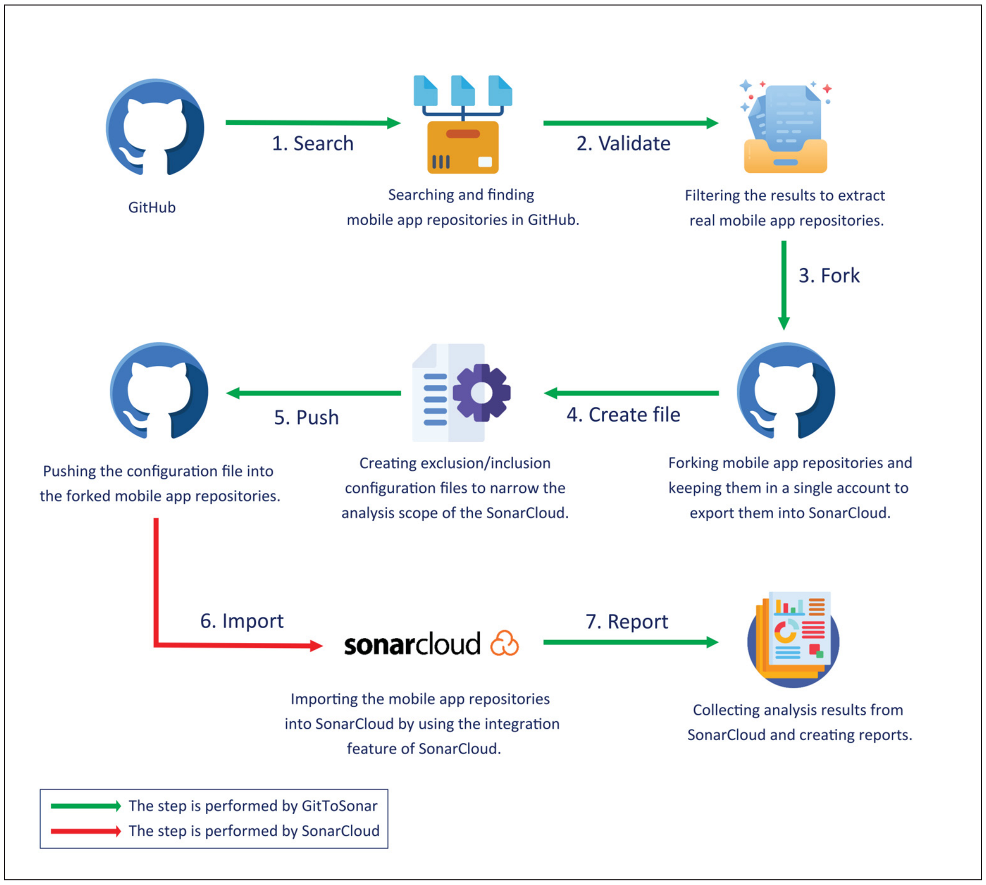

GitHub
1. Search
Searching and finding mobile app repositories in GitHub.
2. Validate
Filtering the results to extract real mobile app repositories.
3. Fork
Forking mobile app repositories and keeping them in a single account to export them into SonarCloud.
4. Create file
Creating exclusion/inclusion configuration files to narrow the analysis scope of the SonarCloud.
5. Push
Pushing the configuration file into the forked mobile app repositories.
6. Import
sonarcloud
Importing the mobile app repositories into SonarCloud by using the integration feature of SonarCloud.
7. Report
Collecting analysis results from SonarCloud and creating reports.
The step is performed by GitToSonar
The step is performed by SonarCloud

Figure 2.16  Overview of the empirical study execution steps and workflow

### 2.3.1 The Searching for mobile apps

The first step in the execution phase is finding and extracting the mobile applications' source codes. Earlier in table 2.2, we defined a size classification of mobile applications in our study. In this section, we use the defined size classes to find mobile applications in each targeted mobile framework by determining the size.

The search API lets developers search for specific items on GitHub. These items can be the repositories, codes, files, commits, labels, and users. However, there are some limitations in using GitHub's APIs. According to its documentation, the search API provides only up to 1,000 results for each search (API (2022a)). In other words, we need to change the search query to get more repositories. However, our size classification enables us to overcome this limitation.

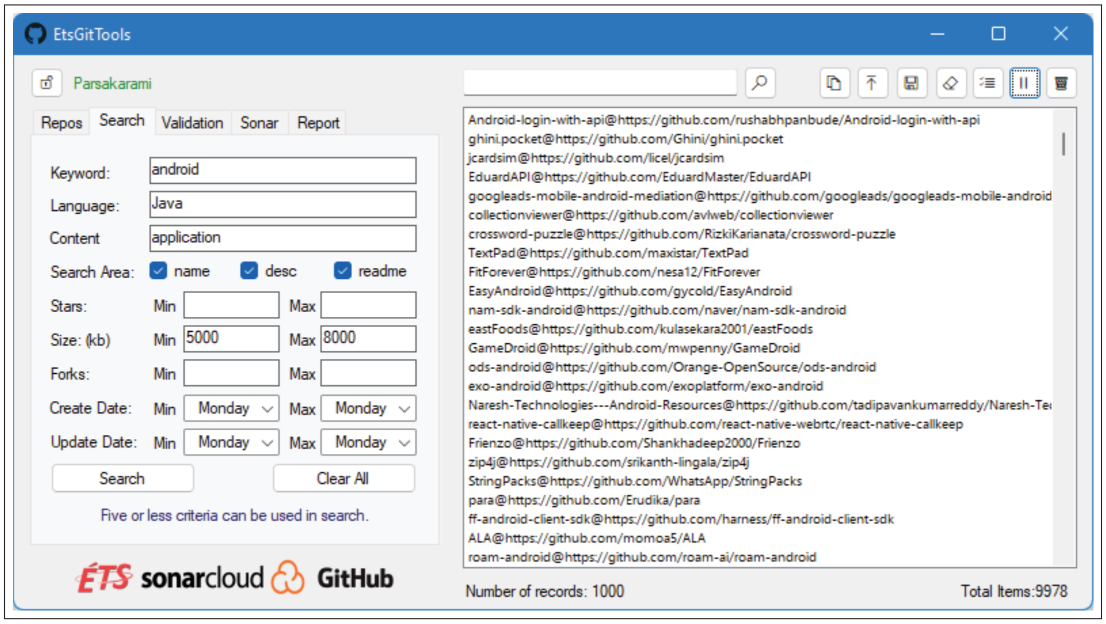
GitHub enables us to narrow our search results using criteria in search queries. We defined criteria to find mobile applications for the study using the GitToSonar tool. First, we use keywords for our search that we choose according to the targeted framework. Second, we select the programming language of the framework. Third, we add text to search within a repository including the repository name, description, and contents inside the README file. Fourth, we put our determined size to have a mobile application with a similar size in our investigation. Fifth, we use a flag in the search query to filter out the archive repositories and only find the active mobile application repositories. Finally, we specified in the query to sort based on the most recently updated repositories. To achieve this, we set the sort parameter when creating the search query.

We used the same search criteria to retrieve similar mobile applications' source codes in each targeted mobile framework. Then, we searched for each size category to collect our target number. Table 2.3 shows our search criteria in each development framework.

Table 2.3 Search criteria for finding mobile application repositories in GitHub

| **Criterion** | **Values** (Android native) | **Values** (React-Native) |
|---|---|---|
| Search Keyword | "android" | "react-native" |
| Programming language | "Java" | "JavaScript" |
| containing words in the name, description, or README file of the repository | "application" | "application" |
| Minimum Size | Lower limit in the classification | Lower limit in the classification |
| Maximum Size | Upper limit in the classification | Upper limit in the classification |
| Archived | false | false |
| Sort | update date-time | update date-time |
| Order | asc | asc |

In the GitToSonar tool, we have this feature to use GitHub's search API. We searched on GitHub and found the mobile application repositories. We can narrow the scope of the search by adding search criteria. Figure 2.17 indicates search results for the first size class of mobile applications' source code developed using the Android native framework. We used the search criteria inside the figures to find mobile applications for the first size category.



Keyword: android
Language: Java
Content application
Search Area: ✓ name ✓ desc ✓ readme
Stars: Min Max
Size: (kb) Min 5000 Max 8000
Forks: Min Max
Create Date: Min Monday Max Monday
Update Date: Min Monday Max Monday

Five or less criteria can be used in search.

```
Android-login-with-api@https://github.com/rushabhpanbude/Android-login-with-api
ghini.pocket@https://github.com/Ghini/ghini.pocket
jcardsim@https://github.com/licel/jcardsim
EduardAPI@https://github.com/EduardMaster/EduardAPI
googleads-mobile-android-mediation@https://github.com/googleads/googleads-mobile-android
collectionviewer@https://github.com/avlweb/collectionviewer
crossword-puzzle@https://github.com/RizkiKarianata/crossword-puzzle
TextPad@https://github.com/maxistar/TextPad
FitForever@https://github.com/nesa12/FitForever
EasyAndroid@https://github.com/gycold/EasyAndroid
nam-sdk-android@https://github.com/naver/nam-sdk-android
eastFoods@https://github.com/kulasekara2001/eastFoods
GameDroid@https://github.com/mwpenny/GameDroid
ods-android@https://github.com/Orange-OpenSource/ods-android
exo-android@https://github.com/exoplatform/exo-android
Naresh-Technologies---Android-Resources@https://github.com/tadipavankumarreddy/Naresh-Te
react-native-callkeep@https://github.com/react-native-webrtc/react-native-callkeep
Frienzo@https://github.com/Shankhadeep2000/Frienzo
zip4j@https://github.com/srikanth-lingala/zip4j
StringPacks@https://github.com/WhatsApp/StringPacks
para@https://github.com/Erudika/para
ff-android-client-sdk@https://github.com/harness/ff-android-client-sdk
ALA@https://github.com/momoa5/ALA
roam-android@https://github.com/roam-ai/roam-android
```

Number of records: 1000 Total Items:9978

Figure 2.17 The GitToSonar tool search results using criteria

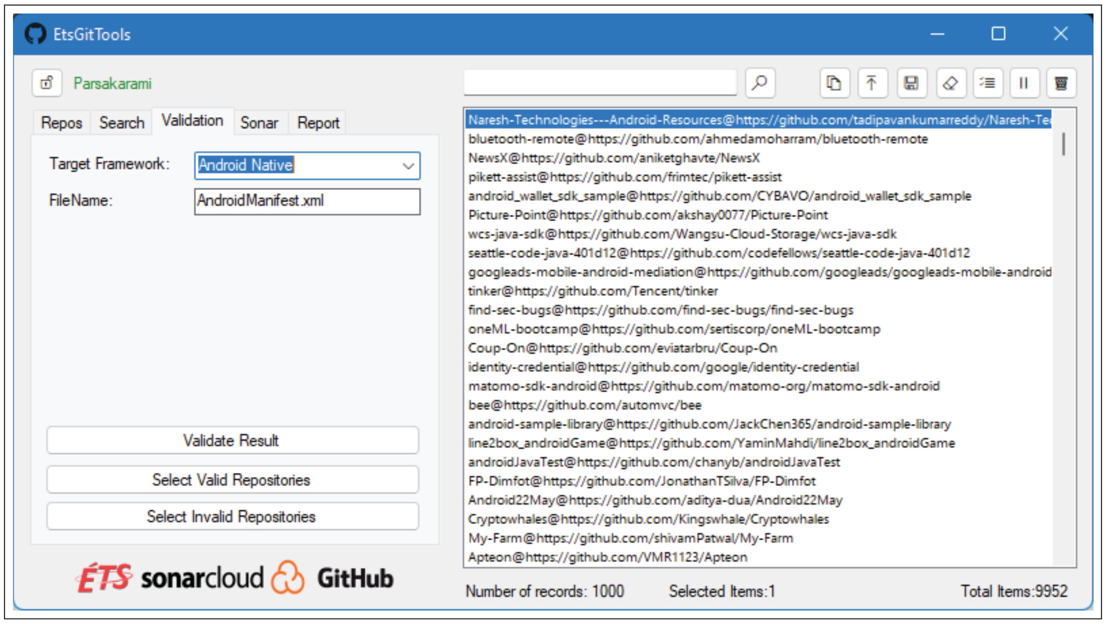
## 2.3.2 Filtering the results from GitHub search

The next step is to filter the findings repositories before starting the analysis phase. During our search process, we found out some repositories had been extracted using our search query, but they are not real mobile applications. There is no straightforward way to find only actual mobile application source code using GitHub search API. Prior to importing the source codes into SonarCloud, we must ensure that the repositories are real mobile applications' source codes. To overcome this limitation, we decided to filter repositories after the search.

