Understanding the Test Automation Culture of App Developers

Pavneet Singh Kochhar[1], Ferdian Thung[1], Nachiappan Nagappan[2], Thomas Zimmermann[2], and David Lo[1]
[1]Singapore Management University
[2]Microsoft Research
{kochharps.2012,ferdiant.2013,davidlo}@smu.edu.sg,{nachin,tzimmer}@microsoft.com

***Abstract*—Smartphone applications (apps) have gained popularity recently. Millions of smartphone applications (apps) are available on different app stores which gives users plethora of options to choose from, however, it also raises concern if these apps are adequately tested before they are released for public use. In this study, we want to understand the test automation culture prevalent among app developers. Specifically, we want to examine the current state of testing of apps, the tools that are commonly used by app developers, and the problems faced by them. To get an insight on the test automation culture, we conduct two different studies. In the first study, we analyse over 600 Android apps collected from F-Droid, one of the largest repositories containing information about open-source Android apps. We check for the presence of test cases and calculate code coverage to measure the adequacy of testing in these apps. We also survey developers who have hosted their applications on GitHub to understand the testing practices followed by them. We ask developers about the tools that they use and "pain points" that they face while testing Android apps. For the second study, based on the responses from Android developers, we improve our survey questions and resend it to Windows app developers within Microsoft. We conclude that many Android apps are poorly tested - only about 14% of the apps contain test cases and only about 9% of the apps that have executable test cases have coverage above 40%. Also, we find that Android app developers use automated testing tools such as JUnit, Monkeyrunner, Robotium, and Robolectric, however, they often prefer to test their apps manually, whereas Windows app developers prefer to use in-house tools such as Visual Studio and Microsoft Test Manager. Both Android and Windows app developers face many challenges such as time constraints, compatibility issues, lack of exposure, cumbersome tools, etc. We give suggestions to improve the test automation culture in the growing app community.**

Keywords*—*Test Automation Culture, App Developers, Android, Microsoft

I. Introduction

Smartphones have become pervasive and platforms such as Android and iOS have gained tremendous popularity recently. According to a Gartner study, worldwide sales of smartphones to end users increased by 42.3% in 2013 as compared to the previous year and Android had 78.4% of the market share of the smartphone sales in 2013 [1]. Furthermore, easy availability of app construction frameworks and dissemination through online app stores such as Google Play[1] and Apple App store[2] have attracted a large number of developers and organizations to develop and market their apps. However, low barriers to development does not ensure that apps are error free. These error-prone apps can significantly impact user experience and may cause harm to the reputation of the developers or the organizations. Therefore, it is important to adequately test these apps before releasing them to the market. A reliable app with few or no bugs is likely to have a higher chance of being well-received by the large user base of these smartphones than the unreliable ones.

Although mobile apps use common technologies such as Java, they significantly differ from web-based and desktop-based applications. An app receives a variety of inputs from users and its environment which makes it difficult to write effective test cases. Thus, many recent studies propose new testing tools that are specifically designed for mobile applications [2], [3], [4]. Despite the growing interest in the software testing and reliability research community to build tools that can automate and improve testing of mobile apps, there has been no study that investigates how developers test these applications in practice. This study is needed to understand the "pain points" that these developers face which can be used to motivate future research that addresses concerns that matters to mobile app developers.

To address this need, we conduct an empirical study which is divided into two parts. In the first one, we analyze over 600 open-source Android apps to examine the current state of testing in the Android development community. Our dataset includes small apps to large and popular apps such as K-9 Mail[3], FrostWire - Downloader/Player[4], OsmAnd Maps & Navigation[5] and OI File Manager[6], which have more than 1,000,000 installs. We use a heuristic to automatically identify the presence of test files in these applications. For applications with test files, we try to build them and run the test cases. We also measure the number of lines and blocks covered by these test cases to investigate the adequacy of testing. Furthermore, we want to understand the common testing tools used by Android developers as well as the challenges faced by them when they test their apps. To do so, we survey over 3,900 Android developers who have hosted their applications on GitHub and collate their responses We received a total of 83 responses.

For the second part, we replicate the above survey study

[1]https://play.google.com/store?hl=en
[2]https://itunes.apple.com/us/genre/ios/id36?mt=8
[3]https://play.google.com/store/apps/details?id=com.fsck.k9
[4]https://play.google.com/store/apps/details?id=com.frostwire.android
[5]https://play.google.com/store/apps/details?id=net.osmand
[6]https://play.google.com/store/apps/details?id=org.openintents.filemanager

978-1-4799-7125-1/15/$31.00 ©2015 IEEE

with app developers from Microsoft. We improve our survey based on the responses of the first study and ask Microsoft developers about the type of testing they do, tools they use and the challenges they face during testing. We survey over 600 developers and received 127 responses.

The main contributions of this study are as follows:

- We are the first to conduct a study to understand the test automation culture in Android and Microsoft app development community.
- We analyze the extent to which Android apps are adequately tested by checking the presence of test cases and calculating coverage.
- We survey many Android and Microsoft app developers and collect their responses to understand testing tools used by them and challenges faced by these developers in testing their applications.

The structure of the remainder of this paper is as follows. In Sections II, we explain our study design which includes the research questions, data collection method, and basic statistics of our collected data. We present the results of current state of Android testing in Section III. We discuss the results of the survey of Android developers in Section IV and survey results of Microsoft developers in Section V. We discuss additional interesting points and describe threats to validity in Section VI and VII, respectively. Related work is discussed in Section VIII. We conclude and mention future work in Section IX.

II. Study Design

In this section, we first present the research questions we investigate in this study. Then, we present how we collected our dataset. Next, we present some basic statistics of over 600 apps in our dataset.

A. Research Questions

We investigate a set of research questions to understand the testing practices commonly followed by Android developers. First, we want to understand the current state of testing in Android apps by analysing whether developers write test cases for their apps. This leads us to our first research question:

RQ1: Are Android applications well tested?

