Noname manuscript No.
(will be inserted by the editor)

To the Attention of Mobile Software Developers: Guess What, Test your App!

Luis Cruz · Rui Abreu · David Lo

the date of receipt and acceptance should be inserted later

Abstract Software testing is an important phase in the software development lifecycle because it helps in identifying bugs in a software system before it is shipped into the hand of its end users. There are numerous studies on how developers test general-purpose software applications. The idiosyncrasies of mobile software applications, however, set mobile apps apart from general-purpose systems (e.g., desktop, stand-alone applications, web services). This paper investigates working habits and challenges of mobile software developers with respect to testing. A key finding of our exhaustive study, using 1000 Android apps, demonstrates that mobile apps are still tested in a very *ad hoc* way, if tested at all. However, we show that, as in other types of software, testing increases the quality of apps (demonstrated in user ratings and number of code issues). Furthermore, we find evidence that tests are essential when it comes to engaging the community to contribute to mobile open source software. We discuss reasons and potential directions to address our findings. Yet another relevant finding of our study is that Continuous Integration and Continuous Deployment (CI/CD) pipelines are rare in the mobile apps world (only 26% of the apps are developed in projects employing CI/CD) – we argue that one of the main reasons is due to the lack of exhaustive and automatic testing.

Keywords Software testing; Mobile applications; Open source software; Software quality; Software metrics.

Luis Cruz
INESC ID, Lisbon, Portugal
E-mail: luiscruz@fe.up.pt

Rui Abreu
INESC ID and IST, University of Lisbon, Lisbon, Portugal
E-mail: rui@computer.org

David Lo
School of Information Systems, Singapore Management University, Singapore
E-mail: davidlo@smu.edu.sg

1 Introduction

Over the last couple of years, mobile devices, such as smartphones and tablets, have become extremely popular. According to a report by Gartner in 2015[1], worldwide sales of smartphones to end users had a record 2014 fourth quarter with an increase of 29.9% from the fourth quarter of 2013, reaching 367.5 million units. Amongst the reasons for the popularity of mobile devices is the ever increasing number of mobile apps available, making companies and their products more accessible to end users. As an example, nine years after the release of the first smartphone running Android[2], there are 3.5 million mobile applications on *Google Play*[3,4].

Mobile app developers can resort to several tools, frameworks and services to develop and ensure the quality of their apps (Linares-Vásquez et al., 2017a). However, it is still a fact that errors creep into deployed software, which may significantly decrease the reputation of developers and companies alike. Software testing is an important phase in the software development lifecycle because it helps in identifying bugs in the software system before it is shipped into the hand of end users. There are numerous studies on how developers test general-purpose software applications. The idiosyncrasies of mobile software apps, however, set mobile apps apart from general-purpose systems (e.g., desktop, stand-alone applications, web services) (Hu and Neamtiu, 2011; Picco et al., 2014).

Therefore, the onset of mobile apps came with a new ecosystem where traditional testing tools do not always apply (Moran et al., 2017; Wang and Alshboul, 2015; Maji et al., 2010): complex user interactions (e.g., swipe, pinch, etc.) need to be supported (Zaeem et al., 2014); apps have to account for devices with limited resources (e.g., limited power source, lower processing capability); developers have to factor in an ever-increasing number of devices as well as OS versions (Khalid et al., 2014); apps typically follow a weekly/bi-weekly time-based release strategy which creates critical time constraints in testing tasks (Nayebi et al., 2016). Moreover, manual testing is not a cost-effective approach to assure software quality and ought to be replaced by automated techniques (Muccini et al., 2012).

This work studies the adoption of automated testing by the Android open source community. We use the term "automated testing" as a synonym of "test automation": the process in which testers write code/test scripts to automate test execution. Automated Input Generation (AIG) techniques were not considered in this study. We analyze the adoption of unit tests, Graphical User Interface (GUI) tests, cloud based testing services, and Continuous Integration / Continuous Deployment (CI/CD). Previous work, in a survey with 83 Android developers, suggests that mobile developers are failing to adopt automated testing techniques (Kochhar et al., 2015). This is concerning since testing is an important factor in software maintainability (Visser et al., 2016). We investigate this evidence by systematically checking the codebase of 1000 Android projects released as Free and Open Source Software (FOSS). Moreover, we delve into a broader set

[1] Gartner's study on smartphone sales: `https://goo.gl/w757Vh` (Visited on February 12, 2019).

[2] The first commercially available smartphone running Android was the HTC Dream, also known as T-Mobile G1, announced on September 23, 2008: `https://goo.gl/QPBdw9`

[3] Google's market for Android apps.

[4] Number of available applications in the Google Play Store from December 2009 to December 2017 available at `https://goo.gl/8P1KD7`.

of testing technologies and analyze the potential impact they can have in different aspects of the mobile apps (e.g., popularity, issues, etc.).

As in related studies (Krutz et al., 2015), we opted to use open source mobile applications due to the availability of the data needed for our analysis. Results cannot be extrapolated to industrial, paid apps, but we provide. In particular, our work answers the following research questions:

RQ1: What is the prevalence of automated testing technologies in the FOSS mobile app development community?

Why and How: It is widely accepted that tests play an important role in assuring the quality of software code. However, the extent to which tests are being adopted amongst the Android FOSS community is still not known. We want to assess whether developers have been able to integrate tests in their projects and which technologies have gained their acceptance. We do that by developing a static analysis tool that collects data from an Android project regarding its usage of test automation technologies. We apply the tool to our dataset of 1000 apps and analyze the pervasion of the different technologies.

Main findings: FOSS mobile apps are still tested in a very *ad hoc* way, if tested at all. Testing technologies were absent in almost 60% of projects in this study. *JUnit* and *Espresso* were the most popular technologies in their category with an adoption of 36% and 15%, respectively. Novel testing and development techniques for mobile apps should provide a simple integration with these two technologies to prevent incompatibility issues and promote test code reuse.

RQ2: Are today's mature FOSS Android apps using more automated testing than yesterday's?

Why and How: We want to understand how the community of Android developers and researchers is changing in terms of adoption of automated testing. In this study, we compare the pervasion of automated tests in FOSS Android apps across different years.

Main findings: Automated testing has become more popular in recent years. The trend shows that developers are becoming more aware of the importance of automated testing. This is particularly evident in unit testing, but GUI testing also shows a promising gain in popularity.

RQ3: How does automated testing relates to popularity metrics in FOSS Android apps?

Why and How: One of the goals of mobile developers is to increase the popularity of their apps. Although many different things can affect the popularity of apps, we study how it can be related to automated tests. We run hypothesis tests over five popularity metrics to assess significant differences between projects with and without tests.

Main findings: Tests are essential when it comes to engaging the community to contribute to mobile open source software. We found that projects using automated testing also reveal a higher number of contributors and commits. The number of *Github Forks*, *Github Stars*, and ratings from *Google Play* users does not reveal any significant impact.

RQ4: How does automated testing affect code issues in FOSS Android apps?

Why and How: The collection of code issues helps developers assess whether their code follows good design architecture principles. It can help developers avoid potential bugs, performance issues, or security vulnerabilities in their software. We use the static analysis tool *Sonar* to collect code issues in our dataset of FOSS Android apps and study whether automated testing brings significant differences.

Main findings: Automated testing is important to assure the quality of software. This is also evident in terms of code issues. Projects without tests have a significantly higher number of minor code issues.

RQ5: What is the relationship between the adoption of CI/CD and automated testing?

Why and How: Previous work showed the adoption of CI/CD with automated testing has beneficial results in software projects Hilton et al. (2016); Zhao et al. (2017). For that reason, the adoption of CI/CD is getting momentum in software projects. We want to study whether CI/CD technologies have been able to successfully address the FOSS Android and whether developers are getting the most out of CI/CD in their projects. We use static analysis to collect data regarding the adoption of CI/CD technologies and compare it to the adoption of automated testing. In addition, we discuss how numbers differ from desktop software.

Main findings: CI/CD adoption in open source mobile app development is not as predominant as in other platforms — only 26% of apps are using it in their development process. We argue that one of the main reasons is the lack of exhaustive and automatic testing — results show evidence that open source projects with CI/CD are more likely to automate tests.

In sum, our work makes the following contributions:

- We created a publicly available dataset with open source apps. The dataset was built by combining data from multiple sources, including metrics of source code quality, popularity, testing tools usage, and CI/CD services adoption. Dataset is available here: `https://github.com/luiscruz/android_test_inspector`.
- We have studied the trends of the adoption of testing techniques in the Android developer community and identified a set of apps that use automated tests in their development cycle.
- We have developed a tool for static detection of usage of state-of-art testing frameworks. Available here: `https://github.com/luiscruz/android_test_inspector`.
- We have investigated the relationship of automated test adoption with quality and popularity metrics for Android apps.

– We have investigated the relationship between automated tests and CI/CD adoption.
– We deliver a list of 54 apps that comply with testing best practices.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. Section 3 outlines the methodology used to collect data in our study. Following, Sections 4–8 describe our methodology and present and discuss the results for each proposed research question. In Section 9, we present a Hall of Fame with apps that comply with the criteria of testing best practices. Threats to the validity are discussed in Section 10. Finally, we draw our conclusions and point directions for future work in Section 11.

2 Related Work

Studies based on data collected from app stores have become a powerful source of information with a direct impact on mobile software teams (Martin et al., 2017). More works have contributed with datasets of open source Android apps (Geiger et al., 2018; Pascarella et al., 2018; Das et al., 2016). Our paper releases a dataset that differentiates by containing information regarding testing practices in Android projects.

