On The Link Between Mobile App Quality And User Reviews

by

Hammad Khalid

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
October 2014

Abstract

Mobile app stores contain millions of apps which users can download and install on their smart phones. Each app has a page in the mobile app store that includes app's description, a download link for the app, and a space for users to review the app. Each review has a 1-5 star rating and a review-comment. Unlike desktop and server side-software where direct user feedback about software quality was difficult to acquire, app developers now have access to the user's perspective of their apps via the reviews. Since apps with high-rating reviews are downloaded statistically significantly more than apps with low-rating reviews, the insights from studying these reviews are very important. Thus in this thesis, we analyze hundreds of thousands of reviews of Android and iOS apps to help developers understand the relationship between app quality, and the feedback in reviews.

In this thesis, we find that low-rated reviews of apps contain 12 different complaint types which have varying impact and frequencies. The most frequent complaints are about functional errors, feature requests, and app crashes; complaints about privacy, ethical and hidden cost issues receive the worst star ratings. For Android developers struggling with device fragmentation, we find that different Android devices give varying star ratings. However, we show that device info from reviews can also be used to identify a subset of Android devices that should be prioritized for testing.

Finally, we show that warnings from FindBugs, a static analysis tool, are related to lower average app ratings and the complaints that users leave in the reviews. This thesis shows how studying the reviews of mobile apps can help developers prioritize their testing efforts to address the concerns of their users.

Co-Authorship

Earlier versions of chapters in this thesis were published as listed below:

1. *What Do Mobile App Users Complain About? – Chapter 3*

   Hammad Khalid, Emad Shihab, Meiyappan Nagappan and Ahmed E. Hassan, IEEE Software, 2014

   My contributions: Drafting the research plan, gathering and analysing the data, writing the manuscript

2. *Prioritizing The Devices To Test Your App On: A Case Study Of Android Game Apps – Chapter 4*

   Hammad Khalid, Meiyappan Nagappan, Emad Shihab and Ahmed E. Hassan, In Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering

   My contributions: Drafting the research plan, gathering and analysing the data, writing the manuscript

Acknowledgments

I consider myself very fortunate to be surrounded by an incredibly supportive group of role-models: Samina (mom), Kat, Laila and Sohail – I am very happy to have you in my life.

Moreover, I would like to thank my supervisor Dr. Ahmed E. Hassan whose support and incredible ability to see the bigger picture were indispensable for this work. Thank you for taking a chance on me and supporting my work.

I am deeply indebted to Dr. Meiyappan Nagappan and Dr. Emad Shihab who are the main collaborators for my research. Their ideas, mentorship, and foresight has had an incredibly positive impact on this work. I am very grateful for their help.

In addition I would like to thank Shane McIntosh and Sai Bala. Their positivity, insights and support were invaluable to me.

I feel very proud for being able to work at the School of Computing in Queen's University. I am very happy with the time that I spent here, the support that I received, and the friends that I made.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over the past few years, the mobile ecosystem has grown at an exceptional rate. Android and iOS operating systems (OS) have been the biggest benefactors of this growth and now dominate the smart phone market [1, 2]. Because of this growth, applications (apps) in both OS affect the lives of millions of users; there are apps for productivity, entertainment, home automation and fitness etc [3]. Since these apps are so interlinked with the lives of users, the expectations of their quality is very high.

Google Play and *iTunes App Store* are the main marketplaces for Android and iOS apps respectively, from where users can search and download apps on their mobile device [4, 5]. Both of marketplaces allow users to give a public *review* of an app. The review includes a 1 to 5 *star rating* (1 being the worst, 5 being the best). These star ratings are aggregated to calculate the overall average app star rating. Each review also includes a *review-comment* which lets the user describe their star rating. When apps don't meet the expected quality, users give them low-rating reviews in their marketplace. Previous research has shown that these reviews are one of the primary indicators of app quality from users [6–8].

Unlike desktop and server-side applications which typically receive feedback from

bug reports, and user studies, the mobile ecosystem also provides public feedback to researchers. Researchers can take advantage of this data and study if and how these reviews can help developers improve the quality of their apps, and better prioritize their Quality Assurance (QA) resources.

1.1 Thesis Statement

The mobile ecosystem provides centralized and publicly available user feedback in the form of reviews. There is a great emphasis on these reviews and high-rated apps receive significantly more downloads than low-rated apps. Now that researchers have access to these reviews, they can perform new large-scale studies to help developers improve the quality of their apps, and better prioritize their QA efforts.

1.2 Overview

Below we give an overview of the three studies that we performed on the reviews of iOS and Android apps.

1. *What Do Mobile App Users Complain About? – Chapter 3*

   Reviews provide a rich data source that can be leveraged to understand user reported issues. By qualitatively studying 6,390 low-rated reviews for 20 free iOS apps, we uncover 12 types of user complaints. We find that functional errors, feature requests and app crashes are the most frequent complaints. Complaints about privacy and ethical issues, and hidden app costs have the most negative impact on the overall star rating of an app. We also find that users attributed their complaint to a recent update of the app in 11% of the reviews. Our study is the first to provide developers insight into the user reported issues of iOS apps,

along with their frequency and impact, helping developers better prioritize their limited QA resources.

2. *Prioritizing The Devices To Test Your App On: A Case Study Of Android Game Apps – Chapter 4*

   Star ratings from mobile app users directly impact the revenue of its developers [8]. At the same time, for popular platforms like Android, these apps must run on hundreds on devices increasing the chance for device-specific problems. Device-specific problems could impact the star rating assigned to an app, given the varying capabilities of devices (e.g., hardware and software). To fix device-specific problems developers must test their apps on a large number of Android devices, which is costly and inefficient.

   We mine the reviews of 99 free game apps and find that, apps receive reviews from a large number of devices: between 38 to 132 unique devices. However, most of the reviews (80%) originate from a small subset of devices (on average, 33%). These findings indicate that focusing on the devices with the most reviews (in particular the ones with negative star ratings), developers can effectively prioritize their limited QA efforts, since these devices have the greatest impact on star ratings. Furthermore, we find that developers of new game apps with no reviews can use the review data of similar game apps to select the devices that they should focus on first. Finally, among the set of devices that generate the most reviews for an app, we find that some devices tend to generate worse star ratings than others. We show that these findings also hold for apps in four other categories of the Google Play market.

3. *Examining the Relationship between FindBugs Warnings and End User Ratings:*

A Case Study On 10,000 Android Apps – Chapter 5

Past research has examined the relationship between static analysis warnings and quality metrics. However, there is no evidence linking the static analysis warnings directly to the user perception of software, as this relationship is difficult to examine by research. In the app ecosystem, user perception is extremely important to study as reviews of apps are highly correlated with downloads and hence revenues. We use FindBugs, which is an automated static analysis tool for Java code, to compare the results of running this tool on 10,000 free-to-download Android apps with the reviews of the apps.

We find that (1) a statistically greater density of static analysis warnings occur in low-rated apps than high-rated apps; (2) specific categories of FindBugs warnings such as the 'Bad Practice', 'Performance' and 'Internationalization' categories are found significantly more in low-rated apps. On examining the relationship between these three categories of warnings and the complaints in reviews, we find that apps with the highest densities of these warning categories, receive significantly more corresponding complaints. These findings provide evidence that certain categories of warnings from FindBugs are closely related to user experience and hence have a strong impact on the star rating of an app. Thus app developers can use static analysis tools such as FindBugs to potentially identify the culprit bugs behind the issues that users complain about, before they release the app.

1.3 Contributions

After in-depth analysis of the reviews of Android and iOS apps, we make the following contributions:

1. We identify the frequency and impact of 12 different complaint types in the reviews of iOS apps. We show that most of the frequent complaints are about functional errors or feature requests. We also show the importance of good business practices as privacy and ethical issues, and hidden cost issues have high impact on star ratings. In addition we identify the importance of regression testing in iOS apps. By studying complaints, we identify why users give low-rated reviews to apps. We compare the frequency and impact of complaints across apps in Android and iOS. This information can help developers better understand the user perceptive, and can thus help them improve the quality of their apps.

2. To aid Android developers struggling with Android fragmentation, we demonstrate how reviews of Android apps can be used to identify a subset of Android devices which should be prioritized for testing. We also show that some devices tend to generate worse star ratings than others. Android developers can allocate additional resources for such devices, or remove support for them if they see fit. By studying reviews of Android apps, we identify the devices which Android developers should prioritize while testing.

3. By comparing the warnings of FindBugs with the star ratings of apps, we show that low-rated apps have a statistically significantly higher density of FindBugs warnings. We also find that three categories of bugs occur significantly more in

low-rated apps. We show how app developers can avoid low-rated reviews by utilizing static analysis.

1.4 Organization of Thesis

The remainder of this thesis is organized as follows: Chapter 2 describes the background of relevant topics, and surveys the related work. Chapter 3 presents our case study on the complaints found in the reviews of iOS apps. Chapter 4 presents our case study on the reviews of Android apps to identify the devices which have the most impact on the star ratings of apps. Chapter 5 presents our case study on the relationship between static analysis warnings on 10,000 Android apps, with their corresponding reviews. Chapter 6 concludes this thesis, and describes its limitations and future work.

Chapter 2

Background and Related Work

In this chapter, we describe the relevant background information for this thesis and survey some of the related work. First, we describe the mobile ecosystems and its recent growth. More specifically, we cover the relevant details of the Android and iOS operating systems. Next, we describe the reviews aspect of the mobile ecosystem. Then, we examine the research which has examined improving the quality of mobile apps. Finally, we describe static analysis, and previous research which has performed static analysis on mobile apps.

2.1 Mobile ecosystem

Over the past few years, the mobile ecosystem has grown at a very impressive rate [3, 9]. Currently this ecosystem includes many operating systems including iOS, Android, Windows Phone and BlackBerry [4, 5, 10, 11]. These operating systems are generally coupled with app marketplaces, which are the distribution platforms for apps built for of these operating systems. For example, Apple hosts the *iTunes App Store* for iOS apps, while Google hosts *Google Play* store through which Android apps are distributed [4, 5]. The revenue from these apps is expected to grow to $74 billion by

2017 [12]. Such lucrative opportunities in the mobile market has drawn the attention of large developer communities, and has made the app market very competitive.

These app marketplaces allow anyone (i.e., independent developers to large corporations) to upload and distribute their apps. Users of these platforms are allowed to rate and review these apps. This process helps these marketplaces rank the apps based on their star ratings. Apps with high-ratings are featured and gain more attention from users, while apps with poor ratings are rarely seen by users. This makes getting and maintaining high-ratings extremely important.

In this thesis, we seek to study the reviews of apps in the mobile ecosystem so that developers can improve the quality of their apps. Next, we cover the relevant info for the iOS and Android operating systems that are two main subjects of this thesis.

2.1.1 iOS

iOS, alongside the first iPhone, was first introduced by Apple in 2007. The main goal behind this product was to create a device that allows users to make phone calls, listen to music, and browse the web with one touch-capable device. Today, seven generations later, iOS has evolved and now powers both iPhone as well as the iPad tablet.

While the operating system itself is not open source, iOS does allow independent developers to make apps on their platforms. These apps are made using Objective C, and the Cocoa Touch framework. These apps are hosted on the iTunes App Store which alone contained more than 1,000,000 apps (as of August 2014) and remains one of the most competitive app markets [3].

2.1.2 Android

Android is an open source operating system which is based on the Linux kernel. Similar to iOS, this operating system supports touch input, and offers support for many other hardware such as accelerometers, gyroscopes and proximity sensors. Android has many marketplaces which distribute Android apps, amongst them are Google Play, Amazon Appstore for Android, and Samsung Apps [4,13,14]. Android apps are generally developed using the Java programming language (although, other options for programming languages are available).

Usage of Android devices has grown at a tremendous rate over the past few years [9]. At the end of 2013, Android accounted for 79% of the smart phone market in the world; iOS accounted for 15.5% [1]. In the United States, however, Android accounts for 52% of the market while iOS accounts for 40% [2].

While the open source nature of Android has allowed it to surpass iOS in market share, the same openness has lead to some challenges for developers working on Android apps. For instance, since any manufacturer can make a device which runs Android, there are hundreds of Android devices in the market. Each one of them have different capabilities, different hardware capabilities, and different target audiences. This is a challenge for developers because now they have to test their app on numerous devices to avoid device-specific issues. This problem is referred to as Android fragmentation.

A recent study by Han *et al.* manually labeled bugs reported for specific vendors of Android devices [15]. They highlighted evidence for Android fragmentation by showing that HTC and Motorola devices have their own vendor specific bugs. They considered this as evidence for Android fragmentation. Ham *et al.* came up with

their own compatibility test system to prevent Android fragmentation problems [16]. They focus on doing code analysis and API pre-testing to identify possible issues. In this thesis, we study how the device information in Android apps reviews can help Android developers deal with Android fragmentation.

The mobile ecosystem has grown at a very impressive rate over the past few years. This interest in mobile apps, along with their usage, has made the app market very competitive for developers. This competition is most intense for iOS and Android app developers. In this thesis, we seek to help developers of apps in these platforms understand how they can improve the perceived quality of their apps and compete in this market.

2.2 Reviews of Mobile Apps

One of the most interesting aspects of the mobile ecosystem is the emphasis on user feedback. Most app marketplaces allow the users to give public reviews to an app. Users can give a star rating to an app (all of which are aggregated and displayed at a version and app level basis), and provide a *review-comment* which describes their rating. If users have a bad experience with an app, they give it a low-rating.

Thus star ratings of apps in the app markets (e.g., Google Play, iTunes App Store) give an insight into the user's opinion of the app. Next, we survey studies which have examined the relevance of these reviews.

2.2.1 Studies on reviews

Reviews have become an important source of information about the perception of customers. There are a slew of studies that have explored star ratings of products and services (e.g., [6–8]).

In fact, Mudambi *et al.* [6] showed that star ratings and reviews play a key role

in the purchasing decision of products at the online retailer *Amazon*. Instead of mining reviews (like [6]), Kim *et al.* [7] interviewed 30 users who bought apps. Kim *et al.* found that word of mouth and star ratings were key determinants in the user's purchase decision of an app. Harman *et al.* [8] mined information from 30,000 BlackBerry apps. They found that there is no correlation between price and star ratings, or between the price and downloads. However, they find that there is a strong correlation between star rating and number of downloads. They associated star ratings to specific features in apps (e.g., having GPS support increases the star ratings of an app).

Vasa *et al.* [17] and Hoon *et al.* [18] analyzed reviews of mobile apps and found that the depth of feedback, and the range of words is higher when the users give a low-rating to an app – highlighting the usefulness of low-rating reviews.

Pagano and Maalej [19] carried out an exploratory study on reviews from iOS apps to determine their potential for requirements engineering processes. They showed that the helpfulness and topics within these reviews vary. There have also been studies on automatically extracting feature requests from the reviews [20, 21].

In Chapter 3 of this thesis, we manually examine the text data within these reviews to identify complaints of users. Next, we discuss previous research that has extracted topics from text data via manual analysis.

2.2.2 Manual Analysis of text data

Past research hash also performed manual analysis to highlight critical information for developers. Thung *et al.* [22] manually categorized the bugs that occur in Machine-learning systems. They also reported that some bug categories have higher severity

than others. Similarly, Tian *et al.* [23] performed manual analysis on the content of Software Engineering microblogs to better understand what developers microblog about. Their manual analysis, was able to categorize these microblogs into distinct categories (e.g., jobs or links for tools).

Reviews are a rich yet rarely studied source of information about the user's perception of app quality. An in-depth understanding of such perception will assist developers in building apps that better meet the needs of users, and help researchers identify research problems for mobile app with the most impact. In this thesis, we examine reviews to better understand the complaints of users, and the issues that impact developers.

2.3 App Quality

The growth in usage of mobile apps, and their relevance in different fields, has made the quality of mobile apps a very important topic. Below we survey the research which relates to the quality and testing of mobile apps.

2.3.1 Studies on the Quality of Mobile apps

Research related to mobile app development is in its early stages, but it is gaining a great amount of interest [24]. However most prior work to date is focused on quality from the perspective of developers, not users. For example Syer *et al.* [25] compare the source code for mobile apps developed for the Android and BlackBerry platforms. They find that BlackBerry apps tend to depend less on the platform compared to Android apps. Ruiz *et al.* [26] compare the extent of reuse of code among different categories of Android apps. They find that there is a tendency to have identical apps serving different locales or areas (e.g., one weather app for each city).

Agarwal *et al.* [27] performed a study involving mobile app developers to better

diagnose unexpected app behaviour. Jha [28] looked at different testing strategies to improve the quality of mobile apps, and cataloged possible concerns when during testing. Kim *et al.* proposed a method of unit testing the performance issues of mobile apps and suggested different strategies that could improve the code quality of mobile apps [29].

Stevens *et al.* examined permission usage in 10,000 Android apps and found a relationship between the popularity of a permission and the number of times it is misused [30]. Khomh *et al.* examined the pre-release field testing data of large enterprise software for mobile apps and proposed two new metrics which they show to be more effective at predicting post-release defects [31].

Linares *et al.* examined API usage of Android apps and found that fault and change proneness within the APIs of Android may have an effect on the quality and star ratings of apps [32]. They found that these changes in the Android APIs can lead to issues which cause malfunctions and crashes in the apps that use them.

In this thesis, we want to help developers improve the quality of their apps by better understanding user feedback. Next, we examine studies which have focused on specific issues of Android apps.

2.3.2 Studies related to testing Android apps

Several recent studies have attempted to reduce the testing burden of Android developers. Hu *et al.* suggested methods for automated test generation and analysis for Android apps [33]. The feed random touch events into an app, and logged the results for analysis. They show that their tool can be used to discover pre-existing and new bugs. Similarly, Machiry *et al.* presented a dynamic input generation system which

can be used by developers for black-box testing [34]. Anand *et al.* also presented an approach which generates input events to execute different event sequences for an Android app [35].

There has also been previous work which has aimed to automate testing in a virtual environment. Amalfitano *et al.* presented a tool which automatically generates tests for Android Apps based on their graphical user interface [36]. DroidMate is another such tool which uses genetic algorithms to generate input sequences (i.e., user interaction, or simulated events) [37].

While testing in a virtual environment is useful for identifying general issues, developers also test their apps on actual Android devices to identify device-specific issues. Google also recommends that that developers test their apps on actual devices before releasing to Google Play [38]. Because of this, major app developers go through the expensive process of thoroughly testing their Android apps on many different devices [39, 40]. Thus, chapter 4 of this thesis focuses on helping developers identify the devices that have the most impact on their app's star rating; developer can prioritize their testing efforts by focusing on these devices.

Research on mobile app quality is in its infancy where most of the prior studies have focused on technical issues related to the quality of mobile apps instead of focusing on the complaints raised by users in the field.

2.4 Static Analysis

Chapter 5 of this thesis uses static analysis to examine the relationship between static analysis warnings and reviews of the apps. In this section we survey the work most related to FindBugs, and other static analysis tools applied on Android apps.

Static analysis via FindBugs: Hovemeyer *et al.* introduced FindBugs as a tool that automatically detects a variety of bugs within Java programs [41]. Their goal was to make a tool that is easily extendible and helps developers identify overlooked bugs and misuses of Java's features. Cole *et al.* performed a study to show the usefulness of the results of FindBugs [42]. They showed that most of the warnings from FindBugs were true errors and could often point out major problems within Java code.

Ayewah *et al.* conducted a study where they examined the evaluations of Google engineers of FindBugs warnings across numerous code bases. They found that while not all issues caused serious problems, most evaluations from engineers suggested fixing the underlying issues of these warnings [43]. On the other hand, Vetro *et al.* compared the FindBugs warnings on 301 student Java projects, with the change information from defects and showed that not all reported warnings are real defects with high precision [44].

Ayewah *et al.* evaluated the effectiveness of FindBugs on multiple projects and software development within Google [45]. They manually evaluated warnings and showed that while there were some warnings which did not lead to bugs, the overall rate of false positives was fairly low.