Software testing is an important part of software development processes as well tested applications are often of high quality and experience few bugs. However, testing is not an easy process as it requires developers to have good understanding of their applications and on how users will use them. Developers also need to be aware of the common testing tools and frameworks available for mobile apps. After analysing several apps, we want to know whether Android developers use any automated testing tools or manually test their apps. If developers use any tools, we want to know what are the common tools used by developers for testing their apps and the challenges faced by them during testing. As such, our second research question is:

RQ2: Do Android developers use automated testing tools to test their applications? What are the tools commonly used by Android developers and the challenges faced by them while testing their applications?

Similar to the second question, here we analyse the tools used by Microsoft developers and the challenges faced by them. Our third research question is:

RQ3: Do Microsoft developers use automated testing tools to test their applications? What are the tools commonly used by Microsoft developers and the challenges faced by them while testing their applications?

Our study is divided into three parts. In the first part, we analyse over 600 apps to understand the current state of testing of Android applications. After analyzing the apps, we want to find out the tools used by developers while testing their apps and the challenges faced by them during the testing procedure. To do this, in the second part, we collect email addresses of Android developers from over 600 apps and ask them questions using a structured survey. Furthermore, in order to understand if Microsoft's Windows app developers face similar issues as their Android counterparts, in the third part of our study, we survey over 600 Windows app developers within Microsoft. We present the above results in Section III, IV and V.

B. Data Collection

We collect URLs of all the applications stored on F-Droid[7] repository and select apps which are hosted on GitHub. In total, we have 627 apps in our dataset.

Test Cases & Coverage: For each app, we examine the presence of test cases by checking for the existence of files which contain the word "Test". We observe that many test files have the word "Test" either in the beginning of their names, e.g., TestUtil.java, or at the end of their names, e.g., AccentTest.java. For projects containing test files, we manually investigate them to build them and run test cases. Some projects fail to compile due to dependencies on external libraries. We try to resolve these dependencies issue by downloading libraries. However, many projects still fail to compile. For projects which compile successfully, we run the test cases present in the project repository and calculate code coverage using Emma code coverage tool[8].

Survey:

First Study - For each of the 627 apps, we collect e-mail addresses of all developers that developed these apps. In total, we sent out e-mails to 3,905 distinct e-mail addresses and ask developers questions about testing tools used by them and challenges that they face while testing their applications. Many of these developers work on both open source and commercial projects. We received a total of 83 responses (response rate of 2.13%). The unit of analysis is individual developer.

Second Study - Based on the responses from the first study, we improve our survey questions and resend the survey to Windows app developers in Microsoft. We sent out e-mails to 678 developers and received a total of 127 responses (response rate of 18.73%). The unit of analysis is individual developer.

[7] https://f-droid.org/

[8] http://emma.sourceforge.net/

For the first study, we use a structured survey which consists of several open-ended questions. The following are the questions that we ask as part of our survey:

1) How do you test your app code?

Free form text

2) Do you use any test automation tools (e.g., monkeyrunner, robotium, robolectric, etc)? If so, what tools do you use and why do you use them (e.g., for generating test cases, for managing test suites etc.)

Free form text

3) What are the challenges you face during testing either manually or using automated tools (e.g., lack of documentation, limited support, unclear benefits, etc.)?

Free form text

For the second study, we also use a structured survey. However, we add additional questions, and provide multiple choices to better understand app developers testing behaviors. The questions that we ask include:

1) How do you test your app code?

Checkbox options: Manually, use automated testing tools, don't test, other

2) If you test your apps, what type of testing do you do on your apps?

Checkbox options: Unit testing, integration testing, system testing, functional testing, regression testing, acceptance testing, load testing, performance testing, beta testing, other

3) If you use automated testing tools for your apps, what are the names of the testing tools?

Free form text

4) If you use automated testing tools, why do you use testing tools?

Checkbox options: Generating test cases, executing test cases, managing test suites, creating and evaluating test execution results, analysing code coverage, finding potential bugs, reporting bugs, performing load testing, other

5) Do you face the following challenges during testing either manually or using automated testing tools and if you do how serious are they?

Challenges: Time constraints, compatibility issues, lack of exposure to tools, emphasis on development rather than testing, lack of support from employer/organization, unclear benefits of tools, poor documentation, lack of experience, steep learning curve.
Seriousness levels: Very serious, serious, insignificant, do not face, no opinion

6) Given the availability of testing tools for app development, in your opinion what are the top 2 things you look for/need/would like to see?

Free form text

C. Basic Statistics

We now present some statistics describing the data collected for our study in terms of number of test cases, lines of code and number of developers.

a) Test Cases: Table I shows the number of apps with and without test cases. We find that 538 (85.81%) apps do not have any test cases, whereas 89 (14.19%) apps have at least one test case. This shows that a large number of Android apps lack test cases.

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

Categories	# of Apps	% of Apps
Without Test Cases	538	85.81%
With Test Cases	89	14.19%

b) Lines of Code (LOC): We count the lines of code for all the apps in our dataset. Figure 1 shows the distribution of LOC. We can observe that 146 apps have sizes between 1 LOC to 1,000 LOC, whereas 234 apps have sizes between 1,000 LOC to 5,000 LOC. Furthermore, 128 apps have sizes between 10,000 LOC to 50,000 LOC and 35 apps are larger than 50,000 LOC. The largest project in our dataset is FrostWire - Downloader/Player[4], which is a native BitTorrent & Cloud file downloader with 1,070,130 LOC.

