Test Automation in Open-Source Android Apps: A Large-Scale Empirical Study

Jun-Wei Lin, Navid Salehnamadi, and Sam Malek
School of Information and Computer Sciences
University of California, Irvine, USA
{junwel1,nsalehna,malek}@uci.edu

ABSTRACT

Automated testing of mobile apps has received significant attention in recent years from researchers and practitioners alike. In this paper, we report on the largest empirical study to date, aimed at understanding the test automation culture prevalent among mobile app developers. We systematically examined more than 3.5 million repositories on GitHub and identified more than 12,000 non-trivial and real-world Android apps. We then analyzed these non-trivial apps to investigate (1) the prevalence of adoption of test automation; (2) working habits of mobile app developers in regards to automated testing; and (3) the correlation between the adoption of test automation and the popularity of projects. Among others, we found that (1) only 8% of the mobile app development projects leverage automated testing practices; (2) developers tend to follow the same test automation practices across projects; and (3) popular projects, measured in terms of the number of contributors, stars, and forks on GitHub, are more likely to adopt test automation practices. To understand the rationale behind our observations, we further conducted a survey with 148 professional and experienced developers contributing to the subject apps. Our findings shed light on the current practices and future research directions pertaining to test automation for mobile app development.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Empirical Study, Automated Testing, Mobile Apps, Android

ACM Reference Format:
Jun-Wei Lin, Navid Salehnamadi, and Sam Malek. 2020. Test Automation in Open-Source Android Apps: A Large-Scale Empirical Study. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), September 21–25, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3324884.3416623

1 INTRODUCTION

Testing is an indispensable phase of software development life cycle. It is the primary way through which quality of software is improved. In comparison with manual testing, automated testing is reported to be more advantageous for a number of reasons, such as reliability, repeatability, and execution speed, especially in the context of continuous integration [16]. Since mobile apps are an integral component of our daily life and used to perform tasks in critical fields such as banking, health, and transportation, automated testing of mobile apps has received significant attention in recent years from researchers and practitioners alike.

For a number of research topics in the area of mobile software engineering, such as automated program repair [5, 55], automated test transfer [6, 40], mutation testing [15, 30, 41], regression test management [9, 31, 32], and test repair [8, 39, 49], understanding the extent mobile tests exist, the type and quality of these tests, and whether the tests are adopted in a particular way is of great importance. For instance, automated program repair of mobile apps [5, 55] is a plausible idea, only if apps come with a substantial number of tests to ensure the repairs are not breaking their functionality. Similarly, automated test transfer [6, 40] is going to yield good results, only if there is a large number of apps with tests, such that tests can be migrated from one app to another. In addition, mobile developers care about why and how to adopt automated testing practices and particularly, whether such adoption impacts the overall quality of their apps and ways in which their apps are perceived by the developer community. Therefore, a holistic view regarding practical adoption of test automation in mobile app development can contribute to both academia and industry.

To understand the test automation culture prevalent among mobile app developers, researchers have investigated the extent to which test automation is adopted in practice [12, 13, 33, 37, 42]. However, those studies are limited in terms of both scale and quality of the curated dataset. First, most prior works have only considered hundreds of apps from a single source, i.e., F-Droid. The findings and conclusions drawn from a relatively small set of sample apps may not generalize to the overall app ecosystem.

Second, previous studies have failed to exclude dummy and invalid tests; an important factor that might severely affect their conclusion. That is, when developers create a new project with Android Studio, the official IDE for Android app development, it generates some example test cases which are irrelevant for the created app. Including these default tests may influence the results of research questions as to the adoption of test automation practices.

Finally, appropriate and representative subjects are of critical importance for an empirical study. In the case of test automation for Android apps, a practical inclusion criterion is to consider only *non-trivial* apps, since it is not cost-effective to write tests for trivial apps such as class assignments, tutorials, or simple apps with only one component. Studying trivial apps cannot reveal useful insights

ASE '20, September 21–25, 2020, Virtual Event, Australia

ACM ISBN 978-1-4503-6768-4/20/09.
https://doi.org/10.1145/3324884.3416623

into the adoption of test automation practices. Nevertheless, no previous study has focused exclusively on non-trivial apps.

In this paper, we report on a large-scale empirical study on open-source Android apps from GitHub from three complementary perspectives: apps, developers, and impacts. We systematically examined more than 3.5 million non-forked repositories in Java and Kotlin, and investigated more than 12,000 real-world apps to determine (1) the prevalence of test automation in mobile app development projects; (2) working habits of mobile app developers with respect to automated testing; and (3) the correlation between the adoption of test automation and the popularity of projects in terms of different metrics, such as contributors and stars on GitHub, and ratings on Google Play Store.

Two important contributions of our work are the scale of study and the way we have curated the dataset. First, we considered more than 12,000 apps across 16 app markets including Google Play Store, F-Droid, and PlayDrone. We also developed novel heuristics to exclude irrelevant and example tests in data collection and analysis. Lastly, the subject apps were selected according to a criteria designated for identifying non-trivial apps (detailed in Section 4). As presented in Section 5, these efforts led to findings that are quite different from prior work.

Another contribution of our work is that we considered both unit tests and UI tests. Given the interactive nature of mobile apps, UI testing, which requires an emulator or a real device to run, is the primary way to examine the functionality and usability of mobile apps. Therefore, in addition to unit tests, we are interested in whether and how automated UI tests are adopted by mobile developers. We discuss related research questions such as developers' preference for unit and UI testing and their compliance with the Testing Pyramid practice [27] in Section 5.

To gather a deeper understanding of the underlying reasons for our observations from the source code, we further conducted a survey with the contributors of the subject apps, and ended up with 148 responses mainly from professional and experienced developers. Interestingly, with respect to some of the research questions, the results obtained from the analysis of project data and survey responses are inconsistent, indicating a gap between what the developers believe they do versus what they actually do.

Overall, this paper makes the following contributions:

- We report on the first large-scale analysis focusing on non-trivial apps in over 12,000 open-source projects from 16 app markets and spanning a period of 5 years, to investigate how test automation is practically adopted.
- We present the working habits of mobile app developers regarding test automation, such as the tendency to write tests or lack thereof and the compliance with the Testing Pyramid practice.
- We discuss how the presence of automated tests, and its extent, impact the popularity of apps in terms of different metrics on GitHub and Google Play Store.
- We present the findings of a survey involving 148 practitioners who developed the subject apps to understand the rationale behind our observations as well as the challenges in Android app testing.
- We create a publicly available dataset for this study [14]. The dataset was built by referring to multiple data sources including GitHub, Google Play Store, F-Droid, and AndroZoo. We believe the dataset can be of great utility for researchers working in the aforementioned research areas (e.g., automated program repair, automated test transfer, mutation testing) that need access to mobile apps with tests.

```
public class ExampleInstrumentedTest {
    @Test
    public void useAppContext() {
        Context appContext = InstrumentationRegistry
                                .getInstrumentation()
                                .getTargetContext();
        assertEquals("com.example",
                    appContext.getPackageName());
    }
}
public class ExampleUnitTest {
    @Test
    public void addition_isCorrect() {
        assertEquals(4, 2 + 2);
    }
}
```

Figure 1: Example Test Classes Generated by Android Studio

The remainder of this paper is organized as follows. Section 2 provides a background on mobile app test automation, followed by a brief review of prior research efforts in Section 3. Section 4 presents our approach for data collection, subject selection, and developer survey. Section 5 details our findings. Section 6 outlines the implications of this study for researchers and practitioners. The paper concludes with a discussion of threats to validity and future work.