Each mobile application source code has some mandatory files regarding the development framework. These files must be included in the source code path in order to build and run the source code without a problem. Our knowledge of this fact enables us to validate the searched results. Our method is to find this mandatory file within the searched results regarding the targeted mobile framework. In this case, the presence of the file indicates it is an actual application.

For the mobile applications developed using the Android native framework, we search for finding *AndroidManifest* file. Every Android application developed using the Android native framework must have an *AndroidManifest.xml* file (exactly that name) at the root of the project source set. The manifest file describes essential information about an application to the Android build tools, the Android operating system, and Google Play. For the mobile applications developed using the react-native framework, we search for finding the *Package.json* file. It is a file created by NPM (Node Package Manager). It contains all the metadata, dependencies, and scripts for the app.

Figure 2.18 shows how to validate the search results. After selecting the target mobile framework from the box, the FileName field automatically gets filled based on the targeted framework.



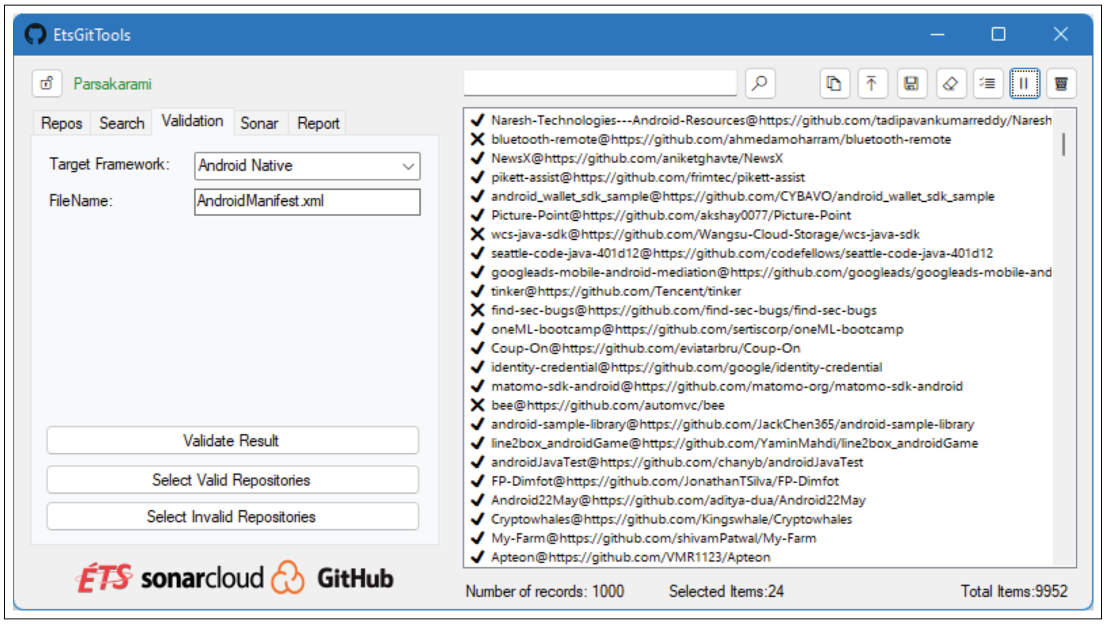
Target Framework: Android Native
FileName: AndroidManifest.xml

```
Naresh-Technologies---Android-Resources@https://github.com/tadipavankumarreddy/Naresh-Te
bluetooth-remote@https://github.com/ahmedamoharram/bluetooth-remote
NewsX@https://github.com/aniketghavte/NewsX
pikett-assist@https://github.com/frimtec/pikett-assist
android_wallet_sdk_sample@https://github.com/CYBAVO/android_wallet_sdk_sample
Picture-Point@https://github.com/akshay0077/Picture-Point
wcs-java-sdk@https://github.com/Wangsu-Cloud-Storage/wcs-java-sdk
seattle-code-java-401d12@https://github.com/codefellows/seattle-code-java-401d12
googleads-mobile-android-mediation@https://github.com/googleads/googleads-mobile-android
tinker@https://github.com/Tencent/tinker
find-sec-bugs@https://github.com/find-sec-bugs/find-sec-bugs
oneML-bootcamp@https://github.com/sertiscorp/oneML-bootcamp
Coup-On@https://github.com/eviatarbru/Coup-On
identity-credential@https://github.com/google/identity-credential
matomo-sdk-android@https://github.com/matomo-org/matomo-sdk-android
bee@https://github.com/automvc/bee
android-sample-library@https://github.com/JackChen365/android-sample-library
line2box_androidGame@https://github.com/YaminMahdi/line2box_androidGame
androidJavaTest@https://github.com/chanyb/androidJavaTest
FP-Dimfot@https://github.com/JonathanTSilva/FP-Dimfot
Android22May@https://github.com/aditya-dua/Android22May
Cryptowhales@https://github.com/Kingswhale/Cryptowhales
My-Farm@https://github.com/shivamPatwal/My-Farm
Apteon@https://github.com/VMR1123/Apteon
```

Number of records: 1000    Selected Items:1    Total Items:9952

Figure 2.18 The GitToSonar tool validation by file name

GitToSonar validates selected repositories and indicates that using a check or cross mark. A validation feature examines each repository individually to see if the files are present inside. Depending on the selection number, this process might take a long time. Figure 2.19 indicates the validation results of sample repositories.



Target Framework: Android Native
FileName: AndroidManifest.xml

```
✓ Naresh-Technologies---Android-Resources@https://github.com/tadipavankumarreddy/Naresh
✗ bluetooth-remote@https://github.com/ahmedamoharram/bluetooth-remote
✓ NewsX@https://github.com/aniketghavte/NewsX
✓ pikett-assist@https://github.com/frimtec/pikett-assist
✓ android_wallet_sdk_sample@https://github.com/CYBAVO/android_wallet_sdk_sample
✓ Picture-Point@https://github.com/akshay0077/Picture-Point
✗ wcs-java-sdk@https://github.com/Wangsu-Cloud-Storage/wcs-java-sdk
✓ seattle-code-java-401d12@https://github.com/codefellows/seattle-code-java-401d12
✓ googleads-mobile-android-mediation@https://github.com/googleads/googleads-mobile-and
✓ tinker@https://github.com/Tencent/tinker
✗ find-sec-bugs@https://github.com/find-sec-bugs/find-sec-bugs
✓ oneML-bootcamp@https://github.com/sertiscorp/oneML-bootcamp
✓ Coup-On@https://github.com/eviatarbru/Coup-On
✓ identity-credential@https://github.com/google/identity-credential
✓ matomo-sdk-android@https://github.com/matomo-org/matomo-sdk-android
✗ bee@https://github.com/automvc/bee
✓ android-sample-library@https://github.com/JackChen365/android-sample-library
✓ line2box_androidGame@https://github.com/YaminMahdi/line2box_androidGame
✓ androidJavaTest@https://github.com/chanyb/androidJavaTest
✓ FP-Dimfot@https://github.com/JonathanTSilva/FP-Dimfot
✓ Android22May@https://github.com/aditya-dua/Android22May
✓ Cryptowhales@https://github.com/Kingswhale/Cryptowhales
✓ My-Farm@https://github.com/shivamPatwal/My-Farm
✓ Apteon@https://github.com/VMR1123/Apteon
```

Number of records: 1000    Selected Items:24    Total Items:9952

Figure 2.19 The GitToSonar tool validation results

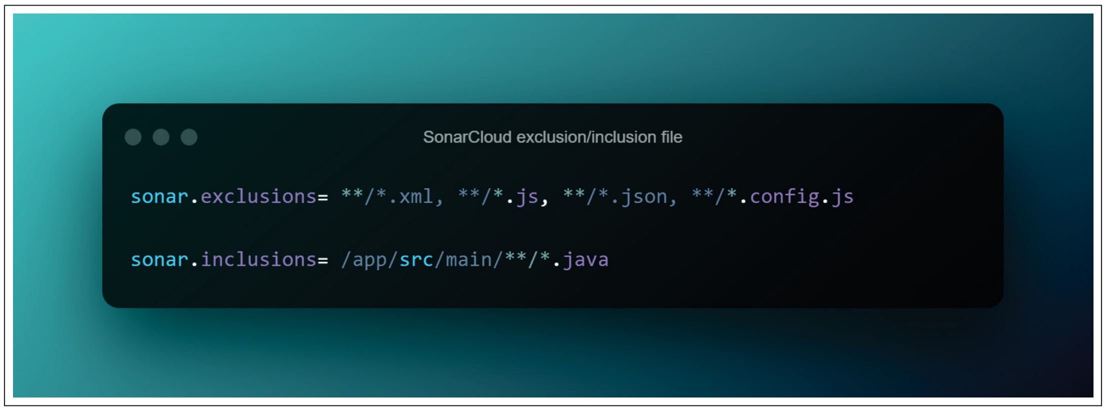
After finding the mobile applications and validating the results, we fork them into a GitHub account to continue our execution phase.

## 2.3.3 Excluding the third-party libraries from the analysis

Mobile applications heavily depend on third-party libraries and plugins. A mobile application usually contains generated code and source code for the third-party libraries or plugins used by the app. Hence, to analyze the application source code in our study, we need to consider only the source code related to the application, not to the libraries and plugins.

As stated, SonarCloud allows us to add specific exclusion and inclusion criteria to narrow the analysis scope. In this step, we exclude the third-party libraries and plugins from the analysis scope in SonarCloud. Moreover, we include the source code files written in the programming language of the targeted framework. For instance, in the Android native framework, we added an inclusion criterion that makes SonarCloud only analyze Java files.

We can define these exclusion and inclusion criteria by adding a file in the project's root path. This SonarCloud feature enables us to restrict the analysis scope by file type and file path. We must add the ".sonarcloud.properties" file (precisely that name) in each repository to define the analysis scope. If the file name is incorrect or the file content is not in a proper format, it will not change the analysis scope. In our study, this file consists of two parts, inclusion and exclusion. In the first line, we defined the exclusion criterion. We exclude certain file types and paths from our analysis based on exclusion criteria. In the second line, we determine the scope of the source code we want to analyze by SonarCloud. Figure 2.20 shows an example of the ".sonarcloud.properties" file.

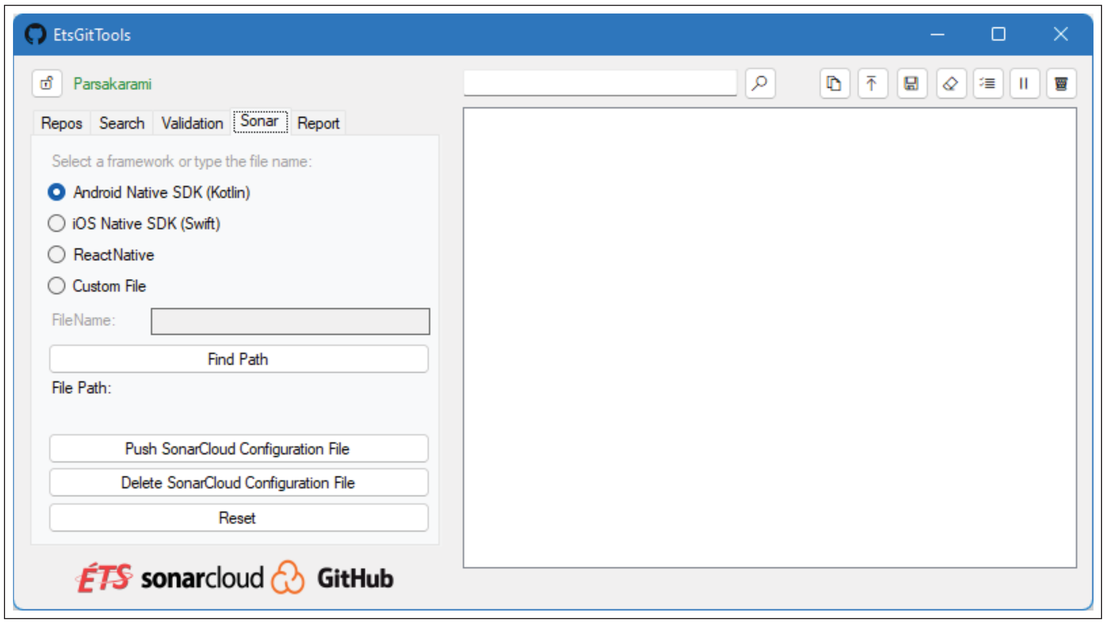


```
sonar.exclusions= **/*.xml, **/*.js, **/*.json, **/*.config.js

sonar.inclusions= /app/src/main/**/*.java
```

Figure 2.20    Example of SonarCloud exclusion/inclusion file

After finding the root path of the source code within each repository, we create the exclusion/inclusion configuration file according to the root path and the targeted framework. Then, GitToSonar creates the exclusion/inclusion configuration file. Finally, we push the file into each repository using our tool. Figure 2.21 shows the Sonar tab in the GitToSonar, which is responsible for creating and pushing exclusion/inclusion configuration files to every GitHub repository.