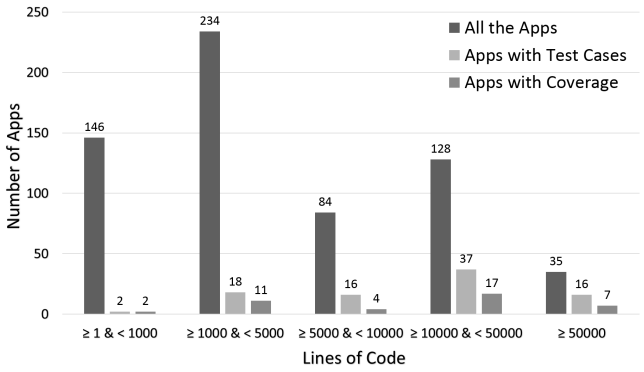

Figure 1: Distribution of Apps in Terms of Total Number of Lines of Code

c) Number of Developers: We want to analyse number of developers involved in the development of an app. We use the information from *git logs* and collect unique e-mail addresses to count the number of developers. Figure 2 shows the distribution of the number of developers who worked on different apps in our dataset. We can observe that a large number of apps (242 apps) are developed by a single developer. Also, 217 apps have more than 1 but less than 5 developers, whereas 75 apps have greater than or equal to 5 developers but less than 10 developers.

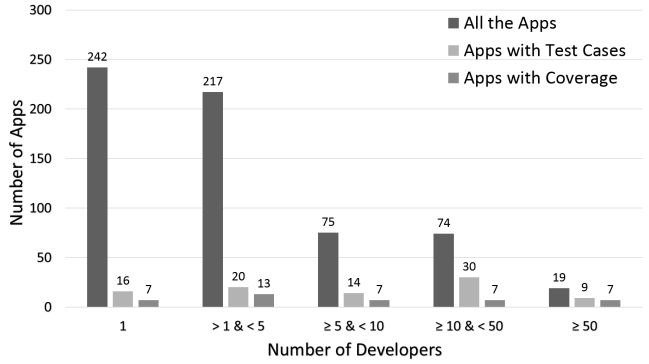

Figure 2: Distribution of Apps in Terms of Number of Developers

III. Current State of Testing in Android Applications

In this section, we report the results of testing 89 out of the 627 applications that contain test suites (i.e., test files).

Figure 3 shows the distribution of test suites for the 89 apps. We find that 19 apps have only 1 test suite, whereas 11 apps have more than 25 test suites. Furthermore, 23 apps have more than equal to 5 but less than 10 test suites. We can observe that most of the apps have very few test suites.

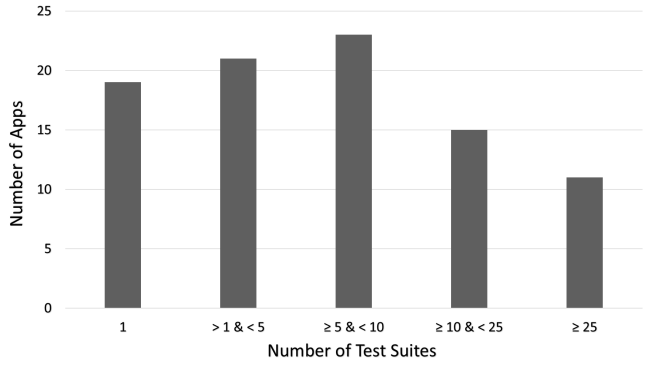

Figure 3: Distribution of Apps in Terms of Total Number of Test Suites

We use coverage as a measure for the adequacy of testing. We want to analyse if the projects which have test cases are thoroughly tested or not. We use two measures of coverage:

1) *Line Coverage* measures the proportion of lines executed during testing.
2) *Block Coverage* measures the proportion of code blocks covered, where each block is a sequence of statements without any jumps or jump targets which is executed as one atomic unit.

Out of the 89 apps, we have 41 apps which compile successfully and we run test cases for these apps. We then calculate code coverage for these apps.

Figures 4 and 5 show the line and block coverage of the 41 projects, respectively. We observe that 37 projects have line coverage of less than 40%, whereas 36 projects have block coverage of less than 40%. The mean and median value of line coverage is 16.03% and 9.33%, whereas the corresponding values for block coverage are 17.22% and 10.65%, respectively. The results show that most of the apps have low coverage, which shows that apps are not adequately tested.

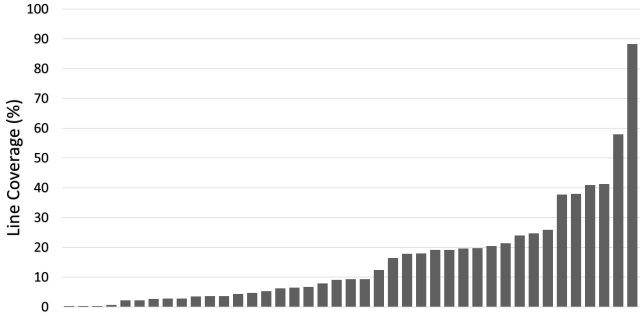

Figure 4: Line Coverage (Ascending Order)

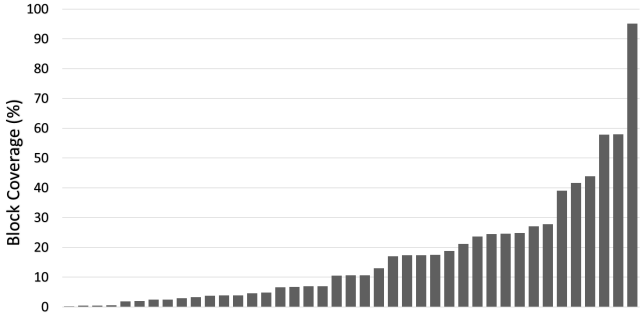

Figure 5: Block Coverage (Ascending Order)

Only 14.19% (89 out of 627) of the apps contain test cases and 9.75% (4 out of 41) of the apps that have executable test cases (i.e., 0.64% of all 627 apps) have line coverage above 40%.

IV. Survey of Android Developers

In this section, we present our results of the survey conducted on Android developers.

A. Android Automated Testing Tools Usage

This section reports our findings on the automated testing tools that are used by Android developers. A large number of automated testing tools are available to test Android applications. Table II shows the number of respondents who use a particular tool. Some developers use more than one tool simultaneously. We briefly explain some of tools commonly used by Android developers.

a) JUnit[9] - A popular unit testing framework for Java. Since Android applications are written in Java, it can be directly used to test the parts of the code that do not call the Android API.