2 TEST AUTOMATION IN ANDROID

2.1 Unit and UI Tests

Given the interactive nature of mobile apps, there are roughly two types of tests in Android: unit tests and UI tests.[1] According to the definition from Google [27], unit tests are small tests that "validate the app's behavior one class at a time". In contrast, UI tests or end-to-end tests are medium or large tests that "validate user journeys spanning multiple modules of the app". The key difference between unit and UI tests, besides the scope of testing, is that unit tests run on a local machine with JVM, while UI tests need an emulated or real device to run, and almost always use the Android OS or Android framework.

In Android Studio, the official IDE for Android app development, unit and UI tests are clearly separated—they are placed in different directories. The tests in the *test* folder are unit tests that run locally on JVM. The tests in the *androidTest* folder are UI tests that require an emulator or real device to run. These two directories are automatically generated when developers create a new project with Android Studio. In this study, we consider the tests under the *test* folder as unit tests, and the tests under the *androidTest* folder as UI tests.

A feature of Android Studio highly related to our study is that, when developers create a new project, it generates not only the folders, but also examples for different types of tests. By default, the

[1] Sometimes they are called local tests and instrumented tests [21].

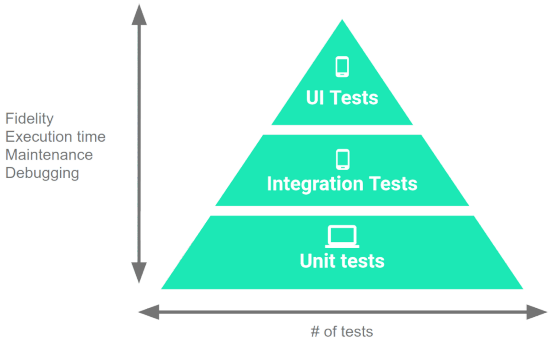

Figure 2: Illustration of the Testing Pyramid Practice from [27]

test folder contains a class called *ExampleUnitTest.java*, and the *androidTest* folder contains a class called *ExampleInstrumentedTest.java*, as shown in Figure 1. They are executable examples of unit and UI tests to help developers get started with test automation. However, including these example files may result in overestimated conclusions for research questions about the prevalence or adoption of automated tests, because developers may accidentally commit these files without an intention to write automated tests. In this study, we exclude these example files when counting number of tests contained in an app.

2.2 The Testing Pyramid Practice

The Testing Pyramid is a mindset or practice to guide developers in terms of how much effort they should put on creating different kinds of automated tests [1, 11, 18, 27, 47]. It essentially says that developers have to balance their automated tests by having many more low-level unit tests than high-level UI tests, as illustrated in Figure 2.

There are many reasons to follow the Test Pyramid practice. First, unit tests make debugging easier because they focus on small modules that can be tested independently. When unit tests fail, developers can quickly pinpoint the root cause of failure and save a lot of time. On the other hand, if there is a failure reported by a UI test, it usually means that the corresponding unit tests are incorrect or missing. Furthermore, unit tests are more robust and run faster in general, while UI tests may be subject to flakiness [45] and almost always run slower. As a result, while UI tests are still important to validate end-to-end workflows, overly relying on them will make testing expensive, slow, and brittle.

Although the proportion of tests for each layer in the Testing Pyramid varies based on different apps, a general recommendation from Google is a 70/20/10 split: 70% unit tests, 20% integration tests, and 10% UI tests [27]. Note that, while there is a layer of integration tests, and they can be understood as tests that "validate the collaboration and interaction of a group of units [27]", the scope for integration tests is controversial [35]. In fact, these three layers are not totally clear-cut and sometimes overlap with each other [48]. In this paper, we leverage the characteristics of Android apps and Android Studio to identify the two major types of tests, unit and UI tests. Furthermore, according to the above guideline, an appropriate ratio of UI tests could be 20% to 30% of the total number of tests.

3 RELATED WORK

Empirical studies on mobile app testing. Previously, researchers have investigated how test automation is practically adopted [12, 13, 33, 37, 42, 43, 50]. Kochhar et al. [37] analyzed over 600 Android apps on F-Droid to check the presence of test cases and computed the code coverage. They also conducted surveys to understand the usage of automated testing tools and the challenges faced by developers while testing. Cruz et al. [13] analyzed 1,000 Android apps on F-Droid to check their usage of automated testing frameworks and continuous integration tools. They also found that projects using automated testing have more contributors and commits on GitHub. Recently, Fabiano et al. [50] analyzed 1,780 Android apps on F-Droid to investigate the prominence of tests developed for the apps, as well as other quality metrics of the tests such as test smells, code coverage, and assertion density. Our work is different from theirs in terms of the scale and data source, as we analyzed over 12,000 apps across 16 app markets.

In addition, Coppola et al. [12] analyzed more than 15,000 apps on GitHub to examine the diffusion, evolution, and modification causes of UI tests in open-source Android apps. While their work is highly related to ours, the key difference is that we focus on only non-trivial apps as they did not factor out toy apps and forks of real apps from their dataset. For example, among the list of 1,042 repositories with tests released by the authors[2], only 42 (4%) of them are considered in our study. That means our study considers a very different set of apps from theirs.

On the other hand, to know the main challenges that developers face while building mobile apps, Joorabchi et al. [33] conducted a qualitative study with 12 mobile developers from 9 companies, followed by a survey with 188 respondents. Linares-Vásquez et al. [42] also analyzed responses from 102 open-source Android app developers to understand their practices and preferences regarding Android app testing. Unlike our work, these papers did not analyze open-source data in the wild and merely relied on interviews and survey responses.

Finally, Linares-Vásquez et al. [43] reviewed the frameworks, tools, and services for automated mobile testing, and their limitations. From a survey, they identified several key challenges that should be addressed in the near future by the researchers in the area of mobile test automation. Nevertheless, their work did not include any source code analysis or developer survey.

Another related topic is the empirical study on automated input generation (AIG) tools [10, 56, 57]. The work by Choudhary et al. [10] focused on the comparison of different AIG tools in terms their usability, compatibility, code coverage and fault detection capability. Another empirical study by Wang et al [56] performed a similar comparison of AIG tools but focused on industrial apps. Zeng et al. [57] further investigate the limitations of Android Monkey, the most widely used AIG tool, in an industrial setting with a popular and commercial messenger app. Our work does not consider AIG tools, rather focuses on automated or scripted test cases created by developers.

Empirical studies on open-source software testing. A number of studies investigate the test adequacy in general open-source

[2]We have contacted the authors to ask for the complete list of repositories under their study but get no response.

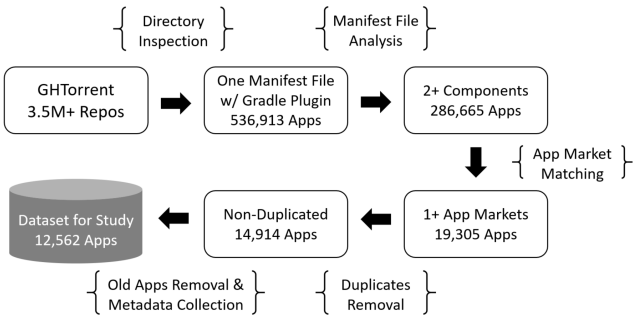

Figure 3: Flow of Data Collection and Analysis in This Study

software. Kochhar et al. [36] studied more than 20,000 projects on GitHub regarding their adoption of testing, and the correlation of test cases with various project development characteristics such as project size and number of bugs. To answer questions related to the usage, costs, and benefits of continuous integration, Hilton et al. [29] analyzed more than 34,000 projects on GitHub. Fraser and Arcuri [19] empirically evaluated the code coverage ability of EvoSuite, a search-based testing tool, with public classes retrieved from 100 Java projects from SourceForge. On the other hand, Beller et al. [7] reported a field study with 416 software engineers in which their development activity was monitored with an Eclipse plugin to understand how and when developers conduct testing. Our work complements these studies by providing insights in the context of Android app development.