Static analysis in Android apps: Guo *et al.* proposed a static analysis tool called *Relda* that can help developers identify resource leaks in Android apps [46]. Their tool analyzes the callbacks in Android framework to locate the resource leaks. Similarly, Payet *et al.* extended a pre-existing static analysis tool called *Julia* and improved the precision of detecting nullness in Android apps [47]. Their work also demonstrated

the usefulness and versatility of static analysis tools in Android apps.

Krishnan *et al.* proposed a process for identifying security issues using existing tools [48]. In addition to unit testing and black box testing, they recommend static analysis via FindBugs to check coding standards.

While previous studies have examined the relationship of FindBugs warnings and code quality, Chapter 5 of this thesis compares Findbugs warnings with the user perceived quality of mobile apps, in the form of user reviews.

Previous research has shown the usefulness of FindBugs and the importance of reviews in the mobile ecosystem. We examine the relationship between the warnings of FindBugs and reviews of apps. We study how FindBugs can help developers improve the user perceived quality of their apps.

Chapter 3

What Do Mobile App Users Complain About?

Summary *– Reviews provide a rich data source that can be leveraged to understand user reported issues. By qualitatively studying 6,390 low-rated reviews for 20 free iOS apps, we uncover 12 types of user complaints. We find that functional errors, feature requests and app crashes are the most frequent complaints. Complaints about privacy and ethical issues, and hidden app costs have the most negative impact on the overall star rating of an app. We also find that users attributed their complaint to a recent update of the app in 11% of the reviews. Our study is the first to provide developers insight into the user reported issues of iOS apps, along with their frequency and impact, helping developers better prioritize their limited QA resources.*

Mobile apps continue to grow in popularity at a rapid pace. The Apple iOS (mobile operating system) App Store alone contained more than 1,000,000 apps and remains one of the most competitive app markets [3]. This competition, and the growth of apps as observed thus far in critical domains such as e-commerce, government and the health care industry has made the quality of apps an increasingly important issue.

The majority of recent work on the quality of apps has focused on quality issues

from the perspective of developers (e.g., [49]). However, one of the first steps in understanding the issues that impact the quality of apps is to determine the challenges or issues that *users* face when using these apps.

iOS apps are distributed through the App Store which lets users review their downloaded apps. In addition to assigning a *star rating* to iOS apps (all of which are aggregated and displayed at a version and app level basis), users can provide a *review-comment* to rationalize their star rating. This data source captures a unique perspective about the perception of users regarding the apps. Such reviews, like product-reviews in online web stores, are highly correlated with download counts (i.e., purchases) and are a key measure of the app's success [6,8]. A good understanding of these issues will help developers understand the concerns of users, avoid low-ratings, and better prioritize their QA resources. Furthermore, such an understanding is crucial in guiding the Software Engineering research community in tackling high-impact research problems in the fastest growing field of software development today.

In order to better understand the complaints of iOS users, we examine the low-rating (1 and 2-star) reviews associated with 20 free[1] iOS apps. Through a manual analysis of a statistically representative sample of 6,390 iOS reviews, we arrive at the following findings:

- We identified 12 types of user complaints in iOS apps ranging from functional to privacy and ethical issues. Users attributed these complaints to a recent update of an app in 11% of the sampled reviews. This highlights the importance of regression testing in iOS apps.

[1]free-to-download; while these apps are labeled 'free' in the App Store, some of them require a fee for premium features.

- The most frequent complaints are about functional errors, feature requests and app crashes. Examining these complaints can help developers identify existing problems, and new features for their app.

- The most negatively-impacting complaints are related to privacy and ethical issues, hidden costs and disliked features that are degrading the end-user experience. Understanding these complaints is important as some complaints can be much more detrimental than others.

Based on our findings (i.e., frequency and impact of complaint types), developers can better anticipate possible complaints, and prioritize their limited QA resources on the complaints most important for them.

The rest of the chapter is organized as follows. Section 3.1 describes our study approach. Section 3.2 presents the results of this chapter. Section 3.3 generalizes the results of this chapter by applying our approach to Android apps. Section 3.4 discusses some of our findings from the previous section. Section 3.5, we describe potential threats to the validity of this study. Section 3.6 concludes the chapter.

Table 3.1: Statistics of the studied iOS apps

	App Name	Category	Rating	Total Low Reviews	Sampled Reviews
High-rated iOS apps (rating above 3.5)	Adobe Photoshop Express	Photo & Video	3.5	1,030	280
	CNN app	News	3.5	1,748	315
	ESPN Score center	Sports	3.5	2,630	335
	EverNote	Productivity	3.5	1,760	315
	Facebook	Social Networking	4	171,618	383
	Four Square	Social Networking	4	1,990	322
	MetalStorm: Wingman	Games	4.5	1,666	312
	Mint.com Personal Finance	Finance	4	1,975	322
	Netflix	Entertainment	3.5	13,403	373
	Yelp	Travel	3.5	2,239	328
Low-rated iOS apps (rating below 3.5)	Epicurious Recipes & Shopping List	Lifestyle	3	940	273
	FarmVille by Zynga	Games	3	10,576	371
	Find My iPhone	Utilities	3	846	264
	Gmail	Productivity	3	1,650	312
	Hulu Plus	Entertainment	2	4,122	351
	Kindle	Books	3	3,188	343
	Last.fm	Music	3	1,418	302
	Weight Watchers Mobile	Health & Fitness	3	1,437	303
	Wikipedia Mobile	Reference	3	1,538	308
	Word Lens	Travel	2.5	1,009	278

3.1 Study Design

Users tend to write reviews when they are either extremely satisfied or extremely dissatisfied with a product [50]. The low-rating reviews have a greater impact on the sales than high-rating reviews since buyers are more likely to react to low-ratings and complaints [51]. Therefore, in order to understand why users give low-ratings to iOS apps, we focus our study on 1 and 2-star reviews. We explore two sources of information in these reviews: 1) the star ratings and 2) the free-form review-comments associated with each review. We manually tag review-comments in order to uncover common complaints across iOS apps. Such manual tagging is time consuming, therefore we focus on a subset of apps (20) and tag a statistically representative sample of their reviews (6,390 reviews across the 20 apps). Figure 3.1 illustrates the design of this chapter. In the following subsections, we describe each step in detail.

3.1.1 Selecting the Apps

We pick the 20 most popular iOS apps, as defined by the iOS Market during June 2012 (see Table 3.1), which are free-to-download. We make sure that the selected apps have at least 750 reviews so that a few users do not skew the tagged reviews that we analyze. We also ensure that half of the selected apps have an overall high-rated (3.5 stars or better) and that the other half of the apps have an overall low-rated (below 3.5 stars), since we want to identify the complaints in both high and low-rated apps. We end up with 20 apps that cover 15 of the 23 categories (e.g., Productivity, Finance) in the iOS market, ensuring the breadth of the studied apps.

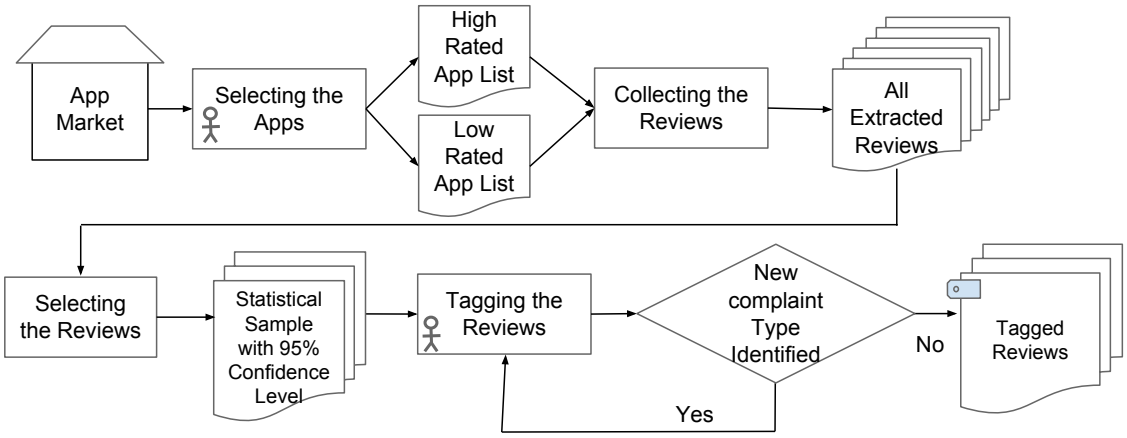


Figure 3.1: Overview of our manual process for tagging reviews

3.1.2 Collecting the Reviews

The main data source for our study are the reviews posted by iOS app users on *iTunes*. However, iTunes does not provide a public API for automatically retrieving reviews. Instead, we obtain the reviews from a web-service called *Appcomments*, which collects reviews of all iOS apps [52]. We build a web crawler which visits each unique page with a specific iOS review and parses the reviews to extract data such as the app name, the review title, the review-comment and the numerical star rating assigned by the user. We collected all the reviews for each of the 20 studied apps during the first week of June 2012.

3.1.3 Selecting the Reviews

As we want to examine the complaints of iOS users, we focus on 1 and 2-star reviews since they are more likely to have user complaints. The studied apps have over 250,000 1 and 2-star reviews. Since manually examining all of these reviews is extremely time-consuming, we study a statistically representative sample for these reviews [53]. The

sample of reviews is randomly chosen to achieve a 95% confidence level and a 5% confidence interval. This means that we are 95% confident that each of the results is within a margin of error of ±5%. For example, 'Adobe Photoshop Express' has a total of 1,030 1 and 2-star reviews. The statistically representative sample for 1,030 reviews, with a 95% confidence level and a 5% confidence interval, is 280 reviews. Thus, we randomly select 280 reviews from the 1,030 1 and 2-star reviews for manual examination.

In total, we manually examine 6,390 reviews. We perform our sampling on a per app basis since different apps have varying number of reviews and we want to capture the complaints across the different apps. The number of randomly sampled reviews for each app ranges from 264 to 383 and is shown in the sixth column of Table 3.1.

3.1.4 Tagging the Reviews

Once we determine the number of reviews to examine, we follow an iterative process called *Coding* as suggested by Seaman *et al.* to identify the different complaint types [54, 55]. The coding is used to turn qualitative information into quantitative data. The author of this thesis read each review to determine the type of complaint mentioned in the review. We follow the procedure below for tagging the reviews:

Inputs = All reviews (each with a review-title and a review-comment), a list of complaint types (which is initially empty)

For each review:
    Manually examine all of the text in the review.

    If review matches an existing complaint type:
        Tag review with a complaint type(s).

    Else:
        Add a new complaint type to the list of complaint types.
        Restart tagging with new list of complaint types.

Outputs = All reviews (tagged with appropriate complaint types), and a list of complaint types

This process is iterative such that each time a new complaint type is identified, we go through all the previously tagged reviews and see if they should be tagged using the new complaint type as well. This iterative process also helps us minimize the threat of human error while tagging the reviews. In total, we ended up having to restart the tagging process 3 times after discovering new complaint types. In certain cases, a user may provide no meaningful comment for their review (e.g., simply saying the app is bad). In such cases, we tag these types of reviews as being 'Not Specific'. Some reviews may also contain multiple complaints; in these cases, we tag the review with multiple complaint types. For example, if a network problem is mentioned in a review that also contains a complaint about the app crashing, the review will be tagged with the 'Network Problem' and 'App Crashing' complaint types.

Table 3.2: Identified complaint types with a description and an example

Complaint Type	**Description**	**Example Review**
App Crashing	The app is often crashing	"Crashes immediately after starting."
Compatibility	App has problems on a specific device or a OS version	"I can't even see half of the app on my ipod touch..."
Feature Removal	Complaint about a disliked feature that is degrading the experience	"This app would be great, but get rid of the notifications!!!"
Feature Request	App needs additional feature(s) to get a better rating	"No way to customize alerts. "
Functional Error	An app specific problem was mentioned	"Not getting notifications unless u actually open the app.."
Hidden Cost	Complaint about the hidden costs for full experience	"Great if you weren't forced to buy coins for REAL money..."
Interface Design	Complaint about the design, controls or visuals	"The design isn't sleek and not very intuitive"
Network Problem	The app has trouble with the network or is slow to respond	"New version can never connect to server!"
Privacy and Ethical	The app invades privacy or is unethical	"Yet another app that thinks your contacts are fair game."
Resource Heavy	The app consumes too much battery or memory	"Makes GPS stay on all the time. Kills my battery."
Uninteresting Content	The specific content is unappealing	"It looks great but actual gameplay is boring and weak."
Unresponsive App	The app is slow to respond to input, or laggy overall	"Bring back the old version. Scrolling lags."
Not specific	A review-comment that's not useful or doesn't point out a problem	"Honestly the worst app ever."

3.2 Results

Once we are done looking through all of the reviews, we end up with 12 different complaint types. Table 3.2 lists the different complaint types, provides a description for each type and gives an example review. We calculate the frequency and impact of each complaint type below.

RQ1) What are the most frequent complaints in low-rated reviews?

We calculate the frequency of the complaint types for each app. Once we have this frequency, we normalize it for each complaint type (i.e., number of complaints of a specific type divided by the total number of sampled reviews for an app), so that we can compare results across different apps with a varying number of reviews. Due to the high deviance of each complaint type between different apps, we use the median, instead of the mean, to summarize the frequency of each complaint type across all the studied apps.

Table 3.3 shows the rank and median percentage of the complaints in column two and three respectively. We see the 'Functional Error' complaints in 26.68% of the reviews, 'Feature Request' in 15.13%, and 'App Crashing' in 10.51%. Together, these three complaint types account for more than 50% of all complaints. Figure 3.2 illustrates these results.

To better understand 'Functional Error', the most frequent complaint type, we examine the most frequently-used terms in these reviews. Then, we read through all the review-comments that use these most frequently-used terms. We find that 4.5% of functional errors are about *location* issues and 7.3% are about *authentication* problems. Below is an example of a functional error review where a user reported an authentication problem.

Don't do the update!!! : when I try to login it just keeps refreshing the screen...

Examining 'Feature Request', the second most frequent complaint type, we find that most requests are very app specific. However, we do find that 6.12% of all feature requests by users are for better notification support in apps.

Overall, we find that 'Network Problem', 'Interface Design' and 'Feature Removal' complaints are also frequent. Another complaint that we identify is 'Compatibility' which is an important issue for iOS devices; this refers to a complaint where the app does not work correctly on a specific device or a version of the OS. Surprisingly, complaints about compatibility, resources and the responsiveness of an app are not as frequent – we expected more of such complaints.

In addition to measuring the frequency, we also examine whether the complaint types vary between high and low-rated apps. To do so, we compare the frequency of each complaint type among the 10 high and the 10 low-rated apps. We carry out this comparison using a two-tailed *Mann-Whitney U-Test* with $\alpha < 0.05$. We find that there is no statistically significant difference between high and low-rated apps.

Our findings highlight the importance of software maintenance activities for iOS apps since many of the low-rating reviews can be avoided by an increased focus on QA (e.g, 'Functional Error', 'App Crashing', 'Network Problem'). In addition, our findings show that low-rating reviews frequently contain information that can help developers identify the features which their users want, or really hate (e.g, 'Feature Request', 'Feature removal').

Functional Error, Feature Request and App Crashing are the most frequent complaints and account for more than 50% of the reported complaints.

Table 3.3: The Most Frequent and Impactful Complaint Types (All results are at 95% confidence level)

	Most frequent		Most impactful	
Complaint Type	**Rank**	**Median (%)**	**Rank**	**1:2 star**
Functional Error	1	26.68	7	2.1
Feature Request	2	15.13	12	1.28
App Crashing	3	10.51	4	2.85
Network Problem	4	7.39	6	2.25
Interface Design	5	3.44	10	1.5
Feature Removal	6	2.73	3	4.23
Hidden Cost	7	1.54	2	5.63
Compatibility	8	1.39	5	2.44
Privacy and Ethical	9	1.19	1	8.56
Unresponsive App	10	0.73	11	1.4
Uninteresting Content	11	0.29	9	1.5
Resource Heavy	12	0.28	8	2
Not specific	-	13.28	-	3.8

RQ2) Which of the complaint types have the most impact on reviews? Having identified the most common complaint types by analyzing 1 and 2-star reviews, we determine which of these complaints are the most negatively-perceived by users. We determine the most negatively-perceived complaints by looking at the ratio of 1 to 2-star ratings for each complaint type (across all apps). For example, a 1 to 2-star ratio of 5 for a complaint type indicates that this complaint type has 5 times as many 1-star ratings as 2-star ratings.

Columns 4 and 5 of Table 3.3 show the rank and the 1:2 star ratio for each complaint type. The most negatively-perceived complaints are different from the most frequent complaints. Observing Table 3.3, we see that 'Privacy and Ethical', 'Hidden Cost' and 'Feature Removal' are the top three most negatively-perceived complaints – meaning that users are most bothered by issues related to the invasion of their privacy and unethical actions of the app developer (e.g., unethical business practices or selling the user's personal data). Developers should only access the data (e.g., contacts of the user, or a user's location) that they specified in the app's description.

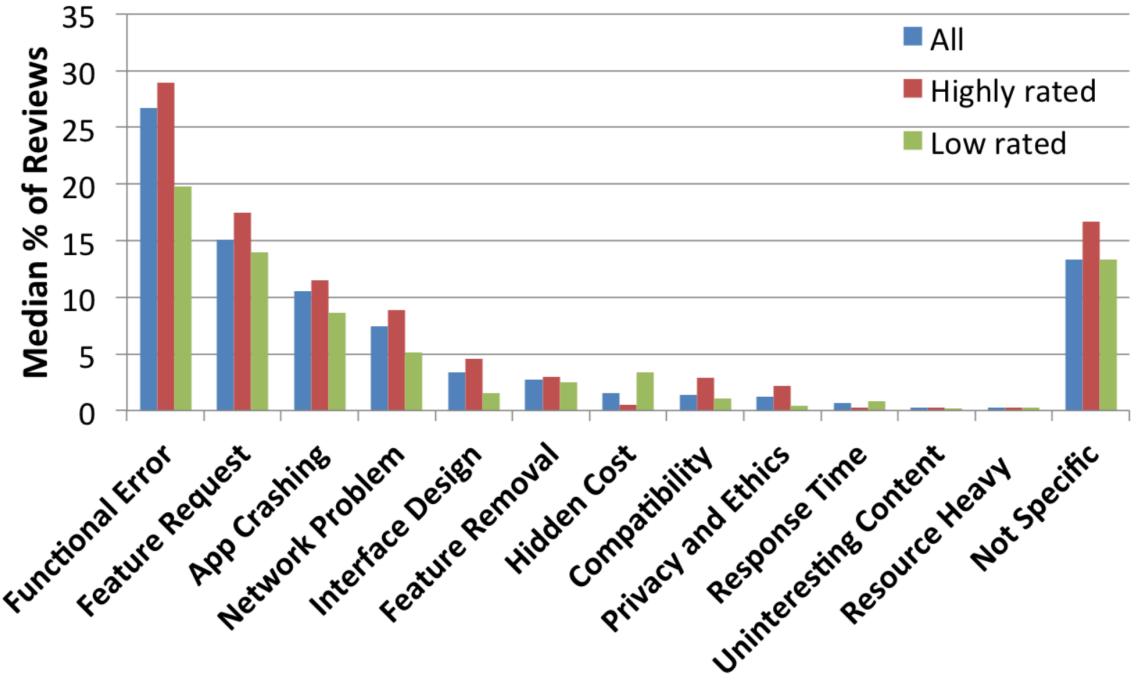


Figure 3.2: The most frequent complaints in user reviews

Figure 3.3 illustrates these results.