[9]http://junit.org/

b) MonkeyRunner[10] - Monkeyrunner tool provides an API to write programs to control an Android device or emulator from outside of the Android code.

c) Robotium[11] - Robotium is a test automation framework, which allows developers to write black-box UI tests for Android apps. Robotium enables developers to write function, system and user acceptance test spanning multiple Android activities.

d) Robolectric[12] - It is a unit test framework for Android, which allows developers to execute test cases in Java Virtual Machine (JVM), rather than running on a mobile device or emulator.

We can observe that JUnit is the most commonly used testing framework. This could be due to the fact that JUnit is one of the mature frameworks and have been used extensively in the industry.

Table II: Automated Testing Tools Usage

Tools	Number of Respondents
JUnit	18
MonkeyRunner	8
Robotium	7
Robolectric	6
Android unit testing framework	6
Monkey	1
Espresso	1
TestNG	1
Other tools	1
No tool	35

Some developers often leverage automated testing tools to test their apps based on the requirements of the project and the functionalities provided by the tool. One of the developers said:

"RoboElectric. I use this pretty heavily for unit testing, but the scope of tests is rather limited at the moment. I run my suite of tests against my data model before checking in code. I find this to be the most mature framework at the moment, but the amount of supported features is still a bit limited as its a community driven project. There have been a number of areas (e.g. the PreferenceActivity and Preferencefragment classes) that are a bit more limited in scope.

MonkeyRunner. I run tests using this generally the night before uploading an app. My UI tends to be fairly stable at this point, so it's not that helpful, but it usually catches any serious functionality that I might have broken.

Robotium. I don't use this at the moment, but I intend to in the future. One of my limitations here right now is that there is no free "recorder" software that I'm aware of at the moment (a recorder would track a series of keystrokes for testing purposes, so I could repeat app tests rather than having to do this by hand). I need to research this a bit further."

Developers have varying opinions over usage of these tools. Some of them regularly use such tools (*"I use Jenkins as a tool for Continuous integration, for testing I use monkeyrunner, roboelectric, it's easy to integrate it with Jenkins. I also use uiautomator for testing the UI interface."*), whereas others prefer to use older frameworks like JUnit (*"I am testing my application logic (ie. service layer) with JUnit and/or TestNG as it is not dependent on Android framework. I usually do not use automation tools for GUI itself, in fact my experience with GUI testing frameworks is somewhat ... unbalanced."*). On the other hand, some developers prefer to write their own scripts to test their applications (*"Honestly I prefer to code instead of spent my time figuring out how complex debugging tools works."*).

Our survey shows that some developers are aware of the new tools coming into the market and they express their intentions of using those tools for future projects (*"...However, I'm interested in Espresso testing tool. It can write clean test code, and runs faster than Robotium. I'll try to use it if I make a next new app."*). Furthermore, some developers who are not satisfied with some tools, plan to use new tools which provide similar functionalities (*"Robotium has been giving us a little bit of trouble by having tests flake, so I'm going to work on migrating those to espresso in the near future, as I've heard nothing but good things about it so far."*).

Several developers prefer to test applications manually because their applications are small. Developers do not find it useful to put in effort and learn something new, when the app can be tested manually in a short amount of time. One of the developers said *"because i only develop some small app. therefore, i don't need any test tool. i just write code, run, debug until it's run correct."*, while another developer mentioned that *"Most of the projects I've worked so far are simple and for short-term. So I was just fine with manual testing."*

Google Play makes it easier for users to search and install apps. Therefore, some developers do not perform much testing, rather they depend on users who download their applications to report problems. One of the respondents mentioned "*... if someone comes across a bug, they can submit on the issue tracker and I will try to fix it."*.

Our survey results also show that a large number of developers prefer testing their apps manually rather than using any automated testing tool. From our analysis, we find that such cases occur due to various reasons. The app to be tested could be simple or it could be difficult to find a tool which can meet the testing requirements of the developers. One developer stated *"I have used robotium for some UI testing, however I haven't found it particularly useful. The things that robotium can test are very easily verified manually and there are a lot of things it can't test AFAIK (layouting, aesthetics, etc)"*.

B. Challenges faced by Android Developers while Testing

This section discusses the challenges faced by the Android developers while testing their apps either manually or when using automated testing tools. Table III shows some of the common challenges confronted by the developers. Some developers that we survey do not mention any challenges and some mention more than one challenges. We describe each of the challenges in detail and quote responses from the developers.

[10]http://developer.android.com/tools/help/monkeyrunner_concepts.html
[11]https://code.google.com/p/robotium/
[12]http://robolectric.org/

Table III: Challenges Faced by Developers while Testing

Challenges	Number of Respondents
Time constraints	20
Compatibility issues	16
Lack of exposure	11
Tool is cumbersome	9
Emphasis on development	6
Lack of organization support	5
Unclear benefits	4
Poor documentation	4
Lack of experience	4
Steep learning curve	2

Time is one of the biggest factors which hinders testing. Most of the developers want to release their applications as soon as possible before someone else develops a similar application. In such cases, developers do not want to invest time in testing but rather develop the application quickly. One developer commented *"...I work as a freelance developer. So often there are time constraints to finish the project. Designing and implementing test cases takes some extra time, which makes it difficult to finish the project in time."*.