4 METHODOLOGY

Figure 3 depicts the flow of data collection and analysis in our study. This study consisted of the following steps: (1) we first collected a large list of GitHub repositories from the GHTorrent database [28]; (2) we set filtering criteria to identify the repositories representing non-trivial Android apps; (3) we further analyzed the identified repositories to collect their meta-data and information about automated tests and popularity; (4) we evaluated the collected dataset to answer research questions about the test automation culture prevalent among mobile app developers; and finally (5) we conducted a survey with the developers of the subject apps to get a deeper understanding of the underlying reasons for our observations from the dataset. We now describe each of these steps in further detail.

4.1 Study Subjects and Selection Criteria

The initial list of GitHub repositories for our study was obtained from the GHTorrent database [28]; a research project that monitors the GitHub public event time line and populates a relational database with the collected information, i.e., meta-data. We downloaded the latest dump of their database [20], and queried the repositories written in Java or Kotlin that are neither forked nor deleted. The query returned a list of more than 3.5 million repositories.

To identify the repositories of non-trivial and real-world Android apps from the returned list, we set the following selection criteria:

(1) The repository must contain exactly one *AndroidManifest.xml*. The manifest file is a must-have for every Android app to provide essential information about the app to the Android build tools [23]. The reason for exactly one manifest file is that the repository containing multiple such files is likely a tutorial or class assignment with multiple demo apps. We used GitHub API to walk through the directory tree of the projects to search for the files.

(2) The repository must contain `build.gradle` with a specific string "`com.android.application`" inside. Android Studio uses Gradle as its build system, and a Gradle plugin with this specific string means that this project has a task to build an Android app. We used GitHub API to search the projects with the specified condition. Most of the repositories were filtered out with these two criteria, with about 537 thousand apps left.

(3) At least two components have to be declared in the manifest file. We parsed the manifest file and looked for the declaration of four Android component types (i.e., Activity, Service, Broadcast Receiver, and Content Provider [24]) inside. We set a threshold of 2 components because we believe it is not cost-effective to write tests for a simple app with only one component. About half of the apps were removed by this step, with 287 thousand apps left.

(4) The package name stated in the manifest file must appear in an app market. We believe that the apps published in app markets, especially the markets that charge fees to join such as Google Play Store, are more likely beyond toy or demo apps, because the developers want the apps to reach general users (and even willing to pay for it). From the manifest file of each app, we retrieved the package name and tried to match it with apps hosted in the following app markets: Google Play Store, F-Droid [17], and the list of package names and markets provided by AndroZoo [3].[3] This criterion was critical to identify non-trivial apps and left us with a list of about 19 thousand apps.

(5) We removed the apps with duplicate package names, and ended up with a list of 14,914 GitHub repositories of non-trivial Android apps.[4]

The above filtering process took two months, primarily because of the rate limit of GitHub API (5,000 requests per hour).

4.2 Data Collection and Analysis

For each of the selected repositories, we used GitHub API to further collect its meta-data: creation date, number of forks, number of stars, number of commits, number of contributors, number of issues, and number of pull requests. If the app is on Google Play Store, we also collected its category and user ratings by crawling the app page.

To collect the information about how test automation is adopted in the project, we used GitHub API to walk through the directory tree of the project, and parsed all the files under the *test* and *androidTest* folders, if any exist. We considered a method as a test case if it is annotated with "@Test". This annotation is used by JUnit-based testing frameworks, including both unit and UI testing frameworks such as JUnit [34], Robolectric [52], Mockito [46], and Espresso [26]. A prior study investigating the usage of testing frameworks in 1,000 apps on F-Droid [13] shows that 100% of the adopted unit testing frameworks and 97% of the UI testing frameworks are JUnit-based. Furthermore, we classify a test case as a

[3] A list of app markets considered by AndroZoo can be found at [4].
[4] Sometimes two repositories contain the same package name because one is a direct copy of the other (not by forking). In this situation, we keep the repository with the oldest creation date.

Table 1: Distribution of Apps by App Market*

Market*	#Apps
Google Play	11265
PlayDrone	539
fdroid	434
anzhi	408
appchina	294
mi.com	70
VirusShare	62
angeeks	41
1mobile	26
freewarelovers	12
slideme	10
torrents	4
praguard	3
hiapk	2
proandroid	1
apk_bang	1

*An app may belong to multiple markets

Table 2: Distribution of Apps by Year Created

Year Created	#Apps
2015	3614
2016	2330
2017	1731
2018	2898
2019	1989
Total	12562

unit test if it is under the *test* folder, and otherwise as a UI test (i.e., under the *androidTest* folder). Finally, as mentioned in Section 2.1, we excluded the example unit and UI test generated by Android Studio.

An assumption of our study is that the subject apps were developed with Android Studio. Because Android Studio has been the official IDE for Android app development since its first stable release in December 2014 [22], we further factored out the repositories before 2015 from the list described in Section 4.1. We finally ended up with 12,562 repositories/apps in our dataset. The distribution of apps by app market is shown in Table 1. While the majority of the apps were published on Google Play Store, the dataset covers apps across 16 app markets. Table 2 shows the distribution of apps by the year they were created. For the apps on Google Play Store, Figure 4 shows the distribution by category.

4.3 Survey

To complement our findings, we conducted an online survey with the developers of the subject apps in our dataset. In this section, we describe the design, participant selection, and data collection of the survey.

4.3.1 Survey Design. The online survey was designed to understand the rationale behind our findings from the dataset as well as the challenges in Android app testing. We first asked demographic questions to understand the respondents' background, such as their

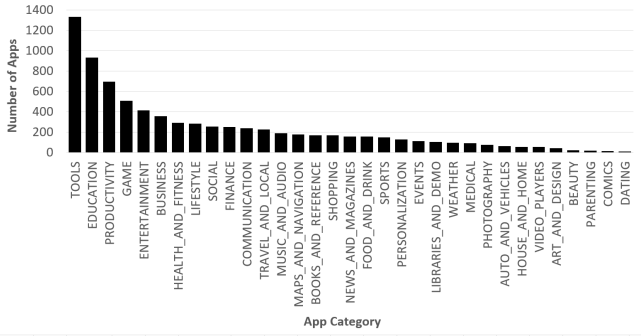

Figure 4: Distribution of the Google Play Apps by Category

experiences in terms of the number of years of Android app development. We then asked them about their current practices of Android app testing. For the respondents reporting the use of automated tests, we further asked them related questions such as the preference for unit and UI testing and whether they follow the Testing Pyramid practice, and the reasons for their choices. Next, we presented some of our findings in the correlation analysis between the adoption of test automation and the popularity of apps, and asked for their opinions on possible explanations. Finally, we asked the respondents for the difficulties in adopting automated tests and general challenges of testing Android apps. For all questions about practices and opinions, we provided a set of choices identified from previous studies [13, 37, 42], as well as an "other" choice with free form text if none of the provided choices apply. A sample of the survey can be found at the companion website [14].

To ensure that the questions were clear and the survey can be finished in 10 minutes, we conducted a pilot survey with graduate students in Computer Science who have experience in Android app development and survey design. We rephrased some questions according to the feedback. The responses from the pilot survey were used solely to improve the questions and were not included in the final results.

4.3.2 Participant Selection. From each subject app in our dataset, we tried to retrieve the email of its main contributor in the following order: (1) the email found in the GitHub profile of the repository's owner; (2) the email of the contributor who made the most commits; and (3) the email of the contributor who made the most recent commit. After removing invalid and duplicate data, we identified 7,490 unique email addresses for our survey.

4.3.3 Data Collection. We used Qualtrics [51] to distribute the survey to the 7,490 targeted email addresses, and 653 of them bounced. From the 6,837 emails successfully sent, we received 148 valid and complete responses with a 2.2% response rate. The response rate is close to the results of previous studies such as 2.1% (83/3905) reported in [37] and 1.0% (102/10000) reported in [42] on very similar surveys with mass developers on GitHub.