Select a framework or type the file name:
- [✓ Android Native SDK (Kotlin)]
- iOS Native SDK (Swift)
- ReactNative
- Custom File

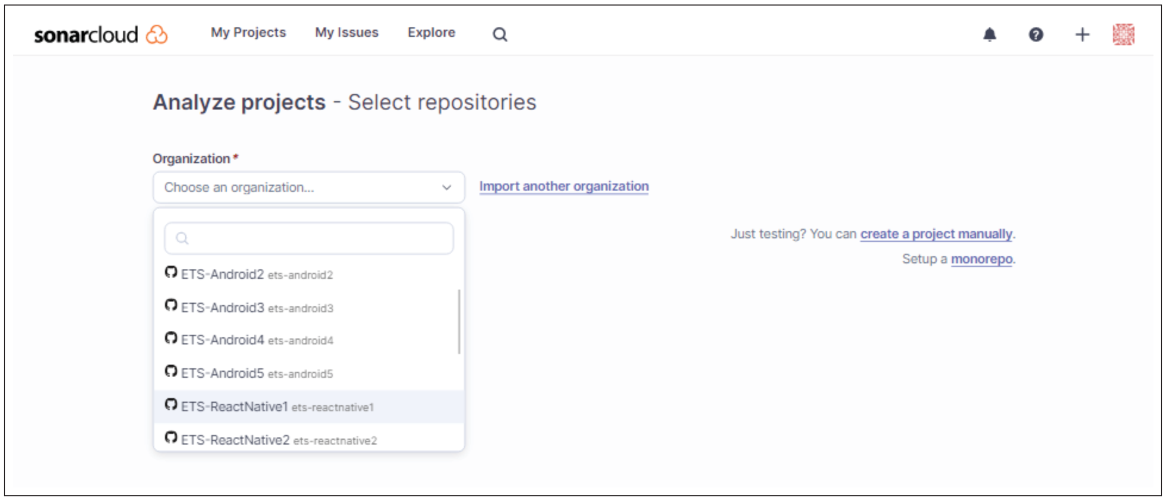
FileName:

File Path:

Figure 2.21 The GitToSonar tool SonarCloud configuration page

### 2.3.4 Importing mobile applications in SonarCloud

After preparing the prerequisites, it is time to start the analysis of the mobile applications. In this step, we only used one of the built-in features in SonarCloud. This feature enables us to log in using the GitHub account and import the mobile application repositories to SonarCloud. We created a GitHub account for our empirical study in order to fork mobile application repositories after the filtering step. This GitHub account has a total of 10 GitHub organizations. We created these organizations based on the targeted mobile frameworks and mobile application size category we defined earlier in the study design section. These organizations allow us to keep the mobile application repositories separately. Then, we import them into SonarCloud for analysis. Figure 2.22 shows the selection of the organization in the SonarCloud.



**Analyze projects** - Select repositories

Organization *

Choose an organization...

Import another organization

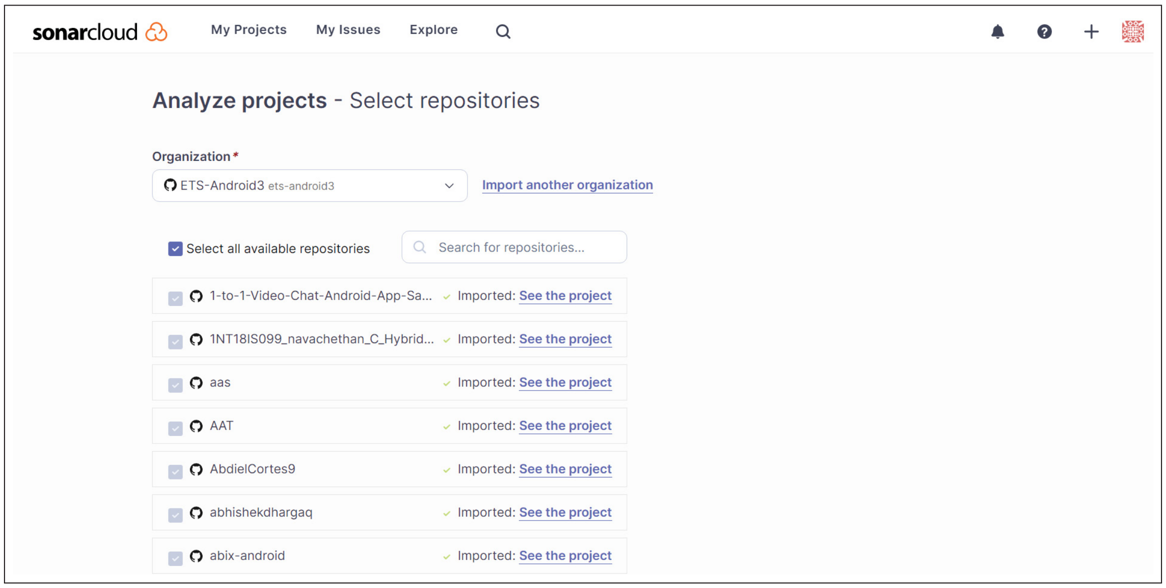
Just testing? You can create a project manually.

Setup a monorepo.

ETS-Android2 ets-android2
ETS-Android3 ets-android3
ETS-Android4 ets-android4
ETS-Android5 ets-android5
ETS-ReactNative1 ets-reactnative1
ETS-ReactNative2 ets-reactnative2

Figure 2.22 Selecting GitHub organization in SonarCloud

After selecting the organization, we select all the and import them into SonarCloud. Figure 2.23 indicates how we import the mobile application source codes in SonarCloud. Soon as the import process is finished, SonarCloud starts to analyze the projects.



**Analyze projects** - Select repositories

Organization *

ETS-Android3 ets-android3

Import another organization

Select all available repositories

Search for repositories...

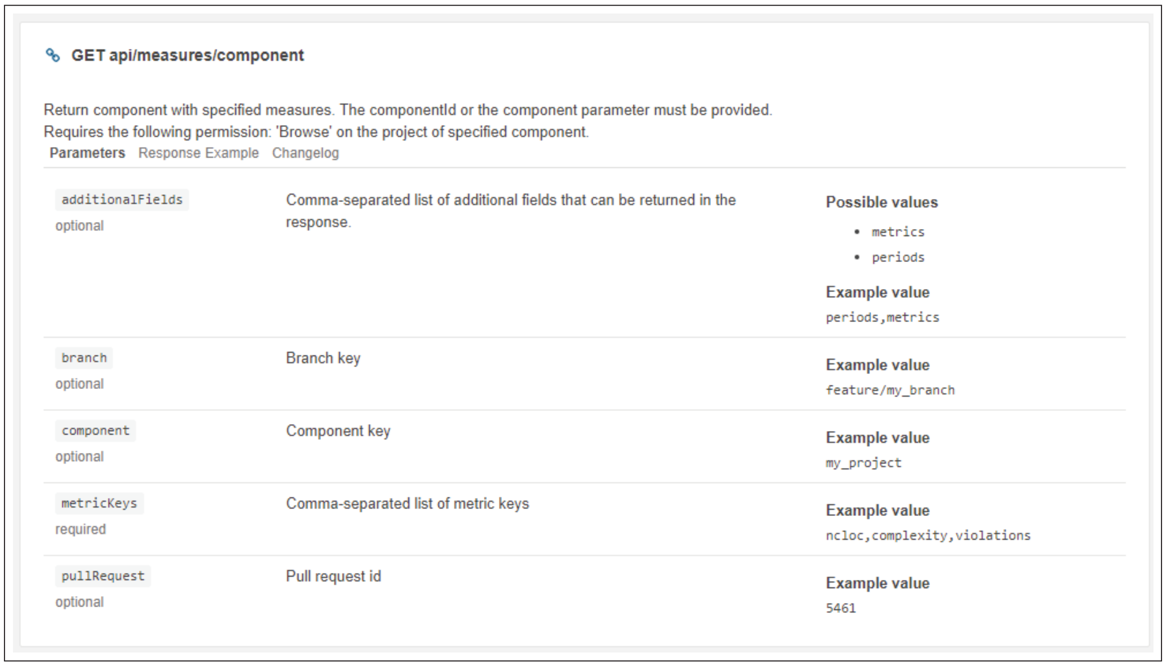
1-to-1-Video-Chat-Android-App-Sa... ✓ Imported: See the project
1NT18IS099_navachethan_C_Hybrid... ✓ Imported: See the project
aas ✓ Imported: See the project
AAT ✓ Imported: See the project
AbdielCortes9 ✓ Imported: See the project
abhishekdhargaq ✓ Imported: See the project
abix-android ✓ Imported: See the project

Figure 2.23 Importing GitHub repositories to SonarCloud

### 2.3.5 Generating reports

After the analysis is completed, we use the GitToSonar tool to collect the metrics values of the mobile applications. In GitToSonar, we used SonarCloud's Rest Web APIs to send requests and retrieve information. After collecting source code metric values, we generate the report files. Later, we will use these files to interpret the analysis results in the next phase. SonarCloud automatically assigns a unique identifier key to each project after importing source codes. We request accessing each project analysis result by passing this key and utilizing SonarCloud's rest APIs. This step is done using our GitToSonar tool.

Figure 2.24 shows the details of the measure API in SonarCloud. The figure also indicates API's parameter. Among these available parameters, we use two of them. The first is the component key which is the unique key for each project. The second is the comma-separated list of metric keys we want to obtain the values.