Previous work collected 627 apps from F-Droid to study the testing culture of app developers (Kochhar et al., 2015). It was found that at the time of the analysis (2015) only 14% of apps contained test cases and that only 41 apps had runnable test cases from which only 4 had line coverage above 40%. In addition, the authors conducted a survey on 83 Android app developers and 127 Windows app developers to understand the common testing tools and the main challenges faced during testing. The most used framework was *JUnit*, being used by 18 Android developers, followed by *Monkeyrunner* and *Espresso*, with 8 and 7 developers, respectively. According to developers in the survey, the main challenges while testing are time constraints, compatibility issues, lack of exposure, cumbersome tools, emphasis on development, lack of organization support, unclear benefits, poor documentation, lack of experience, and steep learning curve. Our work extends and completes the study by Kochhar et al. via a more extensive data sample (1000 Android apps) and additional analyses. We adopt a comprehensive mining-software-repositories-cum-static-analysis approach to collect mobile software code repositories and empirically assess the benefits of having tests, rather than surveying developers. In addition, we compare the presence of tests in the project with potential issues of the app, satisfaction level of end users, among other popularity metrics. Moreover, we assess the use of different testing tools using static analysis and provide insights into observed trends on automated testing in the past years and compare the testing culture with the adoption of CI/CD.

More works have attempted to capture the current picture of app testing. Silva et al. have studied 25 open source Android apps in terms of test frameworks being used and how developers are addressing mobile-specific challenges (Silva et al., 2016). Results show that apps are not being properly tested, and tests for app executions under limited resource constraints are practically absent. It suggests that a lack of effective tools is one of the reasons for this phenomena. Our work differentiates itself by considering a more representative sample of apps and

complements Silva et al. by providing insights on how developers and researchers can help bring new types of tests into the app development community.

Coppola et al. studied the fragility of GUI testing in Android apps Coppola et al. (2017). The authors collected 18,930 open source apps available on Github and analyzed the prevalence of five scripted GUI testing technologies. However, toy apps or forks of real apps were not factored out from the sample — we understand that real apps were underrepresented (Cosentino et al., 2016; Bird et al., 2009). Thus, we restrict our study to apps that were published in F-droid. In addition, we extend our study to a broader set of testing technologies, while studying relationships between automated testing and other metrics of a project.

Corral and Fronza have compared the success of apps with quality code metrics (Corral and Fronza, 2015). They analyzed a sample of 100 apps and consider a number of code metrics: *Weighted Methods per Class*, *Depth of Inheritance Tree*, *Number of Children*, *Response for a Class*, *Coupling between Objects*, *Lack of Cohesion in Methods*, *Cyclomatic Complexity*, and *Logical Lines of Code*. Results demonstrated that these metrics only have a marginal impact on the success of the apps, showing that real drivers of user satisfaction are beyond source code attributes. Given that mobile apps are very different from traditional applications we find the above metrics too generic. We extend Corral and Fronza's work by focusing on the impact of test automation. Furthermore, besides user satisfaction, we also analyze a number of code issues detected using static analysis and popularity metrics important for the survival of an open source project (e.g., number of contributors).

Previous work has studied the state-of-the-art tools, frameworks, and services for automated testing of mobile apps (Linares-Vásquez et al., 2017a). It revealed that automated test tools should aid developers to overcome the following challenges: 1) restricted time/budget for testing, 2) needs for diverse types of testing (e.g., energy), and 3) pressure from users for continuous delivery. Related work surveyed developers of open source apps to understand their main barriers to mobile testing (Linares-Vásquez et al., 2017b). Developers identified easy maintenance, low overhead in the development cycle, and expressiveness of test cases as important aspects that should be addressed by existing frameworks.

Previous work has compared different techniques and tools for AIG (Choudhary et al., 2015; Amalfitano et al., 2017; Zeng et al., 2016). Choudhary et al. have compared AIG testing tools in terms of ease of use, ability to work on multiple platforms, code coverage, and ability to detect faults (Choudhary et al., 2015). A follow-up study showed that AIG techniques are not ready yet for an industrial setting since activity coverage is dramatically low (Zeng et al., 2016). Our work does not scope AIG techniques — we focus on automated testing strategies that require the creation of test cases. In addition, we differ by studying the prevalence of testing tools and which test frameworks have actually gained the acceptance of mobile developers.

Other works have empirically studied tests on open source software. Kochhar et al. studied the correlation between the presence of test cases and project development characteristics (Kochhar et al., 2013a,b). It was found that tests increase the number of lines of code and the size of development teams. Our work adds value to these contributions by providing insights in the context of mobile app development, and by analyzing a broader set of metrics to study the potential benefits of automated tests in mobile app development.

Hilton et al. analyzed 34,000 open source projects on *GitHub* and surveyed 442 developers (Hilton et al., 2016) on the implications of adopting CI/CD in open source software. Results showed that most popular projects are using CI/CD and its adoption is continuously increasing. A similar approach showed that developers are improving automated tests after the adoption CI/CD (Zhao et al., 2017). Our work only focuses on the relation between automated tests and CI/CD in the context of mobile development, bringing some enlightenment on how the adoption of CI/CD differs in mobile app development.

3 Data collection

Data was gathered from multiple sources, as presented in Figure 1. *F-droid*, a catalog that lists 2,800 free and open source Android apps[5], is used to obtain metadata, package name, and source code repository. *GitHub* is used to collect activity and popularity metrics about the development of the app: number of stars, number of contributors, number of commits, and number of forks. Other popularity metrics are also gathered from *Google Play Store*: rating, and the number of users who rated the app. Test coverage information is obtained from the cloud services *Coveralls* and *Codecov*.

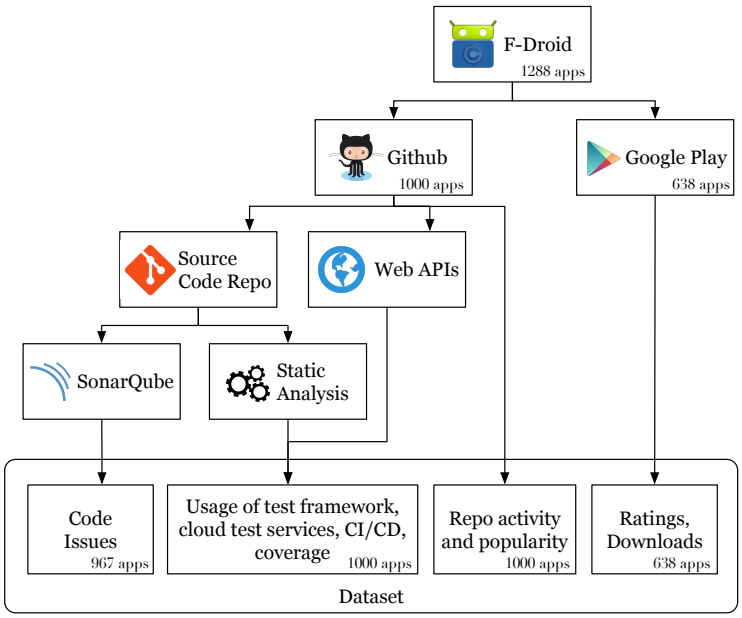

Fig. 1 Flow of data collection in the study.

We extended the data by running the static analysis tool *Sonar*[6] to collect quality-related metrics and potential bugs. We select *Sonar* because it integrates

[5] F-droid's website: `https://goo.gl/NPUusK` (Visited on February 12, 2019).
[6] Sonar's website: `https://goo.gl/svp88G` (Visited on February 12, 2019).

the results of the state-of-the-art analysis tools *FindBugs*, *Checkstyle*, and *PMD*. Furthermore, it has been used with the same purpose in previous work (Krutz et al., 2015).

For each project, we gather the total number of code issues detected by *Sonar*. We also count the number of code issues according to severity, labeled as *blocker* (issue with severe impact on app behavior and that must be fixed immediately; e.g., memory leak), *critical* (issue that might lead to an unexpected behavior in production without impacting the integrity of the whole application; e.g., badly caught exceptions), *major* (issue that might have a substantial impact on productivity; e.g., too complex methods), and *minor* (issue that might have a potential and minor impact on productivity; e.g., naming conventions).

Directly comparing the number of issues in different projects can be misleading: small projects are more likely to have fewer issues than large projects, regardless of projects' code quality. To reduce this effect, we controlled for the size of the project by normalizing the number of issues by the number of files in a project, as follows:

$$I'(p) = \frac{I(p)}{F(p)}, \tag{1}$$

where p is a given project, $I(p)$ the number of issues of p, and $F(p)$ the number of files.

Since one of the main goals in this work is to assess how apps are being tested, we developed a tool to infer which testing frameworks a given project is using[7]. It works by fetching the source code of the app and looking for imported packages and configuration files. The efficacy of this tool was validated with a random sample of apps which was manually labeled.

Table 1 lists all supported tools and frameworks aside with the number of search results in *StackOverflow*, as a proxy of popularity among the developers' community. Unit test tools, user interface (UI) automation frameworks, and cloud based testing services were selected based on a previous survey on tools that support mobile testing (Linares-Vásquez et al., 2017a) and an online curated list of Android tools[8].

We also collect information about the usage of Continuous Integration and Continuous Delivery (CI/CD) services in our study: *Travis CI*, *Circle CI*, *AppVeyor*, *Codeship*, *Codefresh*, and *Wercker*. The selection is based on CI/CD services that have a free plan for open source projects and which adoption can be automatically assessed — i.e., either they save their configuration in the code repository or have an open API that can be accessed with the *GitHub* organization and project name. Self-hosted CI/CD platforms (e.g., *GoCD*, *Jenkins*) are not included in this list. Although this is a subset of CI/CD services that can be used in a project, previous work found that *Travis CI* and *Circle CI* have more than 90% of share in *GitHub* projects using CI/CD services (Hilton et al., 2016).