The 148 received responses are from 45 countries. The top two countries where the respondents reside are United States of America (22.3%) and India (10.8%). 70.3% of the respondents are professional software developers paid by a company, and 68.9% of them have more than 2 years of experience in Android app development.

Table 3: Distribution of Apps in Terms of Presence of Test Cases

Group	#Apps	Percentage
Apps with any tests	1002	7.98%
Apps without tests	11560	92.02%
Apps with unit tests	766	6.10%
Apps with UI tests	502	4.00%
Apps with both unit and UI tests	266	2.22%

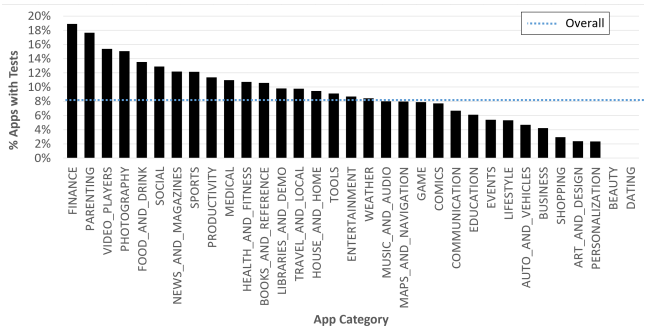

Figure 5: Prevalence of Test Automation of the Google Play Apps by Category

5 RESULTS

In this section, we present the results of our study from three complementary perspectives: apps, developers, and impacts.

5.1 App Perspective

We started by analyzing our curated dataset to understand the state of affairs with respect to test automation adoption in open-source projects. Answers to these questions are important for emerging areas of research interest (e.g., automated program repair, automated test transfer) that rely on the availability of large number of tests.

RQ1. How prevalent is test automation in open-source Android apps, in terms of the presence of unit and UI tests?

Table 3 shows the number of repositories grouped by the presence of different types of tests. The results indicate that only 7.98% of the subject apps contain tests, and most of them are poorly tested in an automated manner—even though they are non-trivial. This percentage is much lower than previous findings: 20% reported in [12], 14% reported in [37], and 40% reported in [13].

There are many possible reasons for the inconsistency between our results and previous findings. First, our analysis excludes the placeholder tests that are automatically generated by Android Studio, as mentioned in Section 2.1. This check was critical for our results, since such tests are common in our dataset (7,017 of the 12,562 apps examined, 56%). We also manually checked the dataset released by Coppola et al. [12], and found such examples in the reported test cases. We are not able to verify the results reported by Kochhar et al. [37] because they are not willing to release their dataset. Regarding the results reported by Cruz et al. [13], since they did not search for test cases (detailed in the next paragraph), we are unable to compare their results with ours.

The way one computes the existence of tests can also influence the results significantly. For example, in the study by Cruz et al. [13], they inspect the build configurations and look for imports related to testing frameworks to determine the presence of tests in a repository. Since having related imports in the build configurations does not necessarily mean there are test cases in the project, their findings about prevalence of test automation is prone to overestimation.

Our inclusion criterion for subjects are different from prior studies too. We excluded the trivial apps (i.e., simple/demo apps with only one component), which is not the case with all prior studies.

Finally, the scale of study might also affect the results. In the papers by Kochhar et al. [37] and Cruz et al. [13], only 627 and 1,000 apps from F-Droid were analyzed, respectively. In contrast, our study considers more than 12,000 apps on GitHub across 16 markets, which is substantially different from their works in terms of scale and source of data.

Another finding from Table 3 is that UI testing is not adopted as extensively as unit testing (i.e., 4% vs. 6.1%). We will further discuss this in Section 5.2.

Observation 1: Only 8% of the non-trivial and real-world apps have automated tests. Automated UI testing is less adopted than unit testing.

RQ2. Is the prevalence of test automation varied across different categories of apps?

To understand whether there are any patterns as to the adoption of automated testing practices across different categories of apps, for the Google Play apps with category information in our dataset, we report their adoption of automated tests by category in Figure 5. As depicted in Figure 5, while overall the prevalence of test automation is 8%, the percentage is substantially higher for some categories of apps such as finance (19%) and video players (15%). On the other hand, some categories of apps such as shopping (3%) and dating (0%) are poorly tested in an automatic manner. This variance could be attributed to the quality requirements for different categories. Note that the observed patterns may not generally apply to apps on Google Play Store, since many commercial and closed-source apps, such as popular shopping apps, are not included in our study.

The observed patterns have practical implications for both researchers and practitioners. For instance, the fact that certain categories of apps contain more tests than others indicates that techniques like automated test transfer [6, 40] may work much better for apps of a certain category than others. The results also provide invaluable hints as to what are the customary development practices for apps of a certain category. This might help developers set up the right development practice for their open-source projects to gain traction and amass contributors.

Observation 2: Some categories of apps, such as finance and video players, are more extensively leveraging test automation techniques than others.

Table 4: The Ways of Testing Android Apps by the Survey Participants

Way	#Respondents
Manually	130
With scripted/automated tests	85
With dedicated QA team or 3rd party testing services	43
With automatic input generation tools	12
Other	6
Not at all	3

Table 5: The Reasons for not Adopting Test Automation by the Survey Participants

Difficulty	#Respondents
Cost to create and maintain automated tests	77
Time constraints	74
Size or maturity of the app	66
Lack of exposure or knowledge of existing frameworks	52
Cumbersome to use	50
Lack of support from management or organization	30
Other	11

Table 6: The Biggest Challengs in Testing Android Apps by the Survey Participants

Challenge	#Respondents
Fragmentation	104
Concurrency	66
Performance	51
Security	44
Energy	43
Functionality	43
Accessibility	35
Other	14

Table 7: The Most Important or Useful Criteria for Evaluating Android App Tests by the Survey Participants

Criterion	#Respondents
Fault detection capability of tests	96
Feature or use case coverage of tests	83
Code coverage of tests	67
Code or test case reviews	59
Other	7

5.2 Developer Perspective

In this section, we present our findings regarding the associations between developers and test automation, including the rationale and preferences reported by the survey participants.

RQ3. How prevalent is test automation and what are the reasons for not adopting it (as reported by developers)? What are the challenges in testing of Android apps in general?

In our survey, we asked the developers how they test their Android apps, and they were allowed to select all options that apply. Table 4 shows the results. Interestingly, over 57% (85/148) of the respondents state that they are using automated tests, yet we do not observe this degree of test automation adoption from the subject apps they develop. One possible explanation for this inconsistency is that the proponents of test automation are more willing to take our survey, while the developers not interested in test automation have no incentive to provide feedback. Another reason could be that the professional developers adopt automated tests at work, but not for their pet projects on GitHub. Finally, it is also possible that the developers only uploaded their source code on GitHub without corresponding tests.

To understand why the observed adoption of test automation is low, we asked the developers to specify the reasons for not adopting test automation. From the results in Table 5, we can see the top three reasons are: (1) cost to create and maintain automated tests, e.g., caused by changing requirements or rapid development; (2) time constraints, e.g., because of time-to-market or customer's schedule; and (3) size or maturity of the app, e.g., the app is not big or complex enough to require automated tests. Note that the third reason corresponds to our insight that it is not cost-effective to write automated tests for trivial apps, and they should be excluded in the empirical study, as we have done. Besides, the respondents also mentioned other interesting difficulties in adopting test automation as follows:

"Legacy code not designed to be tested requires lots of refactoring which makes it harder to justify the additional effort to write tests."

"...hard to test unexpected GUI aspects or unexpected hardware (manufactor firmware) issues or unexpected permission issues or unexpected Android behavoir or unexpected 3rd party data formats."