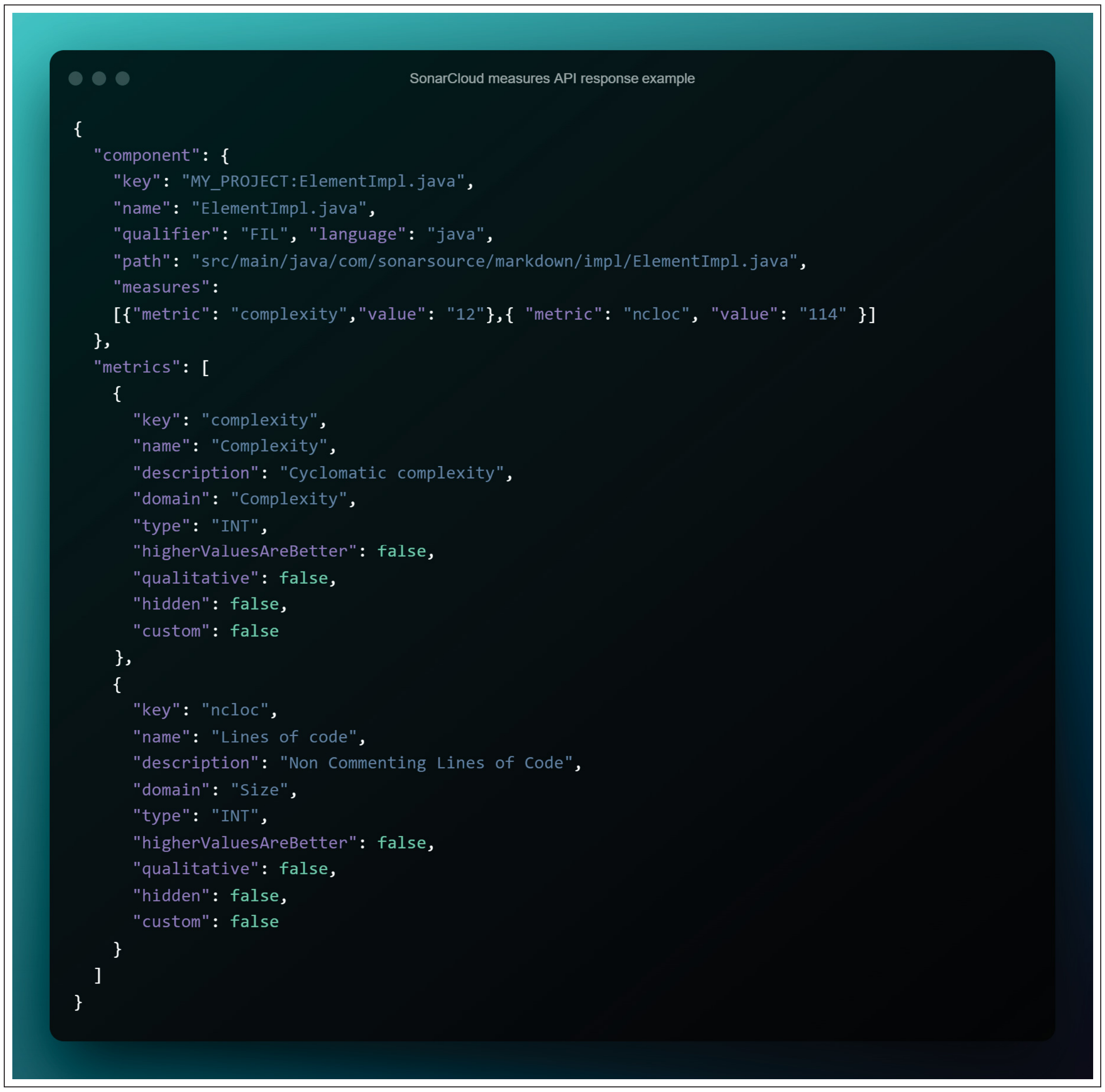

**GET api/measures/component**

Return component with specified measures. The componentId or the component parameter must be provided.
Requires the following permission: 'Browse' on the project of specified component.

| | | |
|---|---|---|
| `additionalFields` optional | Comma-separated list of additional fields that can be returned in the response. | **Possible values** • `metrics` • `periods` **Example value** `periods,metrics` |
| `branch` optional | Branch key | **Example value** `feature/my_branch` |
| `component` optional | Component key | **Example value** `my_project` |
| `metricKeys` required | Comma-separated list of metric keys | **Example value** `ncloc,complexity,violations` |
| `pullRequest` optional | Pull request id | **Example value** `5461` |

Figure 2.24 The measure API description in SonarCloud

Figure 2.25 illustrates the measures API response example. In response to this API call, SonarCloud returns the requested metric values of the component. The return results are provided in JSON format.

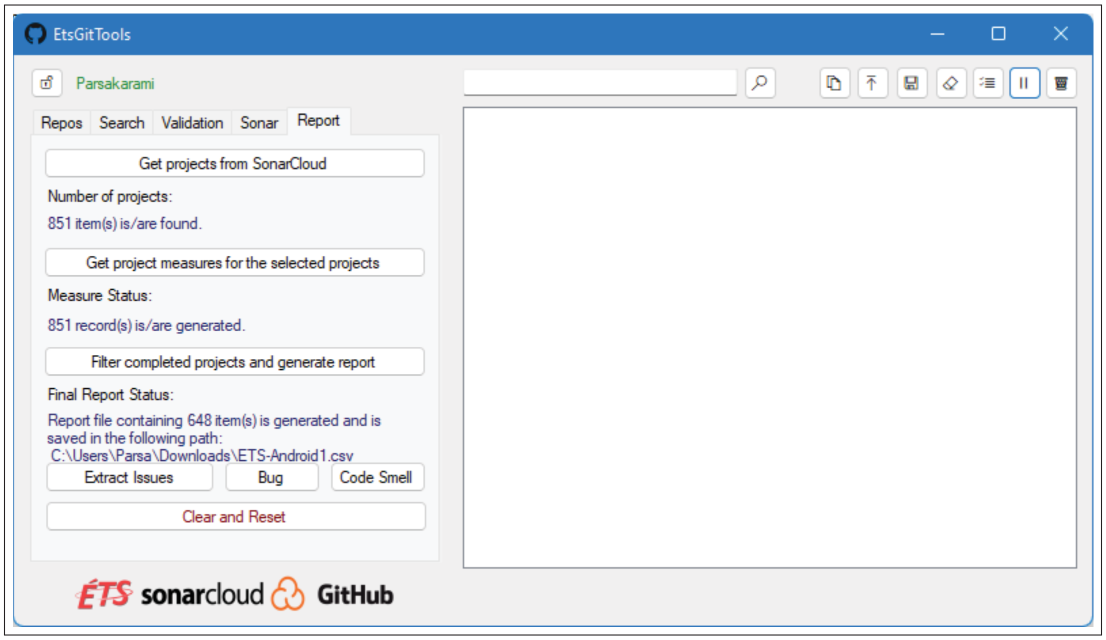


```
{
  "component": {
    "key": "MY_PROJECT:ElementImpl.java",
    "name": "ElementImpl.java",
    "qualifier": "FIL", "language": "java",
    "path": "src/main/java/com/sonarsource/markdown/impl/ElementImpl.java",
    "measures":
    [{"metric": "complexity","value": "12"},{ "metric": "ncloc", "value": "114" }]
  },
  "metrics": [
    {
      "key": "complexity",
      "name": "Complexity",
      "description": "Cyclomatic complexity",
      "domain": "Complexity",
      "type": "INT",
      "higherValuesAreBetter": false,
      "qualitative": false,
      "hidden": false,
      "custom": false
    },
    {
      "key": "ncloc",
      "name": "Lines of code",
      "description": "Non Commenting Lines of Code",
      "domain": "Size",
      "type": "INT",
      "higherValuesAreBetter": false,
      "qualitative": false,
      "hidden": false,
      "custom": false
    }
  ]
}
```

Figure 2.25 The response example of the measure API in SonarCloud

To generate the report file, we followed these three steps. First, the GitToSonar tool retrieves all the keys of the projects imported into SonarCloud. Second, GitToSonar calls measure API for each project iteratively and collects metric values. And finally, the tool generates the two report files for each targeted framework.

Figure 2.26 shows the results for the first category of the mobile app' source codes. GitToSonar collects the values. Then the tool generates the report files and saves them in the specified path.


EtsGitTools
Parsakarami
Repos
Search
Validation
Sonar
Report
Get projects from SonarCloud
Get project measures for the selected projects
Filter completed projects and generate report
Extract Issues
Bug
Code Smell
Clear and Reset
ÉTS sonarcloud GitHub

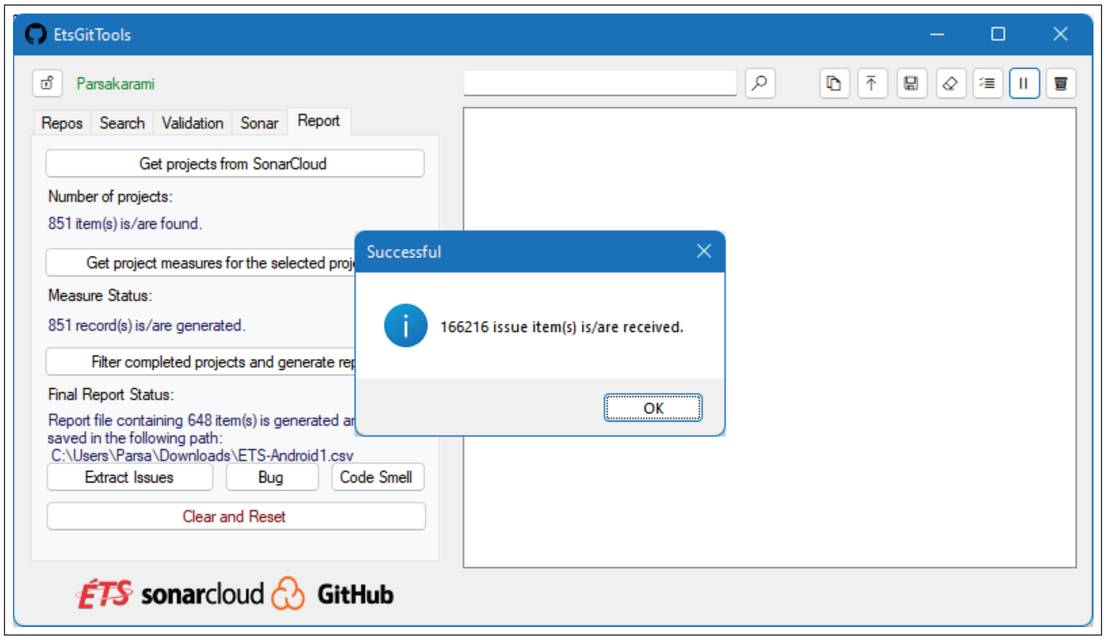
Number of projects:
851 item(s) is/are found.

Measure Status:
851 record(s) is/are generated.

Final Report Status:
Report file containing 648 item(s) is generated and is saved in the following path:
C:\Users\Parsa\Downloads\ETS-Android1.csv

Figure 2.26 The generating report in GitToSonar

In addition, to generate reports, the GitToSonar enables us to obtain the project's issues in detail. It helps us identify the most commonly encountered bugs and code smells in mobile applications developed using each targeted framework. The occurring reason for these issues leads us to proceed with our qualitative analysis. An overview of the number of projects' issues, including bugs and code smells, is shown in Figure 2.29. The tool saves the report file to the specified path when you click the Bug or Code Smell buttons.



Number of projects:
851 item(s) is/are found.

Measure Status:
851 record(s) is/are generated.

Final Report Status:
Report file containing 648 item(s) is generated ar
saved in the following path:
C:\Users\Parsa\Downloads\ETS-Android1.csv

166216 issue item(s) is/are received.

Figure 2.27 The number of collected issues using the GitToSonar tool

## 2.4 Results and Discussion

Here, we address the thesis's primary question using Qualitative and Quantitative analysis of our empirical study and discuss the results. First, we begin with a qualitative assessment of mobile application source code quality based on two issue types identified by SonarCloud. These two types are bugs and code smells. We conduct qualitative analysis to examine the impact of using mobile frameworks on generating bugs and code smells in source code. We scrutinize the five most frequently encountered bugs and code smells in mobile applications in each targeted framework. After analyzing and interpreting qualitative data, we move on to quantitative analysis. There, we apply descriptive statistics methods to answer the thesis's question. Finally, we interpret the results and discuss using statistical values of quality metrics.

### 2.4.1 Qualitative analysis

A qualitative analysis of issues is necessary to answer the thesis's question. We use this analysis to evaluate the impact of choosing a mobile application development framework on the design quality. The bugs and code smells are the two source code quality factors that impact the application design. The fewer bugs and code smells in a source code enhances the quality of source code and improves maintainability. We can collect source code analysis results from SonarCloud through our GitToSonar tool. GitToSonar also helps us to generate an inclusive report file for the bugs and code smells. The generated report file by the tool gives us the list of bugs and code smells for each targeted framework. Then, we ordered and filtered the results to find the most frequent bugs and code smells in each mobile framework. Analyzing the most frequent bugs and code smells in mobile applications' source codes is necessary to answer the thesis's question. Examining the causes of these bugs and code smells can provide insight into the impact of mobile frameworks on design quality. The following tables ( 2.4, 2.5, 2.6, and 2.7) show the most frequent bugs and code smells in mobile applications' source code that were developed using each targeted mobile framework.