'Hidden Cost' is the second most negatively-perceived complaint that indicates the dissatisfaction of users with the hidden costs needed for the full experience of an app. This complaint showed up in 15 out of the 20 studied apps. While the apps we studied are called free apps, the term 'free' only refers to downloading the apps for free – and not necessarily using them for free. We find that when an app is free to download but not free to use, users are disappointed and often end up giving low-rating reviews. This suggests that the trust between the developers and users is extremely important. For example, the 'Hulu Plus' app is free to download, but has a monthly subscription cost and ads in streaming videos. Because of the need for a monthly subscription, over 55% of the low-rating reviews for Hulu were about

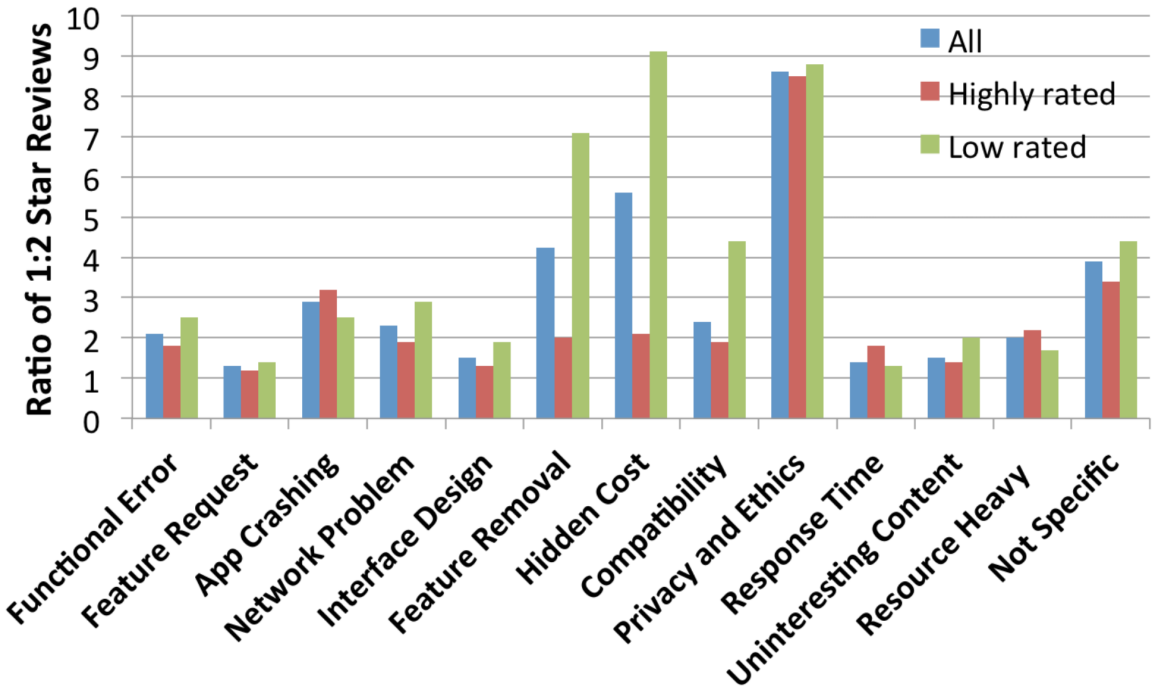


Figure 3.3: The most impactful complaints in user reviews

the hidden costs. On closer examination, we find that the problem is that of a poor description of the app by the developer and/or a misunderstanding by the user.

Developers should devote extra attention to the 'App Crashing', 'Hidden Cost' and 'Feature Removal' complaints as they occur frequently *and* they are negatively-perceived by iOS users (see Table 3.3).

Privacy and Ethical, Hidden Cost and Feature Removal complaints are the most impactful complaints and are mostly found in 1-star reviews. For developers, this finding stresses the importance of establishing trust and expectations with the app users.

Table 3.4: Android apps used in this study with the number of total and sampled reviews

	App Name	Category	Rating	Total Low Reviews	Sampled Reviews
High-rated Android apps	Adobe Photoshop Express	Photo & Video	4.0	960	275
	Facebook	Social Networking	3.6	948	274
Low-rated Android apps	Hulu Plus	Entertainment	3.1	782	258
	Weight Watchers Mobile	Health & Fitness	2.7	790	259

3.3 Generalizing the Results

All of our analysis thus far used reviews for iOS apps. We want to see whether we reach similar findings for the apps which are developed for multiple platforms. In particular, we would like to examine whether the rankings and the negatively-perceived complaints are different on the Android platform. Below we describe our approach for collecting and analyzing reviews for Android apps.

3.3.1 Comparison with Android apps

Google Play is the main hub for Android app reviews. Due to the way that *Google Play* publishes reviews (i.e., each review page loads with AJAX, requiring a browser client), we were not able to use a simple web crawler which we used for the iOS apps. Therefore, we extend our crawler to use *Selenium*, a web automation and testing tool [56]. *Selenium* allows us to mimic a browser session and advance through each page containing the Android app reviews. We parse and extract the same information from each review as we did for the iOS apps.

To determine the effect of the platform on the ranking and the negatively-perceived complaints, we re-do the analysis on four different apps that are developed for both, iOS and Android. Two apps, i.e., 'Adobe Photoshop Express' and Facebook, represent

the high-rated apps, whereas, 'Hulu Plus' and 'Weight Watchers Mobile', represent the low-rated apps. Table 3.4 shows the Android apps that we picked along with the total reviews collected and the sampled reviews. In total, we tag an additional 1,066 Android app reviews.

The most frequent complaint types are different for iOS and Android: Table 3.5 presents the rank and median percentage of reviews for all complaint types. We also show whether these rankings are statistically different. We find that Android seems to have more 'Resource Heavy' complaints than iOS. Also, we find that 'Resource Heavy' is statistically different across the two platforms. In addition, we notice that iOS apps have more 'App Crashing' and 'Hidden Cost' complaints than Android.

Many negatively-perceived complaint types are different on iOS and Android: Table 3.6 shows the ratio of 1 to 2 star ratings for each complaint type. We ignore all complaint types with less than 10 reviews, since we cannot reach conclusions with any confidence because of such a low number of reported reviews. The top three complaints that are most negatively-perceived in Android are related to compatibility issues, hidden costs and functional errors, whereas, for iOS the top three complaints are related to 'Hidden Costs', 'Compatibility Issues' and 'Feature Removal'. We find that 'Functional Error', 'Hidden Cost', 'Network Problem', 'Feature Removal', 'Privacy and Ethical', 'App Crashing', and 'Feature Request' complaints have a statistically significant difference across the two platforms.

Frequent complaints can have low negatively-perceived impact, while high negatively-perceived complaints can be infrequent: It is interesting to note

Table 3.5: Comparison of the most frequent complaints across different platforms (stat. signif. codes: '***'=0.001 '**'=0.01 '*'=0.05)

	4 Android Apps		4 iOS Apps	
Complaint Type	**Rank**	**Median**	**Rank**	**Median**
Functional Error	1	32.05	1	30.63
Feature Request	2	15.63	3	16.39
Network Problem	3	13.35	4	9.98
Resource Heavy*	4	5.68	11	0.13
App Crashing	5	4.96	2	16.83
Interface Design	6	2.58	8	1.56
Unresponsive App	7	2.37	12	0.00
Compatibility	8	2.00	7	1.68
Feature Removal	9	1.85	6	1.93
Privacy and Ethical	10	0.75	9	0.31
Hidden Cost	11	0.58	5	5.66
Uninteresting Content	12	0.18	10	0.13
Not specific	-	8.25		19.41

that 'Functional Errors' are ranked highly in terms of frequency and negatively-perceived impact for Android, whereas in iOS, 'Functional Errors' are ranked highly in frequency but lower in terms of negatively-perceived impact. Also, 'Compatibility Issues' have a high rank in terms of frequency and negatively-perceived impact in Android, whereas in iOS, compatibility issues have a low-frequency but high negatively-perceived impact. Such observations show that apps developed for different platforms have different issues and are perceived differently by these issues. 'Feature Request' while a frequent complaint on both platforms, does not have strong negatively-perceived impact on either.

We find that Android versions of the apps have more 'Resource Heavy' complaints than their iOS counterparts. 'Compatibility', 'Functional Error' and 'Hidden Cost' are the most negatively-perceived complaints for Android, whereas, 'Hidden Cost', 'Compatibility' and 'Feature Removal' are the most negatively-perceived complaints for iOS.

Table 3.6: Most negatively-perceived factors across platform ('-' = low confidence items, stat. signif. codes: '***'=0.001 '**'=0.01 '*'=0.05)

	4 Android Apps			4 iOS Apps		
Complaint Type	**Rank**	**# Reviews**	**1:2 star**	**Rank**	**# Reviews**	**1:2 star**
Compatibility	1	152	3.00	2	29	6.25
Functional Error***	2	362	1.28	5	358	2.31
Hidden Cost***	3	18	1.25	1	229	15.36
Unresponsive App	4	26	1.17	-	8	0.33
Network Problem***	5	206	0.94	4	140	4.83
Resource Heavy	6	64	0.83	-	2	0.00
Interface Design	7	38	0.81	7	21	2.00
Feature Removal***	8	32	0.78	3	67	5.70
Privacy and Ethical*	-	7	0.75	-	6	0.00
App Crashing***	9	102	0.62	6	222	2.17
Feature Request***	10	236	0.61	8	214	1.30
Uninteresting Content	-	2	0.00	-	5	1.50
Not Specific***	-	95	1.16	-	262	5.89

3.4 Discussion

While reading through the complaints, we notice that for many of the complaints, users also report that they recently updated their app. Hence, we want to study what appears to be a relationship between updates and complaints. This can help developers prioritize regression testing for iOS apps. Then, we discuss the relevance of different types of complaints to the various stakeholders of a software project (e.g., developer vs. project managers).

3.4.1 Complaints related to app updates

It is important to mention that we can only know if a complaint is post-update if the user mentions it in the review; other complaints could be because of an update as well.

We find that ∼11% of the sampled reviews mentioned that the recent update

impaired existing functionality. In 22% of these reviews, the users mentioned 'Functional Error' complaints after updating their app. Most of these complaints are app specific. For example, in the review below a user is facing a bug that affects the function key of the 'Adobe Photoshop Express' app:

Useless now: Was very useful till last update... function keys no longer appear during editing

We also find that 18.8% of reviews after a reported update include requests from users for a new or a previously removed *feature*. We also found that 18.2% of post-update reviews complained about the frequent *crashing* of the app.

Developers often release free apps in hopes of eventually monetizing them by transforming free content/features to paid ones. We find that 6.8% of post-update complaint reviews report complaints about this *hidden cost*. Another important complaint that users report with recent updates is related to changes in the *interface design*. We find that 6.2% of post-update complaint reviews report complaints about the user interface.

Based on these findings, we recommend that developers pay special attention (e.g., via regression testing and user focus groups) to features that they might consider removing, to adding additional fees, and to user interfaces changes that they might plan to introduce in an update, since these seem to be some of the more common complaints of iOS app updates. Thus, even if a user previously liked an app, a bad update could be irritating enough to make them give the app a low-rating.

In ∼11% of the sampled reviews, users attributed their complaints to an app update – highlighting the importance of regression testing in mobile development.

3.4.2 Lessons for Practitioners

Since users review apps as a whole, they often raise issues that are not directly the responsibility of the developers; some complaints are directed towards product managers or other team members. To identify these stakeholders, we divide these complaints into three different categories: developer, strategic and content issues.

Developer issues are complaints that are directly related to developmental issues. These issues include 'Apps Crashing', 'Functional Error', 'Network Problem', 'Resource Heavy', and 'Unresponsive App' complaints and accounted for 45.6% of all complaints. Hence, many of the complaints are directly related to problems that developers can address.

Strategic issues are complaints that primarily concern project managers, but could partially target developers as well. These issues include 'Feature Removal', 'Feature Request', 'Interface Design' and 'Compatibility' complaints and makeup 22.7% of all complaints. Strategic issues require a greater knowledge about the project and priorities, and usually do not have a straightforward solution.

Content issues encompass complaints about the content or value of the app itself – developers have little or no control over these issues. These issues include 'Privacy and Ethical', 'Hidden Cost' and 'Uninteresting Content' complaints. Addressing these issues requires rethinking the core strategy of the app (i.e., business model or the content offered). While these issues account for only 3.02% of all complaints, 'Privacy and Ethical' (1:2 star ratio of 8.56) and 'Hidden Cost' (1:2 star ratio of 5.63) issues are the most negatively-impactful complaints.

3.5 Threats to Validity

3.5.1 External Validity:

Our study was performed on a sample of 20 iOS apps. Hence, our results about complaints may not generalize to all iOS apps. To mitigate this threat we maximized the coverage of complaints by studying apps which cover most of the categories in the App Store.

3.5.2 Internal Validity:

The author of this thesis manually tagged the reviews. During this process, human error or subjectivity may have lead to incorrect tagging. This threat was addressed by random inspection of the reviews and the corresponding tags by the second and third authors of this chapter.

3.6 Conclusion

Individual developers and organizations that develop iOS apps are strongly impacted by reviews since low-ratings negatively reflect on the quality of their apps, and thus affect the app's popularity and eventually their revenues. To compete in an increasingly competitive market, app developers must understand and address the concerns of their users. In this study we identify 12 types of complaints and calculate the frequency and impact of each complaint type. *Our findings can help developers better anticipate the complaints and prioritize their limited QA resources towards the most frequent and/or impactful complaints.* At the same time, our findings point to new Software Engineering research avenues, such as the effect of ethics, privacy and

user-perceived quality on mobile apps.

Chapter 4

Prioritizing The Devices To Test Your App On: A Case Study Of Android Game Apps

Summary *– Star ratings from mobile app users directly impact the revenue of its developers [8]. At the same time, for popular platforms like Android, these apps must run on hundreds on devices increasing the chance for device-specific problems. Device-specific problems could impact the star rating assigned to an app, given the varying capabilities of devices (e.g., hardware and software). To fix device-specific problems developers must test their apps on a large number of Android devices, which is costly and inefficient. We mine the reviews of 99 free game apps and find that, apps receive reviews from a large number of devices: between 38 to 132 unique devices. However, most of the reviews (80%) originate from a small subset of devices (on average, 33%). These findings indicate that focusing on the devices with the most reviews (in particular the ones with negative star ratings), developers can effectively prioritize their limited QA efforts, since these devices have the greatest impact on star ratings. Furthermore, we find that developers of new game apps with no reviews can use the review data of similar game apps to select the devices that they should focus on first.*

Finally, among the set of devices that generate the most reviews for an app, we find that some devices tend to generate worse star ratings than others. We show that these findings also hold for apps in four other categories of the Google Play market.

Usage of Android devices has grown at a tremendous rate over the past few years [9]. To capitalize on this growth, both small and large companies are developing an enormous amount of apps, designed to run on Android devices. However, the top-rated or the featured apps in the app markets, are the apps with the most downloads, and hence the most revenue [8, 57]. Also the app market is very competitive, especially for game app developers who have to compete with almost 120,000 game apps already in the Google Play store – more than any other category of apps. To compete in this environment, developers need to get (and maintain) good star ratings for their apps [8]. This can be difficult since users are easily annoyed by buggy apps, and that annoyance could lead to bad star ratings [58, 59]. Hence, app developers need to test their apps thoroughly on different devices to avoid a poor rating.

To make matters worse, there exists a large number of Android devices, each with its own nuances. In fact, dealing with device specific issues of (the many) Android devices is considered one of the biggest challenges developers face when creating an Android app [60]. A 2013 survey from Appcelerator, which has aggregated results from similar such surveys in the past three years, shows that developer interest in Android has fallen to 79% in 2013 from a high of 87% in 2011 [61]. A staggering 94% of the developers that avoid working on Android apps cited Android fragmentation as the main reason [62]. Android fragmentation refers to the concern, that since there are many devices with different screen sizes, different OS versions, and other hardware

specifications, an app that functions correctly on one device might not work on a different one [15]. Joorabchi *et al.* [60] examined the challenges in mobile application development by interviewing 12 mobile developers. One of main the findings of their study is that dealing with device specific issues (e.g., testing these devices) remains a major challenge for mobile app developers. Even Google suggests that developers should test their apps on actual devices before they release the app [38]. There are now even business solutions based on providing devices remotely for testing [40]. However, with costs ranging in approximately a dollar for every 15 minutes of device time [63], the total expense incurred to developers can get very high. These concerns are especially worrisome for game app developers since they have to manually test their apps on-device instead of just relying on automated tests; due to the graphical and non-deterministic nature of video games [64].

Therefore, Android developers (and game developers in particular) need to carefully prioritize their testing and QA efforts on the most important devices. While there has been some previous research in automated testing for Android apps [33, 34, 36, 37], to the best of our knowledge, there has not been any work on prioritizing testing and QA efforts on the devices that have the most impact on the star rating of an app.

We coin the term 'review share', which measures the percentage of reviews for an app from a specific device. For example, a review share of 10% means that a specific device gave 10% of all reviews for an app. We use 'review share' to demonstrate the importance of focusing QA efforts on a smaller subset of devices. Through a study of 89,239 reviews for 99 free game apps from various Android devices, we explore the following research questions:

RQ1. What percentage of devices account for the majority of reviews?

We find that on average 33% of the devices account for 80% of all reviews given to a free game app. With this information, app developers can prioritize their testing and QA efforts by focusing on a small set of devices that have the highest potential review share.

RQ2. How can new developers identify the devices that they should focus their testing efforts on?

We find that developers can use the list of devices with the most review share in all other gaming apps as a good indicator of which devices they should focus their testing and QA efforts on.

RQ3. Do the star ratings from different Android devices vary significantly?

By examining the reviews from different devices for the same app, we find that some devices give significantly worse star ratings. Developers can take corrective actions for such devices or remove support for them if they see fit.

Takeaway: Developers can better prioritize their QA efforts by picking the devices that have the most impact on the star ratings of apps. Some of these devices may give worse star ratings than others – developers can either allocate additional QA resources for these devices, or remove support for them.

The remainder of this chapter is organized as follows: Section 4.1 discusses in detail the data that we analyze in this chapter. Section 4.2 presents the results of this chapter. Section 4.3 performs further analysis on paid apps, and apps in other categories, to see if our findings generalize. Section 4.4 discusses the lessons from our

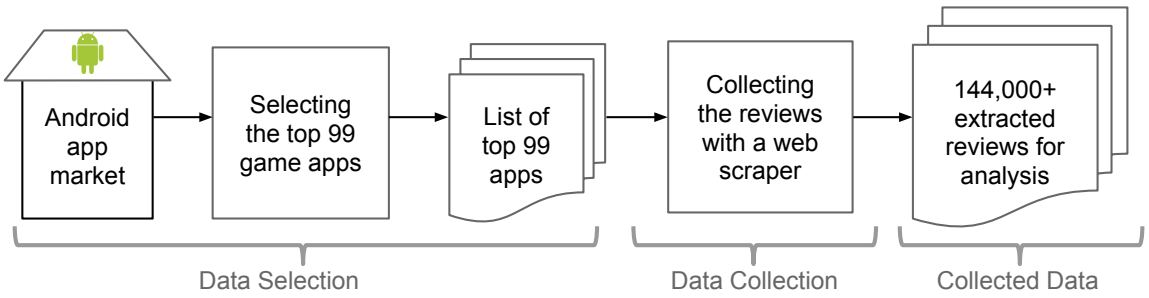


Figure 4.1: Overview of our process

findings for developers. Section 4.5 discusses the potential threats to the validity in this chapter. Section 4.6 concludes the chapter.

4.1 Study Design

In this section we discuss the data used in this chapter, the main sources of this data, and the collection process for this data. The main data used in this chapter are the reviews of the top 99 Android game apps. We collect these reviews from Google Play, which is the main market for Android apps. After collecting these reviews, we identify the devices that these reviews were produced from, as well as, the star ratings associated with each review. Figure 4.1 provides an overview of the process used in this chapter. The following sections describe the data used in this chapter and our data collection method in further detail.