It is worth mentioning that we also asked two general questions to understand (1) the biggest challenges in testing of Android apps; and (2) the most useful criteria for evaluating tests for Android apps. The results are reported in Tables 6 and 7. According to Table 6, the top three challenges are: (1) fragmentation, e.g., multiple Android OS or API versions, devices with different sizes or resolutions, etc.; (2) concurrency, e.g., detecting data races, deadlock, or violation of execution order of methods; and (3) performance, e.g., app's responsiveness such as frames per second for gaming apps. Moreover, from Table 7 we can see that the developers do not consider code coverage as the most important criterion for evaluating tests, which is in line with the prior study [42]. We believe the reported concerns call for additional research and development in test automation frameworks and tools. We take a closer look at the implications of this result in Section 6.

Observation 3: 57% of the survey participants reported the use of test automation, which varies drastically from that observed in the dataset. The top three difficulties in adopting test automation are: cost to create and maintain tests, time constraints, and size or maturity of the app.

RQ4. Do the same developers have the same testing habits across apps?

In this section, we investigate whether developers are following the same test automation habits across apps. To that end, we first clustered all subject apps by their owner, i.e., the GitHub account,

Table 8: Probability of Observing Consistent Behavior on the Apps by the Same Developers. S_a: Clusters of Apps by the Same Developers. S_b: by Different Developers

Set	Size (#Clusters)	#Clusters Same Behavior	Probability	p-value
S_a	985	902	91.57%	4.06e−12
S_b	985	763	77.46%	

and obtained a set of 985 clusters, S_a, in which each cluster contains two or more apps by the same developer. Next, we defined and computed the *test adoption rate* for each cluster C in S_a as follows:

$$rate(C) = \frac{\#Apps\ with\ test\ in\ C}{\#Apps\ in\ C}$$

A cluster with a rate of 1 or 0 means the developer has followed the same behavior across apps. That is, the developer either wrote tests for all of her apps or did not write tests at all. We further computed the probability of observing the same behavior in S_a by dividing the number of clusters showing the same behavior (i.e., achieve test adoption rate of 1 or 0) by the size of S_a.

Moreover, to understand if the probability observed in S_a is high, we created another set of clusters, S_b, as a control group. The number of clusters and the size of each cluster in S_b is exactly the same as S_a. However, the apps in S_b were randomly selected from the apps not in S_a. We computed the test adoption rate for each cluster in S_b using the same equation, and the probability of observing the same behavior in S_b accordingly.

Finally, to determine if the observed difference between S_a and S_b is statistically significant, we applied hypothesis testing on the rate distribution of S_a and S_b using the non-parametric test Mann-Whitney U [44] with a significance level of 0.05. We chose the Mann-Whitney U test because S_a and S_b are not normally distributed and did not pass the normality test of Shapiro-Wilk [54].

The results in Table 8 show that in S_a, the set of clusters in which each cluster consists of the apps by the same developer, it is more likely to observe a cluster manifesting the same behavior. In other words, for a group of apps by the same developer, the probability that either all or none of them have tests (91.57%) is higher than a group of apps by different developers (77.46%). The difference between S_a and S_b is statistically significant, because the null hypothesis that S_a and S_b are from the same distribution is rejected by the Mann-Whitney U test with a *p-value* of 4.06×10^{-12}. This finding vouches for the effect of software engineering education regarding test automation: once learned, developers keep their habits.

Observation 4: App developers tend to follow the same test automation practices across projects.

RQ5. Do developers prefer unit or UI testing and why?

From Table 3 in Section 5.1, we see that the apps adopting unit tests (6.1%) are more than UI tests (4%). To validate our observation and understand the reasons behind this, for the developers reporting the use of test automation, we further asked what type of testing (unit testing or UI testing) they do mostly and why. Among the 83 respondents, the majority of them (55/83, 66%) prefer unit testing.

Table 9: The Reasons for the Preference of Unit Testing by the Survey Participants

Reason	#Respondents
Speed	39
Scope	30
Simpleness	28
Robustness	26
Other	6

Table 10: Distribution of the Number of Tests in the Apps with Both Types of Tests. 1Q: 1st quartile. 2Q: 2nd quartile (median). 3Q: 3rd quartile.

	Distribution					
	Min	Max	Mean	1Q	2Q	3Q
#Unit Tests	1	685	34.75	3	11	27.25
#UI Tests	1	178	14.32	2	7	17
Ratio of UI Tests to All Tests	0.3%	97.1%	41.9%	17.5%	40.0%	64.4%

This is in line with our observation from the dataset. Furthermore, 27% (22/83) of the respondents have no preference and 7% (6/83) of them prefer UI testing.

We also asked the proponents of unit testing for their rationale. Table 9 shows that the top three reasons by the developers are: (1) speed, e.g., unit tests run faster than UI or end-to-end tests; (2) scope, e.g., unit tests focus on small or independent modules, thereby simplify the debugging; and (3) simpleness, e.g., unit tests are easier to learn and write. On the other hand, developers preferring UI testing indicate that the interactivity is the top reason, because UI or end-to-end tests can test the app in a more interactive and straightforward way. In Section 6, we discuss how these insights could be used for possible improvements of UI testing tools and libraries.

Observation 5: Majority of the developers prefer unit testing, corroborated through both project dataset and survey results. The top three reasons are speed, scope and simpleness.

RQ6. Is the practice of Test Pyramid followed by developers?

As mentioned in Section 2.2, the Testing Pyramid practice is a guideline for developers to have a balanced portfolio of different types of automated tests. To understand if the guideline is appropriately followed by the developers, we analyzed the 266 apps containing both unit tests and UI tests in our dataset by counting the number different types of tests. Furthermore, we computed the ratio of the number of UI tests to the total number of tests as follows:

$$\frac{\#UI\ tests}{\#Unit\ tests + \#UI\ tests} \times 100$$

Table 10 shows that the distribution of the numbers of unit and UI tests in the apps are skewed, because the averages are much larger than the medians (i.e., 34.75 vs. 11 for unit tests, and 14.32 vs. 7 for UI tests). That means some apps contain many more tests than others. On the other hand, the third quartile shows that 75%

of the apps have fewer than 27.25 unit tests and 17 UI tests. We believe these are reasonable numbers for general apps.

Regarding the developers' compliance with the Test Pyramid practice, Table 10 shows that in more than half of the apps, the ratio of UI tests is higher than 40%, which differs from the recommended ratio of 20%-30% by Google [27]. In other words, the developers put more effort than recommended in writing UI tests. A possible explanation is that the interactive nature of mobile apps drives the developers to write more UI tests. However, while UI tests are essential to validate certain types of requirements such as business logic and usability, overly relying on them may make testing and debugging cumbersome, as mentioned in Section 2.2.

In our survey, we asked the participants whether they are following the Testing Pyramid practice, and 52% (51/98) of them said no, which is consistent with our observation from the dataset. A prominent reason from the respondents reporting the non-compliance is the lack of exposure or knowledge about the Testing Pyramid practice (40/51, 78%). Other interesting reasons include "special needs for my team or projects" and "the Testing Pyramid practice is misleading/flawed".

Observation 6: Developers put more effort than recommended in writing UI tests, as the average ratio of UI tests to all tests is 40%.

5.3 Impact Perspective

Mobile app developers often strive to have their apps become popular. As members of an open-source community, developers are pleased to see their apps receive more attention from other developers in terms of stars, forks, contributors, etc. on GitHub. As product owners, developers want their apps to satisfy the users and receive good ratings and feedback on the market. While these popularity metrics are not necessarily related to the development process of apps, we would like to investigate whether they are impacted by the adoption of test automation. Specifically, we consider the following popularity metrics on GitHub: number of stars, forks, contributors, commits, issues[5], and pull requests. Moreover, we consider user ratings on Google Play Store as the metric of user satisfaction. These metrics were collected in the manner described in Section 4.2. Table 11 presents the distribution of data in terms of different metrics.