We analyzed Android apps that are open source and published in *F-droid*. The most popular version control repository is *GitHub*, being used by around 80% of

[7] Source code repository of the tool created to inspect automated testing technologies in Android projects: `https://github.com/luiscruz/android_test_inspector`

[8] List of Android tools curated by Furiya: `https://goo.gl/yLrWgW` (Visited on February 12, 2019).

Table 1 Android tools analyzed

Name	StackOverflow Mentions*
Unit testing	
JUnit	67,153
AndroidJunitRunner	164
RoboElectric	245
RoboSpock	23
GUI testing	
AndroidViewClient	474
Appium	9,687
Calabash	1,856
Espresso	4,374
Monkeyrunner	1,299
PythonUIAutomator	0
Robotium	3,019
UIAutomator	1,918
Cloud testing services	
Project Quantum	0
Qmetry	27
Saucelabs	1,087
Firebase	100,350
Perfecto	224
Bitbar(Kaasila et al., 2012)	16
CI/CD services	
Travis CI	3,662
Circle CI	377
AppVeyor	655
CodeShip	564
CodeFresh	6
Wercker	200
*StackOverflow mentions as of January 26, 2018	

projects. To make data collection clean, only projects using *GitHub* were considered. No other filtering was applied except in particular analyses that required data that was not available for all apps (e.g., *Google Play*'s ratings).

Although *F-droid*'s documentation reports that it hosts a total 2,800 apps[9], only 1288 actually make it to the end user catalog. As we restrict our study to projects using *GitHub*, in total we analyze 1000 Android apps, roughly 35GB of source code collected between September 1–8, 2017. Apps in the dataset are spread amongst 17 categories, as presented in Figure 2, and are aged up to 9 years. The distribution of apps by age is presented in Figure 3.

Since in a few projects the static analysis tool *Sonar* does not successfully run, we collect code issues data for 967 apps, analyzing a total of 329,676 files. Additional data gathered from the *Google Play* store is available for 638 apps.

Reproducibility-oriented Summary

To power reproducibility, based on previous guidelines for app store analyses (Martin et al., 2017), our work is best described as follows:

[9] As reported in the *F-droid*'s wiki page *Repository Maintenance*: `https://goo.gl/VfEQMg` (Visited on January 26, 2018).

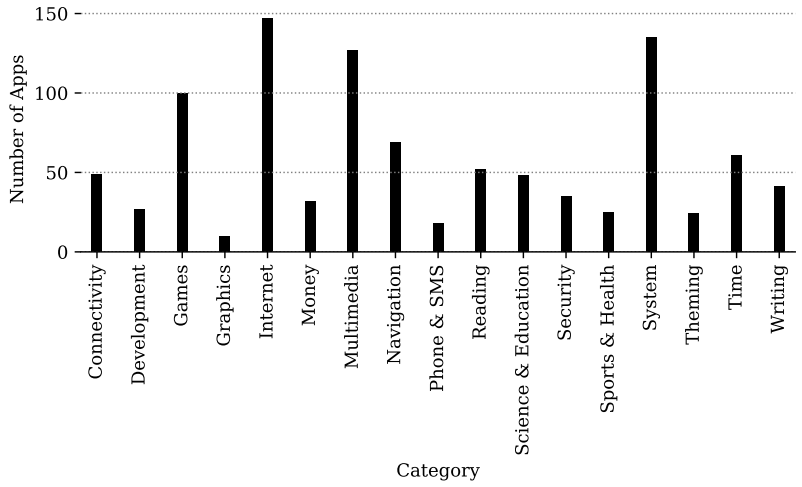

Fig. 2 Categories of apps included in our study with the corresponding app count for each category.

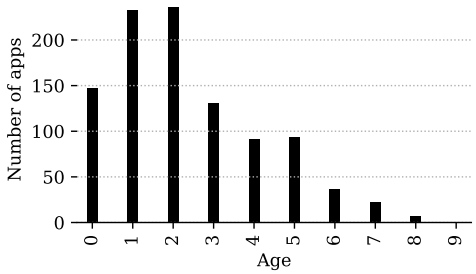

Fig. 3 Distribution of apps by age.

App Stores used to gather collections of apps. We use apps available on *F-Droid* and combine it with data available on *Google Play* store.
Total number of apps used. The study comprises 1000 apps.
Breakdown of free/paid apps used in the study. Only free apps are listed in our dataset.
Categories used. Apps in this study are spread across 17 categories. The distribution of apps is illustrated with the bar chart of Figure 2.
API usage. We collect usage of APIs related to test automation exclusively.
Whether code was needed from apps. Source code was required given the nature of analyses performed in the study.
Fraction of open source apps. Open source apps are used exclusively.
Static analysis techniques. We analyze source code with a self-developed tool for detection of tools, frameworks, and services' usage in the app's project and the static analysis techniques provided by *SonarQube* to gather code issues.

All scripts and tools developed in this work are publicly available with an open source license: `https://luiscruz.github.io/android_test_inspector/`. The same applies to the whole dataset, for the sake of reproducibility.

4 What is the prevalence of automated testing technologies in the FOSS mobile app development community? (RQ1)

Testing is an essential task in software projects, and mobile apps are no different. Given the specific requirements of mobile apps, conventional approaches do not always apply. Thus, we want to assess how the FOSS mobile app development community is addressing automated testing. In particular, we study which testing approaches and technologies are most popular while discussing potential factors.

We compare the frequency of the automated testing technologies employed in the development of the apps in the dataset. The state-of-the-art technologies listed earlier in Table 1 were included, dividing them into three different categories: Unit testing, GUI testing, and Cloud testing services. We resort to data visualizations and descriptive statistics to analyze the frequency of technologies in the dataset.

4.1 Results

Figure 4 shows, out of 1000 apps, the number of projects using a test framework. We include results for *Unit Testing*, *UI Testing*, and *Cloud Testing* frameworks. The first bar shows the number of apps that use any test tool. About 41% of apps have tests. We can see that unit tests are present in 39% of projects while *JUnit* is the most popular tool, with 36% of projects adopting it. This means that 89% of projects with automated tests are using *JUnit*.

Only 15% of projects have automated User Interface (UI) tests. *Espresso* is the most used framework — almost every project with UI tests is using *Espresso*. *UIAutomator*, *Robotium*, and *Appium* are used by a very small portion of projects in our dataset, while *AndroidViewClient*, *Calabash*, *Monkeyrunner*, and *PythonUIAutomator* are not used in any project.

With less than 3% of projects employing them, cloud testing services have not found their way into the open source mobile app development community. In total, 28 projects use *Google Firebase*, whereas only 1 project uses *Saucelabs*. All the other cloud test services in this study are yet to be adopted.

4.2 Discussion

Most mobile apps published in *F-droid* do not have automated tests. Developers are relying on manual testing to ensure proper functioning of their apps, which is known to be less reliable and to increase technical debt (Stolberg, 2009; Bavani, 2012; Karvonen et al., 2017).

Given their simplicity, unit tests are the most common form of tests. *JUnit* is the main unit testing tool and the reason lies in the official Android Developer documentation for tests[10], which introduces *JUnit* as the basis for tests in Android.

[10] *Getting Started with Testing* Android guide available at: `https://goo.gl/RxmHq2` (Visited on February 12, 2019).

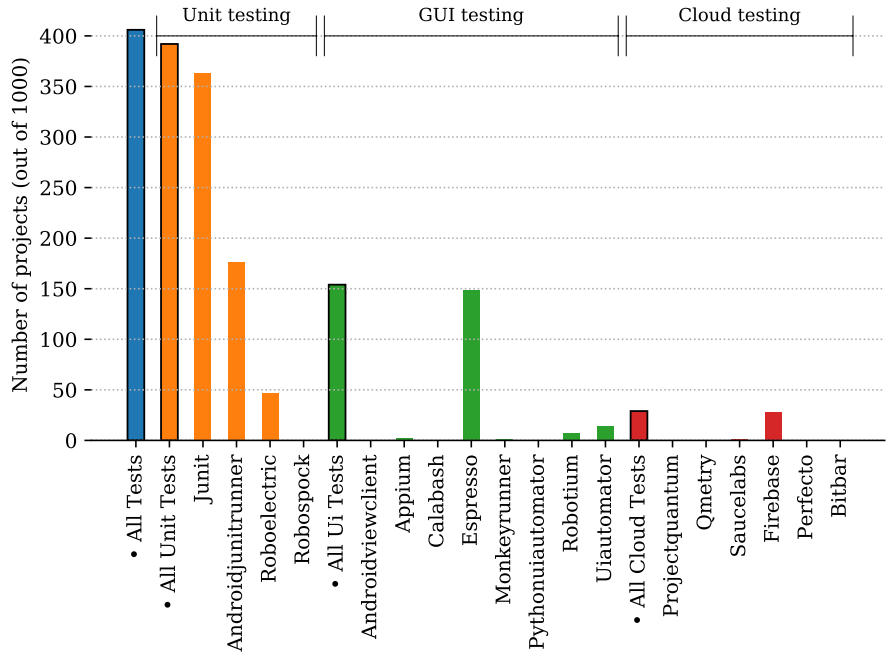

Fig. 4 Number of projects per framework.

Furthermore, other test tools often rely internally on *JUnit* (e.g., *AndroidJunitRunner*).