4.1.1 Data Selection

We selected the top 99 free game apps as ranked by Google Play. Our proposed method is general (i.e., other apps can be examined – See Section 4.3 for a discussion about our analysis across the different categories of the market). Nevertheless, we

picked these top game apps for two reasons (a) since game apps are the most popular apps in the Android market, and thus our results could have the greatest impact, and (b) top game app developers like Red Robot Labs, Pocket Gems, Storm8, and Animoca (More than 400 millions app download across their app portfolio), in an article on TechCrunch discussed the issues they are having with testing their apps on many devices [39]. Currently they rely on their experience (and app analytics data when available) for choosing 30-50 devices to test on.

4.1.2 Data Collection

To collect the reviews, we build a web crawler using a web automation and testing tool called Selenium [56]. Our crawler extracts data such as the app name, the review title, the review description, the device used to review the app, and the numerical star rating of the review. This crawler opens a new instance of the Firefox browser, visits the URL of one of the selected apps, and then clicks through the review pages of this app. The required information is located on each page using Xpath then stored into a database [65]. We used a similar crawler to collect the URLs of the top apps. This crawler is much more limited and slower than typical web-crawling since Google Play puts heavy restrictions on crawlers. We found this crawling approach to be the only reliable method for extracting the reviews. All of the crawling is done using an anonymous user.

Also, Google Play limits the total number of reviews that a user can view to a maximum of 480 for each of the star ratings (i.e., a user is restricted to viewing a maximum of 2,400 reviews for each app as there are 5 levels of star ratings (5*480)). Recent work by Pagano and Maalej shows that there exists a large amount of noise

in reviews, for example one word reviews [19]. Hence even Apps Markets have moved to using more complex methods to calculate a global star rating or ranking of an app instead of simply summing up all the reviews [66]. We use 'the most helpful reviews' as they are considered by App stores and users, as one of the most reliable sources of user feedback. The helpfulness is determined by other users voting for reviews. Such crowd-based filtering helps weed out spam reviews.

Summary of Collected Data In total we collect 144,689 reviews (across the 5 levels of star rating) from the studied apps. From this set of reviews, we only consider the ones that have a device associated with them. This reduces the set of reviews to 89,239, each submitted by a unique user. The implications of our review selection is discussed in more detail in Section 4.5.2. We limit our reviews to those given only after October 2012 to January 2013, a 3-month window since the rate of churn in devices is very high.

4.1.3 Preliminary Analysis

Prior to delving into our research questions, we perform some preliminary analysis on our dataset as a whole, i.e. using data from all 99 game apps taken together. We perform this analysis to determine whether our review dataset contains reviews from many different devices or a small set of devices. In other words, we would like to determine if Android fragmentation does exist in our data or not. For this, we look at how the reviews are distributed across the devices, for the apps taken as a whole. Using all of the reviews, we identify the number of reviews that each device gave.

In total, our dataset contains 89,239 reviews from 187 unique Android devices.

Out of these 187 devices, 114 of these devices have provided more than 100 reviews. These facts highlight the magnitude of the Android fragmentation problem and the importance of identifying the devices that give the most reviews to apps, for prioritizing QA efforts.

4.2 Results

Now that we have determined that our dataset contains reviews from many different Android devices, in this section we answer our research questions.

RQ1) What percentage of devices account for the majority of reviews?

Motivation: As mentioned earlier, Android fragmentation is a major concern of developers who are seeking to develop high quality Android apps [16, 60, 62]. There may be hundreds of devices that developers may need to test their apps on. Testing on such a large number of devices is not feasible for developers with limited time and budget. Therefore, our goal is to determine what percentage of devices account for the majority of reviews.

Approach: To answer this question, we use the reviews that we collected for the 99 free game apps. For each review, we determine the device that the review was posted from. Then we calculate the 'review share' for each device. We define 'review share' as the percentage of reviews from one device compared to the total number of reviews from all devices. For example, a review share of 10% means that a device gave 10% of all reviews. Initially we consider the reviews from all apps taken together. We then determine what percentage of devices is required to cover X% of the reviews. In our

case, X varies from 0 to 80%.

We also examine the results by breaking down the reviews by star ratings. Instead of looking at individual star ratings, we combine 1 and 2-star reviews into a group which we call 'bad reviews', and combine 4 and 5-star reviews into another group which we call 'good reviews'. We label 3-star reviews as 'medium reviews'. We then calculate the review share for each device, when we exclusively consider only bad, medium or good reviews.

To ensure that our grouping makes sense, we run a sentiment analysis tool over the text of the reviews in these groups to validate that these groupings are appropriate (i.e., 'good reviews' actually have the most positive reviews and vice versa) [67]. This tool assigns an integer to the sentiment expressed in the reviews (where a negative integer represents a negative review). In our case, the 'negative reviews' group was assigned a score of -0.32, the 'medium reviews' group was assigned a score of 0.44, while the 'good reviews' group was assigned a score of 1.25. These scores support our grouping scheme.

Findings: Figure 4.2 shows the percentage of devices (x-axis) vs. the cumulative percentage of reviews (y-axis) in the different star rating groups (different coloured curves). From this figure, we find that *20% of the devices account for approximately 80% of the posted reviews.* This finding suggests that developers working on free game apps, only need to focus their testing efforts on 20% of the devices to cover the majority (i.e., approximately 80%) of the reviews.

Observing Figure 4.2, we note that the bad, medium and good star ratings curves are very similar. After comparing how different devices gave bad, medium and good

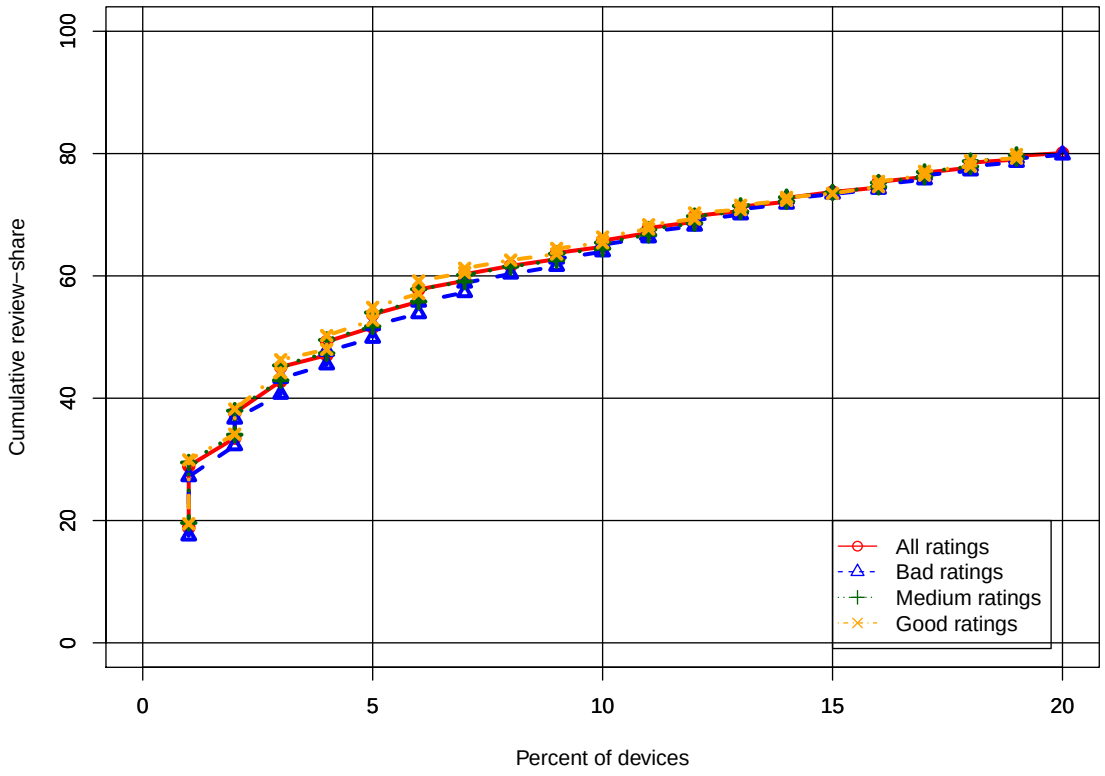


Figure 4.2: Percent of Android devices used to give X% (X ranges from 0 - 80) of the reviews for free game apps

reviews, we find that while the devices that gave the bad star ratings were fairly identical to the devices that gave good star ratings, there were a few discrepancies. The set of devices that had a cumulative sum of 80% of the bad star ratings, had four devices that were not in the set of devices that gave most of the good reviews. This finding suggests that there is additional variation in how specific devices rate these game apps.

Discussion: From RQ1, we know that a few devices account for most of the reviews

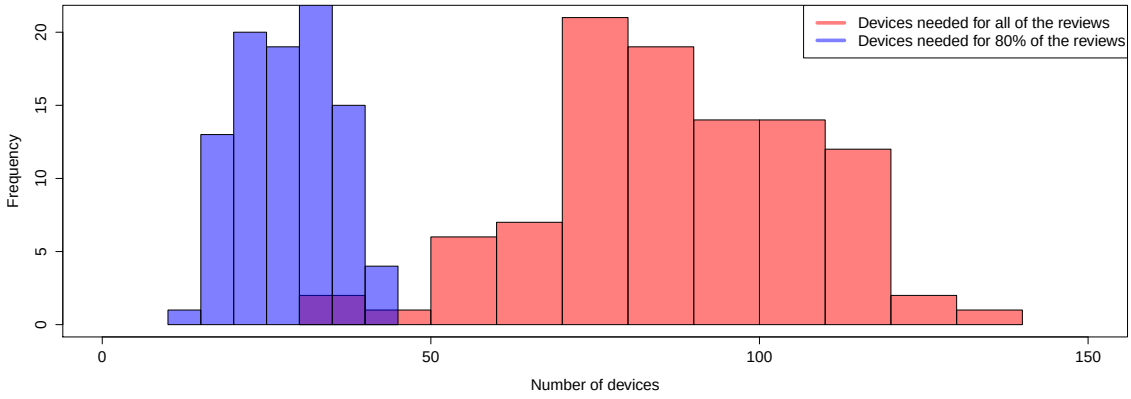


Figure 4.3: Number of Android devices that account for 80% of the reviews (left purple histogram), compared with the total devices that review the app (right red histogram)

in an app. We wanted to dig deeper and find if a similar trend exists at the app level as well, i.e., for each app, if most of the reviews came from a small subset of devices.

The two histograms in Figure 4.3 compare the number of total unique devices that rate an app, with the number of devices that account for 80% of the reviews for an app. The right red histogram shows the number of unique devices that rate each of the 99 game apps. The minimum number of devices is 38, the median is 87 devices, and while 132 is the maximum number of unique devices that rate an app. This finding clearly indicates that fragmentation exists even at the app level.

The left purple histogram in Figure 4.3 show the number of devices that account for 80% of the reviews. As this figure shows, this number is much lower than the total number of unique devices. We find a minimum of 13, a median of 30, and a maximum of 45 devices accounted for 80% of the reviews, per app. While these numbers may seem large, one would have to keep in context that Android developers often have to think about testing their apps on hundreds of devices, otherwise.

To get a better idea of the comparison above, we calculate the percentage of

devices that account for 80% of the reviews in each of the 99 free game apps. We do this on a per app basis. We observe that, 80% of the reviews can be addressed by considering a minimum of 22% of the devices, and at most 53% of the unique devices. The average percentage of devices required to cover 80% of the review share is 33% (and the median is 32%). This finding indicates that app developers can cover the majority (i.e., 80%) of the reviews by focusing their QA efforts on 33% of the devices on average.

A small set of devices are needed to cover 80% of the reviews. On average, 33% of all devices account for 80% of reviews given to free game apps.

RQ2) How can new developers identify the devices that they should focus their testing efforts on?

Motivation: Thus far, we have shown that a small percentage of devices make up the majority of reviews. An implication of this finding is that, if developers carefully pick their set of focus devices, then they can maximize the effectiveness of their QA efforts. To illustrate, consider a scenario, where a team of developers is working on a new free game app but they can only afford to buy 10 devices to test their app on. Identifying the optimal set of devices is even more important for such developers with limited resources who can only afford a few devices.

One method of picking these devices is to aggregate the review data of every high-rated app in a category, and select the devices which would lead to the most review share. This method can provide developers a working set of devices to start from, which they can later augment with other devices. In this RQ we examine the effectiveness of this method.

Using our review data, we generate Figure 4.4 which contains a stacked area chart

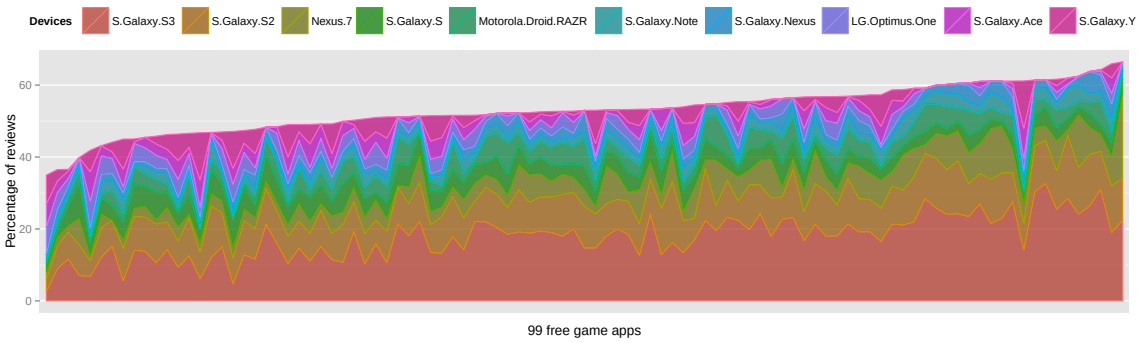
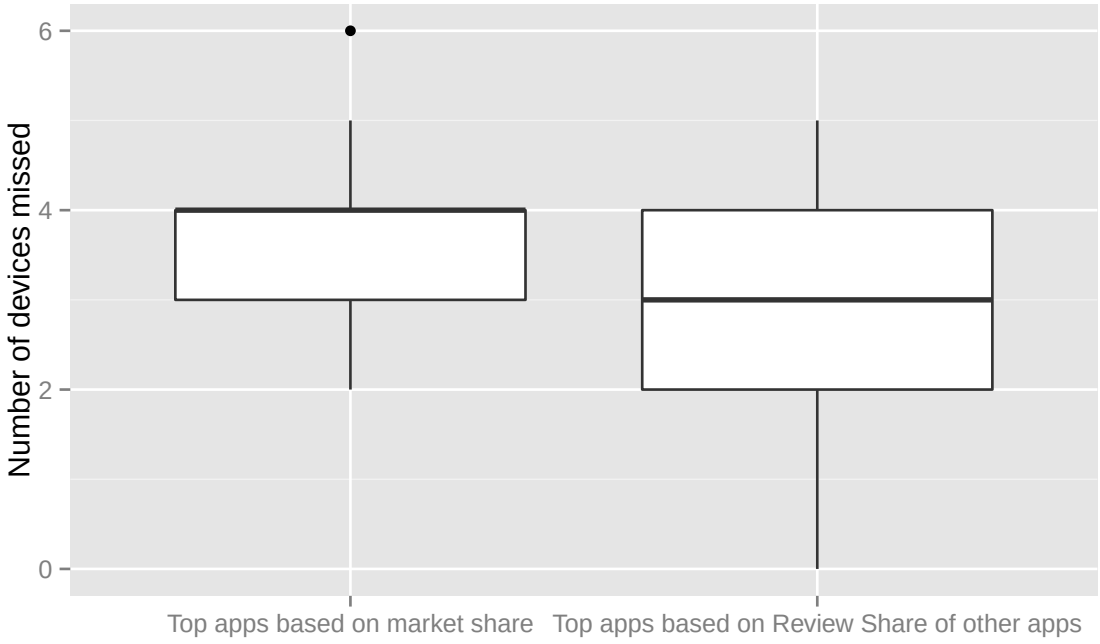


Figure 4.4: Percent of reviews of each app accounted for, by selecting the 10 devices with the most review share for all 99 free game apps taken together

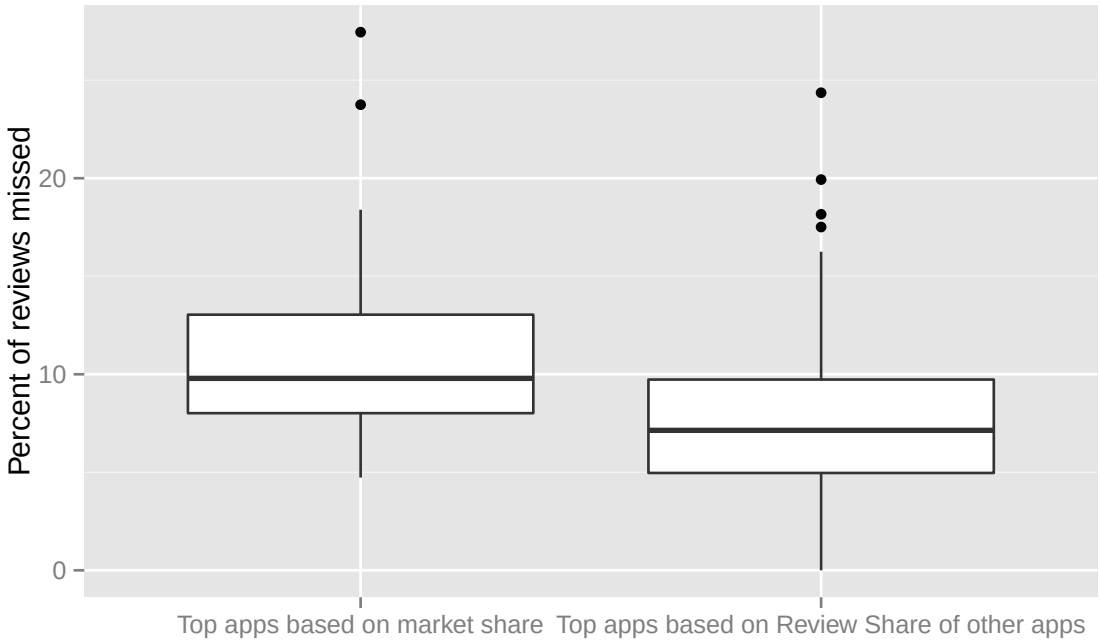
that shows the percentage of reviews that 10 devices with highest review share would cover. This figure shows that even a small number of devices, if carefully chosen (i.e., by looking at which devices frequently post reviews for apps), can account for a considerable number of reviews – more than half of the reviews in most cases.

This leads to the question - how can developers without any reviews for their app pick the best set of devices that they should focus their QA efforts on? For example, can a developer use the review share data from all the high-rated apps that are currently present in the game app category? Should they just pick the most popular devices? We explore this issue in RQ2.

Approach: To answer this RQ (working with our dataset of 99 free game apps), we identify a set of 10 devices with the most review share for each app, and compare this set with the set of 10 devices with the most review share in the remaining 98 apps combined. Doing so allows us to simulate an app developer who picks the 10 devices that provide the most reviews for other apps, and using these devices to test his or her own app. Since we have the reviews for that appas well in our dataset, we use



(a) Number of devices missed



(b) Percent of review share missed

Figure 4.5: Number of devices and percent of review share missed if 10 devices with the most reviews for all the remaining apps are chosen

these reviews to measure the review share that would be covered for an app based on the selected 10 devices. In other words, this analysis allows us to identify the devices that were in the set of devices that gave the most reviews for an app, but not in the set of devices that gave the most reviews for the rest of the group (essentially a set difference). If such devices are identified, we sum the review share of these devices to highlight the review share that the developer would have lost/missed if he or she had picked the 10 devices with highest review share among all the other 98 apps in that category. These results are shown in Figure 4.5.