RQ7. How does test automation relate to project popularity?

We would like to know whether apps with tests are different from apps without tests in terms of the popularity metrics on GitHub. First, to eliminate the effect caused be app size, we excluded the apps that have fewer than 3 components (the 1st quartile) and more than 8 components (the 3rd quartile) in our dataset, ending up with a set of 7,664 apps under consideration. Next, we conducted statistical analysis for each metric with the following steps:
(1) We divided the data into two disjoint sets, R_w and R'. R_w consists of the metric values from the apps with tests. R' consist of the metric values form the apps without tests.

[5]Issues may be considered as an indicator of app quality. In fact, the topics posted with issues can be very broad, such as feature request or usage discussion. Therefore, we consider it as an indicator of popularity.

Table 11: Distribution of the Popularity and Satisfaction Metrics of the Apps. 1Q: 1st quartile. 2Q: 2nd quartile (median). 3Q: 3rd quartile.

		Distribution					
	Sample Size	Min	Max	Mean	1Q	2Q	3Q
Stars	12533	0	7897	15.09	0	0	1
Forks	12533	0	2209	4.49	0	0	1
Contributors	12533	0	451	2.30	1	1	2
Commits	12527	1	13844	75.95	4	15	55
Issues	12527	0	2442	5.5	0	0	0
Pull Requests	12527	0	1679	3.80	0	0	0
Ratings	3937	1	5	4.23	3.91	4.38	4.78

(2) We applied the Z-score method [38] with a threshold of three times of standard deviation to remove the outliers from both sets.
(3) Since the apps without tests are much more than the apps with tests in our dataset, R_w and R' are extremely unbalanced in terms of their sizes. Given that unequal sample sizes may generally reduce statistical power of equivalence tests [53], we created R_o with the same size as R_w by randomly selecting the values in R'.
(4) We computed the mean and median of R_w and R_o and the difference between the mean and median.
(5) To determine if the difference observed in R_w and R_o is statistically significant, as in Section 5.2, we performed hypothesis testing on R_w and R_o using the Mann-Whitney U test with a significance level of 0.05. The null hypothesis on R_w and R_o is that they were selected from populations having the same distribution. For example, in the case of stars, the null hypothesis is that "an app with tests (from R_w) has the same number of stars on GitHub as an app without tests (from R_o)". We chose the Mann-Whitney U test because R_w and R_o are not normally distributed and did not pass the normality test of Shapiro-Wilk.
(6) The above process is repeated for all the popularity metrics.

Table 12 shows the results of our statistical analysis. The statistical evidence shows that test automation is associated with all popularity metrics. Namely, on average, open-source Android apps with tests are expected to have more stars, forks, contributors, commits, issues, and pull requests on GitHub. Our finding is not exactly in line with the prior work by Cruz et al. [13], in which they only found such correlation with contributors and commits but not other metrics. We believe this inconsistency is caused by similar reasons discussed in Section 5.1.

We presented this correlation to the survey participants and asked for their opinions as to the possible explanations. 57% (84/148) of the respondents believe that there is a cause-and-effect relationship between test automation and popularity. The causation, however, could be direct, reverse, bidirectional, etc., as explained by some of the respondents below:

"I would say they have a direct connection since the quality and rigidness of the app's code can definitely influence an app's popularity." (direct)

"Projects can only grow to large numbers if they are stable. Automated testing can ensure this happens to some degree." (direct)

Table 12: Impact of Having Tests on the Popularity of Apps. R_w: Apps with Tests. R_o: Apps Without Tests.

	Stars*				Forks*			
	Size	Mean	Median	p-value	Size	Mean	Median	p-value
R_w	629	10.95	0	4.07e−11	630	3.74	0	3.59e−12
R_o	629	4.57	0		630	1.31	0	
Δ		6.38	0			2.43	0	

	Contributors*				Commits*			
	Size	Mean	Median	p-value	Size	Mean	Median	p-value
R_w	630	2.76	2	1.75e−14	628	147.21	84.5	3.53e−67
R_o	630	1.63	1		628	39.93	14	
Δ		1.13	1			107.28	70.5	

	Issues*				Pull Requests*			
	Size	Mean	Median	p-value	Size	Mean	Median	p-value
R_w	635	10.39	0	1.40e−27	633	8.76	0	1.95e−29
R_o	635	1.35	0		633	0.85	0	
Δ		9.04	0			7.91	0	

*The difference is statistically significant.

"First you build the app, then it gets popular, then you get resources/motivation to increase it's quality. That's when you go to UI tests." (reverse)

"I think because the projects were big they were motivated to create a comprehensive testing suite." (reverse)

"Projects that become popular end up writing more tests because they need to ensure the stability of the project. As the project becomes more stable (due to more testing) it provides a positive feedback loop. The project, in part, is more likely to be popular if it is perceived as stable, and testing helps to increase that stability." (bidirectional)

On the other hand, 34% (50/148) of the respondents consider this correlation to be more of a connection than causation. For example, the following responses claim common causes for them:

"Common cause: Experienced developer who cares about making code evolvable."

"Popular projects are usually bigger, with multiple developers and with more management. Tests is just a part of that process."

Observation 7: Popular projects are more likely to adopt test automation practices. 57% of the developers believe it implies causality between them.

RQ8. How does test automation relate to user satisfaction?

Following the same steps, we conducted statistical analysis to investigate whether test automation relates to user satisfaction in terms of Google Play ratings. As shown in Table 13, we do not find the association between them with statistical significance.

Surprisingly, when we presented this to the survey participants and asked for their opinions, 52% (77/148) of the respondents believe that test automation and user ratings should be somehow related. Namely, the developers do not believe our finding is correct. Examples of their reasons are as follows:

"I think it would depend on the type of application. Games and such are harder to test and the quality of test does not correlate with how fun the game is. For a banking application tests are essential and do effect the quality of the final product."

Table 13: Impact of Having Tests on the User Satisfaction of Apps. R_w: Apps with Tests. R_o: Apps Without Tests.

	Size	Mean	Median	p-value
R_w	211	4.14	4.25	0.0689
R_o	211	4.2	4.33	
Δ		-0.06	-0.08	

"Play Store ratings are a noisy metric of app quality and overall user experience, so the no apparent correlation doesn't convince me that app quality isn't impacted at least somewhat by automated testing"

Observation 8: Users' satisfaction with apps appears to be unrelated to the adoption of automated testing practices in their development, while half of the developers think differently.

6 DISCUSSION

Automated testing is not widely adopted. Only 8% of the subject apps in our study have adopted automated testing. As mentioned earlier, this finding contradicts earlier studies that have reported substantially higher adoption rate [12, 13, 37], but it is in line with the general perception that it is challenging to find complex and open-source apps with lots of tests for research purposes, as noted by Adamsen et al. [2]. Nevertheless, our study addresses this issue by providing a dataset of real-world and non-trivial apps with automated tests, which can by of significant utility for emerging areas of research interest, such as automated program repair [5, 55], automated test transfer [6, 40], and mutation testing [15, 30, 41]. Moreover, researchers may hold out hope on specific categories of apps when looking for automated tests for their experiments, since our results indicate that the prevalence of test automation is varied across different categories. Note that the focus of our study is on automated or scripted tests. The subject apps may have gone through proper manual testing by the developers, but that is outside the scope of this study.

Automated testing can be useful and important. We found a strong correlation between the adoption of automated testing practices and the popularity of development projects. The majority of the survey respondents (91%, 134/148) believe that the correlation is either causation or a connection. On the other hand, while users' satisfaction appears unrelated to test automation, a considerable amount of survey participants think that automated testing contributes to app quality in terms of stability and maintainability, and has impacts on users' satisfaction. As noted by previous studies [16, 37], automated testing is not universally applicable, but can be useful and important, especially for apps that update regularly and frequently.