Automated testing tools are generic in nature and are developed to suit many applications. However, several apps contain custom functionalities which make it difficult for developers to use automated testing tools. A number of developers were of the opinion that some parts of the code are hard to test using automated testing tools, which forces them to resort to manual testing. Also, automated testing tools are designed to work on specific technology. When developers use different technology, these automated testing tools no longer work well. One developer lamented *"I tried robolectric, but ran into several issues, that were probably also related to the fact that I am using Scala on Android."*

Some developers are not aware of automated testing tools available in the market. One developer admitted *"I have not been aware of them ."* Furthermore, lack of discussion about the importance of automated tests worsens the problem. One developer commented *"... but it's not a common thing to 'do' so there isn't a lot of discussion around it."*

Usability of a tool is one of the key characteristics of it being used by a large number of developers. A tool which is easier to use will appeal to more developers as compared to a tool whose usage is complex. Several developers responded that they tried to use a particular tool but due to its cumbersome usage, they discarded the tool. One respondent mentioned *"I think Monkey runner is kinda cumbersome, and breaks easily when changing layout options."*. Yet another commented *"There is some coordination problems with Robotium which can be painful to workaround sometimes"*.

Functionalities of an application are one of the key factors which decide whether the app is useful or not. If an app provides functionalities which suits the need of a large number of users, the app will be popular. For example, one of the apps in our dataset, i.e., Open Explorer[13], has between 100,000 - 500,000 installs. Therefore, developers are often more focussed on adding new features of an app. Thus, they devote most of their time towards development rather than testing. One developer commented "*... I spend most of the time I dedicate to this project to implementing features"*.

With the increasing size of an application, it becomes imperative to adequately test the application. However, larger application means significant investment in terms of cost involved in testing it, which can act as a hindrance for many developers and organizations. If organizations are not able to provide resources to the developers, it would be difficult for developers to do much testing or invest time to learn automated testing tools. One developer commented *"The advice I was given was ... not bother with trying to use the Android testing tools/frameworks"*. In several cases, clients are not willing to pay extra for doing automated testing. One developer commented "*...few clients were ready to pay more for automatic tests : they did manual tests themselves. We never used automatic tests for this reason.*"

Testing tools can play an important role during software development life cycle as they assist developers in writing and running test scripts and creating test results automatically as compared to manually testing the application. However, it is important to clarify how the tools would be beneficial to a developer or organization who wants to use it. Unclear benefits would resist developers from venturing into the arena of automated testing. One developer stated *"The pain points for me would be assessing what automated test tools are available, assessing their applicability to my applications and writing comprehensive test scripts or whatever the tools require. That is probably more effort than what went into writing the applications in the first place.*"

Learning new tools and techniques requires developers to read documentation and try out examples before they can apply the tool to their app. A good documentation makes it easier for novice as well as experienced developers to grasp the functionality of a tool and get started quickly on using the tool to test their apps, whereas a poor documentation will act as a hindrance for developers to adopt the tool. Therefore, a good and easy to understand documentation is a must for a new tool. Four developers in our survey mentioned that lack of documentation is one of the challenges. One of them mentioned *"Testing is documented there, but not very well and there should be far more information (for instance, how to test interaction with data providers - there's only a chapter how to test OWN data providers, but that's not what we need).*"

Developers who have prior experience of using automated testing tools are more likely to use new tools. Our survey responses show that developers with no experience of using automated testing tools are reluctant to use Android test automation tools. One developer mentioned *"For that, I haven't used any kind of tool for testing pourposes. The reason? Well, for starters, I have no experience with testing tools for any language/platform, so I don't really know how to tackle that..."*.

Some of the developers perceive that using testing tools involve steep learning curve. One developer mentioned *"I fear it would represent a strong learning curve."*. Another developer commented that *"I know what automated testing is how to write a test case or prepare a test suite. But I don't know how can I use automated testing effectively. Learning this will take considerable amount of time and effort.*"

[13]https://play.google.com/store/apps/details?id=org.brandroid.openmanager

Most of the developers prefer using well-known testing tools such as JUnit. Others experiment with new tools specifically designed for Android such as Robolectric, Robotium and Monkeyrunner. However, a large number of developers prefer to stay away from using tools and perform testing manually. Our survey shows that Android developers face challenges in adopting automated testing tools such as time constraints, compatibility issues, lack of exposure, cumbersome tools, emphasis on development, lack of organization support, unclear benefits, poor documentation, lack of experience, and steep learning curve.

V. Survey of Microsoft Developers

In this section, we present our results of the survey conducted on Microsoft developers.

A. Types of Testing

114 out of 127 developers use manual testing whereas 68 developers use automated testing tools. Some developers use both manual and automated testing. 4 developers responded that they do no test their apps. Figure 6 shows number of developers who perform different types of testing while developing apps. Most of the developers i.e., 103 in our survey perform functional testing. 97 developers perform unit testing, 75 perform integration testing, 74 perform performance testing, 63 perform regression testing, 47 perform system testing, 45 perform acceptance and load testing and 43 perform beta testing.

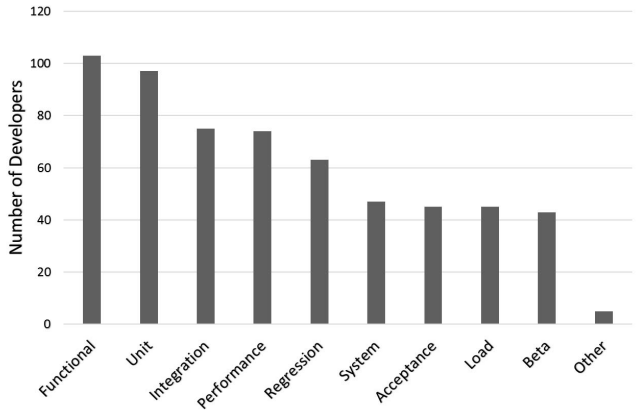

Figure 6: Types of Testing

B. Automated Testing Tools Usage

Table IV shows some of tools used by app developers in Microsoft. The results show that developers prefer using in-house tools. We also analyze why developers use automated testing tools. Figure 7 shows why developers use automated testing tools and the corresponding number of developers for each type of usage. 64 developers use tools for executing test cases, 48 use them for finding potential bugs, 43 use them for analysing code coverage, 37 each use them creating & evaluating test execution results and for performing load testing, 33 each use them for generating test cases and managing test suites, whereas 27 developers use tools for reporting bugs.