Findings: Using our proposed method to prioritize the devices works well overall. In the majority of the cases the developer will identify the 7 most review-producing devices. Figure 4.5(a) (right box plot) shows the distribution of these missed devices for the 99 free game apps. We observe that the median number of missed devices is 3 devices, with a max of 5 devices and a min of 0.

Now, we want to know what percentage of the review share would be impacted due to these missed devices. Figure 4.5(b) (right box plot) contains a box plot that shows the percentage of review share impacted due to the missed devices. The median of missed review share is 7.1%. This is not a large loss in review share, especially given that, using this method (i.e., using devices that have a large review share in the app's category), developers will have the benefit of picking an adequate set of devices even before releasing their app to the market.

Discussion: While identifying the devices with the most review share within an app's category is useful, some developers may opt to pick the devices with the most

overall *market share* (which is posted on many mobile analytics websites, e.g., App Brain [68]) to prioritize their QA effort. Market share is generally determined by the number of active Android devices.

To compare this method of simply using market share to our proposed method, which uses the reviews, we compare the devices with the most market share to the devices that have the most review share for each of the game apps. We obtain the list of devices with the most market share from App Brain, which gives us a list of the 10 devices (shown in Table 4.1) for our studied time period (October 2012 to January 2013).

Figure 4.5(a) (left box plot) shows the distribution of missed devices for each the 99 free game apps, when market share devices are compared with the 10 devices that have the highest review share for the corresponding app. We find that the median number of devices missed if we consider the market share devices is 4, which is higher than if we use review share in the app's category. Moreover, Figure 4.5(b) (left box plot) shows the distribution of the percentage of review share impacted due to the missed devices. Once again, we see that the median value is 9.8%. This rate is higher than the median if our method was used (which has a median of 7.1%). The difference in the number of missed devices and missed review share between choosing our review-share method and the market-share method is statistically significant (p-value < 0.01 for a paired Mann Whitney U test). Our findings suggest that simply using the market share is not sufficient, and using our method, which uses reviews from apps in the same category can identify devices that have a greater chance to review the app. Thus by using our method, a developer can improve the effectiveness of their device prioritization efforts, since they will be able to identify devices that

Table 4.1: The market share of the most popular Android devices for our studied time period (October 2012 to January 2013)

Top market share device	Market share (%)
Samsung Galaxy S3	9.4
Samsung Galaxy S2	7.8
Samsung Galaxy S	2.6
Samsung Galaxy Ace	2
Samsung Galaxy Note	1.9
Samsung Galaxy Y	1.9
HTC Desire HD	1.5
Asus Nexus 7	1.1
Samsung Galaxy Tab 10.1	1.1
Motorola Droid RAZR	1.1

have a greater impact on the star ratings of an app.

While examining the 10 devices with the most review share in the game category, we notice not all of the devices rate apps the same way. This makes us wonder if some specific devices give worse star ratings than others, and thus need special attention from developers. We explore this question next in RQ3.

Devices with the most review share across all existing high-rated game apps are a good indicator of which devices are likely to have a large review share for each game app. Using this list of devices, a developer can focus their QA efforts even before they release the first version of their app.

RQ3) Do the star ratings from different Android devices vary significantly?

Motivation: Since different devices have varying specifications, it could be the case that users of the different Android devices perceive and rate the same app differently. Understanding how different devices rate apps will allow developers to understand why their apps were given the star ratings that they receive (i.e., is a device issue or an app issue). If different devices give varying star ratings, this would imply that not all reviews of apps should be treated the same way. With this kind of information,

the app developer can do one of two things: 1) they can either allocate even more QA efforts (e.g., testing or focus group) to devices that give a poor rating, when the revenue from that device is critical or 2) if there are not enough users of the app on the particular device, then developers can manually exclude these devices [69], from the list of supported devices on Google Play. In either case knowing if certain devices give worse star ratings than others will help developers prioritize their QA efforts even further.

Approach: We separately compare the bad (1 and 2-star reviews), medium (3-star reviews) and good (4 and 5-star reviews) reviews from the 20 devices with the most review share of the 99 free game apps. For example, to compare the star ratings of the devices that gave low-ratings to game apps, we create a table where the columns are the 99 game apps and the rows are a percentage of star ratings from the top 20 devices. Each cell in the table has a bad-ratings:all-ratings percentage (which is the number of 1 and 2-star reviews over the total number of reviews) given to an app, by a specific device. Table 4.2 is an example of such a table. For instance, the ratio 21:100 in the cell that corresponds to the row 'Samsung Galaxy S3' and the column 'Angry Birds', means that 21 out of every 100 star ratings that S3 devices gave to Angry Birds game were bad (i.e., 1 and 2-star). Similarly, 33 out of every 100 star ratings from the 'Samsung Galaxy S2' device for the 'Temple Run' game were bad.

To compare how the top review share devices give bad, medium and good star ratings, we use the Scott-Knott test [70]. The Scott-Knott test is a statistical multi-comparison procedure based on cluster analysis. The Scott-Knott test sorts the percentage of bad reviews for the different devices. Then, it groups the devices into two

Table 4.2: An example of the table used to compare bad star ratings given by devices

Device	Angry Birds	Temple Run
Samsung Galaxy S3	21:100	19:100
Samsung Galaxy S2	29:100	33:100

different groups that are separated based on their mean values (i.e., the mean value of the percentage of bad reviews to all reviews for each device). If the two groups are statistically significantly different, then the Scott-Knott test runs recursively to further find new groups; otherwise, the devices are put in the same group. In the end of this procedure, the Scott-Knott test comes up with groups of devices that are statistically significantly different in terms of their percentage of bad reviews to all reviews.

Findings: The 20 devices (which we examine in this RQ) are divided into 4 statistically significantly different groups. Table 4.3 shows the significantly different groups of devices as indicated by the Scott-Knott test. Table 4.3 also lists the devices that are in the group and the mean percentage of bad reviews for each of the devices.

Our findings show that indeed, the users of some devices such as the 'Motorola Droid X2' give more bad star ratings to apps than others. We find that this device has a significantly higher ratio of bad star ratings than the devices that give the least ratio of low-ratings to all star ratings (i.e., Samsung Galaxy Y). The Scott-Knott test also shows that this device has the lowest ratio for high-ratings. A reason behind these poor star ratings could be manufacturer specific problems. A recent study by Han *et al.* provided evidence of vendor specific problems when they compared bug reports of HTC devices with Motorola devices [15]. A report of this device from Android Police (an Android dedicated web blog) describes its sluggish performance and poor screen resolution [71]. The poor screen resolution may be the main issue with this device

Table 4.3: Scott-Knott test results when comparing the mean percentage of bad star ratings given from each device to free game apps, divided into distinct groups that have a statistically significant difference in the mean

Group	Device	Mean % of bad star ratings for the device per app
G1	Motorola Droid X2	45.79
G2	Droid Bionic	39.25
	Motorola Droid X	39.20
	HTC Sensation 4G	39.10
	HTC Evo 4G	39.03
	HTC Desire HD	36.81
	Samsung Galaxy Nexus	35.72
	HTC EVO 3D	35.53
	HTC One S	35.31
G3	Motorola Droid RAZR	33.51
	Samsung Galaxy S	33.26
	LG Optimus One	31.11
	HTC One X	32.76
G4	Samsung Galaxy Ace	30.02
	Samsung Galaxy Note	29.68
	Samsung Galaxy S3	28.19
	LG Cayman	28.17
	Samsung Galaxy S2	27.83
	Asus Nexus 7	26.90
	Samsung Galaxy Y	26.78

since most game apps require a good screen resolution and performance.

On the other hand, we notice that the users of some devices such as the 'Samsung Galaxy Y', 'Asus Nexus 7' and 'Samsung Galaxy S2' give less bad star ratings than other devices. To better understand the results, we further investigated the data to see whether the bad reviews were given to the same app or whether the bad reviews were given from different apps. We discover that the bad reviews are different for all devices (i.e., it is not the same apps which are receiving the bad star ratings across the different devices). For example, 92% of all star ratings given from the 'HTC Sensation 4G' device to the 'Tap Tap Revenge 4' game app are bad star ratings while

the median percentage of bad star ratings given by the same device is 37.5%. We also find that none of the other top devices gave this app such poor star ratings. Thus we see that this particular app just does not work well on the 'HTC Sensation 4G' device. On further examination, we find many complaints by users of this device on the forum of the developer of 'Tap Tap Revenge 4'. Examining the reviews from this device, we see that this app crashes after most major events (i.e., a song ending in the app) [72].

Discussion: Our findings imply that developers should be aware that a few devices may give significantly worse reviews than others. These bad star ratings may be given because the device itself provides a poor user experience (i.e., 'Motorola Droid X2'), or that the app itself does not work well on a device (as in the case of 'HTC Sensation 4G' device and the 'Tap Tap Revenge 4' app). In either case, developers aware of this finding can specifically address the concerns expressed in the reviews from such devices (e.g., do detailed testing) or remove the support for such devices. We are not suggesting that developers only need to test on devices that give statistically different star ratings than others; developers just need to devote specific attention towards problematic devices. Monitoring how different devices rate apps can reveal devices that are bringing down the overall star rating of an app. Additionally, our findings suggest that the user's perception of the quality of an app depends on the device on which it runs. Hence research on testing Android apps, should factor in the effect of the device.

Another possible reason why some devices give worse star ratings than others could be that those devices are simply older. Older devices also tend to have worse

hardware specifications than new devices so the performance difference may be the main reason for these bad star ratings. To test this theory, we identify the release dates of the 20 devices with the most review share. Then we do a correlation test of their release dates and the median of the percentage of bad star ratings to all star ratings for the 20 devices. Using the 'Spearman' correlation test we find a correlation of -0.61. Since the correlation is negative, it means that newer devices have a lower percentage of bad star ratings. Therefore, this result implies that when it comes to the top 20 devices, the age of the device may be a factor for bad star ratings. It is important to note here that what we are observing is a correlation, not causation.

For the devices that give the most reviews for game apps, we find statistical evidence suggesting that some of these devices give worse star ratings than others. Developers can take corrective actions for such devices by allocating more QA effort, or removing support for these devices if they see fit.

4.3 Generalizing the Results

We focus on free game apps since this lets us examine a very focused context, thus avoiding other confounding factors such as cost, functionality, and end user profile. We now wish to examine whether our findings generalize. First we compare our results of free game apps with paid ones. Then, we compare our results for different categories.

4.3.1 Comparison with Paid Game Apps

Although it is likely that many devices review the paid game apps as well, we believe that there will be a more even distribution of reviews among the devices. The move even distribution is because users of paid apps may be more inclined to give reviews,

since they paid for the apps [19]. To conduct our comparison, we collected all the reviews for the paid game apps. In total, we collected 61,996 reviews, of which 42,110 were associated with one of 159 devices.

Similar to RQ1, we identify the percentage of devices that are needed to account for the majority of the reviews given to the paid game apps. We use the same methods that we used in RQ2, to determine an ideal method to choose the top 10 devices for focussed QA efforts - based on the reviews from other apps, or based on market share. We use the same method that we used in RQ3 to identify if star ratings from different devices vary, for paid game apps.

What percentage of devices account for the majority of reviews? For paid game apps, we find that the median number of unique devices that review each paid game app is 50. Compared to free game apps, which have a median of 87 devices, we find that paid game apps have much fewer unique devices that rate an app. In terms of percentage of devices needed to account for 80% of the reviews on a per app basis, we find that while the median is the same as free game apps, the range of these percentages is more for paid game apps in comparison to free game apps. The minimum percentage is 16.7% whereas the maximum percentage is 69.2%.

Figure 4.6 shows the percentage of devices vs. the cumulative percentage of reviews in the different star rating groups for all the paid game apps taken together. Again, compared to free game apps, we find that much less devices are needed to account for 80% of the reviews. We find that only 12.6% of the devices are needed to cover 80% of the reviews (compared to the 20% for the free game apps). Our finding suggests that developers working on paid apps should be even more attentive of their

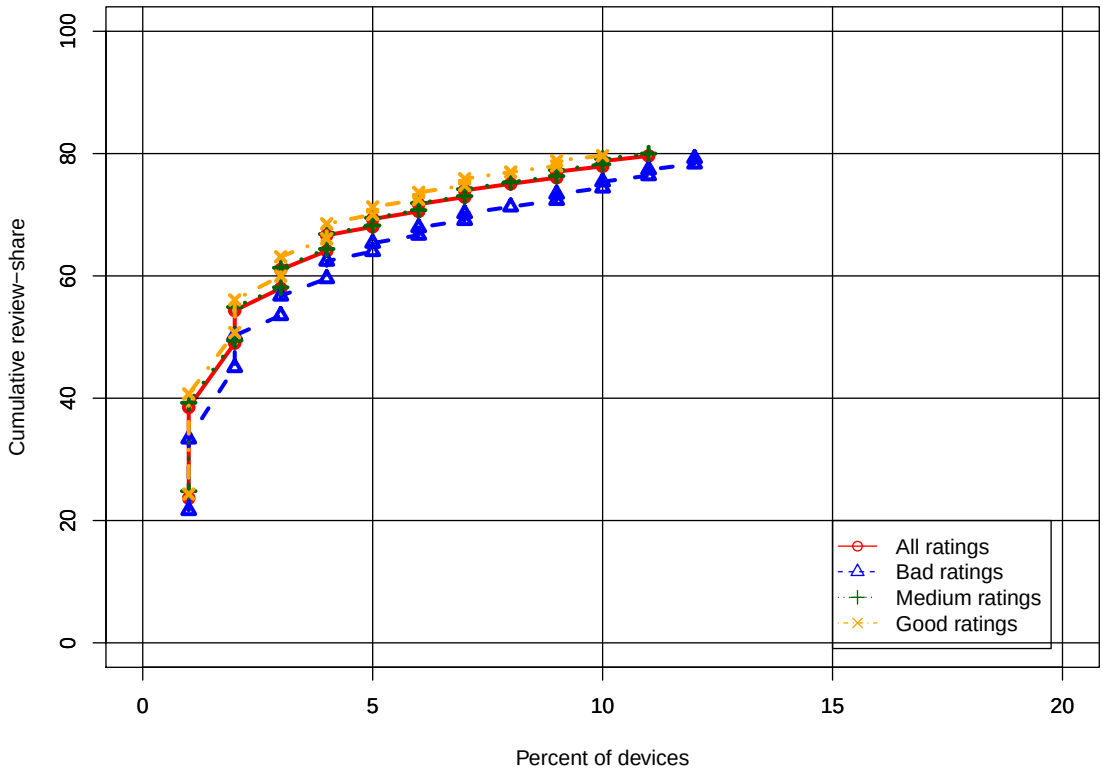


Figure 4.6: Percent of Android devices which contribute X% (X ranges from 0 - 80) of the reviews for all paid game apps

analytics since in some cases a select few devices have a huge impact on their star ratings, and hence their future revenue.

How can new developers identify the devices that they should focus their testing efforts on? From Figure 4.7(a) and Figure 4.7(b), we can see that it is indeed more beneficial to target devices based on the review share of the other paid game apps instead of using the market share, since we will be targeting devices that are used to review the apps more. The difference in both the cases is statistically

Table 4.4: Scott-Knott test results when comparing the mean percentage of low-ratings given from each device to free game apps, divided into distinct groups that have a statistically significant difference in the mean

Group	Device	Mean % of low-ratings for the device per app
G1	EeePad Transformer TF101	42.83
	Motorola XOOM	41.20
	HTC Evo 4G	40.32
	Samsung Nexus S	39.16
	Galaxy Tab 10.1	38.79
	HTC Desire HD	36.55
	Droid Bionic	36.26
	EeePad Transformer TF300	35.69
G2	Samsung Galaxy Note	33.55
	HTC Sensation 4G	33.25
	HTC EVO 3D	32.33
	Samsung Galaxy S	32.84
	SEMC Xperia Play	31.53
	HTC One S	31.29
	Motorola Droid RAZR	30.29
	HTC One X	29.29
	Samsung Galaxy S2	29.11
	Samsung Galaxy Nexus	29.04
G3	Samsung Galaxy S3	26.72
	Asus Nexus 7	22.63

significant (p-value < 0.01 for Mann Whitney U test). This result is similar to the result for the free game apps. However one noticeable difference is that the number of devices missed (median of 5 devices) and the review share missed (median of 15.9%) for paid apps when using the market share data is slightly higher than in the case of free game apps (where the median values are 4 devices and 9.8%). Thus we can see that in the case of paid game apps, the market share data is much less accurate in helping the developer identify the devices to test their app on first.

Do the star ratings from different Android devices vary significantly? For

the paid game apps, we illustrate the differences in the percentage of low-ratings from each device in Table 4.4. Our findings show that indeed, even for paid game apps, different devices provide different levels of low-ratings to apps. The Scott-Knott test groups the devices into 3 statistically significantly different groups. For example, the 'Asus Nexus 7' and the 'Samsung Galaxy S3' devices give significantly better ratings to paid game apps than many of the other devices (i.e. Motorola Xoom, EeePad TF101, HTC Evo 4G). We also find that the set of devices that give a higher percentage of bad star ratings in paid apps is different from the set of devices that give a higher percentage of bad star ratings in free game apps. For example, the Motorola Xoom and EeePad TF101 are not even in the top 20 devices that review free game apps. *Thus developers need to be careful about using free games apps to prioritize their QA efforts for paid game apps, as the devices that the users use for paid game apps do vary.* Next, we examine the devices that review apps in other categories (not just games).

Trends and results in paid game apps are similar to free game apps as well. However, the argument for prioritization is more pronounced in the case of paid game apps. Therefore paid game app developers can make optimal use of their QA budget by prioritizing their efforts based on the share of reviews from a device.

4.3.2 Analysis of Apps in Other Categories

While the findings of the study so far are most relevant for developers of game apps, we want to see if our findings hold for apps in other categories as well. More specifically, we want to examine if the reviews of apps in the Business, Education, Sports and Entertainment categories have similar patterns to those we found in our study – this can help other developers deal with Android fragmentation as well. Statistics about

Table 4.5: Reviews collected and number of devices for the other 4 categories of free apps

Category	# of Reviews	# Reviews Linked to a Device	# of devices
Business	21,365	13,901	153
Education	14,097	9,000	168
Sports	16,790	12,102	157
Entertainment	64,690	40,399	180

the data used to perform this analysis is summarized in Table 4.5.

What percentage of devices account for the majority of reviews? Figure 4.8 shows the percentage of devices vs. the cumulative percentage of reviews in the different star rating groups for each of the 5 categories. We find that when considered together, 21.1%, 22.2% and 22.6% of the devices account for 80% of the reviews given to apps in the Entertainment, Business and Education categories respectively. For apps in the Sports category, only 17.1% of the devices account for 80% of the reviews. These numbers are similar to the 20% in the free game apps category. Similar to game apps, these findings imply that developers working on these categories only need to focus on a small subset of devices to cover the majority of the reviews given to their apps. Our finding suggests that developers working on apps in categories other than games can also greatly improve their efficiency by focusing on the few important devices since these devices make up the majority of the reviews given to an app.

How can new developers identify the devices that they should focus their testing efforts on? Similar to our results for free game apps, we find that developers get more coverage of the reviews, if they focus on the devices with the most review share instead of the devices with the most market share. We find that by focusing

on the devices with the most review share, instead of the devices with the most market share, developers can gain an extra 7.69%, 8.51%, 6.48%, 7.91% review coverage on apps in the Business, Education, Entertainment and Sports categories respectively.