Other unit testing tools such as *AndroidJunitRunner* and *Roboelectric* do not have a substantial adoption. These tools help running unit tests within an Android environment, instead of the desktop's JVM. This is important given the complexity of an Android app's lifecycle, which might affect test results. However, many apps still do not cross that limit, providing only unit tests for parts of the software that can run absent from the mobile system. Since many apps follow a similar structure, based on Android's framework enforced design patterns, easily customizable boilerplate tests should be delivered along with those patterns.

UI tests are not so popular (15%), which can be explained by their cumbersome maintainability reported in previous work (Gao et al., 2016; Coppola, 2017; Li et al., 2017). Although there are many UI testing frameworks available, *Espresso* is the only one with substantial adoption. This is consistent with the phenomenon of *JUnit* for unit tests: *Espresso* is also promoted in the official Android Developer documentation. In fact, it is distributed with the Android Software Development Kit (SDK). Another strength is that *Espresso* provides mechanisms to prevent flakiness and to simplify the creation and maintenance of tests.

Previous work has considered *Espresso* as the most energy efficient GUI testing framework. The fact that these projects are already using it leaves an open door for the creation of energy tests. On the other hand, *Espresso* still provides a limited set of user interactions, which can be a barrier to high test coverage (Cruz and Abreu, 2018).

Unfortunately, studied cloud testing services have not reached the open source app community. This is probably due to the recency of the introduction of these technologies and the lack of a testing culture in mobile app development, as shown in our results.

The good news is that we observe an increasing adoption of unit and UI tests in the last two years. This trend can be observed by comparing our findings with previous work (Kochhar et al., 2015); while the previous study highlights that the prevalence of automated tests in mobile apps was merely 14%, in this work, we observe that 41% of FOSS apps are developed with automated testing tools.

These findings provide useful implications for the development of new testing tools and techniques. Previous work has shown the importance of creating new types of tests for mobile apps (e.g., energy tests, security tests) (Linares-Vásquez et al., 2017a; Muccini et al., 2012; Wang and Alshboul, 2015). Our results show the importance of simplifying the learning curve and the project's setup. Hence, new types of tests should be compatible at least with *JUnit* and *Espresso*, avoiding reinventing the wheel or complicating usage with new dependencies.

In addition, the adoption of these tools by the FOSS community is highly sensitive to the quality and accessibility of documentation. The fact that *Google* has control over the official documentation does not help third-parties to come aboard. Perhaps the official documentation should feature more tools that are not delivered with the Android SDK. The same concern applies to the academia that is developing many interesting tools for mobile development and testing. Often the lack of documentation is a big barrier to the adoption of innovative techniques by the software industry (Gousios et al., 2016; Kochhar et al., 2015).

Only 41% *of FOSS apps have automated tests. Unit testing frameworks are the most popular, comprising* 39% *of projects. GUI testing is being used by* 15% *of projects, while the adoption of Cloud testing platforms is negligible (*3%*).*

5 Are today's mature FOSS Android apps using more automated testing than yesterday's? (RQ2)

Android testing tools are in constant evolution to fit the ever-changing constraints and requirements of mobile apps. Although we are currently far from having a satisfactory prevalence of automated testing, the evolution from past years can provide actionable information. We study which technologies and types of testing have gained momentum, and which ones are still failing to be perceived as beneficial in FOSS mobile app projects.

Thus, we analyze how the adoption of automated testing relates to the age of an app and the time of an app's last update. We dig further and study the adoption of automated testing in mature FOSS apps by years since the last update. Trends on automated testing adoption over time are analyzed using scatter plots.

5.1 Results

The percentage of apps that are doing tests grouped by their age is presented in the plot of Figure 5. The data is presented from older to newer projects (i.e., 9–0 years old). The size of each circle is proportional to the number of apps with that age (e.g., older projects have smaller circles, meaning that there are fewer projects for those ages.). It is used to show the impact of results in each case. E.g., since projects that are six or more years old have small circles, they comprise a small number of projects. Hence, trends in those age groups are not significant.

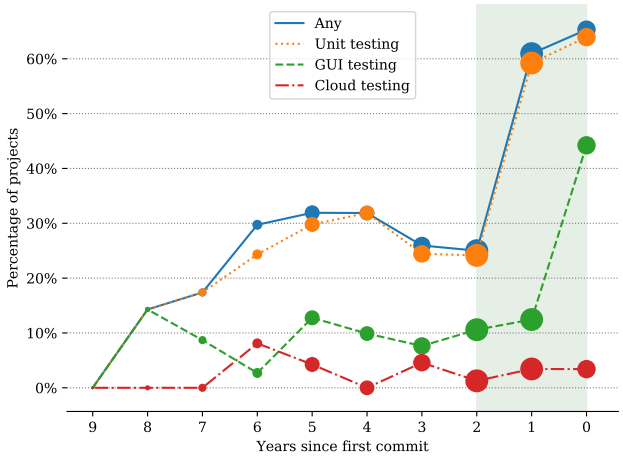

Fig. 5 Percentage of Android apps developed in projects with test cases over the age of the apps.

The timeline in Figure 5 shows that apps that are less than two years old have significantly more tests than older apps. Moreover, the usage of GUI testing frameworks has increased among apps that are under two years old.

In addition, we present in Figure 6 how new apps have been changing the overall test automation adoption. In the past two years (shaded region) the slope of projects with tests is higher than projects without tests. However, this recent change is not able to change the overall pervasion of test automation: most projects are not doing it.

Finally, we present the timeline of the adoption for different kinds of testing techniques in Figure 7. The aforementioned trend is observable for unit testing and GUI testing, which have a higher slope in the past two years (shaded region).

5.2 Discussion

Results show a significant increase in automated testing amongst new FOSS apps. However, the fact that older apps have a lower adoption rate of automated testing can be explained by two phenomena: 1) automated testing is becoming more accessible to developers, who are becoming more aware of its benefits, 2) at some

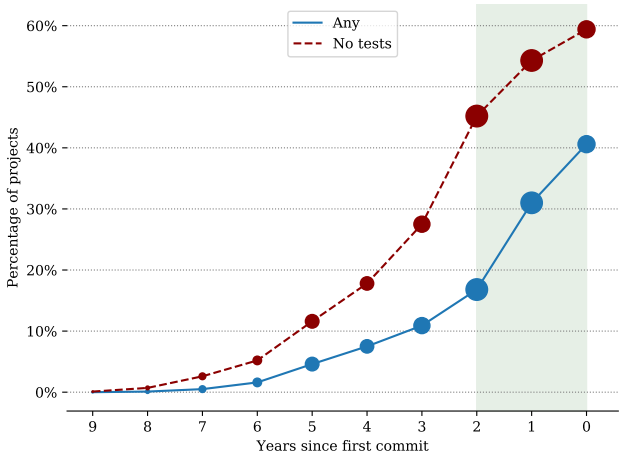

Fig. 6 Cumulated frequency of projects with and without tests (from 9 to 0 years old), normalized by the total number of projects.

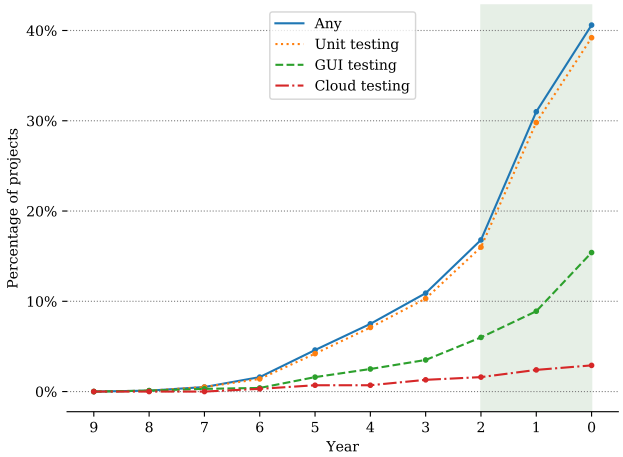

Fig. 7 Cumulated percentage of projects with tests (from 9 to 0 years old), normalized by the total number of projects. All test categories are represented.

point during the lifespan of a project, developers realize that the overhead of maintaining automated testing is not worth the benefits and decide to remove it. While the first phenomenon reveals a positive trend, the latter is quite alarming — automated testing does not provide a long-term solution.

Giving a better sense of which phenomenon is more likely to happen, Figure 6 reveals that automated testing has been gaining popularity in the last two years.

It is worth noting that this increase is happening in both unit testing and GUI testing. The fact that GUI testing is gaining popularity is important — unit testing per se does not provide means to achieve high test coverage in mobile apps. This

increase provides more case studies for researchers to study new types of mobile testing (e.g., energy, security, etc.).

Open source mobile developers are becoming more aware of the importance of using automated tests in their apps. This is observed more for apps that are updated recently than those updated several years ago.

6 How does automated testing relates to popularity metrics in FOSS Android apps? (RQ3)

In this study, we compare popularity metrics with the adoption of automated testing practices in FOSS Android apps. The following popularity metrics were selected:

Number of Stars. The number of Github users that have marked the project as favorite.
Number of Forks. The number of Github users that have created a fork of the repository.
Number of Contributors. The number of developers that have contributed to the project.
Number of Commits. The number of commits in the repository.
Average Rating. The average user rating from *Google Play* store.
Number of Ratings. The number of users rated the app on *Google Play*.

These metrics depend on a myriad of factors, which do not necessarily relate to mobile app development processes. Yet, they are notable metrics that developers do care about. Typically, developers need to drive their development process based on multiple sources of feedback (Nayebi et al., 2018). We want to investigate whether there is any kind of relationship between these features and automated testing. Relationships can help motivate mobile app developers employing tests in their projects.