Table IV: Automated Testing Tools Usage

Tools	Number of Respondents
Visual Studio	35
Internal Tool	8
Selenium	7
Microsoft Test Manager	5
Others (QUnit, Robotium etc.)	27

C. Challenges faced by App Developers

In this section, we discuss the challenges faced by the app developers at Microsoft while testing their apps either manually or using automated testing tools. Figure 8 shows the challenges encountered by developers along with their perceived severity levels. We can observe that 35 developers consider time constraints as a very serious challenge and 56 consider it as a serious challenge. Poor documentation is the next big challenge which was mentioned by 19 developers as very serious and by 32 developers as serious. Lack of exposure, emphasis on development and compatibility issues were mentioned by several developers as a very serious challenge among others.

D. Developer Needs

In this section, we discuss the needs of the developers from the automated testing tools they use. We ask developers for two additional things they would like to see in the automated testing tools.

Poor documentation is one of the barriers for learning a new tool. Several developers expressed that a good documentation will increase their likelihood of using the tool. One of the developers commented *"Proper documentation so that a person new to the system can easily ramp up using these documents or articles."*

Developers often struggle to meet the deadlines due to the amount of the work they are assigned and the corresponding amount of time allotted for completion. To worsen the problem, developers are unaware of the testing tools which would be helpful for them. Examples of testing from successful projects would go a long way in motivating developers to use these tools. One of the developers mentioned *"We should have more internal material on proven practices about how to do testing, which Tools to use and many many samples and how-to Videos would be great. There is a lot of stuff about .NET code testing but not much about XAML App testing (at least not enough Deep digging Content)"*.

Although there are lot of testing tools available, developers have to put in significant effort in activities such as generating and executing test cases. An automated testing tool which accepts the requirements and perform testing would do wonders for the developers. A developer mentioned *"Test case generation on most of the testing tools I came across needs to be generated by manually. this needs to be reduced with tools automation."*, while another one commented *"There should a tool which should accept the requirement from Dev. and should be able to develop the test suite to run test cases. It reduces lot of testing efforts."*

In general, developers expect tools which are easy to use. One of developers opined *"I would love to see testing tools that are simple to learn and straightforward to use. Most tools*

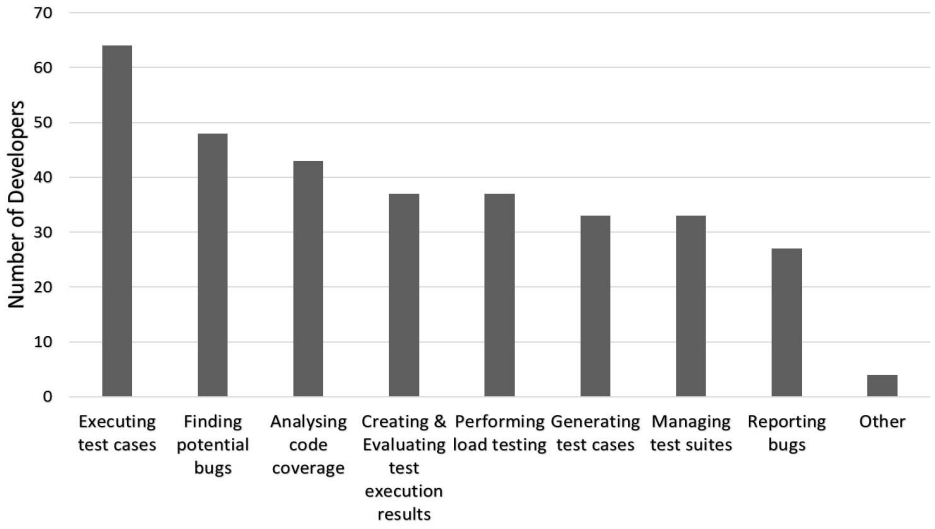

Figure 7: Usage of Automated Testing Tools

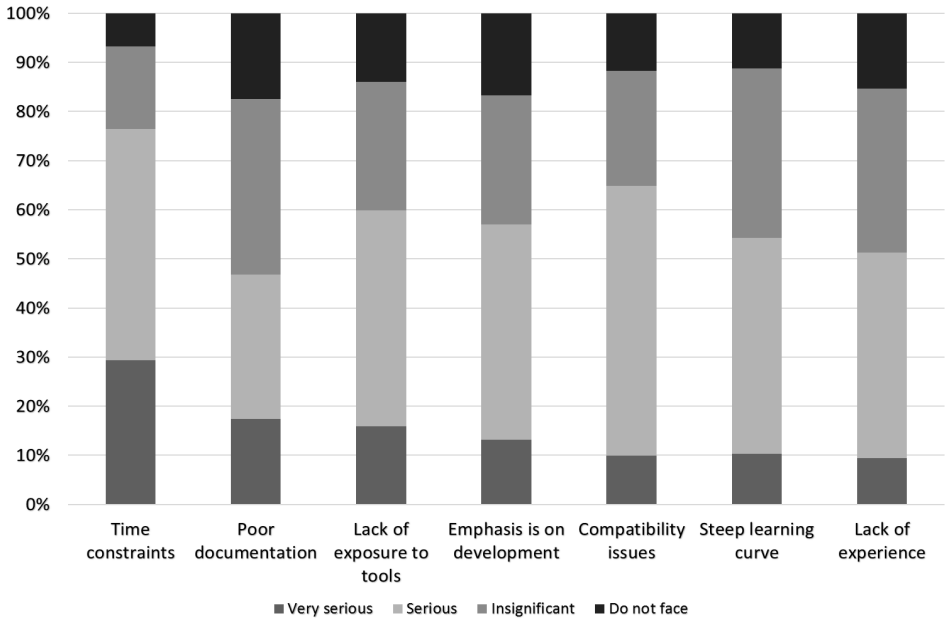

Figure 8: Challenges Faced by Developers

are cumbersome, lacking documentation and support is poor. Most of the existing UI Framework testing Tools for XAML feel incomplete."