Do the star ratings from different Android devices vary significantly? We now compare how different devices review free apps in Entertainment, Business, Sports, Game and Education categories. Using the Scott-Knott test, we observe that the variation of star ratings from different devices is also present for the apps in these categories of apps. In each category, the Scott-Knott test divided the set of devices into four statistically significantly different groups based on the percentage of bad reviews to all reviews in an app. We note that many of the devices that give the most star ratings to apps in these categories vary in terms of their low and high-rating. We also note that the Asus Nexus 7 gave more low-ratings, and less high-ratings for the apps in the Entertainment category than it did in the game category. The fact that the Asus Nexus 7 gave more bad star ratings in the Entertainment category may be because it is a tablet and not all apps scale well to larger screens, indicating that some devices may rate apps in different categories with varying criteria. Our findings further suggest that developers working on Android must be attentive of their analytics as it could help them identify problematic devices.

After comparing the results of our RQs for the free game apps, with paid game apps and apps in four other categories, we find evidence that our results do generalize. Our finding suggests that developers working on apps in other categories can also use our methods to better prioritize their QA efforts.

When working on apps in the Entertainment, Business, Sports, and Education categories, developers only need to focus their testing efforts on a small set of devices to cover the majority of the reviews for their apps. Moreover, we find that many of the devices give different star ratings to apps in these categories.

4.4 Lessons for Developers

While we think that the results from this study will be useful for developers, device usage may have changed by the time most developers view this study. The change is because of the rapid growth and evolution of the mobile industry where newer devices are constantly being released [9]. Thus, developers should focus on our general findings and method rather than the device specific results.

For example, developers can take away the idea of prioritizing their QA efforts on a subset of impactful devices rather than all devices. Moreover, they can use our method of analyzing reviews per device to potentially identify devices that consistently give poor star ratings. Once developers identify these problematic devices, they can remove support for them to avoid additional QA and their bad reviews altogether. If developers feel that the additional downloads (which can generate high ad revenue) from these devices are worth potentially lower average star ratings, they can allocate additional QA resources for these devices.

While it is ultimately up to the developers to decide how they are going to prioritize their QA efforts, we think that focusing on the devices that have the most impact on the app's star ratings is an effective method (especially because star ratings are directly correlated to the number of downloads [8], and thus the revenue generated by apps).

4.5 Threats to validity

In this section we discuss the perceived threats to our work and how we address them.

4.5.1 Construct Validity

We compared the bad, medium and good reviews given from different devices to identify if certain devices give different (and worse) star ratings than other devices. Note that we are not raising a causal link here in this thesis. We are not claiming that an app gets a poor rating because of a device. We are just saying that apps get rated frequently and sometimes poorly from a small set of devices. This is similar to the vast literature on using software metrics for QA prioritization. Such literature do not claim a causal link between software metrics and software defects, but just suggest that software metrics like churn can be used to prioritize QA efforts. The underlying reason for poorer or more frequent reviews from a particular device could be the hardware specification of the devices, the OS running on the devices, or just that the people using a particular device may have a specific profile. More research has to be conducted to identify the underlying causes. Note that data on user profile or which OS version is running on a device is currently not available openly to be mined by researchers. Hence, a major data collection effort has to be launched to examine these underlying factors.

4.5.2 Internal Validity

Since we limited our reviews to only the reviews from a few months (i.e., October, 1st, 2012 till January, 15, 2013), our data may not accurately represent star ratings for the entire year or the entire life of devices. However, to mitigate this threat, we

made sure to apply statistical tests, where applicable, to ensure that our findings are statistically significant. Also note that we extract the device information from reviews. This information, as far as we can tell is very accurate, and cannot be faked, since a user cannot manually change this information when posting a review. The device information is automatically taken from the device from which the review is posted.

Since we require reviews to contain the device information, we had to ignore reviews that were not linked to a device. To determine the impact of this issue on our findings, we measured the average star rating from reviews that were linked to a device and reviews that were not linked to a device. We found that for free game apps, the average star rating for reviews that were linked to a device is 3.22, while it is 3.43 for reviews that were not linked to a device. For paid game apps, the average star rating for reviews linked to a device is 3.53, while it is 3.52 for reviews that are not linked to a device. If we consider all of the apps, not just game apps, we found that the average star rating for reviews that are linked to a device is 3.37 and for reviews that are not linked to a device is 3.51 (and we found similar results when we took each category of apps separately). We performed the Wilcoxon statistical significance test and found that there is a statistically significant difference for free game apps and all apps, however, there is no statistically significant difference for paid game apps. In all statistically significant cases, we find the star rating tend to be lower for reviews that are linked to a device. This finding indicates that reviews that are linked to a device are more critical, and hence, are more important to developers who are trying to avoid negative reviews.

Reviews can contain some spam reviews that serve as noise in our dataset [19]. To

mitigate this issue and ensure the quality of the reviews used in our study, we selected the 'most helpful reviews', since they provide us with the most reliable information.

All of our findings are derived from reviews. In certain cases, reviews may not directly correlate with other measures of quality such as defects, for example. However, prior research showed that reviews are directly correlated with revenues (even for free apps, which make their money through ads). Therefore, we believe that using reviews is a good proxy of success of an app.

4.5.3 External Validity

Since this study was performed on 99 mobile apps, our results may not generalize to all game apps. To address this threat, we pick apps which are labeled as 'Top apps' by Google Play. We feel that these apps are an appropriate representation of the apps in the game category, and a better choice than hand picking apps. In addition we extended our study to paid game apps, and free apps from four other app categories in Google Play. We found that, our results were often consistent, and sometimes even more pronounced in these apps, when compared to our results for free game apps.

Given that the Android OS and app ecosystems are quickly evolving, the device specific analysis in this thesis may not be applicable in a few years (or even months). However, we would like to emphasize that the main takeaways from this study are not about specific devices, but are about our generalizable method for prioritizing QA efforts.

4.5.4 Conclusion Validity

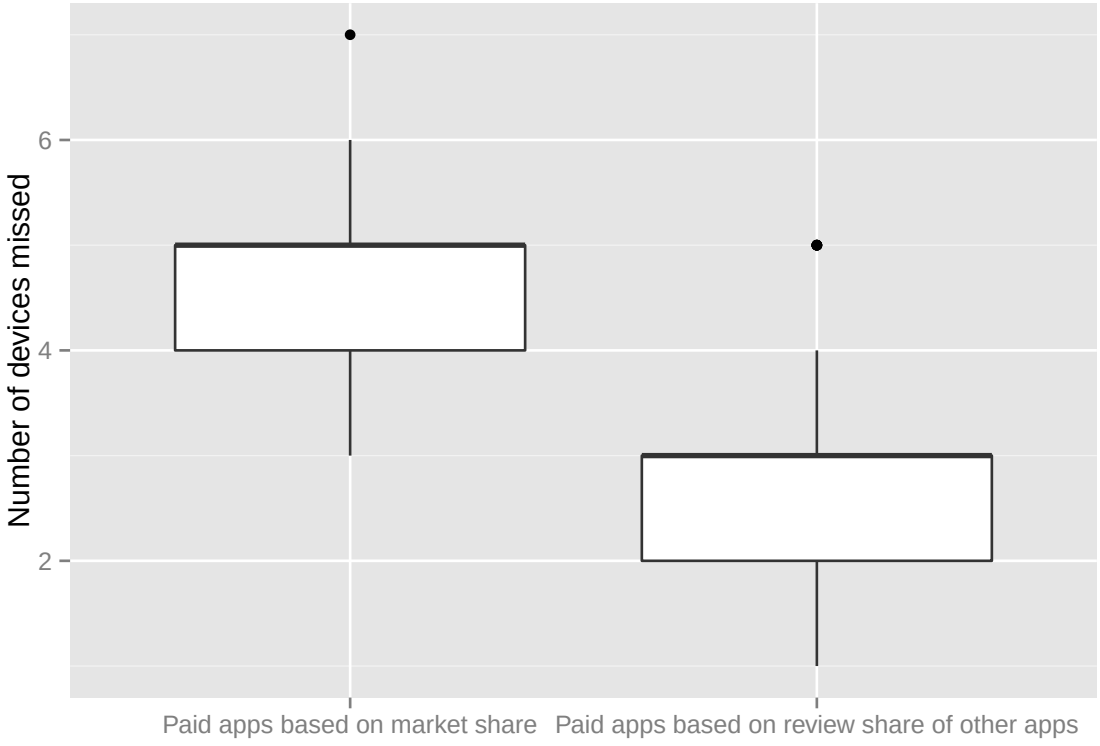
We assume that testing apps or conducting focus groups for certain devices will find problems (e.g., bugs) which can be fixed by the developer of an app, thereby improving the quality and hence the revenues of the app. Even though it may seem logical, it still is an assumption. For example, we did not verify whether the test effort prioritization does actually improve quality or increase revenues. However, this assumption (testing finds bugs that can be fixed to improve quality) is the basis for most testing efforts. Nevertheless more in-depth studies are needed to study such assumptions.

4.6 Conclusion

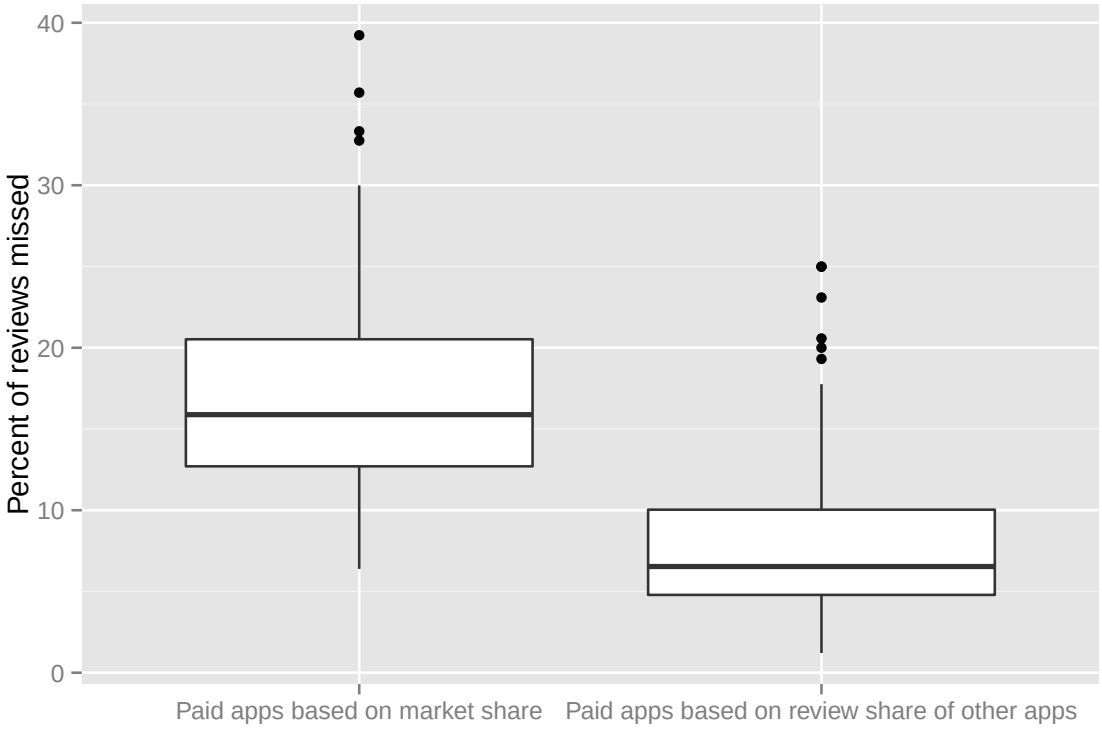
In this chapter we sought to help game app developers deal with Android fragmentation by picking the devices that have the most impact on their app star ratings, thus aiding developers in prioritizing their QA efforts. By studying the reviews of game apps, we find that a small percentage of devices account for most of the reviews given to apps. Thus, developers can focus their QA efforts on a small set of devices. New developers can use data from other apps in the same app category to prioritize their testing and other QA efforts if they do not already have reviews for their app. We also find that some devices give statistically significantly worse star ratings than others. Therefore, developers should identify particularly problematic devices, and prioritize their QA efforts even further towards such devices. Finally we find that the results from the free game category to generalized to paid game apps, and free apps in the four other categories that were examined.

In conclusion, developers can adopt our method of analyzing Android reviews in

order to effectively alleviate the QA challenges brought forth by Android fragmentation.



(a) Number of devices missed



(b) Percent of review share missed

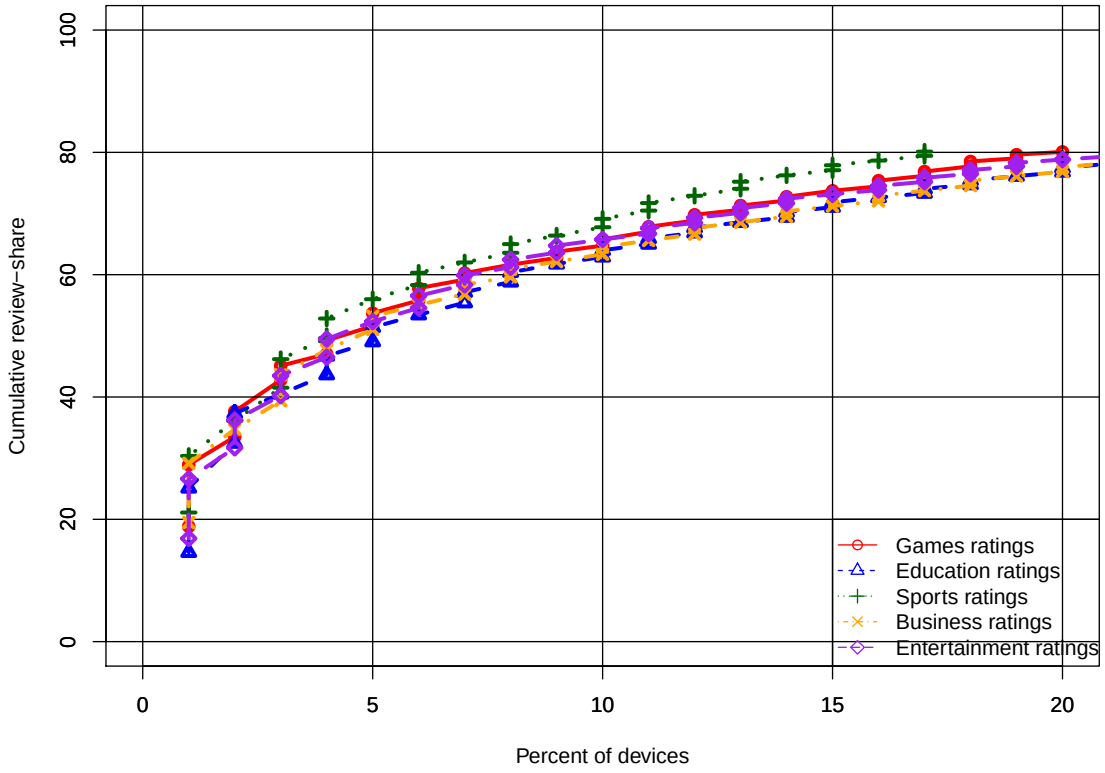


Figure 4.8: Percent of Android devices used to give X% (X ranges from 0 - 80) of the reviews of free apps in 5 categories

Chapter 5

Examining the Relationship between FindBugs Warnings and End User Ratings: A Case Study On 10,000 Android Apps

Summary *– Past research has examined the relationship between static analysis warnings and quality metrics. However, there is no evidence linking the static analysis warnings directly to the user perception of software, as this relationship is difficult to examine by research. In the app ecosystem, user perception is extremely important to study as reviews of apps are highly correlated with downloads and hence revenues. We use FindBugs, which is an automated static analysis tool for Java code, to compare the results of running this tool on 10,000 free-to-download Android apps with the reviews of the apps. We find that (1) a statistically greater density of static analysis warnings occur in low-rated apps than high-rated apps; (2) specific categories of FindBugs warnings such as the 'Bad Practice', 'Performance' and 'Internationalization' categories are found significantly more in low-rated apps. On examining the relationship between these three categories of warnings and the complaints in reviews, we find that apps with the highest densities of these warning categories, receive significantly*

more corresponding complaints. These findings provide evidence that certain categories of warnings from FindBugs are closely related to user experience and hence have a strong impact on the star rating of an app. Thus app developers can use static analysis tools such as FindBugs to potentially identify the culprit bugs behind the issues that users complain about, before they release the app.

Static analysis tools are used by developers to identify possible issues before a software is released. These tools automatically examine source code and produce warnings that help developers find possible issues. Previous research has confirmed that static analysis tools can help identify warnings within the code, many of which are actual software bugs [42–44]. Thus addressing the results of static analysis can be used to improve the quality of a software. However, there are many categories of static analysis warnings and it is not clear if some of these categories lead to software bugs that impact the user's perception of app quality.

User perception in the mobile ecosystem, represented as star ratings, are statistically significantly related to the downloads, and hence the revenue generated by an app [8]. When mobile app users are dissatisfied with the quality of an app, they often give a low-ratings to the app (1-star indicates bad quality while a 5-star indicates good quality). In addition to these star ratings, app users can also leave review-comments that are a text description about why the app was given a low-rating. If low-rated reviews of apps are related to the warnings generated from static analysis tools, then developers can use static analysis tools to potentially identify some of the bugs that lead to low-ratings.

Thus in this case study, we study the static analysis tool "FindBugs" which is an

open source program that automatically identifies warnings in Java code for potential bugs [73]. This tool can be used to analyze Jar files and strives to reduce the number of false positives warnings [41]. In this case study, we examine the warnings from running FindBugs on 10,000 free-to-download Android apps. By studying a large corpus of apps, *we want to empirically examine the relationship between the FindBugs warnings in an app, and the star rating assigned to the app by the end user.* We add another dimension of evidence to the relationship, by comparing the complaints in the reviews, and the warnings from FindBugs for a subset of apps. We want to understand this relationship to determine if developers can trust the FindBugs warnings and use them to fix crucial issues that affect users. More specifically, we seek to answer the following research questions:

RQ1. Are there more FindBugs warnings in low-rated apps compared to high-rated apps?

By comparing the densities of FindBugs warnings in low and high-rated apps, we find that low-rated apps have higher warning densities than high-rated apps. In addition, low-rated apps have a greater density of high-priority warnings.

RQ2. Which warnings occur more frequently in low-rated apps than high-rated apps?

We find that warnings in the 'Bad Practice', 'Internationalization' and 'Performance' categories have significantly higher densities in low-rated apps than high-rated apps.

The results from these two research questions suggest that developers can benefit from running static analysis tools (i.e., FindBugs) on their Android apps as this can help

them identify software bugs in their app that could result in low-rating reviews, before they release the app.

The remainder of this chapter is organized as follows: In Section 5.1, we provide the necessary background on FindBugs. In Section 5.2, we provide an overview of our approach. In Section 5.3, we present the motivation, approach and the findings for each RQ. In Section 5.4, we discuss the potential threats to validity. We discuss the lessons from this chapter and conclude this chapter in Section 5.5.

5.1 Background On FindBugs

We pick FindBugs as the static analysis tool for our case study. While there are other prominent static analysis tools, we select FindBugs because it strives to reduce the number of false positive warnings [41]. This, we feel is what makes a static analysis tool useful since developers look for low-cost, high-effectiveness tools. Moreover, we can run FindBugs on Jar files as we do not have access to the unpackaged source code. While there are other static-analysis tools which also work well on Android apps, we choose FindBugs since it focuses on reducing false positives and can analyze Jar files. Moreover, FindBugs is widely used, and it focuses on the bugs that cause functional problems [41, 43].