Table 2.4 Most frequent bugs in mobile applications developed using Android native framework

| **Bug Description** | **Number of occurrences** |
|---|---|
| Cast one of the operands of this "type A" to a "type B" | 9744 |
| Either re-interrupt this method or re-throw the "InterruptedException" that can be caught here | 8299 |
| Do something with the "type A" value returned by "method name" | 3071 |
| Use another way to initialize this instance | 1947 |
| Strings and Boxed types should be compared using "equals()" | 1632 |

Table 2.5 Most frequent bugs in mobile applications developed using React Native framework

| Bug Description | Number of occurrences |
|---|---|
| Expected an assignment or function call and instead saw an expression | 7553 |
| Consider using "forEach" instead of "map" as its return value is not being used here | 1454 |
| This function expects [#N] argument, but [#M] were provided | 1419 |
| Unreachable code | 1031 |
| Add a "return" statement to this callback | 865 |

Table 2.6 Most frequent code smells in mobile applications developed using Android native framework

| Code Smell Description | Number of occurrences |
|---|---|
| Rename this [ local variable, constant, method, package, ... ] to match the regular expression " ^[a-z][a-zA-Z0-9] * $" | 421436 |
| This block of commented-out lines of code should be removed | 150324 |
| Make "A" a static final constant or non public and provide accessors if needed | 103138 |
| #N duplicated blocks of code must be removed | 93994 |
| Refactor this method to reduce its Cognitive Complexity from [#N] to the [#M] allowed | 63119 |

To perform a qualitative analysis, we must identify the source of bugs and code smells on each targeted framework. Some code smells and bugs in source code may be caused by developers. Developers can cause issues by using bad practices in coding. These bad practices include typos in code, hard-coding parts of code, naming conventions, etc. In these cases, the bugs and smells are unrelated to the framework's choice. But in other cases, the issues can be produced by the

Table 2.7 Most frequent code smells in mobile applications developed using React Native framework

| Code Smell Description | Number of occurrences |
|---|---|
| Remove the unused function parameter "A" or rename it to "B" to make intention explicit | 88326 |
| Remove this unused import of "A" | 76577 |
| "property A" is already declared in the upper scope | 69882 |
| Remove this commented out code | 47361 |
| Unexpected empty [block, function, method, ...] | 24773 |

framework. If a bug or code smell is generated by the framework, consider it as an impact of choosing the framework on design quality.

To investigate the source of each issue, we selected the top 5 most frequent bugs and code smells from each targeted development framework. In order to determine the cause of each bug or code smell, we selected ten projects randomly. A key goal is identifying whether the development framework or the developer causes the issues in these projects. We proceeded in the following manner to randomly select the ten projects. First, we picked two projects in each size category in each targeted development framework. We choose projects of different sizes to ensure our samples cover all scopes. Second, we check if the project has that specific bug or code smell. Then, we investigate the reason for the occurrences. If the selected project does not have that bug or code smell, we choose another project from the same size category. This process allowed us to choose ten projects for each bug or code smell. Finally, we check the issue's origin to find out its occurrence. The following tables present the investigation results of bugs and code smells in the targeted framework. The first column in the tables indicates the bug or code smell in each development framework. The second column shows the number of occurrences for each issue in total. The rest of the columns show each project ID we took in our evaluation. In order to indicate the reason for each bug or code smell in these projects, we used two letters. If the development framework generates an issue, we use the letter "F" to indicate that and we use the letter "D" to demonstrate the developer produces that bug or code smell.

Table 2.8 The reason for occurrences of the most frequent bugs in mobile applications developed using the Android native framework

| **Description** | **Total** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Cast one of the operands of this "type A" to a "type B" | 9744 | D | D | D | D | D | D | D | D | D | D |
| Either re-interrupt this method or re-throw the "InterruptedException" that can be caught here | 8299 | D | D | D | D | D | D | D | D | D | D |
| Do something with the "type A" value returned by "method name" | 3071 | D | D | D | D | D | D | D | D | D | D |
| Use another way to initialize this instance | 1947 | D | D | D | D | D | D | D | D | D | D |
| Strings and Boxed types should be compared using "equals()" | 1632 | D | D | D | D | D | D | D | D | D | D |

Table 2.9 The reason for occurrences of the most frequent bugs in mobile applications developed using the React Native framework

| **Description** | **Total** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Expected an assignment or function call and instead saw an expression | 7553 | F | F | F | F | F | F | F | F | F | F |
| Consider using "forEach" instead of "map" as its return value is not being used here | 1454 | D | D | D | D | D | D | D | D | D | D |
| This function expects [#N] argument, but [#M] were provided | 1419 | D | F | D | F | F | F | D | D | F | D |
| Unreachable code | 1031 | D | D | D | D | D | D | D | D | D | D |
| Add a "return" statement to this callback | 865 | D | D | D | D | D | D | D | D | D | D |

Table 2.10 The reason for occurrences of the most frequent code smells in mobile applications developed using the Android native framework

| **Description** | **Total** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Rename this [local variable, constant, method, package, ...] to match the regular expression " ̂[a-z][a-zA-Z0-9] * $" | 421436 | D | D | D | D | D | D | D | D | D | D |
| This block of commented-out lines of code should be removed | 150324 | D | D | D | D | D | D | D | D | D | D |
| Make "A" a static final constant or non public and provide accessors if needed | 103138 | D | D | D | D | D | D | D | D | D | D |
| #N duplicated blocks of code must be removed | 93994 | D | D | D | D | D | D | D | D | D | D |
| Refactor this method to reduce its Cognitive Complexity from [#N] to the [#M] allowed | 63119 | D | D | D | D | D | D | D | D | D | D |

Table 2.11 The reason for occurrences of the most frequent code smells in mobile applications developed using the React Native framework

| **Description** | **Total** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Remove the unused function parameter "A" or rename it to "B" to make intention explicit | 88326 | D | D | D | D | D | D | D | D | D | D |
| Remove this unused import of "A" | 76577 | D | D | D | D | D | D | D | D | D | D |
| "property A" is already declared in the upper scope | 69882 | D | D | D | D | D | D | D | D | D | D |
| Remove this commented out code | 47361 | D | D | D | D | D | D | D | D | D | D |
| Unexpected empty [block, function, method, ...] | 24773 | D | D | D | D | D | D | D | D | D | D |

According to our examination, developers caused the bugs and code smells in most cases. But in some cases, we found out bugs are generated by the chosen mobile framework. Based on the results, developers are the main reason for causing the top five most frequent bugs in the Android native framework. The results also indicate the same fact for the code smells in this framework. As a result, we can conclude that developers are the primary source of issues in mobile applications' source codes written with the Android native framework in our investigation. In other words, choosing an Android native framework as a mobile application development framework does not directly impact the design quality of applications.

We also investigated the mobile applications developed using React Native development framework. The results are not quite the same in React Native compared to the Android native framework. In the case of code smells, both frameworks have the same results. We found the most frequent code smells caused by developers among the randomly selected projects. As a result, we can state choosing a mobile framework can increase the number of code smells in source codes. In the case of bugs, sometimes the framework generated the bug, specifically the React Native framework. In our experiment, the most common bug in the projects developed using the React Native framework is "Expected an assignment or function call and instead saw an expression.". In JavaScript functions, it is not mandatory to use a return statement explicitly. Hence, it is not a bug from a JavaScript developer's point of view. However, in the React Native framework, all the component generator functions must return a value, and SonarCloud considered this a bug. Therefore, this difference might confuse the developers, and we marked "F" for this bug.

Another example is the third most frequent bug in React Native (see Table 2.9). By looking into the analyzed projects, we found that in some cases, this bug is related to the developer and, in others React framework causes the bug, as shown in Table 2.9. This bug happens when developers pass fewer or more arguments than the actual number of parameters defined in the function signature. In some projects, we found out it happened because of the developer's mistake and more parameters passed by the developer in a function call. If developers pass fewer parameters in a function call, this is not considered a bug in the JavaScript programming language. But, React Native framework considers this as an error. In some projects, we found developers passed more parameters in function calls.

To summarize, we identified two frequent bugs related to the React Native framework. Since these two bugs are not bugs in JavaScript, developers might be confused about them, and we consider this as the impact of the development framework on source code quality. So, we can state a cross-platform mobile framework can impact some quality metrics.

### 2.4.2 Quantitative analysis

Here, we use a statistical analysis method to determine the impact of choosing a mobile application development framework on source code quality. The GitToSonar tool collects the analysis results from SonarCloud and generates a report file that includes the two quality metrics presented in table 2.1. Then, we analyze the results quantitatively to find if meaningful similarities or differences happened between metrics in our two sets of applications' source codes. Projects developed using the Android native framework are included in the first set. The second set of applications' source code contains those developed using React Native.

Table 2.12 reports the metrics for a sample of projects for both Android and React Native frameworks. The name of the project, and the framework used to develop the project, are given in the first row. The projects in this table were selected randomly.

Table 2.12 A small sample of projects and their metrics

| **App Name (framework)** | yemmsg (Android) | litho (Android) | musicm (Android) | EDX (Android) | Lupa (React) | meala (React) | accApp (React) | elektra (React) |
|---|---|---|---|---|---|---|---|---|
| **Number of Bugs** | 107 | 42 | 165 | 24 | 41 | 23 | 21 | 53 |
| **Number of Code Smells** | 9602 | 3139 | 5165 | 1997 | 2960 | 655 | 874 | 548 |

We use statistical hypothesis testing to address the thesis's question. We define our hypothesis to prove whether using mobile frameworks impacts source code quality. Our null hypothesis ($H_0$)

is that using a mobile framework does not negatively impact source code quality, specifically by adding bugs and code smells in source code. This hypothesis indicates the sum of the number of bugs and code smells in the two groups does not meaningfully differ. On the other hand, our alternative hypothesis ($H_a$) stated using a mobile framework impacts the source code quality or that choosing a framework increases the number of bugs and code smells in source code.

Table 2.13 shows the results of the descriptive analysis for each code metric besides the data distribution in our dataset. The "Normal_dist" column determines if the data is distributed normally or not. Moreover, the table also displays mean, median, and standard deviation (SD) values for the code metrics for each framework.

Table 2.13 The descriptive statistical of the dataset

| **Variable** | **Framework** | **Normal_dist** | **Mean** | **SD** | **Median** |
|---|---|---|---|---|---|
| Bugs | Android SDK | No | 8.76 | 23.85 | 2 |
| Bugs | React Native | No | 7.18 | 38.58 | 1 |
| CodeSmell | Android SDK | No | 501.48 | 1549.8 | 165 |
| CodeSmell | React Native | No | 197.56 | 714.73 | 45 |

According to the results of the descriptive statistics, none of the variables in our dataset followed a normal distribution. The distribution of our variables leads us to select our analysis method. In the case of non-normally distributed data, the Mann-Whitney U-test is recommended (Ramachandran & Tsokos (2021)).

We hypothesized that using a mobile framework does not impact code quality and does not cause more bugs and code smells in source code. According to the boxplots in 2.28 and 2.29, the number of bugs (box plot a in Group A) is not meaningfully different among frameworks. But for code smells (box plot a Group B), it shows the number of code smells in android is meaningfully higher than in React Native on average. According to the descriptive statistical table, the median value for the Android native framework is about four times bigger than its value in the react native framework. As a result, the boxplots cannot support our null hypothesis.

In other words, it indicates using a mobile framework impacts some source code quality metrics. Table 2.14 reports the Mann-Whitney test results.

Table 2.14 The result of the Mann-Whitney test

| **variable** | **mw.p.value** | **mw.estimate** | **effect_size** |
|---|---|---|---|
| Bugs | < 0.05 | 0.999996653898586 | 0.272 (small) |
| CodeSmell | < 0.05 | 82.9999447657932 | 0.712 (large) |

To do so, we assessed the *P-value* of the Mann-Whitney test to support or reject our null hypothesis. The p-value (probability value) indicates whether the null hypothesis is true. A low *P-value* (less than 0.05) rejects the null hypothesis and supports the alternative hypothesis. Based on table 2.14, the null hypothesis is rejected since the *P-value* for *bug* and *code smell* are less than the significant level (0.05). In other words, it indicates a mobile framework can impact the source code quality by increasing the bugs or code smells.

In addition, we drew a box plot for each metric for the targeted mobile frameworks. Drawing the box plot chart for each code metric helps us to find the possible similarities and differences. Figures 2.28 and 2.29 show the box plots for each code metric for each targeted framework. Figure 2.28 shows the number of bugs metric has similar values distributions between the two targeted frameworks. Figure 2.29 displays the number of code smells does not have similar distributions in the two frameworks. The average value for the number of code smells meaningfully is lower in mobile applications developed using React Native framework. As a result, it seems the React Native development framework enhances the quality of mobile application source codes.