To remove atypical cases, we perform an outlier detection using the Z-score method with a threshold of three standard deviations. In addition, we perform the normality test *Shapiro-Wilk*, which tests the null hypothesis that data follows a normal distribution.

Then we apply hypothesis testing, using the non-parametric test Mann-Whitney U, with a significance level (α) of 0.05. We may also consider a parametric test (e.g., the standard t-test), in case we find variables that follow a Normal distribution. In addition, since we are conducting multiple comparisons, the Benjamini-Hochberg procedure is used to correct p-values and control false discovery rate.

The independent variable is whether an app has tests in its project source code while the dependent variables are the popularity metrics.

The hypothesis test is formulated as follows, with populations WO and W as the population of **apps without tests** and the population of **apps with tests**, respectively:

$$H_0 : P(W > WO) = P(WO > W)$$
$$H_1 : P(W > WO) \neq P(WO > W)$$

In other words, we test the null hypothesis (H_0) that a randomly selected value from population W is equally likely to be less than or greater than a randomly selected value from sample WO.

We perform hypothesis testing for each of the aforementioned metrics, formulated as follows:

Number of Stars
H_0 : a project with tests (W) has the same number of Github stars as a project without tests (WO).
H_1 : the number of Github Stars in projects with tests is different from the number of stars in a project without tests.
Number of Forks
H_0 : a project with tests (W) has the same number of forks as a project without tests (WO).
H_1 : the number of forks in projects with tests is different from the number of stars in a project without tests.
Number of Contributors
H_0 : projects with tests (W) have the same number of contributors as a project without tests (WO).
H_1 : the number of forks in projects with tests is different from the number of contributors in a project without tests.
Number of Commits
H_0 : a project with tests (W) has the same number of commits as a project without tests (WO).
H_1 : the number of commits in projects with tests is different from the number of commits in a project without tests.
Average Rating
H_0 : a project with tests (W) has the same rating as a project without tests (WO).
H_1 : the rating of a randomly selected project with tests is different from the rating in a project without tests.
Number of Ratings
H_0 : a project with tests (W) has the same number of rating as a project without tests (WO).
H_1 : the number of ratings of a randomly selected project with tests is different from the number of ratings in a project without tests.

In addition, we perform effect size analyses for variables showing statistical significance. We compute the mean difference ($\Delta\bar{x} = \bar{x}_W - \bar{x}_{WO}$), the difference of the medians ($\Delta Md = Md_W - Md_{WO}$), and the Common Language Effect Size (CL) (McGraw and Wong, 1992).

The mean difference ($\Delta\bar{x}$) measures the difference between the means of apps with tests (W) and apps without tests (WO) for a particular popularity metric. We compute it for being a conventional effect-size metric. In addition, since the distribution is not necessarily normal, we compute the difference of the medians (ΔMd) between apps with tests (W) and apps without tests (WO). Given that

the median of a sample is the value that separates the higher half from the lower half of the sample, ΔMd measures how different this median value is in the two distributions.

There are nonetheless a few cases in which ΔMd does not capture differences in the two distributions (Kerby, 2014). We complement the effect size analysis with the Common Language (CL) measure. CL is the recommended measure when there is no assumption on the shape of the distributions of the two samples being tested and it is commonly used in tandem with Mann-Whitney U test (Leech and Onwuegbuzie, 2002). One advantage of using CL to measure effect size is that it can be easily interpreted (Brooks et al., 2014): it is the probability that the value from an individual randomly extracted from one sample will be higher than the value from an individual randomly extracted from another.

6.1 Results

The distributions of the popularity metrics are depicted in the boxplots of Figure 8. The medians are represented by the orange solid lines, while the means are by green dashed lines. The results of the normality tests *Shapiro-Wilk* yielded a low p-value ($p < 0.001$) for all metrics. Thus, none of the metrics follows a normal distribution, which highlights the suitability of using the Mann-Whitney U test over the standard t-test.

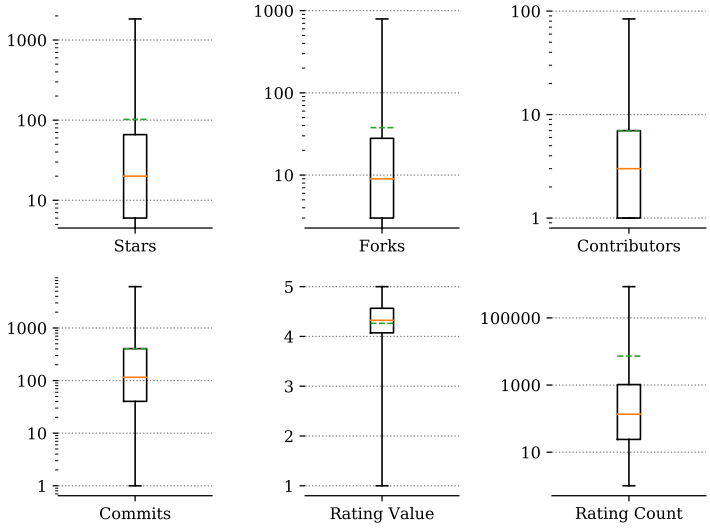

Fig. 8 Boxplots with the distributions of the popularity metrics. Note that the y-axis is in log-scale for all metrics but ratings.

Hypothesis testing results are shown in Table 2 along with the effect size analysis: mean difference ($\Delta\bar{x}$), difference of median (ΔMd), and CL expressed in percentage. The bigger the effect size is, the bigger is the metric for apps with

tests. The effect size analysis is only relevant in cases with statistical significance, which are highlighted in bold text.

Table 2 Statistical analysis of the impact of tests on the popularity of apps.

	p-value	$\Delta\bar{x}$	ΔMd	CL (%)
Stars	0.2130	54.78	3.00	52.74%
Forks	0.4525	11.39	1.00	51.40%
Contributors	**0.0052**	**2.17**	**0.00**	**55.80%**
Commits	**0.0008**	**247.58**	**49.00**	**57.13%**
Rating Value	0.0835	0.05	0.05	54.77%
Rating Count	0.2465	−894.26	−56.00	47.03%

There is statistical evidence that FOSS Android apps with tests are expected to have more **commits** and more **contributors**. Note, however, that this evidence does not imply that tests boost these variables. Conclusions must analyze the causality of this relationship (i.e., whether tests are cause or consequence) and the fact that there are many external variables that are expected to have a significant impact (e.g., target users, originality of idea, design, marketing, etc.). Nonetheless, no statistical significance was found between having automated tests and the number of *GitHub* stars, forks and ratings on *Google Play*.

Projects with tests have on average more 248 commits in the whole project. The CL effect size is small but substantial: the probability of a project with tests having more commits than a project without tests is 57%. Although the number of commits increases, one can argue that the number of commits can be related to overhead created by tests maintenance.

Projects with tests have a small but substantial CL effect size: the probability that a project with tests will have more contributors is 56%. Nevertheless, the direction of this relationship cannot be assessed with these results — i.e., there is no evidence of whether the presence of tests is a consequence of the high number of contributors in the project or, in contrary, it is a way of attracting more developers to contribute.

Tests and Contributors: developers' perception? We decided to conduct a follow-up study to assess the developers' perception of whether tests can lead to more contributors. We contacted 343 mobile open source developers to answer a survey with two close-ended questions:

1. *Do you think that more tests benefit/attract new-comers?*
   Possible answers were: *Yes*; *No*; and *Maybe*.
2. *Is the presence of tests a reason or a consequence of a big community of contributors?*
   Possible answers were: *Most likely a reason*; *Most likely a consequence*; *Both equally*; and *No impact*.

Respondents had an additional box where they could optionally leave their comments or feedback on the subject. Developers were selected by being active in an open source mobile application available on *GitHub*. In the end, we had 44 responses. Data collected was anonymized and it is available online[11].

[11] Questionnaire responses are available online: `https://goo.gl/6CFDb9`

As shown in the pie chart of Figure 9, 45.5% of our respondents believe that tests help new developers to contribute in a project, while 38.6% are not sure, and only 15.9% disagree with the statement. The pie chart of Figure 10 shows that, despite the recognized improvement from having tests, the majority of respondents believe that the presence of tests is more likely a consequence from having a big community of contributors (43.2%). A smaller part of respondents (25%) believe that the presence tests and the size of the community do not affect each other — i.e., they both depend on a different variable. Other respondents believe both variables affect each other (22.7%), while only 9.1% reckon tests as the cause.

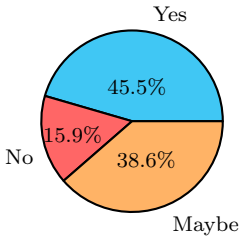

Fig. 9 Do tests attract newcomers?

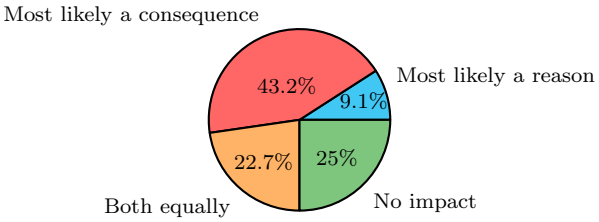

Fig. 10 Tests: cause or consequence of a big community?

Feedback submitted by some developers provided some insights on their personal experience. Some developers pointed out that the adoption of CI/CD is probably "more influential than the actual tests". Other developers emphasized the importance of having tests as "a good starting point for newcomers to get familiar with the project's code and its features". Finally, some developers state that the "maintenance burden of automated tests is really high" and that they can block major refactorings in software projects.