Overall, FindBugs identifies warnings for over 400 possible bugs. These warnings are grouped into the following 8 categories: 'Bad Practice', 'Correctness', 'Internationalization', 'Malicious Code Vulnerability', 'Multi-threaded Correctness', 'Performance', 'Security' and 'Dodgy Code'. Moreover, FindBugs also assigns different priorities to each warning; the priority level of the warnings is dependent on how

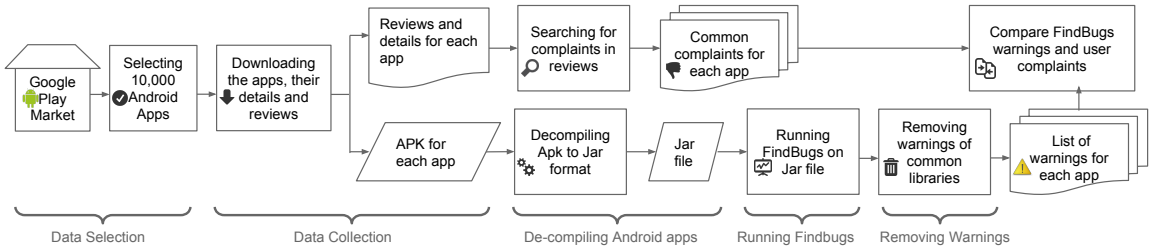


Figure 5.1: Overview of our process

confident FindBugs is, in whether what it detects is in fact a bug.

In the next section, we provide a detailed overview of our study, and how we used FindBugs to identify warnings in Android apps.

5.2 Study Design

In this case study, we examine the results of running FindBugs on 10,000 Android apps. Figure 5.1 illustrates the different steps for this study in order, and we describe them in the following subsections.

5.2.1 Data Selection

The data sample in our study are 10,000 free-to-download Android apps from the Google Play market. These apps are randomly selected and cover a broad range of star ratings and categories (we selected our app from a list of apps generated by Dienst *(*et al.) [74]). Figure 5.2 illustrates the distribution of the star ratings of these apps. The minimum star rating of our sample apps is 1.3 stars while the maximum is 5 stars. The median star rating of these apps is 4.1 stars. Moreover, we only select apps, which have a minimum of 30 individual star ratings. This ensures that a few users don't skew the star rating of these apps. The median of the number

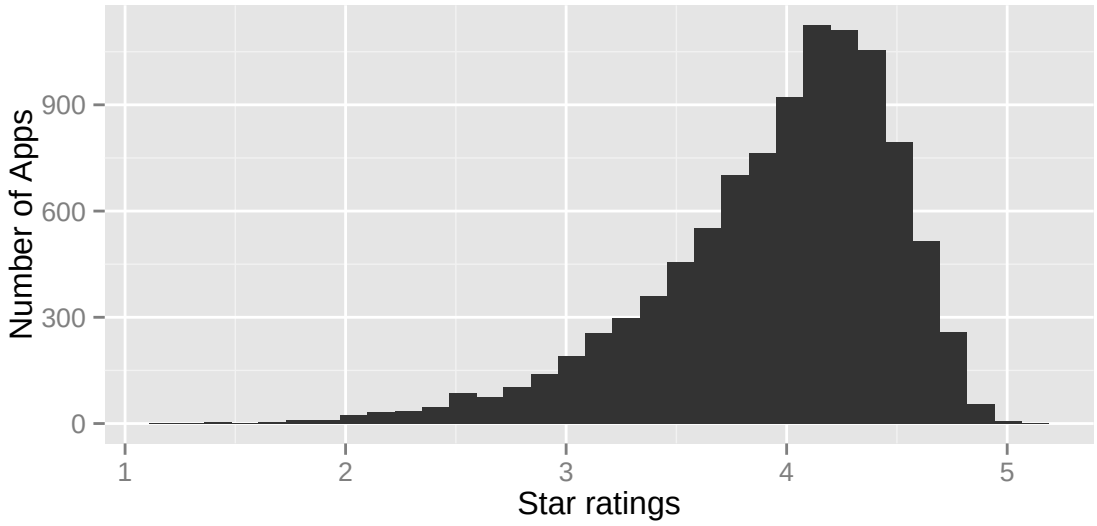


Figure 5.2: Rating distribution of the 10,000 Android apps

of individual star ratings in our selected apps is 181. Our sample of apps is composed of apps in all categories in the Google play market. Collectively, game apps account for the highest number of apps, while weather apps account for the lowest.

5.2.2 Data Collection

To collect these apps, along with their details, we write a script using an open source crawler called *Google Play Crawler* [75]. For each of these apps, we download the APK (Android application package) file, their overall star rating, and their reviews. We collect up to 500 of the newest reviews for each of the apps (Google Play limits the total number of reviews that non-developers can see). Each review consists of the star rating assigned to the app by the user, and the text of the review-comment that the users enter.

Once we have all of the required data, we de-compile the apps into a format that we can run FindBugs on.

5.2.3 De-compiling Android apps

After downloading the selected apps, we extract the Jar files from the APKs since this is the format which FindBugs requires. We use an open source tool called *Dex2Jar* to extract Jar files from APKs [76].

5.2.4 Running FindBugs on Android apps

We run FindBugs using recommended settings that detects high and medium priority warnings, but ignores low priority warnings – these often include false positives and are thus not a part of the recommended configuration [73]. In addition, we ignore all style and naming convention warnings (since we are looking at the decompiled binary of the original code), and the warnings from automatically generated files.

After running FindBugs on each of the apps, we extract the density of each warning per app. Warning density in FindBugs is defined as *warnings per thousand lines of non-commenting source statements*. In addition to this, we also identify the counts of warnings in each of the 8 categories, and the occurrence of each warning along with the name of the class where this warning occurred.

5.2.5 Removing warnings of common libraries

Android apps, like all software, are built with numerous external libraries. Since we want to examine the relationship of reviews and warnings within each app, we cannot have interference in our analysis because of common libraries (and warnings for these libraries). For example, attributing the warnings of the *Android.support* library (a set of code that provide backward-compatibility and are found across many Android apps) adds unnecessary noise to the data.

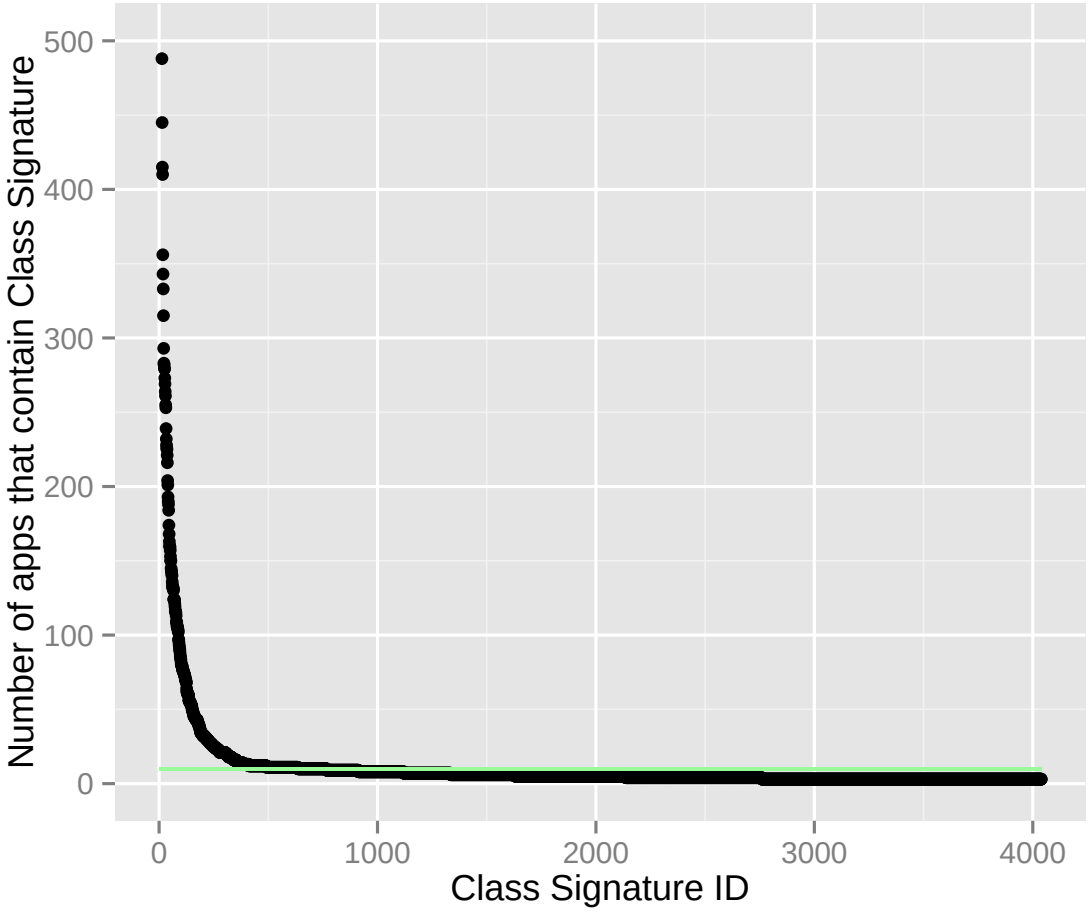


Figure 5.3: Number of apps that contain the 4039 shared class signatures (we examine all class signatures above the green line)

The first step in removing these libraries is identifying the ones that are found across many apps. We identify the external libraries using the packaging information and class names of the base classes. For example, the base class *android.support.app.Fragment* lets us identify that this app uses the *Android.support* library.

We count the number of apps that each package is found in. Figure 5.3 illustrates the 4039 shared packages and the number of apps they are in. This figure does not show the number of apps for the top 10 most common packages as they skew

the graph – the max of which is *com.google*, included in 5,611 apps. As this figure demonstrates, there are a few packages that are found in many of the apps. After the first few hundred popular classes, the frequency quickly declines.

For this study, we manually examine 766 packages that are shared in 10 or more apps and the libraries that they are a part of. We examine these to make sure that they are actually a library and not a commonly used package name (for example, *com.myapp.ButtonFragment*). For each of these potential libraries, we examined public code bases *e.g., Github, Ohloh* that match their signature ID. After examining hundreds of potential libraries, we end up flagging a total of 329 libraries and ignore the warnings from the classes of these libraries in our analysis.

5.3 Results

In this section, we discuss the motivation, approach and findings for both research questions.

RQ1) Are there more FindBugs warnings in low-rated apps compared to high-rated apps?

Motivation: Our hypothesis is that apps with low-ratings have more FindBugs warnings. If this is proven, then this will be the first evidence for a direct link between FindBugs warnings and star ratings. Developers could then use the warnings from FindBugs to gauge the quality of their app, and find potentially critical bugs that could lead to low-ratings (before they even deploy the app). We discuss our approach for comparing the warnings of high and low-rated apps next.

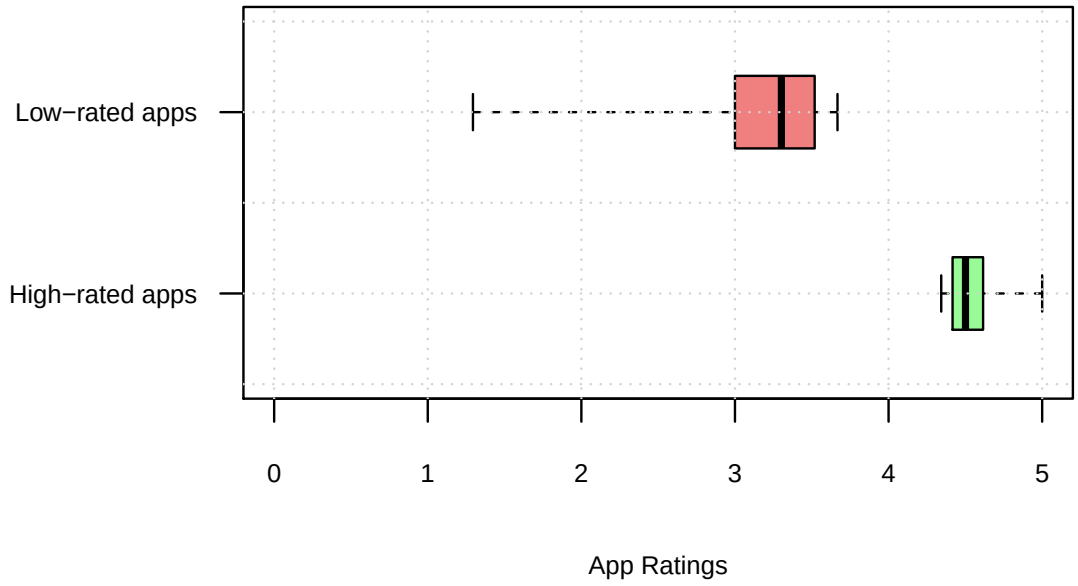


Figure 5.4: Comparing the star ratings of 2,500 high and low-rated apps

Approach: To answer this RQ, we want to check if densities of FindBugs warnings are different between high and low-rated apps. To do this, we first sort the 10,000 apps by their ratings. We identify the 25%(2,500) apps with the best star ratings as high-rated apps, and 25% with the worst star ratings as low-rated apps. The box plot in Figure 5.4 illustrates the star ratings of high and low-rated apps. The high-rated apps range from a star rating of 4.3 to 5 stars, while the low-rated apps range from 1.3 to 3.7 stars. We perform our analysis on warning densities instead of raw warning numbers to control for the size of the apps.

To compare the densities of FindBugs warnings the high and low-rated apps, we perform a one-tailed *Mann-Whitney U-test* with $\alpha < 0.05$ [77]. In the next section, we present the results of this analysis.

Findings: We find that low-rated apps, when compared to high-rated apps, tend to have statistically significantly higher warning densities (p-value of *<0.001*). Moreover, we find that both high and medium priority warnings are statistically higher in low-rated apps. Note that FindBugs is more confident that high priority warnings will lead to a real bug.

The median of the total warning densities for low-rated apps is 2.4, while high-rated apps have a median density of 1.9. However, both high and low-rated apps do have some outliers (out of 2,500 apps in each set). Among the low-rated apps, there is an app which has a high warning density of 76; this is an e-book app which has a star rating of 3.6 and has complaints in its reviews about crashing and text display issues.

The results from this RQ show that when FindBugs analyzes apps it reports a higher density of warnings for low-rated apps. This finding provides evidence for a link between star ratings and the density of FindBugs warnings. Thus, there is a value in developers using static analysis before they release an app. They should consider more QA tasks for an app with high warning densities.

Discussion: The results from RQ1 suggest that developers should be wary of a large density of FindBugs warning in their apps. We now want to identify the specific warning density at which developers should start paying additional attention. To do this, we first sort the 10,000 apps by their total warning densities. Then, we divide the apps into two sets based on their warning densities. Initially, we divide the two sets with an arbitrary value of $X = 2$ warning density. The first set consists of all app with a warning density of less than 2, while the second set contains all of the

apps with a warning density of 2 or higher. We compare the star ratings of the two sets with the *Mann-Whitney U-test*. We keep decreasing the value of X in steps of 0.05, until the p-value is >0.05.

We find that 0.74 is the warning density at which the star ratings of the two sets start to statistically significantly differ. Apps with a warning density of 0.74 or higher have a statistically significantly lower rating. This suggests that 0.74 or higher is the warning density that the developers should be careful about. At this point, developers should consider investing additional resources in QA.

We find that low-rated apps have a statistically significant higher density of warnings than high-rated apps. In addition, we find that low-rated apps also have higher density of warnings for medium and high-priority warnings.

RQ2) Which warnings occur more frequently in low-rated apps than high-rated apps?

Motivation: One of the common criticisms of static analysis tools is that they often produce numerous false positives. This means that even if developers incorporate static analysis tools into their workflow, they might end up wasting their time solving warnings that don't have an impact on their software.

While FindBugs explicitly focuses on reducing false positives as much as possible, it is not perfect either. Some of the warnings it finds within apps could be benign as well. Therefore we want to identify the warnings that are most related to the lowest-rated apps. This will help developers prioritize the warnings found in their app, which could be the culprit behind the issues that users complaint about.

Approach: The approach for this RQ builds on our work from RQ1. In this case, we identify the specific type of warning by its ID, thus counting each individual type

Table 5.1: Categories of FindBugs warnings that have a statistically significantly higher density of warnings in low rated apps compared to high rated apps.

FindBugs Warning Category	**MWU test p-value**	**Median Warning Density**	
		High-Rated Apps	Low-Rated Apps
Bad Practice	0.011	0.21	0.24
Internationalization	1.57e-11	0.11	0.18
Performance	4.03e-05	0.39	0.48

of warning. Several types of warnings are aggregated to a category of warning as specified by FindBugs (i.e., Performance). Therefore, we count the warnings based on their categories. Following this step, we turn the raw counts into densities. The densities for each of the different categories of warnings, are determined by adding the density of every warnings that they are composed of.

Once we have the densities for each category, and type of warning, for the high and low-rated apps (same as RQ1), we compare them using a one-tailed *Mann-Whitney U-test*. In the next subsection, we present the warnings that were statistically different.

Findings: We find that three categories (out of eight categories) of warnings occur statistically significantly more in low-rated apps, than high-rated apps. As shown in Table 5.1, Bad Practice warnings, Internationalization warnings, and Performance warnings, have a statistically higher warning density in the low-rated apps as compared to the high-rated apps. Bad Practice warnings are violations of essential coding practices (e.g., equals problems, dropped exceptions, misuse of finalize). Internationalization warnings are warnings where developers misuse character encodings. Performance warnings are for code that is slow. We also present the median density values for these three categories among the high- and

low-rated apps.

These findings imply that developers should prioritize warnings in these three categories over others as they tend to be found more in low-rated apps and could lead to low ratings.

Discussion: After identifing the three categories of FindBugs warnings that occur signficantly more in low-rated apps, we now examine if users explicitly mention the issues related to these warnings in the reviews of apps. To do so, we compare the complaints in the reviews of the apps that have the highest densities of these warnings, with the apps that have the lowest densities. To focus on the complaints we only analyze the reviews-comments which have a rating of 3 stars or less [78]. We filter away the apps that have less than 10 review-comments (so that a few review-comments will not skew the overall complaints) [79]. We are left with a total of 4,708 apps. From these 4,708 apps, we identify the top 25% apps (1,177) which have the highest and the lowest reported densities for the warnings in Bad Practice, Performance and Internationalization categories. Thus, we have the 1,177 apps with the highest Bad Practice warning density, and 1,177 apps with the lowest Bad Practice warning density. Similarly we have subsets of apps for the Performance and Internationalization categories as well.

To analyze the review-comments, we first identify the keywords that we should look for. Judging by the nature of these three categories, and our prior experience with manually categorizing reviews of mobile apps in Chapter 3 [58], we select the keywords shown in Table 5.2 for our analysis. We count the number of review-comments per app that has a keyword related to a warning category. For example,

we count the number of review-comments per app that has the keywords slow, hang, lags, slug and attribute it to 'Performance' complaints from users. This list also includes stemmed versions of each of these words (e.g., lags, lagging, lagged). We only count one occurrence of these keywords per review, so if both 'slow' and 'hangs' are mentioned in a review, we only count this as one occurrence of a Performance complaint. We do this since we only care *if* a review-comment contains a particular type of complaint.

Table 5.2: Keywords used to identify user complaints in review-comments associated with a particular warning category and the results of the analysis in the discussion subsection.

FindBugs Warning Category	**Keywords**	**MWU Test p-value**	**Percentage of Review-Comments with the Corresponding Complaint (Mean %)**	
			Low density apps	**High density apps**
Bad Practice	Bug, Buggy, Issue Problem, Broke	4.02e-09	5.6	6.7
Internationalization	Country, Language Word, International Internationalization UTF, Encoding	0.0002261	3.5	3.8
Performance	Slow, Hang Lag, Slug	0.0004456	3.9	6.0

Thus we get the frequency of review-comments with complaints corresponding to a particular warning category, along with the total number of review-comments for the app (i.e., *total-review-comments: 500, performance-complaint-count: 50*). Following this step we turn the raw count into the percentage values (i.e., *total-review-comments: 500, performance-complaint-percentage: 10*). We calculate these performance-complaint-percentage for each of the 1,177 apps with the highest performance warning densities, and the 1,177 apps with the lowest performance warning densities. Then we compare the performance-complaint-percentage values across these 2 subsets of apps using the one-tailed Mann-Whitney U (MWU) test, to see if users complain about performance in apps that have a higher density of performance warnings. We repeat this process for the other two categories of warnings as well.