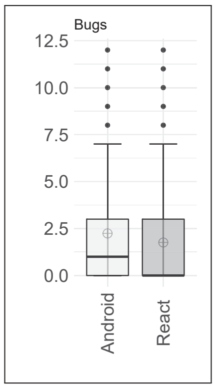


Figure 2.28 Descriptive statistical - Number of bugs box plot

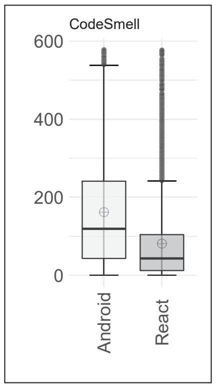


Figure 2.29 Descriptive statistical - Number of code smells box plot

## 2.5 Limitations of the Study

In this section, I discuss the limitations of the empirical investigation in my thesis. The first limitation of the empirical study is related to the code analyzer used in the study, which is SonarCloud. The results of the study are dependent on the SonarCloud metrics evaluations. This dependency can be considered a threat to the validity of the outcomes of our empirical investigation.

The selection of mobile applications' source code is the second limitation of our study. The size of the mobile application is a factor that might affect the results. To reduce this effect, we defined a size classification to collect mobile applications with various sizes. Moreover, mobile applications have been developed in various domains like Gaming, Online shopping, etc. Mobile frameworks might affect the application source code quality differently in various domains. In our investigation, we did not target a specific domain to find mobile applications. Hence, the validity of the results can be compromised by comparing mobile applications in different domains. A workaround to mitigate this effect is to use GitHub tags to find mobile applications from a similar domain.

Another limitation in our study is related to the heuristics we used to exclude third-party libraries and plugins from the analysis scope of the SonarCloud. There is a possibility that our approach does not completely exclude all third-party libraries and plugins. Therefore, it again can impact the analysis phase and final results. This should be tackled in future works.

Finally, our study was limited to two development frameworks. Future work should analyze more frameworks and more applications.

## 2.6 Conclusion

Following finding research gaps in our SLR, we designed and conducted the empirical study. In this study, we targeted two mobile frameworks and evaluated two sets of mobile application source codes to find whether selecting a mobile framework impacts code quality.

The empirical study consisted of three phases. In the first phase, we set the study scope and the tools to be used. We selected two mobile frameworks (Android native framework and ReactNative) and the source code analyzer tool. Moreover, we defined the mobile application size classification we wanted to analyze. In the second phase, we executed the study and collected data. We developed a tool called GitToSonar, to automate the execution phase. Our execution phase had seven steps. First, we searched for mobile applications from GitHub based on our criteria. In the second step, we validated our findings and filtered out real mobile application repositories. Third, we keep the filtered results in one account. In the fourth step, we excluded the third-party and plugin source codes from our analysis by creating SonarCloud exclusion inclusion configuration files. Fifth, we pushed those configuration files to each mobile application repository accordingly. Sixth, we imported all mobile applications to SonarCloud in order to analyze them. Finally, after SonarCloud completed the analysis, we collected the results and generated report files. In the last phase of the empirical study, we interpreted the results.

We used both qualitative and quantitative analysis to answer the question of the study. In the qualitative part, we found some mobile applications' source code in which a mobile development framework impacts the source code quality. We figured out this result by looking into the most frequent bugs and code smells in mobile application source codes. In the quantitative analysis, we hypothesized a mobile framework does not impact the source code quality. Specifically, it does not add bugs or code smells to the source code. We used the Mann-Whitney U test since the data distribution was not normal. The p-values (probability values) of the code metrics in two groups of mobile applications are less than the significance level (less than 0.05). Hence, we rejected our hypothesis. Quantitative and qualitative analysis results led us to conclude a mobile framework can impact the source code quality in some metrics. In the qualitative analysis, we saw a mobile framework can generate some bugs and reduce the source code quality of a mobile application. In the quantitative analysis, we saw the average value for the number of code smells in React Native framework is lower than the Android native framework. In other words, it seems the React Native framework enhances source code quality compared to Android native framework.

# CONCLUSION AND RECOMMENDATIONS

This thesis's goal was to investigate the impact of mobile development frameworks on the quality of the applications. To achieve this goal, we first carried out a systematic literature review of existing studies that evaluated mobile development frameworks. For this purpose, we followed the approach outlined in (Kitchenham (2004)). In the SLR, we reviewed sixty-one existing studies that compared mobile frameworks. The SLR identified the most studied mobile frameworks. The SLR also revealed the gaps between the most researched and most used mobile frameworks in practice. The SLR enabled us to identify the evaluation criteria, perspectives, and methodologies used by other researchers. We observed that: 1) all the studies that used experiments to analyze the mobile frameworks used a prototype application to do so; 2) none of these studies used a large set of mobile applications in their analysis, and 3) the evaluation criteria did not include criteria about the source code quality. As a result, we designed an empirical study that uses a large set of mobile applications to evaluate the impact of mobile frameworks on the quality of the applications.

In our empirical study, we selected two popular mobile frameworks to investigate the possible impact of choosing a framework on design quality. We targeted native and cross-platform frameworks to compare source codes. Then, we collected six thousand mobile applications' source codes (three thousand for each targeted framework) from GitHub. Then, we imported those source codes to the SonarCloud source code analyzer tool to analyze them. Since the number of our dataset is large enough, we developed a tool to do the tasks systematically. Our GitToSonar tool handles every step of our empirical study, from finding mobile source codes in GitHub to generating reports after analysis.

We found that choosing a mobile application development framework can affect the quality of the source code in terms of producing bugs or code smells. In our qualitative analysis, we analyzed the top five most frequent bugs and code smells in both targeted frameworks in more depth. We saw a mobile framework can generate some bugs and reduce the source code quality of a mobile application. The qualitative analysis results show the average value for the number of code smells in mobile applications' source code developed using the React Native framework is lower than this value in mobile applications developed using the Android native framework. It led us to figure out that a mobile framework can impact the source code quality of mobile applications. The qualitative analysis shows the React Native framework enhances source code quality compared to the Android native framework.

Moreover, we made our GitToSonar tool open-source in order to make it accessible to all researchers. The tool's source is available on GitHub at the following address:
https://github.com/Parsakarami/GithubTool.

This study touched the surface of analyzing mobile application development and mobile frameworks. We believe there is potential for extending this work in different directions. The first direction would be to use more source code quality metrics and employ other source code analyzer tools. The second direction is targeting more mobile frameworks in the study. Finally, the third direction would be to collect more mobile applications' source codes for the empirical investigation.

# APPENDIX I

## SELECTED PAPERS

**P1** F.Botella, P. Escribano, A.P. Benavent. "Selecting the best mobile framework for developing Web and hybrid mobile apps" Moreno L., de la Rubia Cuestas E.J., Penichet V.M.R., García-Peñalvo F.J. (Eds.), Proceedings of the XVII International Conference on Human Computer Interaction, Interacción 2016, Salamanca, Spain, September 13, - 16, 2016, ACM (2016), pp. 40:1-40:4.

**P2** M.Palmieri, I. Singh, and A. Cicchetti, "Comparison of Cross-Platform Mobile Development Tools" 16th International Conference on Intelligence in Next Generation Networks, 2012, pp. 179–186.

**P3** H.Heitkötter, S. Hanschke, T.A. Majchrzak "Evaluating Cross-platform Development Approaches for Mobile Applications" LNBIP, 140, Springer (2013), pp. 120-138.

**P4** S.Dhillon and Q. H. Mahmoud, "An Evaluation Framework for Cross-Platform Mobile Application Development Tools," Softw. Pract. Exp., vol. 45, no. 10, pp. 1331–1357, Oct. 2015.

**P5** Ville Ahti, Sami Hyrynsalmi, and Olli Nevalainen. 2016. "An Evaluation Framework for Cross-Platform Mobile App Development Tools: A case analysis of Adobe PhoneGap framework". In Proceedings of the 17th International Conference on Computer Systems and Technologies 2016 (CompSysTech '16). Association for Computing Machinery, New York, NY, USA, 41–48. DOI: https://doi.org/10.1145/2983468.2983484

**P6** Christoph Rieger, Tim A. Majchrzak, "Towards the definitive evaluation framework for cross-platform app development approaches", Journal of Systems and Software, Volume 153, 2019, Pages 175-199, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2019.04.001.

**P7** Michael Gonsalves, "Evaluating the mobile development frameworks Apache Cordova and Flutter and their impact on the development process and application characteristics", Master thesis, 2018.

**P8** Henning Heitk¨otter, Tim A. Majchrzak, Benjamin Ruland and Till Weber. "Evaluating Frameworks for Creating MobileWeb Apps", WEBIST 2013 - 9th International Conference on Web Information Systems and Technologies, 2013.

**P9** Xiaoping Jia, Aline Ebone, Yongshan Tan. "A Performance Evaluation of Cross-Platform Mobile Application Development Approaches", ACM/IEEE 5th International Conference on Mobile Software Engineering and Systems, 2018.

**P10** Dalmasso, Isabelle; Datta, Soumya Kanti; Bonnet, Christian; Nikaein, Navid. "Survey, Comparison and Evaluation of Cross Platform Mobile Application Development Tools", 9th International Wireless Communications and Mobile Computing Conference, IWCMC, 2013.

**P11** Andreas BIØRN-HANSEN, Tor-Morten GRØNLI, Gheorghita GHINEA. “A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development” , ACM Computing Surveys, Vol. 51, No. 5, Article 108, 2019.

**P12** Humayoun, Shah Rukh; Ehrhart, Stefan; Ebert, Achim. "Developing mobile apps using cross-platform frameworks: A case study", Human-Centred Design Approaches, Methods, Tools, and Environments - 15th International Conference, HCI International 2013

**P13** Sommer, Andreas ; Krusche, Stephan. "Evaluation of cross-platform frameworks for mobile applications", Multi-Conference on Software Engineering, 2013

**P14** Xanthopoulos, Spyros; Xinogalos, Stelios. "A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications", 6th Balkan Conference in Informatics, 2013

**P15** Granlund, Daniel; Johansson, Dan; Andersson, Karl; Brännström, Robert. "A Case Study of Application Development for Mobile and Location-Based Services", 15th International Conference on Information Integration and Web-Based Applications and Services, iiWAS, 2013

**P16** Peixin Que, Xiao Guo, Maokun Zhu. "A Comprehensive Comparison Between Hybrid and Native App Paradigms", 2016 8th International Conference on Computational Intelligence and Communication Networks, CICN 2016.

**P17** Ng Moon Hui, Liu Ban Chieng, Wen Yin Ting, Hasimah Hj Mohamed, Muhammad Rafie Hj Mohd Arshad. "Cross-Platform Mobile Applications for Android and iOS", 2013 6th Joint IFIP Wireless and Mobile Networking Conference, 2013.

**P18** Song Sun, Sanxing Cao. "The Web development technology research of Cross platform mobile application", International Conference on Machine Tool Technology and Mechatronics Engineering, ICMTTME, 2014.

**P19** Matteo Ciman, Ombretta Gaggi, Nicola Gonzo. "Cross-Platform Mobile Development: A Study on Apps with Animations", 29th Annual ACM Symposium on Applied Computing, SAC, 2014.

**P20** Ribeiro, André, Da Silva, Alberto Rodrigues. "Survey on cross-platforms and languages for mobile apps", 8th International Conference on the Quality of Information and Communications Technology, QUATIC, 2012.

**P21** Vishal Kumar, Kushwaha, Ajay Shriram. "Mobile Application Development Research Based on Xamarin Platform", 4th International Conference on Computing Sciences, ICCS, 2018.

**P22** Salma Charkaoui, Zakaria Adraoui, El Habib Benlahmar. "Cross-platform mobile development approaches", 3rd IEEE International Colloquium in Information Science and Technology, CIST, 2014.

**P23** Delia Lisandro, Galdamez Nicolas, Thomas Pablo, Corbalan Leonardo, Pesado Patricia. "Multi-Platform Mobile Application Development Analysis", IEEE RCIS 2015 - 9th International Conference on Research Challenges in Information Science, Proceedings 2015.

**P24** Mounaim LATIF, Younes LAKHRISSI, El Habib NFAOUI, Najia ES-SBAI. "Review of mobile cross platform and research orientations". International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS), 2017.