6.2 Discussion

Results show that FOSS Android projects with tests have more commits and more contributors. The increase in the number of commits can be explained by an overhead of commits induced by the maintenance and configuration of tests.

Responses to the questionnaire show that the presence of tests is more likely a consequence of having a big community. In addition, tests can help new developers contribute to the project. Since one of the main concerns in open source projects is to foster the community to contribute[12], the importance of tests for this purpose cannot be discarded. Conventionally, maintainers of open source projects target this goal by inviting contributors, providing social and communication tools, and making sure that instructions on how to contribute are well documented. These results show that tests should also be part of their agenda.

This relationship is consistent with previous work. Automated tests help new developers be more confident about the quality of their contributions (Gousios et al., 2016). Contributors are able to create *Pull Requests* (PR) to a project with a reasonable level of confidence that other parts of the software will not break. The same applies to the process of validating a PR. Integrators usually have some barriers when accepting contributions from newcomer developers (Gousios et al., 2015). The presence of automated tests helps reduce that barrier, and contributions with tests are more likely to be accepted (Gousios et al., 2015). Another aspect of automated tests that contributes to this trend is the reported ability to provide up-to-date documentation of the software (Van Deursen, 2001; Beck, 2000).

Previous work that shows that app store's ratings are not able to capture the quality of apps (de Langhe et al., 2016; Ruiz et al., 2017). Our results show that this is also the case for tests: there is no relationship between using tests and rating on *Google Play*.

Our findings have direct implications for different stakeholders of mobile software projects. Developers have to start using automated tests in their code in order to assure quality in their contributions. Open source project maintainers must promote a testing culture to engage the community in their projects.

Automated testing is important to foster the community to contribute. There is statistical evidence that FOSS Android projects with tests have more contributors and more commits. Number of GitHub Stars*,* Github Forks *and ratings on* Google Play *did not reveal any significant impact.*

7 How does automated testing affect code issues in FOSS Android apps? (RQ4)

Code issues are related to potential vulnerabilities of software. It is a major concern of developers to ship software with a minimal number of code issues. We study whether automated testing can help developers deploy mobile app software with fewer code issues.

We use the issues detected by the static analysis Sonar tool as a proxy of software code issues. We apply Sonar to our dataset of 1000 Android apps. As mentioned in Section 3, Sonar issues are divided into four categories, based on

[12] *Five best practices in open source: external engagement* by Ben Balter: `https://goo.gl/BQRZBa` (Visited on February 12, 2019).

the severity of their impact. We evaluate the number of issues normalized for the number of files in the project ($I'(p)$).

We apply the same approach used in Section 6: we use hypothesis testing with the Mann-Whitney U test using a significance level (α) of 0.05. Benjamini-Hochberg procedure is used to correct p-values since four tests are performed in the same sample. Mean difference ($\Delta\bar{x}$), difference of median (ΔMd), relative difference ($\frac{\Delta Md}{Md_W}$), and CL are used to analyze effect size.

7.1 Results

We successfully collected code issue reports from 967 apps. It was not possible to collect data from 33 apps: Sonar failed due to characters invalid with UTF-8 encoding. This was the case of the reading app *FBReaderJ* and its file `ZLConfigReader.java` that contained characters that not even Github is able to render[13]. Since these apps consisted of a small portion of our dataset (3%), we decided to leave them out of this part of the study.

Table 3 presents descriptive statistics of the number of code issues per file $I'(p)$ for each level of severity — size of the sample (N), median (Md), mean ($\bar{x}$), and standard deviation (s). The table also presents the results of normality tests with the p-values for Shapiro-Wilk tests ($X \sim N$), showing that none of the metrics follows a normal distribution. Statistics are presented for both apps with tests (W) and apps without tests (WO).

Table 3 Descriptive statistics of code issues on apps with (W) and without (WO) tests

	Tests	N	Md	$\bar{x}$	s	$X \sim N$
Blocker	W	398	0.00	0.02	0.04	$p < 0.0001$
	WO	569	0.00	0.05	0.59	$p < 0.0001$
Critical	W	398	0.24	0.34	0.39	$p < 0.0001$
	WO	569	0.26	0.48	0.90	$p < 0.0001$
Major	W	398	0.50	0.73	0.80	$p < 0.0001$
	WO	569	0.52	0.84	1.09	$p < 0.0001$
Minor	W	398	0.61	0.87	0.93	$p < 0.0001$
	WO	569	0.73	1.27	2.12	$p < 0.0001$

Figure 11 illustrates the distribution of the $I'(p)$ in projects with tests (blue line, hatch fill) and without tests (red line, empty fill) for different types of issues. The mean for each group is depicted with a dashed green line, while the median with a solid orange line. Types of issues with a statistically significant difference between W and WO are highlighted with thicker lines. Results show that projects with tests have significantly less minor code issues than projects without tests.

Table 4 reports the resulting p-values and computes the effect-size metrics: mean difference ($\Delta\bar{x}$), difference of median (ΔMd), relative difference ($\frac{\Delta Md}{Md_W}$), and CL.

The number of minor issues per file increases significantly in projects without tests. The difference of median shows that projects without tests are expected to

[13] Example of a source code file incompatible with Sonar tool: `https://git.io/fxNg9` (Visited on February 12, 2019).

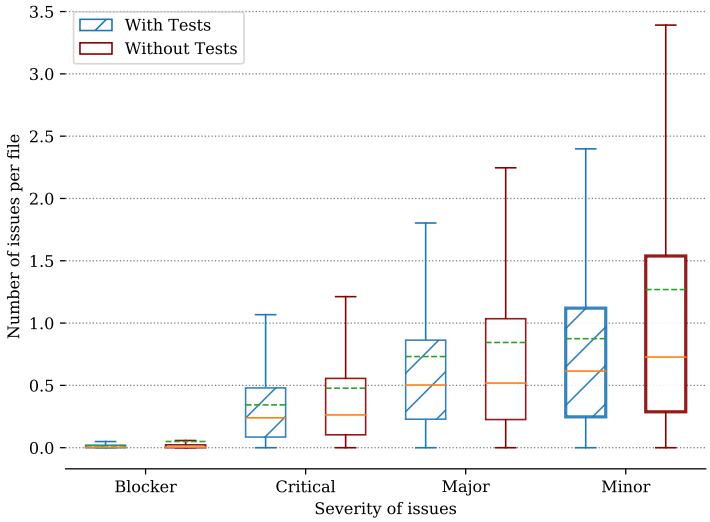

Fig. 11 Comparison of the number of issues per file in projects with and without tests. Green dashed lines in each box represent the mean value, while orange solid lines represent the median.

have 0.11 more minor issues per file (increase of 18%). Furthermore, as reported with the CL effect-size, projects without tests have more minor issues than projects with tests with a probability of 54%. The number of issues for higher severity levels is not significantly affected.

Table 4 Statistical analysis of the impact of tests in mobile software code issues

Severity	p-value	$\Delta\bar{x}$	ΔMd	$\frac{\Delta Md}{Md_W}$ (%)	CL (%)
Blocker	0.1643	0.0337	0.0014	48	52.09%
Critical	0.1150	0.1337	0.0234	9	52.97%
Major	0.2939	0.1130	0.0157	3	51.02%
Minor	**0.0440**	**0.3940**	**0.1127**	**18**	**54.32%**

7.2 Discussion

Results show that there is a statistically significant and substantial relationship between using automated testing and the number of minor code issues that appear in the project. FOSS Android projects without automated testing have significantly more minor code issues. Given that only 41% of apps in this study have automated tests, mobile developers need to be aware of the importance of testing their apps.

On the other hand, although the normalized number of blocker, critical, and major bugs is higher for apps without tests than those with tests, the difference is not statistically significant. Other alternatives, such as manual testing, code

inspection, or static analysis, are probably preventing such issues. Our sample size may also not be large enough to make the result to be statistically significant.

There is statistical evidence that FOSS Android projects without tests have 18% *more minor code issues per file. In our sample, projects without tests also had more code issues for other severity levels: major (3%), critical (9%), and blocker (48%).*

8 What is the relationship between the adoption of CI/CD and automated testing? (RQ5)

CI/CD has been proved to be beneficial in software projects and to have even better results when employed along with automated testing Hilton et al. (2016); Zhao et al. (2017). Thus, we study whether mobile app developers are using CI/CD in its full potential. Moreover, we delve into how mobile app projects set themselves apart from conventional software projects in terms of CI/CD adoption.

To answer this research question, we start by comparing the adoption of the different studied CI/CD technologies in Android FOSS projects. In addition, we compare the frequency of projects that have adopted one of the studied CI/CD tools with the frequency of projects using automated testing.

For this analysis, we resort to data visualizations. To validate the relationship between automated testing and CI/CD we use Pearson's chi-squared test with a significance level of 0.05. This test was selected for being commonly used to compare binary variables.

8.1 Results

We first analyze which apps are using CI/CD pipelines in their development practices. The distribution of CI/CD pipelines among these platforms is given in Figure 12. *Travis CI* is the most popular platform with 249 apps using it (25%), followed by *Circle CI*, being used by 2% of apps. However, in total, only 27% have adopted CI/CD.

The relationship between the prevalence of CI/CD and prevalence of tests is depicted by the mosaic plot in Figure 13. The size of each area is proportional to the number of apps in each group. Nearly 50% of apps are not having tests nor adopting CI/CD (region A). 26% of apps, despite having tests, are not using CI/CD (region B). 12% of apps are using CI/CD but are not doing any automated tests (region C). Only 15% of apps are using CI/CD effectively, with automated tests (region D). In addition, the mosaic plot suggests that automated testing is more prevalent in projects with CI/CD than projects without. This is confirmed by the Pearson's chi-squared test: $\chi^2 = 31.48, p = 2.009\text{e-}8$.