Our survey shows that most of the Microsoft developers test the apps that they build and perform various kinds of testing. They employ a number of automated testing tools for various purposes; the top three reasons are executing test cases, finding potential bugs, and analyzing code coverage. The top 3 challenges faced by Microsoft developers in testing their apps are time constraints, compatibility issues, and lack of exposure to tools. Microsoft developers expect automated testing tools which are more well documented and promoted in the company, easy to use, and automating more manual tasks.

VI. Discussion

Automated testing is not a panacea but is useful and important especially for apps that will be regularly updated in many releases over a long period of time. One of the Android developers commented that "*The problem with automatic test is, that you need to update them with the code, and it's not unusual, that you change a code, which needs a bunch of unit tests to be rewritten, or if you change the layout of an app, then your complex ui tests needs to be updated. This can add more development time than you save on not having bugs, when you are doing a one-off app development, but* ***can really save you a lot of time, when you have frequent releases of the same app for years***."

Despite the benefits of software testing, our study finds that most open-source Android apps are not adequately tested - nearly 86% of the apps do not contain any test cases and the median line coverage across all the apps that have executable test cases is only 9.33%. Our study is the first one that empirically demonstrates this phenomenon over a large number of open-source Android apps. Therefore, there is a need to improve testing for Android apps.

Both Android and Microsoft developers find some parts of code are hard to test with existing automated testing tools. Many also find existing tools hard to use. Thus, there is a room to develop new automated testing tools that are both more powerful and more user friendly. The tools should be able to help developers test difficult cases. On the other hand it needs to be easy to use such that developers do not need to invest much effort and resources to learn these tools. One thing that make tools hard to use is the unavailability of good documentations. A documentation of a tool might be spread across tutorials, read me files, blogs, forums, question & answer sites, etc. It will be interesting to develop a tool that can aggregate and summarize these pieces of information together to make it easier for developers to learn how to use automated testing tools effectively.

Many developers are not aware of existing testing tools. This highlights the need for researchers to not only release research prototypes online (e.g., in GitHub) but also promote these tools through various channels that developers often use to get information. Singer et al. highlighted various social channels that developers have used to get information [5]. It would be interesting to investigate ways to automatically propagate relevant information to practitioners using these channels.

VII. Threats to Validity

Threats to External Validity. Threats to external validity relate to the generalizability of our results. We have investigated over 600 Android apps from F-Droid, which is one of the largest repositories of open-source Android apps. Our dataset consists of many kinds of apps from small ones to large and popular ones that contain more than one million lines of code or downloads. Still, it is unclear if our findings would generalize to all Android applications. In the future, we plan to reduce this threat further by analyzing more Android apps. Our respondents might not be representative of the entire population of app developers and thus our results might not generalize to all app developers. We have tried to reduce this threat to validity by surveying more than 200 developers of Android and Windows, which are the two most popular mobile app platforms. To the best of our knowledge, our study is the largest study on app developers to date.

Threats to Internal Validity. Threats to internal validity relate to the conditions under which experiments are performed. We automatically identify apps which contain test cases by using the following heuristics: we treat *.java* files whose names contain the word "test" as test files. We might miss some test files or mistakenly consider a file to be a test file when it is not. Furthermore, we manually analyse 89 apps which contain test files and calculate the coverage of test cases contained in these files. Out of the 89 apps, many failed to compile mainly due to missing dependencies. We tried our best to resolve all the dependencies by finding and downloading needed external libraries. However, we still cannot resolve many of them. We only compute coverage for 41 apps that we can successfully compile. To calculate the number of developers, we use information from git logs. There may be cases where the same developer uses different e-mail addresses to commit to the same git repository and we may have wrongly counted the number of developers.

VIII. Related Work

Android Testing Techniques. There have been many studies on techniques for testing Android applications. Hu and Neamtiu proposed automatic testing framework for Android applications which combines test case and automatic event generation with runtime monitoring and log file analysis [6]. Amalfitano et al. proposed a GUI crawling approach to automate testing in Android applications [7]. Recent study by Amalfitano et al. proposed MobiGUITAR, a model based tool that allows automated testing of mobile applications [8]. Anand et al. proposed Acteve, a tool for performing automatic concolic testing in smartphone applications [9]. Mirzaei et al. proposed a technique to test Android applications using symbolic execution [4]. Jensen et al. proposed a targeted event sequence generation based technique to automate Android application testing [10]. Machiry et al. proposed Dynodroid, a tool for generating relevant inputs for Android applications [2]. Gomez et al. proposed RERAN, a tool for recording and replaying inputs to Android applications, which can be used for generating automated UI tests [3].

The above studies signify a growing interest in the software engineering research community to support automated testing of Android applications. However, no tool/technique will ever be useful if no one wants to use it. To the best of our knowledge, there has not been any study about testing practices in the Android community, especially on the adoption of automated testing tools. In this work, we fill this gap by presenting the current state of testing practice in Android community and describing the challenges in adopting automated solutions for testing Android applications.

Empirical Studies on Android. There have been many empirical studies on Android. Takala et al. reported experiences on applying model based user interface testing on Android applications [11]. Kropp and Morales investigated strengths and weaknesses of two approaches for testing mobile GUI applications: the Android instrumentation framework and Positron framework [12]. Bhattacharya et al. performed an analysis on bug reports and bug fixing process of Android applications [13]. McDonnell et al. studied the stability and adoption of APIs in Android ecosystem [14]. Syer et al. studies 15 most popular Android applications and compare them with 3 desktop applications [15]. Ruiz et al. investigated the practice of reuse in Android ecosystem [16]. Maji et al. characterize failures in Android and Symbian mobile OSes [17].

In this work, we perform empirical study on automated testing practices in Android community. We surveyed Android developers and mined source code repositories of the applications that they develop to understand the current practices for testing Android applications.