**P25** Robin Nunkesser. "BeyondWeb/Native/Hybrid: A New Taxonomy for Mobile App Development", ACM/IEEE 5th International Conference on Mobile Software Engineering and Systems, 2018.

**P26** Rahul Raj C.P, Seshu Babu Tolety, "A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach", Annual IEEE India Conference, 2012.

**P27** Lamia Gaouar, Abdelkrim Benamar, Fethi Tarik Bendimerad. "Desirable Requirements of Cross Platform Mobile Development Tools", International Journal of Computer & Information Science, v 16, n 4, p 11-19, Oct.-Dec. 2015.

**P28** Pavel Smutný. "Mobile development tools and cross-platform solutions", 13th International Carpathian Control Conference (ICCC 2012), p 653-6, 2012.

**P29** Matteo Ciman and Ombretta Gaggi. "Measuring Energy Consumption of Cross-Platform Frameworks for Mobile Applications", Lecture Notes in Business Information Processing, v 226, p 331-346, 2015, Web Information Systems and Technologies - 10th International Conference, WEBIST 2014.

**P30** Nabil Litayem, Bhawna Dhupia, Sadia Rubab. "Review of Cross-Platforms for Mobile Learning Application Development", (IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 6, No. 1, 2015.

**P31** Darwin Mena and Marco Santorum. "Maintainability and Portability Evaluation of the React Native Framework Applying the ISO/IEC 25010", Systems and Information Sciences. ICCIS 2020, 27-29 July 2020.

**P32** Esteban Angulo and Xavier Ferre. "A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX", Proceedings of the XV International Conference on Human Computer Interaction, Article No.: 27, Pages 1–8. 2014

**P33** Paulo Meirelles, Carla S. R. Aguiar, Felipe Assis, Rodrigo Siqueira, Alfredo Goldman. "A Students' Perspective of Native and Cross-Platform Approaches for Mobile Application Development", International Conference on Computational Science and Its Applications ICCSA 2019: Computational Science and Its Applications – ICCSA 2019 pp 586-601. 2019

**P34** LACHGAR Mohamed, ABDALI Abdelmounaïm. "Decision Framework for Mobile Development Methods", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 8, No. 2, 2017.

**P35** Andreas Biørn-Hansen, Christoph Rieger, Tor-Morten Grønli, Tim A. Majchrzak, Gheorghita Ghinea. "An empirical investigation of performance overhead in cross-platform mobile development frameworks" , Empirical Software Engineering, v 25, n 4, p 2997-3040, 2020.

**P36** Rui Oliveira, Gabriel Pontes, José Machado, António Abelha. "Analysis of Cross-Platform Development Frameworks for a Smartphone Pediatric Application" , 2013 IEEE International Conference on Industrial Engineering and Engineering Management, 2013.

**P37** Paulo R. M. de Andrade, Adriano B. Albuquerque. "Cross platform app a comparative study", International Journal of Computer Science & Information Technology (IJCSIT) Vol 7, No 1, February 2015.

**P38** Arvind Hudli, Shrinidhi Hudli, Raghu Hudli. "An Evaluation Framework for Selection of Mobile App Development Platform", MobileDeLi 2015 - Proceedings of the 3rd International Workshop on Mobile Development Lifecycle, p 13-16, 2015.

**P39** AlexanderZibula and TimA.Majchrzak. "Developing A Cross-platform Mobile Smart Meter Application Using Html5,jquery Mobile And Phonegap", In Proceedings of the 8th

International Conference on Web Information Systems and Technologies (WEBIST-2012), pages13-23, 2012.

**P40** Iván Tactuk Mercado, Nuthan Munaiah, Andrew Meneely. "The Impact of Cross-Platform Development Approaches for Mobile Applications from the User's Perspective", WAMA 2016 - Proceedings of the International Workshop on App Market Analytics, co-located with FSE 2016, p 43-49, 2016.

**P41** Rieger, Christoph; Majchrzak, Tim A. "Weighted evaluation framework for cross-platform app development approaches" , 9th EuroSymposiumon Information Systems: Development, Research, Applications, Education, SIGSAND/PLAIS, 2016.

**P42** Leonardo Corbalan, Juan Fernandez, Alfonso Cuitiño, Lisandro Delia, Germán Cáseres, Pablo Thomas, Patricia Pesado. "Development Frameworks for Mobile Devices: A Comparative Study about Energy Consumption" , 2018 ACM/IEEE 5th International Conference on Mobile Software Engineering and Systems, 2018.

**P43** Carlos Manso Pinto, Carlos Coutinho . "From Native to Cross-Platform Hybrid Development" , 2018 International Conference on Intelligent Systems (IS), 2018.

**P44** Andreas Biørn-Hansen, Gheorghita Ghinea. "Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development" , Hawaii International Conference on System Sciences, 2018.

**P45** Andreas Biørn-Hansen, Tor-Morten Grønli, Gheorghita Ghinea. "Animations in Cross-Platform Mobile Applications: An Evaluation of Tools, Metrics and Performance" , Sensors (Switzerland), v 19, n 9, May 1, 2019; ISSN: 14248220; DOI: 10.3390/s19092081, 2019.

**P46** Andreas Biørn-Hansen, Tor-Morten Grønli, Gheorghita Ghinea, and Sahel Alouneh. "An Empirical Study of Cross-Platform Mobile Development in Industry". Hindawi Wireless Communications and Mobile Computing Volume 2019, Article ID 5743892, 12 pages https://doi.org/10.1155/2019/5743892. 2019

**P47** Matteo Ciman, Ombretta Gaggi. "An empirical analysis of energy consumption of cross-platform frameworks for mobile development" Pervasive and Mobile Computing, v 39, 214-30, Aug. 2017.

**P48** Mia ČARAPINAa, Igor MEKTEROVIĆ, Tomislav JAGUŠT, Neven DRLJEVIĆ, Jelena BAKSA, Petar KOVAČEVIĆ, Ivica BOTIČKI. "Developing a multiplatform solution for mobile learning". Proceedings of the 23rd International Conference on Computers in Education. 2015

**P49** Taneja Kavita,Taneja Harmunish, Bhullar Rohit K. "Cross-platform application development for smartphones: Approaches and implications". 3rd International Conference on Computing for Sustainable Global Development, INDIACom 2016, p 1752-1758, October 27, 2016.

**P50** Tim A. Majchrzak, Andreas Biørn-Hansen, Tor-Morten Grønli. "Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks", Proceedings of the 50th Hawaii International Conference on System Sciences | 2017.

**P51** Michiel Willocx, Jan Vossaert, Vincent Naessens. "A Quantitative Assessment of Performance in Mobile App Development Tools", 2015 IEEE International Conference on Mobile Services, 2015.

**P52** Hyung Jin Kim, Sudara Karunaratne, Holger Regenbrecht, Ian Warren, Burkhard Claus Wunsche. "Evaluation of Cross-platform Development Tools for Patient Self-Reporting on Mobile Devices", Proceedings of the 8th Australasian Workshop on Health Informatics and Knowledge Management (HIKM 2015), Sydney, Australia, 27 - 30 January 2015.

**P53** Sarah Fahlesson, Dan Johansson. "A Case Study of Cross-Platform Web Application Capability", IIMC International Information Management Corporation, 2013 ISBN: 978-1-905824-36-6.

**P54** Lisandro Delía, Nicolás Galdamez, Leonardo Corbalan, Patricia Pesado, Pablo Thomas. "Approaches to Mobile Application Development: Comparative Performance Analysis". Computing Conference 2017, 18-20 July 2017 | London, UK, 2017.

**P55** Andreas Biørn-Hansen, Tim A. Majchrzak, Tor-Morten Grønli. "Progressive Web Apps: The Possible Web-native Unifier for Mobile Development". In Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST 2017), pages 344-351 ISBN: 978-989-758-246-2, 2017.

**P56** Andreas Biørn-Hansen, Tim A. Majchrzak, Tor-Morten Grønli. "Progressive Web Apps for the Unified Development of Mobile Applications". WEBIST 2017, LNBIP 322, pp. 64–86, 2018.

**P57** Ngu Phuc Huy, Do vanThanh. "Evaluation of mobile app paradigms". ACM International Conference Proceeding Series, p 25-30, 2012.

**P58** Tena Vilček, Tomislav Jakopec. "Comparative analysis of tools for development of native and hybrid mobile applications". 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2017.

**P59** Gatis Vitols, Ingus Smits, Oleg Bogdanov. "Cross-platform Solution for Development of Mobile Applications". In Proceedings of the 15th International Conference Enterprise Information Systems, pages273-277, ICEIS 2013.

**P60** William Jobe. "Native Apps vs. Mobile Web Apps". iJIM – Volume 7, Issue 4, October 2013.

**P61** Yousuf Hasan, Mustafa Zaidi, Najmi Haider, W.U.Hasan, I.Amin. "Smart Phones Application development using HTML5 and related technologies: A tradeoff between cost and quality". IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 3, No 3, May 2012.

# APPENDIX II

# CRITERIA

Here is the list of all criteria that were applied in the surveyed studies. The list also contains the definition for each. In the following catalog, we divided items into two groups. The first is for the criteria used to evaluate mobile frameworks from users' points of view, and the second is from the developers' point of view. Moreover, Table III-1 in appendix III and table A in appendix IV show the criteria distribution from each point of view.

1. *Performance:* The performance criterion is used to evaluate and compare the application resource utilization and execution speed.
2. *User Interface:* The feel during working with the application and the looking compliance of the rendered elements based on the native platform components is considered in this criterion. The smoothness of animations running and the reaction to the user's interaction evaluate using this criterion.
3. *Usability:* The usability elaborates on the learnability and understandability of the developed applications. The application should be easy to use, and users should learn how to use the application easily.
4. *Platform API Accessibility:* Assessing the access to platform APIs is the goal of defining this criterion. It analyzes the quality and the support of accessing platform APIs. The platform APIs in mobile applications contain the following items: access to local storage to save and restore data, the ability to send and receive notifications, access to the gallery, contacts, etc.
5. *Hardware and Sensors Accessibility:* This criterion evaluates the ability of the mobile frameworks to access the device hardware APIs. These APIs include GPS, camera, accelerometer, compass, etc.
6. *Framework maturity:* The maturity of a development framework includes the following items: framework popularity in practice, community size, the shortness of release

cycles, and the speed of fixing bugs. This criterion is used to compare the framework maturity.

7. *Development Tools:* Each framework provides tools to let programmers develop the application faster and easier. The development environment, the debugging features, the emulators for running applications, and code auto-completion during programming are development tool features that make using a framework easier. The development tools criterion considers the mentioned items to analyze the mobile frameworks.
8. *Documentation:* The completeness and updated documentation of a software framework let framework consumer find their way faster with more peace of mind. This criterion looks into the documentation of the mobile frameworks.
9. *Targeted operating systems:* Unlike native frameworks, cross-platform mobile frameworks publish applications for more than one operating system. The criterion assesses the available targeted platform for a mobile framework.
10. *User interface Design :* The user interface design assesses the easiness of designing the user interface elements in mobile frameworks. Some frameworks provide UI builder tools to see the application design before running. Moreover, these tools make user interface maintenance more manageable. Implementing a user interface according to device screen size is another item in this criterion.
11. *Distribution:* After developing applications is crucial to deliver to the audience. The possibility of publishing the application in the application stores (e.g., Apple's app store and Google's play store) in a framework is analyzed using this criterion.
12. *License:* Mobile frameworks have a different license for use by developers. License is a factor that should be considered before selecting a framework for developing an application. The License criterion looks into the license of the mobile frameworks.
13. *Learning Curve:* The criterion compares the hardness of learning a new framework. It contains the costs, time, and skills a developer needs to develop an application using the framework.