Online coverage trackers are useful tools that play well with CI/CD platforms. They help ensure that the code is fully covered. Nevertheless, only 19 projects are using it — 9 use *Coveralls* and 12 use *Codecov*, having 2 projects using both platforms. However, only 4 have line coverage above 80%, and no meaningful results can be extrapolated.

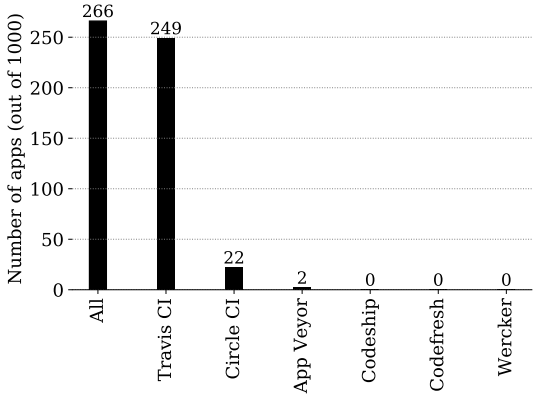

Fig. 12 Android apps using CI/CD platforms.

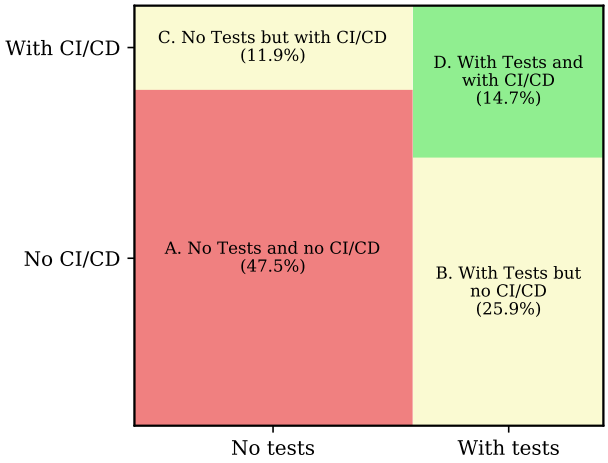

Fig. 13 Relationship between apps using CI/CD and apps using tests.

8.2 Discussion

CI/CD is not as widely adopted by mobile app developers as compared to developers of general OSS projects — only 26% of apps have adopted CI/CD services while the adoption in general open source software hosted by *GitHub* is 40% (Hilton et al., 2016).

There are 12% of apps that, despite using CI/CD, do not have automated tests. In practice, these projects are only using CI/CD tools to run static analyses. Yet, they rely on a pipeline that requires an approver to manually build and test the app.

The fact that there are projects that have tests but did not adopt CI/CD (26%) is also concerning. One of the main strengths of adopting CI/CD is improving software quality through test automation (Zhao et al., 2017). Although

CI/CD services have made a good work in simplifying the configuration of Android specific requirements (e.g., SDK version, emulator, dependencies, etc.), developers have reported that the main obstacle in adopting CI/CD in a project is having developers who are not familiar with it (Hilton et al., 2016). Nevertheless, since these projects are already using automated tests, they could potentially benefit from a CI/CD pipeline with little effort. More research needs to be conducted to assess why mobile developers are not adopting CI/CD in their projects.

Travis CI and *Circle CI* are the most used CI/CD services, as expected from previous results for other types of software (Hilton et al., 2016). Although the other platforms have a well documented support for Android, they are not being used by the community.

Even more surprising is the fact that, from the 147 apps with both CI/CD and tests, only 19 are actually promoting full test coverage with coverage tracking services. This suggests that coverage is not a top priority metric for mobile developers, which is in sync with concerns by Gao *et al.* who have reported the need for coverage criteria to meet the idiosyncrasies of mobile app testing (Gao et al., 2014). In particular, *Coverall* and *Codecov* platforms only report line coverage. Different coverage criteria, such as event/frame coverage, would be more suitable in the context of mobile apps.

More education and training is needed to get full benefits of CI/CD for mobile apps. Developers that are already performing automated tests in their apps should explore the integration of a CI/CD pipeline in their projects. This is also a good opportunity for newcomer developers willing to start contributing to open source projects.

CI/CD in mobile app development is not as prevalent as in other platforms; Automated testing is more prevalent in projects with CI/CD.

9 Hall of Fame

We have selected a set of apps from our dataset that we consider good candidates for studying best practices from the mobile app development community. We perform a systematic selection by choosing projects that perform unit tests, UI tests and are using CI/CD. In total, 54 apps satisfy these requirements[14]. We present in Table 5 one app for each category based on the popularity of that app among developers, using the number *GitHub Stars* as a proxy. Some categories, namely *Games*, *Money*, and *Phone & SMS*, did not have any app that meets the requirements.

Note, however, that although these projects follow best practices, they are not necessarily the ones with the highest ratings (e.g., rating in *Google Play*, number of Forks in *Github*). The success of apps also depends on a myriad of other factors. Nevertheless, the impact of best practices is not negligible and for that reason, these projects can be used as role models for new projects or subjects for case studies for further research.

[14] The whole set of apps in the Hall of Fame can be accessed online: `https://luiscruz.github.io/android_test_inspector/`.

Table 5 Hall of fame

Category	Organization	Project Name
Internet	k9mail	k-9
Multimedia	TeamNewPipe	NewPipe
Writing	federicoiosue	Omni-Notes
Theming	Neamar	KISS
Time	fossasia	open-event-android
Sports & Health	Glucosio	android
Navigation	grote	Transportr
System	d4rken	reddit-android-appstore
Reading	raulhaag	MiMangaNu
Security	0xbb	otp-authenticator
Science & Education	EvanRespaut	Equate
Connectivity	genonbeta	TrebleShot
Development	Adonai	Man-Man
Graphics	jiikuy	velocitycalculator

10 Threats to validity

Construct validity Code issues collected with *SonarQube* are used to measure the quality of code. Some projects might not follow common development guidelines due to specific requirements. Thus, generic static rules might not be able to capture the quality of such projects. Nevertheless, we expect that this is the case of a minimal number of apps and results are not affected. Metrics from *Google Play* and *GitHub* are used as proxies to measure user satisfaction, and popularity of apps. These metrics are affected by a number of factors and not always are sufficiently dynamic to cope with changes in the app (Ruiz et al., 2017).

Furthermore, the online coverage trackers investigated in this study only support line coverage. Coverage metrics for events or UI frames are more suitable for mobile applications. These metrics were not evaluated as they are not available in the state-of-the-art online coverage trackers. Finally, we did not consider AIG techniques since they are more advanced and thus are not popularly used in mobile app development yet.

Internal validity The usage of a test framework or service was assessed through a self-developed automatic tool based on static analysis and Web requests to service's APIs. To validate the accuracy of our tool we have manually labeled a random sample of 50 apps and compared the results. Our tool has successfully passed our validation with no false positives and no false negatives, but we understand that some corner cases may not have been checked yet. The same applies to the static analysis tool *SonarQube* used to collect code issues — it provides an approximation of the actual set of code issues in a project. Some issues detected by *SonarQube* may be false positives or may not generalize to other, distinct projects. Nevertheless, we argue such cases are rare and they are not expected to have a significant effect in results.

External validity Our work has focused on free and open source apps. Our 1000-app dataset comprises a good proportion of these apps that are currently available for Android users. Findings in this work are likely to generalize to types of apps with a caveat: private companies usually have a different approach from open source

organizations on software testing (Joorabchi et al., 2013). We did not include testing services without a free plan for open source projects; Paid apps have different budgets and might be more willing to use paid services in their projects. Legal and copyright restrictions do not allow us to scope apps with commercial licenses. This is a known barrier for research based on app store analysis (Krutz et al., 2015; Nagappan and Shihab, 2016; Martin et al., 2017).

The adoption of CI/CD is based on a subset of CI/CD services available, as described in Section 3. This subset is equivalent to the one used by Hilton et al. to study CI/CD adoption in general software projects (Hilton et al., 2016).

11 Conclusion

Testing is a crucial activity during the software development lifecycle to ascertain the delivery of high quality (mobile) software. This study is about testing practices in the mobile development world. In particular, we investigated working habits and challenges of mobile app developers with respect to testing.

A key finding of our large-scale study, using 1000 Android apps, is that mobile apps are still tested in a very *ad hoc* way, if tested at all. We show that, as in other types of software, testing increases the quality of apps (demonstrated in the number of code issues). The adoption of tests has increased over the last two years and that Espresso and *JUnit* are the most popular frameworks. Furthermore, we find that availability of tests plays a positive role in engaging the community to contribute to open source mobile app projects. Yet another relevant finding of our study is that CI/CD pipelines are rare in the mobile app world (only 26% of the apps are developed in projects leveraging CI/CD) – we argue that one of the main reasons is due to the lack of a good set of test cases and adoption of automatic testing. We have discussed possible reasons behind the observed phenomena and some implications for practitioners and researchers.

As future work, our empirical study can be expanded in several ways: 1) study how mobile app projects address tests for particular types of requirements (e.g., security, privacy, energy efficiency, etc.); 2) based on the test practices collected from mobile app repositories, provide a set of best practices to serve as rule of thumb for other developers; and 3) verify that these findings also hold for other platforms.