Empirical Studies on Testing. A number of studies investigate test adequacy of open-source projects. Kochhar et al. investigated the correlation between the presence of test cases and various project development characteristics, including the lines of code and the size of development teams [18]. They extended their study to include characteristics such as number of bugs, number of bug reporters and the programming languages [19]. Moreover, in their latest study they investigated the adequacy of testing by analysing correlations between code coverage and software metrics such as lines of code, cyclomatic complexity, and number of developers [20].

In this work, we uncover the current state of automated testing practices in app development community which has not been studied by prior works.

IX. Conclusion and Future Work

Testing is a crucial activity during the software development lifecycle. With an ever-growing app community, testing holds much more importance to ensure that apps are adequately tested and reliable, which would lead to better user experience and overall growth of the community.

Our study reports the current state of testing of over 600 Android apps from F-Droid. We analyze the source code repositories of these apps and count the number of applications that have test cases. We also run these test cases and compute their line and block coverages. Furthermore, we survey developers of these apps to understand the test automation culture in Android app and Windows app development community. We ask developers about tools used by them and challenges faced by them while testing their apps.

The following is a summary of our findings:

1) Only around 14% of the apps contain test cases and only around 9% of the apps that have executable test cases have coverage above 40%.
2) Android app developers prefer using standard framework such as JUnit, but they also use Android specific testing tools such as Monkeyrunner, Robotium and Robolectric. However, many Android developers prefer to test their applications manually without the help of any testing framework or tools. Most Windows app developers make use of Visual Studio, Coded UI, Selenium, and Microsoft Test Manager to test their apps.
3) Android and Windows app developers face numerous challenges in testing their apps and in using automated testing tools. These challenges include time constraints, compatibility issues, lack of exposure, cumbersome tools, emphasis on development, lack of organization support, unclear benefits, poor documentation, lack of experience, and steep learning curve.

App developers can use our findings to gain insights into tools used and challenges commonly faced by their counterparts. Software organizations can use our findings to provide support to developers to overcome these challenges.

Our study is the first exploratory step to understand the automated testing practices in the app development community. In the future, we want to expand our empirical study by analyzing more apps and by surveying more developers to get more responses. We also plan to develop tools that address the "pain points" that are faced by developers. For example, we want to investigate automated ways to improve documentations of automated testing tools by aggregating and summarizing information from various blogs, forums, and question and answer sites (e.g., StackOverflow). We also want to build powerful and yet user-friendly testing tools that will make testing Android and Windows apps much easier for developers.

Acknowledgement

We would like to thank all the survey participants for sharing their time and experience, and the ICST reviewers for their valuable feedback on this paper.

References

[1] Gartner, "Gartner says annual smartphone sales surpassed sales of feature phones for the first time in 2013," in *http://www.gartner.com/newsroom/id/2665715?fnl=search*, Last accessed on October 22, 2014.

[2] A. MacHiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for Android apps," in *FSE*, pp. 224–234, 2013.

[3] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "RERAN: timing-and touch-sensitive record and replay for Android," in *ICSE*, pp. 72–81, 2013.

[4] N. Mirzaei, S. Malek, C. S. Păsăreanu, N. Esfahani, and R. Mahmood, "Testing Android apps through symbolic execution," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 6, pp. 1–5, 2012.

[5] L. Singer, F. M. F. Filho, and M.-A. D. Storey, "Software engineering at the speed of light: how developers stay current using Twitter," in *ICSE*, pp. 211–221, 2014.

[6] C. Hu and I. Neamtiu, "Automating GUI testing for Android applications," in *AST*, pp. 77–83, ACM, 2011.

[7] D. Amalfitano, A. R. Fasolino, and P. Tramontana, "A GUI crawling-based technique for Android mobile application testing," in *ICSTW*, pp. 252–261, 2011.

[8] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. Ta, and A. Memon, "MobiGUITAR – a tool for automated model-based testing of mobile apps," *IEEE Software*, vol. 99, 2014.

[9] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in *FSE*, p. 59, 2012.

[10] C. S. Jensen, M. R. Prasad, and A. Møller, "Automated testing with targeted event sequence generation," in *ISSTA*, pp. 67–77, 2013.

[11] T. Takala, M. Katara, and J. Harty, "Experiences of system-level model-based GUI testing of an Android application," in *ICST*, pp. 377–386, 2011.

[12] M. Kropp and P. Morales, "Automated GUI testing on the Android platform," *Testing Software and Systems*, p. 67, 2010.

[13] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru, "An empirical analysis of bug reports and bug fixing in open source Android apps," in *CSMR*, pp. 133–143, 2013.

[14] T. McDonnell, B. Ray, and M. Kim, "An empirical study of API stability and adoption in the Android ecosystem," in *ICSM*, pp. 70–79, 2013.

[15] M. D. Syer, M. Nagappan, B. Adams, and A. Hassan, "Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open source Android apps," in *CASCON*, pp. 283–297, 2013.

[16] I. J. M. Ruiz, M. Nagappan, B. Adams, and A. E. Hassan, "Understanding reuse in the Android market," in *ICPC*, pp. 113–122, 2012.

[17] A. Kumar Maji, K. Hao, S. Sultana, and S. Bagchi, "Characterizing failures in mobile oses: A case study with Android and Symbian," in *ISSRE*, pp. 249–258, 2010.

[18] P. S. Kochhar, T. F. Bissyandé, D. Lo, and L. Jiang, "Adoption of software testing in open source projects-a preliminary study on 50, 000 projects," in *CSMR*, pp. 353–356, 2013.

[19] P. S. Kochhar, T. F. Bissyandé, D. Lo, and L. Jiang, "An empirical study of adoption of software testing in open source projects," in *QSIC*, pp. 103–112, 2013.

[20] P. S. Kochhar, F. Thung, D. Lo, and J. Lawall, "An empirical study on the adequacy of testing in open source projects," in *APSEC*, 2014.