Automated testing needs more attention, organizationally and culturally. Despite the benefits of automated testing, our study shows that it is not adequately adopted in practice. Many reported difficulties in adopting test automation, however, can be addressed from the perspective of organization and culture. For instance, management or organization could provide more support

in terms of budget or schedule to allow for the introduction and maintenance of automated tests. In other words, adoption of test automation involves a culture change; organizations need to be willing to incur the additional cost and effort of setting up test automation practices early on for the promise of producing higher-quality apps at a faster pace later on.

Tools and libraries have room for improvement. One of the difficulties reported by developers in adopting automated testing practices is cumbersome tools, including steep learning curve, poor documentation, usability, and compatibility issues. A possible improvement of such tools is a comprehensive information hub that aggregates and summarizes scattered pieces of information from tutorials, forums, blogs, case studies, etc., to make the learning and use of such tools easier. Besides, in our study, UI testing is less adopted than unit testing, and developers have concerns about the speed, simpleness, and robustness of UI testing. As a result, current UI testing tools could be improved by addressing these concerns. For example, supporting "headless mode" such as done by Robolectric [52] can let developers run UI tests without an emulator and save a massive amount of execution time. In addition, interactive tools such as Espresso Test Recorder [25] can help developers create UI tests without writing test code. Finally, efforts to prevent or resolve flakiness of UI tests may increase the robustness and attract more users.

Awareness matters. Our study indicates a primary reason for not following specific practices in automated testing is the lack of exposure or knowledge about them. Moreover, we found that once developers learn and begin to use test automation techniques, they maintain that habit across other projects. Therefore, raising the developers' awareness of existing test automation frameworks, tools, and practices may increase their adoption.

7 THREATS TO VALIDITY

External validity. The major external validity is the generalization of our findings to all open-source Android apps. We mitigated this threat by including more than 12,000 apps that vary in terms of size, created year, category, published market, and popularity metrics on GitHub. However, findings in this study may not be applicable to trivial apps or commercial apps developed privately. Furthermore, the respondents of our survey may not be representative of the entire developer community of the subject apps, or the global community of Android app developers. We tried to reduce this threat by collecting the responses of 148 developers from 45 countries with various years of professional experience. The number of responses to our survey is also comparable to other similar studies of mobile developers [33, 37, 42].

Internal validity. We proposed certain heuristics to automatically identify non-trivial apps. While we may have missed some complex and published apps, e.g., apps with single Activity and multiple fragments, we believe that the findings in this paper are still useful for practitioners and researchers regarding test automation. Moreover, we automatically determine the number of test cases contained in a repository based on the assumption that the test cases are written in JUnit-based testing frameworks. While JUnit-based testing frameworks overwhelmingly dominate Android app testing (e.g., 97% to 100% according to a prior study [13]), it is possible that some test cases built on top of other types of frameworks are not included in our study. To mitigate this threat, we manually verified a small set of projects in our dataset and did not find any missed test cases. As a result, we argue that such cases are rare and would not significantly impact our conclusions.

8 CONCLUSION

This paper provides a holistic view regarding how and why test automation is practically adopted in open-source Android apps. With the analysis of more than 12,000 non-trivial apps on GitHub and a survey of 148 developers of these apps, we investigated (1) the prevalence of test automation in mobile app development projects; (2) working habits of mobile app developers; and (3) the correlation between the adoption of test automation and the popularity of projects. Among others, we found that: (1) only 8% of the non-trivial apps contain automated tests; (2) developers tend to follow the same test automation practices across apps; and (3) popular projects are more likely to adopt test automation practices. We believe the findings in this paper shed light on the current practices and future research directions pertaining to test automation for mobile app development. In our future work, we plan to incorporate additional open-source projects, such as those hosted on Bitbucket, and investigate new research questions, e.g., questions related to the interplay between test automation techniques and continuous integration practices.

ACKNOWLEDGMENT

This work was supported in part by award number 1823262 from the National Science Foundation.

REFERENCES

[1] 360logica. 2020. *A sneak peek into test framework and testing pyramid.* Retrieved January 19, 2020 from https://www.360logica.com/blog/sneak-peek-test-framework-test-pyramid-testing-pyramid/

[2] Christoffer Quist Adamsen, Gianluca Mezzetti, and Anders Møller. 2015. Systematic Execution of Android Test Suites in Adverse Conditions. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis* (Baltimore, MD, USA) *(ISSTA 2015)*. Association for Computing Machinery, New York, NY, USA, 83–93. https://doi.org/10.1145/2771783.2771786

[3] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 468–471. https://doi.org/10.1145/2901739.2903508

[4] AndroZoo. 2020. *AndroZoo Markets.* Retrieved January 19, 2020 from https://androzoo.uni.lu/markets

[5] Larissa Azevedo, Altino Dantas, and Celso G. Camilo-Junior. 2018. DroidBugs: An Android Benchmark for Automatic Program Repair. *CoRR* abs/1809.07353 (2018). arXiv:1809.07353 http://arxiv.org/abs/1809.07353

[6] F. Behrang and A. Orso. 2019. Test Migration Between Mobile Apps with Similar Functionality. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 54–65.

[7] Moritz Beller, Georgios Gousios, Annibale Panichella, and Andy Zaidman. 2015. When, How, and Why Developers (Do Not) Test in Their IDEs. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 179–190. https://doi.org/10.1145/2786805.2786843

[8] N. Chang, L. Wang, Y. Pei, S. K. Mondal, and X. Li. 2018. Change-Based Test Script Maintenance for Android Apps. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 215–225.

[9] W. Choi, K. Sen, G. Necul, and W. Wang. 2018. DetReduce: Minimizing Android GUI Test Suites for Regression Testing. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 445–455.

[10] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. 2015. Automated Test Input Generation for Android: Are We There Yet? (E). In *Proceedings*

of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (ASE '15). IEEE Computer Society, Washington, DC, USA, 429–440. https://doi.org/10.1109/ASE.2015.89

[11] Mike Cohn. 2010. *Succeeding with agile: software development using Scrum*. Pearson Education.

[12] Riccardo Coppola, Maurizio Morisio, Marco Torchiano, and Luca Ardito. 2019. Scripted GUI testing of Android open-source apps: evolution of test code and fragility causes. *Empirical Software Engineering* 24, 5 (01 Oct 2019), 3205–3248. https://doi.org/10.1007/s10664-019-09722-9

[13] Luis Cruz, Rui Abreu, and David Lo. 2019. To the attention of mobile software developers: guess what, test your app! *Empirical Software Engineering* 24, 4 (01 Aug 2019), 2438–2468. https://doi.org/10.1007/s10664-019-09701-0

[14] The Dataset. 2020. *The Dataset*. Retrieved January 19, 2020 from https://github.com/seal-hub/ASE20Empirical

[15] Lin Deng, Jeff Offutt, Paul Ammann, and Nariman Mirzaei. 2017. Mutation operators for testing Android apps. *Information and Software Technology* 81 (2017), 154–168.

[16] Paul M Duvall, Steve Matyas, and Andrew Glover. 2007. *Continuous integration: improving software quality and reducing risk*. Pearson Education.