Acknowledgment

This work is financed by the ERDF — European Regional Development Fund through the Operational Program for Competitiveness and Internationalization - COMPETE 2020 Program and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia with reference UID/CEC/50021/2019, and within projects GreenLab (POCI-01-0145-FEDER-016718) and FaultLockerRef (PTDC/CCI-COM/29300/2017). Luis Cruz is sponsored by an FCT scholarship grant number PD/BD/52237/2013.

References

M. Linares-Vásquez, K. Moran, and D. Poshyvanyk, "Continuous, evolutionary and large-scale: A new perspective for automated mobile app testing," in *33rd IEEE International Conference on Software Maintenance and Evolution (ICSME'17), page to appear*, 2017.

C. Hu and I. Neamtiu, "Automating GUI testing for android applications," in *Proceedings of the 6th International Workshop on Automation of Software Test*. ACM, 2011, pp. 77–83.

G. P. Picco, C. Julien, A. L. Murphy, M. Musolesi, and G.-C. Roman, "Software engineering for mobility: reflecting on the past, peering into the future," in *Proceedings of the on Future of Software Engineering*. ACM, 2014, pp. 13–28.

K. Moran, M. Linares-Vásquez, and D. Poshyvanyk, "Automated GUI testing of Android apps: from research to practice," in *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 2017, pp. 505–506.

Y. Wang and Y. Alshboul, "Mobile security testing approaches and challenges," in *Mobile and Secure Services (MOBISECSERV), 2015 First Conference on*. IEEE, 2015, pp. 1–5.

A. K. Maji, K. Hao, S. Sultana, and S. Bagchi, "Characterizing failures in mobile oses: A case study with android and symbian," in *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*. IEEE, 2010, pp. 249–258.

R. N. Zaeem, M. R. Prasad, and S. Khurshid, "Automated generation of oracles for testing user-interaction features of mobile apps," in *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*. IEEE, 2014, pp. 183–192.

H. Khalid, M. Nagappan, E. Shihab, and A. E. Hassan, "Prioritizing the devices to test your app on: A case study of android game apps," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 610–620.

M. Nayebi, B. Adams, and G. Ruhe, "Release practices for mobile apps–what do users and developers think?" in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, vol. 1. IEEE, 2016, pp. 552–562.

H. Muccini, A. Di Francesco, and P. Esposito, "Software testing of mobile applications: Challenges and future research directions," in *Proceedings of the 7th International Workshop on Automation of Software Test*. IEEE Press, 2012, pp. 29–35.

P. S. Kochhar, F. Thung, N. Nagappan, T. Zimmermann, and D. Lo, "Understanding the test automation culture of app developers," in *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 1–10.

J. Visser, S. Rigal, R. van der Leek, P. van Eck, and G. Wijnholds, *Building Maintainable Software, Java Edition: Ten Guidelines for Future-Proof Code*. " O'Reilly Media, Inc.", 2016.

D. E. Krutz, M. Mirakhorli, S. A. Malachowsky, A. Ruiz, J. Peterson, A. Filipski, and J. Smith, "A dataset of open-source android applications," in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 2015, pp. 522–525.

M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*. IEEE, 2016, pp. 426–437.

Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu, "The impact of continuous integration on other software development practices: A large-scale empirical study," *32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017.

W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE transactions on software engineering*, vol. 43, no. 9, pp. 817–847, 2017.

F.-X. Geiger, I. Malavolta, L. Pascarella, F. Palomba, D. Di Nucci, and A. Bacchelli, "A graph-based dataset of commit history of real-world android apps," in *Proceedings of the 15th International Conference on Mining Software Repositories, MSR. ACM, New York, NY*, 2018.

L. Pascarella, F.-X. Geiger, F. Palomba, D. Di Nucci, I. Malavolta, and A. Bacchelli, "Self-reported activities of android developers," in *5th IEEE/ACM International Conference on Mobile Software Engineering and Systems, New York, NY*, 2018.

T. Das, M. Di Penta, and I. Malavolta, "A quantitative and qualitative investigation of performance-related commits in android apps," in *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*. IEEE, 2016, pp. 443–447.

D. B. Silva, A. T. Endo, M. M. Eler, and V. H. Durelli, "An analysis of automated tests for mobile android applications," in *Computing Conference (CLEI), 2016 XLII Latin American*. IEEE, 2016, pp. 1–9.

R. Coppola, M. Morisio, and M. Torchiano, "Scripted gui testing of android apps: A study on diffusion, evolution and fragility," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2017, pp. 22–32.

V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Findings from github: methods, datasets and limitations," in *Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on*. IEEE, 2016, pp. 137–141.

C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The promises and perils of mining git," in *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 2009, pp. 1–10.

L. Corral and I. Fronza, "Better code for better apps: a study on source code quality and market success of android applications," in *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*. IEEE Press, 2015, pp. 22–32.

M. Linares-Vásquez, C. Bernal-Cárdenas, K. Moran, and D. Poshyvanyk, "How do developers test android applications?" in *33rd IEEE International Conference on Software Maintenance and Evolution (ICSME'17), page to appear*, 2017.

S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for Android: Are we there yet? (E)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 429–440.

D. Amalfitano, N. Amatucci, A. M. Memon, P. Tramontana, and A. R. Fasolino, "A general framework for comparing automatic testing techniques of android mobile apps," *Journal of Systems and Software*, vol. 125, pp. 322–343, 2017.

X. Zeng, D. Li, W. Zheng, F. Xia, Y. Deng, W. Lam, W. Yang, and T. Xie, "Automated test input generation for android: Are we really there yet in an industrial case?" in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 987–992.

P. S. Kochhar, T. F. Bissyandé, D. Lo, and L. Jiang, "Adoption of software testing in open source projects — A preliminary study on 50,000 projects," in *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*. IEEE, 2013, pp. 353–356.

——, "An empirical study of adoption of software testing in open source projects," in *Quality Software (QSIC), 2013 13th International Conference on*. IEEE, 2013, pp. 103–112.

J. Kaasila, D. Ferreira, V. Kostakos, and T. Ojala, "Testdroid: Automated Remote UI Testing on Android," in *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, ser. MUM '12. New York, NY, USA: ACM, 2012, pp. 28:1–28:4.

S. Stolberg, "Enabling agile testing through continuous integration," in *Agile Conference, 2009. AGILE'09.* IEEE, 2009, pp. 369–374.

R. Bavani, "Distributed agile, agile testing, and technical debt," *IEEE software*, vol. 29, no. 6, pp. 28–33, 2012.

T. Karvonen, W. Behutiye, M. Oivo, and P. Kuvaja, "Systematic literature review on the impacts of agile release engineering practices," *Information and Software Technology*, 2017.

Z. Gao, Z. Chen, Y. Zou, and A. M. Memon, "Sitar: Gui test script repair," *IEEE transactions on software engineering*, vol. 42, no. 2, pp. 170–186, 2016.

R. Coppola, "Fragility and evolution of android test suites," in *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 2017, pp. 405–408.

X. Li, N. Chang, Y. Wang, H. Huang, Y. Pei, L. Wang, and X. Li, "ATOM: Automatic maintenance of GUI test scripts for evolving mobile applications," in *Software Testing, Verification and Validation (ICST), 2017 IEEE International Conference on*. IEEE, 2017, pp. 161–171.

L. Cruz and R. Abreu, "Measuring the energy footprint of mobile testing frameworks," in *Software Engineering Companion (ICSE-C), 2018 IEEE/ACM 38th International Conference on*. IEEE, 2018.

G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: The contributor's perspective," in *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 2016, pp. 285–296.

M. Nayebi, H. Cho, and G. Ruhe, "App store mining is not enough for app improvement," *Empirical Software Engineering*, pp. 1–31, 2018.

K. O. McGraw and S. Wong, "A common language effect size statistic." *Psychological bulletin*, vol. 111, no. 2, p. 361, 1992.

D. S. Kerby, "The simple difference formula: An approach to teaching nonparametric correlation," *Comprehensive Psychology*, vol. 3, p. 11.IT.3.1, 2014. [Online]. Available: https://doi.org/10.2466/11.IT.3.1

N. L. Leech and A. J. Onwuegbuzie, "A call for greater use of nonparametric statistics." 2002.

M. E. Brooks, D. K. Dalal, and K. P. Nolan, "Are common language effect sizes easier to understand than traditional effect sizes?" *Journal of Applied Psychology*,

vol. 99, no. 2, p. 332, 2014.

G. Gousios, A. Zaidman, M.-A. Storey, and A. Van Deursen, "Work practices and challenges in pull-based development: the integrator's perspective," in *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 2015, pp. 358–368.

A. Van Deursen, "Program comprehension risks and opportunities in extreme programming," in *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*. IEEE, 2001, pp. 176–185.

K. Beck, *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

B. de Langhe, P. M. Fernbach, and D. R. Lichtenstein, "Navigating by the Stars: Investigating the Actual and Perceived Validity of Online User Ratings," *Journal of Consumer Research*, vol. 42, no. 6, pp. 817–833, 2016.

I. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. Hassan, "An examination of the current rating system used in mobile app stores," *IEEE Software*, 2017.

J. Gao, X. Bai, W.-T. Tsai, and T. Uehara, "Mobile application testing: a tutorial," *Computer*, vol. 47, no. 2, pp. 46–55, 2014.

M. E. Joorabchi, A. Mesbah, and P. Kruchten, "Real challenges in mobile app development," in *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*. IEEE, 2013, pp. 15–24.

M. Nagappan and E. Shihab, "Future trends in software engineering research for mobile apps," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, vol. 5. IEEE, 2016, pp. 21–32.