Nat. Volatiles & Essent. Oils, 2021; 8(6): 6260-6270

NVEO
Natural Volatiles &
Essential Oils

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

DHANYA.V.J[1] , Mr. M. Imayavaramban[2]

[1]Assistant Professor, Department of Information Technology, Dhanalakshmi Srinivasan College of Engineering and Technology.

[2]Assistant Professor, Dhanalakshmi Srinivasan College of Engineering and Technology

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.

KEYWORDS: Empirical Study, Automated Testing, Mobile Apps, Android.

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. 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. 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 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

public class Example Instrumented Test {

@Test

public void use App Context() {

Context app Context = Instrumentation Registry .get Instrumentation () .get Target Context (); assert Equals("com. example", app Context. Get Package Name());

public class Example Unit Test {

@Test

public void addition_ is Correct() {

assert Equals(4, 2 + 2);

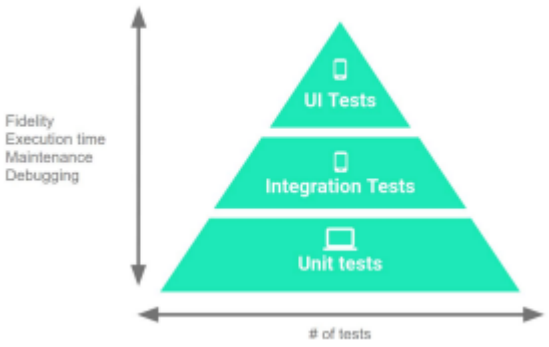
Figure 1: Example Test Classes Generated by Android Studio.

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. 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

2.1Unit 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 endto-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 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

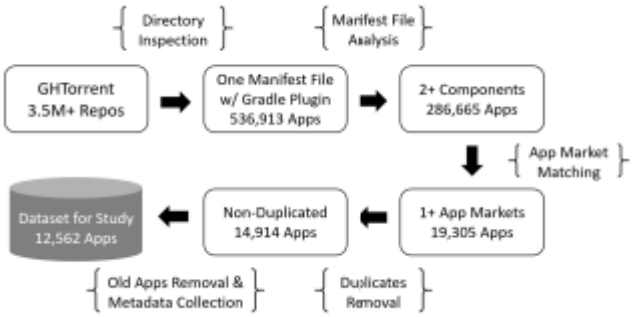
2.2The Testing Pyramid Practice

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 authors2, 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



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 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

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

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.

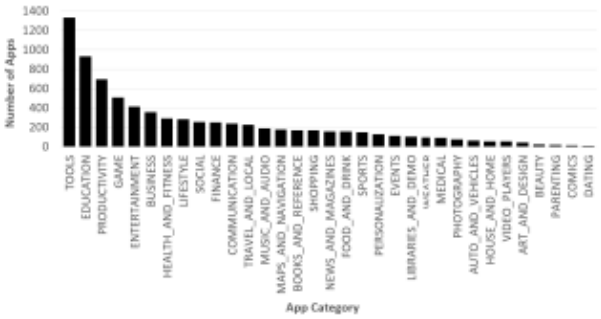


Figure 4: Distribution of the Google Play Apps by Category

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.

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%

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

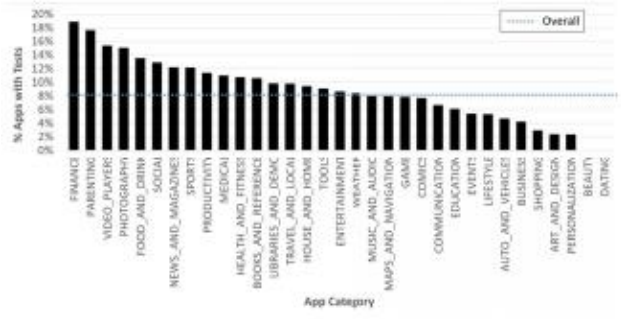


Figure 5: Prevalence of Test Automation of the Google Play Apps by Category

5.2Impact 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, issues5, 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.

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

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 nontrivial 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

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-testframework-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.2015.8