[17] F-Droid. 2020. *F-Droid - Free and Open Source Android App Repository*. Retrieved January 19, 2020 from https://f-droid.org

[18] Martin Fowler. 2020. *Test Pyramid*. Retrieved January 19, 2020 from https://martinfowler.com/bliki/TestPyramid.html

[19] G. Fraser and A. Arcuri. 2012. Sound empirical evidence in software testing. In *2012 34th International Conference on Software Engineering (ICSE)*. 178–188. https://doi.org/10.1109/ICSE.2012.6227195

[20] GHTorrent. 2020. *GHTorrent Downloads*. Retrieved January 19, 2020 from https://ghtorrent.org/downloads.html

[21] Google. 2020. *Advanced Android in Kotlin: Testing Basics*. Retrieved January 19, 2020 from https://codelabs.developers.google.com/codelabs/advanced-android-kotlin-training-testing-basics/index.html#4

[22] Google. 2020. *Android Studio release notes*. Retrieved January 19, 2020 from https://developer.android.com/studio/releases

[23] Google. 2020. *App Manifest Overview*. Retrieved January 19, 2020 from https://developer.android.com/guide/topics/manifest/manifest-intro

[24] Google. 2020. *Application Fundamentals*. Retrieved January 19, 2020 from https://developer.android.com/guide/components/fundamentals.html

[25] Google. 2020. *Create UI tests with Espresso Test Recorder*. Retrieved January 19, 2020 from https://developer.android.com/studio/test/espresso-test-recorder

[26] Google. 2020. *Espresso*. Retrieved January 19, 2020 from https://developer.android.com/training/testing/espresso

[27] Google. 2020. *Fundamentals of Testing*. Retrieved January 19, 2020 from https://developer.android.com/training/testing/fundamentals

[28] Georgios Gousios. 2013. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (San Francisco, CA, USA) *(MSR '13)*. IEEE Press, Piscataway, NJ, USA, 233–236. http://dl.acm.org/citation.cfm?id=2487085.2487132

[29] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore) *(ASE 2016)*. Association for Computing Machinery, New York, NY, USA, 426–437. https://doi.org/10.1145/2970276.2970358

[30] Reyhaneh Jabbarvand and Sam Malek. 2017. MuDroid: An Energy-Aware Mutation Testing Framework for Android. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 208–219. https://doi.org/10.1145/3106237.3106244

[31] Reyhaneh Jabbarvand, Alireza Sadeghi, Hamid Bagheri, and Sam Malek. 2016. Energy-Aware Test-Suite Minimization for Android Apps. In *Proceedings of the 25th International Symposium on Software Testing and Analysis* (Saarbrücken, Germany) *(ISSTA 2016)*. Association for Computing Machinery, New York, NY, USA, 425–436. https://doi.org/10.1145/2931037.2931067

[32] B. Jiang, Y. Wu, Y. Zhang, Z. Zhang, and W. K. Chan. 2018. ReTestDroid: Towards Safer Regression Test Selection for Android Application. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 01. 235–244.

[33] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. 2013. Real Challenges in Mobile App Development. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, Maryland, USA, October 10-11, 2013*. 15–24. https://doi.org/10.1109/ESEM.2013.9

[34] JUnit. 2020. *JUnit*. Retrieved January 19, 2020 from https://junit.org

[35] Kostis Kapelonis. 2020. *Software Testing Anti-patterns*. Retrieved January 19, 2020 from http://blog.codepipes.com/testing/software-testing-antipatterns.html

[36] P. S. Kochhar, T. F. Bissyandé, D. Lo, and L. Jiang. 2013. An Empirical Study of Adoption of Software Testing in Open Source Projects. In *2013 13th International Conference on Quality Software*. 103–112. https://doi.org/10.1109/QSIC.2013.57

[37] Pavneet Singh Kochhar, Ferdian Thung, Nachiappan Nagappan, Thomas Zimmermann, and David Lo. 2015. Understanding the Test Automation Culture of App Developers. In *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*. IEEE, 1–10.

[38] Erwin Kreyszig. 2009. *Advanced Engineering Mathematics, 10th Eddition*. Wiley.

[39] X. Li, N. Chang, Y. Wang, H. Huang, Y. Pei, L. Wang, and X. Li. 2017. ATOM: Automatic Maintenance of GUI Test Scripts for Evolving Mobile Applications. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 161–171. https://doi.org/10.1109/ICST.2017.22

[40] J. Lin, R. Jabbarvand, and S. Malek. 2019. Test Transfer Across Mobile Apps Through Semantic Mapping. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 42–53.

[41] Mario Linares-Vásquez, Gabriele Bavota, Michele Tufano, Kevin Moran, Massimiliano Di Penta, Christopher Vendome, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. 2017. Enabling Mutation Testing for Android Apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 233–244. https://doi.org/10.1145/3106237.3106275

[42] M. Linares-Vásquez, C. Bernal-Cardenas, K. Moran, and D. Poshyvanyk. 2017. How do Developers Test Android Applications?. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 613–622. https://doi.org/10.1109/ICSME.2017.47

[43] M. Linares-Vásquez, K. Moran, and D. Poshyvanyk. 2017. Continuous, Evolutionary and Large-Scale: A New Perspective for Automated Mobile App Testing. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 399–410. https://doi.org/10.1109/ICSME.2017.27

[44] H. B. Mann and D. R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *Ann. Math. Statist.* 18, 1 (03 1947), 50–60. https://doi.org/10.1214/aoms/1177730491

[45] Atif M. Memon and Myra B. Cohen. 2013. Automated Testing of GUI Applications: Models, Tools, and Controlling Flakiness. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA) *(ICSE '13)*. IEEE Press, 1479–1480.

[46] Mockito. 2020. *Most popular mocking framework for Java*. Retrieved January 19, 2020 from https://github.com/mockito/mockito

[47] Duncan Nisbet. 2020. *Test Automation Basics – Levels, Pyramids & Quadrants*. Retrieved January 19, 2020 from http://www.duncannisbet.co.uk/test-automation-basics-levels-pyramids-quadrants

[48] Chairat Onyaem. 2020. *Separate Unit, Integration, and Functional Tests for Continuous Delivery*. Retrieved January 19, 2020 from https://medium.com/pacroy/separate-unit-integration-and-functional-tests-for-continuous-delivery-f4dc240d8f2f

[49] M. Pan, T. Xu, Y. Pei, Z. Li, T. Zhang, and X. Li. 2019. GUI-Guided Repair of Mobile Test Scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 326–327.

[50] Fabiano Pecorelli, Gemma Catolino, Filomena Ferrucci, Andrea De Lucia, and Fabio Palomba. 2020. Testing of Mobile Applications in the Wild: A Large-Scale Empirical Study on Android Apps. In *28th IEEE/ACM International Conference on Program Comprehension (ICPC)*.

[51] Qualtrics. 2020. *Qualtrics Survey Software*. Retrieved January 19, 2020 from https://www.qualtrics.com/

[52] Robolectric. 2020. *Test-drive your Android code*. Retrieved January 19, 2020 from http://robolectric.org/

[53] Shayna Rusticus and Chris Lovato. 2014. Impact of Sample Size and Variability on the Power and Type I Error Rates of Equivalence Tests: A Simulation Study. *Practical Assessment, Research and Evaluation* 19 (01 2014).

[54] S. S. SHAPIRO and M. B. WILK. 1965. An analysis of variance test for normality (complete samples)†. *Biometrika* 52, 3-4 (12 1965), 591–611. https://doi.org/10.1093/biomet/52.3-4.591 arXiv:https://academic.oup.com/biomet/article-pdf/52/3-4/591/962907/52-3-4-591.pdf

[55] S. H. Tan, Z. Dong, X. Gao, and A. Roychoudhury. 2018. Repairing Crashes in Android Apps. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 187–198.

[56] Wenyu Wang, Dengfeng Li, Wei Yang, Yurui Cao, Zhenwen Zhang, Yuetang Deng, and Tao Xie. 2018. An Empirical Study of Android Test Generation Tools in Industrial Cases. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 738–748. https://doi.org/10.1145/3238147.3240465

[57] Xia Zeng, Dengfeng Li, Wujie Zheng, Fan Xia, Yuetang Deng, Wing Lam, Wei Yang, and Tao Xie. 2016. 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* (Seattle, WA, USA) *(FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 987–992. https://doi.org/10.1145/2950290.2983958