14. *Reusability:* Reusing the same codes and components in different projects can reduce the time and cost of the development team significantly. The criterion evaluates both code and developers' knowledge reusability of a framework.
15. *Architectural pattern implementation:* Developers use different architectural patterns in order to develop maintainable mobile applications. The possibility of implementing different architectural patterns (e.g., MVC, MVVM, MVP, etc.) in mobile frameworks is considered by this criterion.
16. *Maintainability:* Maintainability is a requirement for every mobile application. This criterion evaluates the comprehensibility of the source code and modularity of projects in frameworks. Supporting powerful modularity in a mobile framework leads to the maintainability level.
17. *Extensibility:* The criterion refers to access to third-party libraries, UI widgets, open-source custom components, etc. The higher level of extensibility helps develop applications faster in a framework.
18. *Reliability:* Mobile applications must work reliably under different circumstances. The reliability criterion assesses mobile frameworks' fault tolerance mechanism and how that framework acts to errors. The error prevention technique in the framework leads the application to be more reliable and makes the users satisfied with to use.
19. *Testing:* The testing criterion in the mobile framework assesses the facilities that a framework provides to developers to ease them to deliver high-quality applications to users. This criterion analyzes frameworks in terms of the easiness and quality of implementing different tests in mobile projects.
20. *Security:* Each mobile application may use the sensitive and personal information of the users. The framework features should help developers to keep the application's data safe and confidential. Encrypting and decryption of sensitive data, having compliance with the platform security policies, possibility of using source code obfuscation are analyzed under this criterion.

21. *Development Cost:* The cost of the development can be varied when using different platforms. The development cost includes human resources, office, license of the tools, registering for learning courses of a framework, etc. This criterion assesses the mobile frameworks for these costs.
22. *Development Speed:* The speed of development is an essential factor in selecting mobile frameworks. The higher pace in the development phase leads to a shorter time to market, which is vital in converting an idea into a product. The criterion uses to compare frameworks in development speed.
23. *Functionality:* The functionality indicates features that improve the application's suitability for the end-users. These features include the application running when the device is offline, supporting the implementation of multithread, intra-application communication, in-app browser, implementing internationalization, etc.
24. *Monetization:* The criterion analyzes the mobile frameworks in terms of the possibility of providing payment facilities and supporting application store features to ease the monetization process.
25. *Scalability:* The level of modularity provided by frameworks is considered scalability. This criterion assesses the possibility of using modularity in the development phase of mobile frameworks.
26. *Adaptability:* This criterion assesses the adaptability level of the developed application by a framework to different versions of operating systems. Moreover, it evaluates adaptability to hardware environments in mobile frameworks.
27. *Portability:* Portability refers to the ability to port the application to multiple platforms. The criterion assesses the possibility and how we port a project from one operating system to another without requiring major and massive rework.

# APPENDIX III

# CRITERIA DISTRIBUTION IN SURVEYED PAPERS (USERS' POINT OF VIEW)

Table-A III-1 Criteria (users' point of view) distribution in reviewed papers

| **Criterion** | **Numbers** | **Paper(s)** |
|---|---|---|
| Performance | 37 | P3 - P5 - P6 - P7 - P8 - P9 - P10 - P12 - P13 - P14 - P16 - P23 - P27 - P28 - P29 - P30 - P32 - P35 - P37 - P40 - P41 - P42 - P43 - P44 - P45 - P46 - P47 - P48 - P49 - P51 - P52 - P54 - P55 - P56 - P57 - P60 - P61 |
| User interface | 27 | P1 - P3 - P4 - P5 - P6 - P7 - P8 - P9 - P11 - P12 - P14 - P16 - P18 - P25 - P27 - P32 - P33 - P34 - P37 - P38 - P41 - P45 - P46 - P48 - P53 - P57 - P61 |
| Usability | 1 | P40 |

## APPENDIX IV

## CRITERIA DISTRIBUTION IN SURVEYED PAPERS (DEVELOPERS' POINT OF VIEW)

Table-A IV-1 Criteria (developers' point of view) distribution in reviewed papers

| **Criterion** | **Numbers** | **Paper(s)** |
|---|---|---|
| Hardware and Sensors Accessibility | 36 | P1 - P2 - P3 - P4 - P6 - P7 - P11 - P13 - P14 - P15 - P16 - P17 - P19 - P20 - P22 - P23 - P24 - P25 - P26 - P27 - P30 - P31 - P33 - P36 - P37 - P38 - P39 - P41 - P43 - P44 - P50 - P53 - P57 - P59 - P60 - P61 |
| Platform API Accessibility | 36 | P1 - P2 - P3 - P4 - P6 - P7 - P8 - P11 - P14 - P15 - P16 - P17 - P19 - P20 - P22 - P23 - P24 - P25 - P26 - P27 - P30 - P31 - P33 - P36 - P37 - P38 - P39 - P41 - P43 - P44 - P50 - P53 - P57 - P59 - P60 - P61 |
| Development Tools | 26 | P1 - P2 - P3 - P4 - P5 - P6 - P7 - P8 - P13 - P14 - P16 - P18 - P19 - P21 - P27 - P28 - P30 - P31 - P37 - P38 - P41 - P43 - P46 - P50 - P57 - P58 |
| Framework maturity | 26 | P3 - P5 - P6 - P7 - P8 - P13 - P19 - P20 - P21 - P22 - P23 - P24 - P33 - P34 - P36 - P38 - P41 - P44 - P46 - P48 - P50 - P57 - P58 - P59 - P60 - P61 |
| Documentation | 21 | P1 - P3 - P5 - P6 - P7 - P8 - P13 - P16 - P19 - P21 - P27 - P33 - P34 - P36 - P38 - P41 - P43 - P48 - P50 - P58 - P59 |

| | | |
|---|---|---|
| Targeted operating systems | 19 | P2 - P3 - P4 - P6 - P13 - P15 - P17 - P19 - P20 - P21 - P27 - P28 - P41 - P43 - P48 - P50 - P58 - P59 - P61 |
| User Interface Design | 17 | P3 - P4 - P6 - P7 - P8 - P11 - P13 - P19 - P23 - P24 - P28 - P34 - P41 - P46 - P50 - P57 - P60 |
| Distribution | 16 | P3 - P6 - P13 - P14 - P16 - P18 - P23 - P25 - P27 - P31 - P38 - P39 - P41 - P43 - P48 - P57 |
| License | 14 | P2 - P3 - P4 - P6 - P7 - P8 - P19 - P21 - P36 - P38 - P41 - P43 - P48 - P59 |
| Learning Curve | 11 | P3 - P6 - P7 - P8 - P13 - P19 - P21 - P34 - P37 - P41 - P43 |
| Reusability | 10 | P1 - P3 - P6 - P7 - P8 - P23 - P25 - P31 - P41 - P50 |
| Maintainability | 9 | P3 - P6 - P8 - P18 - P25 - P27 - P31 - P41 - P61 |
| Architectural pattern implementation | 9 | P2 - P4 - P6 - P17 - P21 - P24 - P26 - P38 - P41 |
| Extensibility | 8 | P6 - P7 - P8 - P21 - P38 - P41 - P57 - P59 |
| Functionality | 7 | P6 - P13 - P38 - P39 - P41 - P43 - P57 |
| Development Speed | 7 | P3 - P6 - P8 - P9 - P18 - P34 - P41 |
| Development Cost | 7 | P3 - P6 - P8 - P13 - P18 - P33 - P43 |
| Security | 7 | P4 - P6 - P11 - P27 - P38 - P40 - P41 |
| Testing | 7 | P6 - P13 - P16 - P31 - P38 - P41 - P57 |
| Reliability | 7 | P6 - P13 - P31 - P40 - P41 - P49 - P61 |
| Monetization | 5 | P4 - P6 - P41 - P57 - P60 |
| Portability | 3 | P31 - P39 - P61 |
| Adaptability | 3 | P1 - P3 - P15 |
| Scalability | 3 | P3 - P6 - P41 |

# LIST OF REFERENCES

API, G. S. (2022a). GitHub search API. Retrieved from: https://docs.github.com/en/rest/search.

API, S. (2022b). SonarCloud API. Retrieved from: https://sonarcloud.io/web_api/api/project_analyses.

Bankmycell. (2022). Number of smartphone users worldwide [Report]. Retrieved from: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world.

Biørn-Hansen, A., Grønli, T.-M. & Ghinea, G. (2018). A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development. *ACM Comput. Surv.*, 51(5). doi: 10.1145/3241739.

Biørn-Hansen, A., Rieger, C., Grønli, T.-M., Majchrzak, T. A. & Ghinea, G. (2020). An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Empirical Software Engineering*, 25(4), 2997–3040. doi: 10.1007/s10664-020-09827-6.

Ciman, M. & Gaggi, O. (2017). An empirical analysis of energy consumption of cross-platform frameworks for mobile development. *Pervasive Mob. Comput.*, 39, 214-230.

Ciman, M., Gaggi, O. & Gonzo, N. (2014, 03). Cross-Platform Mobile Development: A Study on Apps with Animations. doi: 10.1145/2554850.2555104.

Corbalan, L., Fernandez, J., Cuitiño, A., Delia, L., Cáseres, G., Thomas, P. & Pesado, P. (2018). Development Frameworks for Mobile Devices: A Comparative Study about Energy Consumption. *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, (MOBILESoft '18), 191–201. doi: 10.1145/3197231.3197242.

El-Kassas et al., W. (2015). Taxonomy of Cross-Platform Mobile Applications Development Approaches. *Ain Shams Engineering Journal*, 8. doi: 10.1016/j.asej.2015.08.004.

Fahlesson, S. & Johansson, D. (2013, 07). A case study of cross-platform web application capability.

GitHubAPI. (2022). GitHub personal access token. Retrieved from: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token.

Humayoun, S. R., Ehrhart, S. & Ebert, A. (2013). Developing Mobile Apps Using Cross-Platform Frameworks: A Case Study. pp. 371–380.

Jia, X., Ebone, A. & Tan, Y. (2018, 05). A performance evaluation of cross-platform mobile application development approaches. pp. 92-93. doi: 10.1145/3197231.3197252.

Joorabchi, M. E., Mesbah, A. & Kruchten, P. (2013). Real Challenges in Mobile App Development. *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 15-24. doi: 10.1109/ESEM.2013.9.

Kitchenham, B. (2004). Procedures for Performing Systematic Reviews. *Keele, UK, Keele Univ.*, 33.

Latif, M., Lakhrissi, Y., Nfaoui, E. H. & Es-Sbai, N. (2016, 03). Cross platform approach for mobile application development: A survey. pp. 1-5. doi: 10.1109/IT4OD.2016.7479278.

Majchrzak, T. A., Biørn-Hansen, A. & Grønli, T.-M. (2017, 01). Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks. doi: 10.24251/HICSS.2017.745.

Nagappan, M. & Shihab, E. (2016). Future Trends in Software Engineering Research for Mobile Apps. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 5, 21-32. doi: 10.1109/SANER.2016.88.

Ramachandran, K. M. & Tsokos, C. P. (2021). Chapter 12 - Nonparametric Statistics. In Ramachandran, K. M. & Tsokos, C. P. (Eds.), *Mathematical Statistics with Applications in R (Third Edition)* (ed. Third Edition, pp. 491-530). Academic Press. doi: https://doi.org/10.1016/B978-0-12-817815-7.00012-9.

Sommer, A. & Krusche, S. (2013, 03). Evaluation of cross-platform frameworks for mobile applications.

StatCounter. (2022). Mobile Operating System Market Share Worldwide [Report]. Retrieved from: https://gs.statcounter.com/os-market-share/mobile/worldwide.

Statista. (2022a). Number of mobile app downloads worldwide from 2016 to 2021 [Report]. Retrieved from: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/.

Statista. (2022b). Number of smartphone subscriptions worldwide from 2016 to 2027 [Report]. Retrieved from: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

Statista. (2022c). Mobile operating systems market share [Report]. Retrieved from: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

Mobile development framework. (2022). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/w/index.php?title=Mobile%20development%20framework&oldid=1138209189.

Wikipedia contributors. [[Online; accessed 23-June-2022]]. (2022). SonarSource — Wikipedia, The Free Encyclopedia. Retrieved from: https://en.wikipedia.org/w/index.php?title=SonarSource&oldid=1073207132.

Willocx, M., Vossaert, J. & Naessens, V. (2015). A Quantitative Assessment of Performance in Mobile App Development Tools. *2015 IEEE International Conference on Mobile Services*, pp. 454-461. doi: 10.1109/MobServ.2015.68.

Willocx, M., Vossaert, J. & Naessens, V. (2016, 05). Comparing performance parameters of mobile app development strategies. pp. 38-47. doi: 10.1145/2897073.2897092.

Xanthopoulos, S. & Xinogalos, S. (2013, 09). A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. doi: 10.1145/2490257.2490292.