We find that apps with the highest densities of warning for a category has a statistically significantly higher rate of the corresponding complaints. We present the p-values of the MWU test and the mean percentage of review-comments in an app that has complaints pertaining to a particular category of warning (i.e., warnings in the Bad Practice, Performance and Internationalization categories) in Table 5.2.

An example of such an app is *Media Player (trial)* which has a Performance warning density of 6.4. A quick examination of the reviews for this app reveals numerous comments mentioning performance issues and crashes such as *"Takes an age to search SD card, then when you try to play a video it just says Buffering until you get bored and close it. Rubbish."*

Many of these internationalization warnings are found in apps where the user is complaining about the encoding, or being forced to use a specific language. Thus, we are able to establish that warnings identified with FindBugs can directly manifest

in the user's review-comments as complaints about the apps, and thus impact the ratings of the apps.

We find that three categories of warnings have a statistically higher density in low-rated apps than high-rated apps: Bad Practice, Internationalization and Performance. This suggests that developers should prioritize their QA efforts on addressing these warnings, as their resultant bugs could have a detrimental affect on the rating of their app.

5.4 Threats to validity

In this section we discuss the perceived threats to our work and how we address them.

5.4.1 Construct Validity:

In this study, we did not analyze the code of the apps in their original form. We ran FindBugs on the de-compiled versions of the 10,000 apps (some of which could be obfuscated). While this may have affected the results, we are limited to this approach, since we do not have access to the source code. For the APK files that we do have, we use Dex2Jar that is also used in other studies to de-compile APK files [26].

When analyzing the warnings found in the Android apps, we removed the warnings from third party libraries. It could be the case that the warnings in these libraries maybe even more problematic than the warnings in the apps. However, we feel that removing these libraries is a better approach since it is shared across many apps. Assigning the warning attributes of the third party libraries to the apps themselves also adds a *low reliability of measures* threat which could lead us to invalid conclusions.

It could also be the case that the set of common libraries in low-rated apps is different than high-rated apps, and that we only removed one of these sets of libraries

from our analysis. To mitigate this threat, we did verify that the relationship between FindBugs warnings and star ratings holds even when all libraries are included.

5.4.2 Internal Validity:

The specific tools that we used for de-compiling the apps and identifying the warnings are not perfect. Hence, their usage may affect our results. However, we used standard tools for reverse engineering Android apps. We are also restricted to this approach because of the large scope of the study. We also used the recommended setting of FindBugs to analyze the Jar files.

In the discussion of RQ2, we identified the mention of different complaints based on the usage of some keywords in reviews. It could be the case that that there were additional words used to describe these complaints. However, manually analyzing thousands of reviews is outside the scope of this study. The keywords that we did pick, are based on our experience with manually analyzing complaints of apps [58]. By focusing on the reviews which gave a star rating of 3 or less, we made sure that we focus on the complaints.

5.4.3 External Validity:

It could be the case that our findings don't generalize to all free-to-download Android apps. However we feel that studying 10,000 apps is a considerable sample. To mitigate the threat of generalization, we also maximize the coverage of our apps by studying apps that cover all of the categories of apps. In addition our 10,000 apps cover a

range of star ratings that is similar to all apps in the Google play market [80].

5.5 Conclusion and Lessons

One of the main criticisms of static analysis tools are the number of false positive warnings. These warnings can direct developers towards fixing issues that have no impact on the quality of their software. In this chapter, we address some of these concerns for developers by studying the relationship between the star ratings of apps with the warnings generated by running FindBugs on their code. We find that that there are certain types of FindBugs warnings that are closely related to lower star ratings in Android apps.

For developers, this chapter highlights the utility, and importance of running FindBugs on Android apps. While static analysis tools does not replace the process of manually testing the apps on different devices, they complement manual testing and offer a low-cost way to help developers identify warnings that could cause issues that lead to low-ratings. The most important thing that developers can take away is that there are three categories of warnings that appear significantly more in low-rated apps. This means that during usage of FindBugs, these warnings lead to some bugs that have a degrading affect on the quality of the app (hence resulting in low-ratings). For app developers this means that they should not neglect running FindBugs (or other static analysis tools) as it is a low-cost method of finding the solutions to some of the user complaints. If the overall warning density for their app is too high, then they should look at the categories of bugs that seem to have a high warning density, and address those warnings before they release the app.

For researchers this chapter provides a direct link between static analysis warnings

and software quality (expressed as star ratings). In the future, we plan to examine other static analysis tools and how they could help developers improve the quality of their apps.

Chapter 6

Summary and Conclusions

In the mobile ecosystem, the success of an app is often determined by the reviews that it receives from users. This has led to an intense level of competition for developers. Developers have to get good reviews in order to stand out from other apps. This pressures developers to always improve the quality of their apps. While developers can improve their app quality by traditional means, they can also exploit the reviews in the mobile ecosystem and examine the user perception as a means of feedback.

However, there hasn't been much research which examined these reviews from the perspective of developers. Therefore, in this thesis we examine the reviews of Android and iOS apps to validate the following research statement:

The mobile ecosystem provides centralized and publicly available user feedback in the form of reviews. There is a great emphasis on these reviews and high-rated apps receive significantly more downloads than low-rated apps. Now that researchers have access to these reviews, they can perform new large-scale studies to help developers improve the quality of their apps, and better prioritize their QA efforts.

6.1 Summary

In Chapter 3 we examine the reviews of iOS apps to identify the different types of complaints that developers can find in them. In Chapter 4 we analyze the device data in the reviews of Android apps to study how Android app developers can better deal with Android fragmentation. Finally in Chapter 5 we compare the star rating and review data of Android apps with their static analysis warnings. We describe each of these chapters in further detail below:

What Do Mobile App Users Complain About?

Chapter 3 presents an empirical study on the reviews of 20 iOS apps. By studying the low-rating reviews of these apps, we found 12 types of user complaints. We found that functional errors, feature requests and app crashes are the most frequent complaints. On the other hand, we found that complaints about privacy and ethical issues, and hidden app costs have the most negative impact on the star rating of an app. Moreover, we found that users attributed their complaint to a recent update of the app in 11% of the reviews.

From Chapter 3, we conclude that reviews can contain a wealth of information that can help developers improve the quality of their apps. They can use the reviews of their apps to frequently identify existing issues or potential new features. Moreover, reviews can help developers identify the otherwise unknown issues (i.e., privacy or ethical issues) which have the worst impact on an app's star ratings. Overall, developers can use the information from reviews to better prioritize their limited QA resources. In the next chapter, we study how Android developers can further optimize their QA resources.

Prioritizing The Devices To Test Your App On: A Case Study Of Android

Game Apps

In Chapter 4 we present a case study where we analyzed the device information present within the reviews of Android apps to better understand of Android developers can deal with device fragmentation. Since there are hundreds of Android devices in the market, each of which could have their own device-specific problems, Android developers must carefully test their Android apps on numerous devices. We found that while each app is rated from numerous devices (38 to 132 unique devices for game apps), most of the reviews (80%) originate from a small subset of devices (on average, 33%). We also found that some of these devices tend to give worse star ratings than others. Moreover, we found that new developers can use the review data from similar apps to identify the devices which they should focus on first.

From Chapter 4 we conclude that the device information from reviews can help developers better prioritize their QA resources. Developers can focus on the devices with the most reviews, and the ones with negative reviews, since these devices have the greatest impact on star ratings. Studying these reviews can help both the owners of the apps, and new developers who are getting started. In the next chapter, we study how the review data compares with the warnings from a static analysis tool.

Examining the Relationship between FindBugs Warnings and End User Ratings: A Case Study On 10,000 Android Apps

In Chapter 5 we present a study which examined the relationship between static analysis warnings from FindBugs and the user's perception. We find that a low-rated apps have a statistically significantly greater density of FindBugs warnings than high-rating apps. We also found that FindBugs warnings categories 'Bad Practice', 'Performance' and 'Internationalization' are found significantly more in low-rated

apps. Finally, by comparing the warnings of the previous 3 categories of warnings with the complaints in the apps, we found that some correspondence does exist.

From Chapter 5 we conclude that some FindBugs warnings negatively impact the user experience and hence have a strong impact on the star rating of an app. Developers can use static analysis tools to potentially identify culprit bugs behind the issues that users complain about, before they release the app.

6.2 Limitations and Future Work

The limitations specific to each of Chapter 3, 4 and 5 are presented in their respective *Threats to Validity* section. Below, we discuss the common limitations of these chapters and the future direction for our this research.

- Since we rely on reviews as our main data source, the validity of the findings in this thesis are limited to the quality of the reviews. Some of the reviews could be inauthentic or contain false information. In the worst case scenario, some of these reviews could have been fake reviews by spammers which artificially increased or reduced the star rating of an app. To mitigate the effects of these potential false reviews, we used a large number of apps or reviews whenever possible.

- Another limitation of our study, in terms of its implications for mobile apps, is that it is limited to iOS and Android apps. Moreover, our set of reviews may not generalize to all the reviews in the market. Future research can expand on this thesis by considering more apps and comparing our findings across other mobile platforms. For our work concerning the different complaint types, future work can study whether other complaints have the same complaints.

We studied low-rated reviews to identify the most frequent and impactful complaints. Future research can more platforms differ in terms of complaints and user expectations. Moreover, future research should also examine the reasons why users give high-rated reviews to apps. The evolution of these complaint types will also help developers better anticipate possible issues across the life time of their apps.

For our work dealing with the device info in Android apps, future work can investigate the problems reported from each device, and why certain Android devices have certain kinds of problems, thereby enabling developers to identify the problems. Future work can also examine if developers can minimize the test device subset by only picking one device from a family of devices; a family of devices may share common characteristics and it may be redundant to test each one of them.

Our examination of static analysis focused on FindBugs. Future research can examine other static analysis tools and how they could help developers improve the quality of their apps. This approach can also be applied to the code of apps in other platforms (e.g., iOS, or Windows Phone) and their respective programming language (e.g., Objective C, C#).

6.3 Conclusions

As the mobile ecosystem continues its growth and impact on different aspects of our daily lives, the quality of mobile is becoming increasingly important. Reviews of mobile apps provide a direct access to the user's perspective of an app, thus studying them can help developers better understand the needs of their users. These reviews contain information relevant to many stakeholders including developers, project managers and content creators. These stakeholders can work together to better prioritize

their QA efforts and improve the quality of mobile apps. Future research can build on our findings to help these stakeholders succeed in the ever-evolving mobile ecosystem.

Bibliography

[1] "Android captured 79% share of global smartphone shipments in 2013," April 2014. [Online]. Available: http://blogs.strategyanalytics.com/WSS/post/2014/01/29/Android-Captured-79-Share-of-Global-Smartphone-Shipments-in-2013.aspx

[2] "Us smartphone market share q2 2013," April 2014. [Online]. Available: http://bgr.com/2013/08/06/us-smartphone-market-share-q2-2013-nielsen/

[3] "Apple press info: App store sales top $10 billion in 2013," January 2014. [Online]. Available: http://www.apple.com/pr/library/2014/01/07App-Store-Sales-Top-10-Billion-in-2013.html

[4] Google, "Android apps on Google Play," April 2014. [Online]. Available: https://play.google.com/store/apps/

[5] "Apple itunes everything you need to be entertained," April 2014. [Online]. Available: http://www.apple.com/itunes/

[6] S. M. Mudambi and D. Schuff, "what makes a helpful online review? A study of customer reviews on Amazon.com," *MIS Quarterly*, vol. 34, no. 1, pp. 185–200, 2010.

[7] H.-W. Kim, H. L. Lee, and J. E. Son, "An exploratory study on the determinants of smartphone app purchase," in *Proceedings of the 11th International DSI and the 16th APDSI Joint Meeting*, Taipei, Taiwan, July 2011.

[8] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: Msr for app stores," in *Proceedings of the 9th Working Conference on Mining Software Repositories (MSR '12)*, Zurich, Switzerland, 2-3 June 2012.

[9] L. Goasduff and C. Pettey, "Gartner says worldwide smartphone sales soared in fourth quarter of 2011 with 47 per cent growth," *Gartner, Inc*, vol. 15, 2012.

[10] "The smartphone reinvented around you — windows phone," April 2014. [Online]. Available: http://www.windowsphone.com/

[11] "Blackberry smartphones - apps," April 2014. [Online]. Available: http://www.blackberry.com

[12] Louis Columbus, "Roundup of mobile apps and app store forecasts, 2013," June 2013. [Online]. Available: http://www.forbes.com/sites/louiscolumbus/2013/06/09/roundup-of-mobile-apps-app-store-forecasts-2013/

[13] "Amazon appstore for Android," April 2014. [Online]. Available: http://www.amazon.com/mobile-apps/b?node=2350149011

[14] "Samsung apps," April 2014. [Online]. Available: http://apps.samsung.com/mars/main/getMain.as?COUNTRY_CODE=CAN

[15] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia, "Understanding Android fragmentation with topic analysis of vendor-specific bugs," in *Reverse*

Engineering (WCRE), 2012 19th Working Conference on. IEEE, 2012, pp. 83–92.

[16] H. Ham and Y. Park, "Mobile application compatibility test system design for Android fragmentation," *Software Engineering, Business Continuity, and Education*, pp. 314–320, 2011.

[17] R. Vasa, L. Hoon, K. Mouzakis, and A. Noguchi, "A preliminary analysis of mobile app user reviews," in *Proceedings of the 24th Australian Computer-Human Interaction Conference.* ACM, 2012, pp. 241–244.

[18] L. Hoon, R. Vasa, J.-G. Schneider, and K. Mouzakis, "A preliminary analysis of vocabulary in mobile app user reviews," in *Proceedings of the 24th Australian Computer-Human Interaction Conference.* ACM, 2012, pp. 245–248.

[19] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *Requirements Engineering Conference (RE), 2013 21st IEEE International*, July 2013, pp. 125–134.

[20] L. V. Galvis Carreño and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013, pp. 582–591.

[21] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, May 2013, pp. 41–44.

[22] F. Thung, S. Wang, D. Lo, and L. Jiang, "An empirical study of bugs in machine learning systems," in *Proceedings of the 23rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2012, pp. 271–280.

[23] Y. Tian, P. Achananuparp, I. N. Lubis, D. Lo, and E.-P. Lim, "What does software engineering community microblog about?" in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 2012, pp. 247–250.

[24] "Workshop on mobile software engineering." [Online]. Available: www.mobileseworkshop.org

[25] M. D. Syer, B. Adams, Y. Zou, and A. E. Hassan, "Exploring the development of micro-apps: A case study on the blackberry and android platforms," *Int'l Working Conference on Source Code Analysis and Manipulation*, pp. 55–64, 2011.

[26] I. J. M. Ruiz, M. Nagappan, B. Adams, and A. E. Hassan, "Understanding reuse in the android market," in *Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC)*, June 2012.

[27] S. Agarwal, R. Mahajan, A. Zheng, and V. Bahl, *Diagnosing mobile applications in the wild*. ACM Press, 2010, pp. 1–6.

[28] A. K. Jha, "A risk catalog for mobile applications by," *Interface*, no. February, 2007.

[29] H. Kim, B. Choi, and W. E. Wong, "Performance testing of mobile applications at the unit test level," in *IEEE Int'l Conf. on Secure Software Integration and Rel. Improvement*, ser. SSIRI '09, 2009, pp. 171–180.

[30] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen, "Asking for (and about) permissions used by android apps," in *Proceedings of the Tenth International Workshop on Mining Software Repositories*. IEEE Press, 2013, pp. 31–40.

[31] F. Khomh, B. Chan, Y. Zou, A. Sinha, and D. Dietz, "Predicting post-release defects using pre-release field testing results," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*. IEEE, 2011, pp. 253–262.

[32] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "Api change and fault proneness: a threat to the success of android apps," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013, pp. 477–487.

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

[34] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for android apps," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013, 2013, pp. 224–234.

[35] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 59.

[36] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using gui ripping for automated testing of android applications,"

in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2012, pp. 258–261.

[37] Z. Jamrozik, Gross, "Droidmate: Fully automatic testing of android apps," September 2013. [Online]. Available: http://www.droidmate.org/

[38] "Using hardware devices — android developers." [Online]. Available: http://developer.android.com/tools/device.html

[39] K.-M. Cutler, "How do top android developers qa test their apps?" June 2012. [Online]. Available: http://techcrunch.com/2012/06/02/android-qa-testing-quality-assurance/

[40] "Appthwack — test your android, ios, and web apps on real devices." [Online]. Available: https://appthwack.com/

[41] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *ACM Sigplan Notices*, vol. 39, no. 12, pp. 92–106, 2004.

[42] B. Cole, D. Hakim, D. Hovemeyer, R. Lazarus, W. Pugh, and K. Stephens, "Improving your software using static analysis to find bugs," in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM, 2006, pp. 673–674.

[43] N. Ayewah and W. Pugh, "The google findbugs fixit," in *Proceedings of the 19th international symposium on Software testing and analysis*. ACM, 2010, pp. 241–252.

[44] A. Vetro, M. Morisio, and M. Torchiano, "An empirical validation of findbugs issues related to defects," 2011.

[45] N. Ayewah, W. Pugh, J. D. Morgenthaler, J. Penix, and Y. Zhou, "Evaluating static analysis defect warnings on production software," in *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. ACM, 2007, pp. 1–8.

[46] C. Guo, J. Zhang, J. Yan, Z. Zhang, and Y. Zhang, "Characterizing and detecting resource leaks in android applications," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 2013, pp. 389–398.

[47] É. Payet and F. Spoto, "Static analysis of android programs," *Information and Software Technology*, vol. 54, no. 11, pp. 1192–1201, 2012.

[48] P. Krishnan, S. Hafner, and A. Zeiser, "Applying security assurance techniques to a mobile phone application: An initial approach," in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*. IEEE, 2011, pp. 545–552.

[49] S. Agarwal, R. Mahajan, A. Zheng, and V. Bahl, "Diagnosing mobile applications in the wild," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 22.

[50] N. Hu, P. A. Pavlou, and J. Zhang, "Can online reviews reveal a product's true quality?: empirical findings and analytical modeling of online word-of-mouth communication," in *Proceedings of the 7th ACM conference on Electronic commerce*, ser. EC '06, 2006, pp. 324–330.

[51] J. A. Chevalier and D. Mayzlin, "The effect of word of mouth on sales: Online book reviews," *Journal of marketing research*, vol. 43, no. 3, pp. 345–354, 2006.

[52] appComments, "Read user reviews online and by rss," June 2012. [Online]. Available: http://appcomments.com/

[53] "Sample size calculator - creative research systems," February 2014. [Online]. Available: http://www.surveysystem.com/sscalc.htm

[54] C. B. Seaman, F. Shull, M. Regardie, D. Elbert, R. L. Feldmann, Y. Guo, and S. Godfrey, "Defect categorization: making use of a decade of widely varying historical data," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2008, pp. 149–157.

[55] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *Software Engineering, IEEE Transactions on*, vol. 25, no. 4, pp. 557–572, 1999.

[56] selenium, "selenium: Web browser automation," Jun. 2012. [Online]. Available: http://seleniumhq.org/

[57] "Distimo - app analytics, conversion tracking, app download and revenue data." [Online]. Available: http://www.distimo.com/

[58] H. Khalid, "On identifying user complaints of ios apps," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 1474–1476.

[59] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about? a study on free ios apps," in *Accepted to be published in IEEE Software*. IEEE Press, 2014.

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

[61] "Appcelerator / IDC - Q4 2013 mobile trends report," November 2013. [Online]. Available: http://www.appcelerator.com.s3.amazonaws.com/pdf/q4-2013-devsurvey.pdf

[62] H. Bodden, "Android's fragmentation problem," November 2012. [Online]. Available: http://greyheller.com/Blog/androids-fragmentation-problem

[63] "Appthwack — pricing for mobile device testing." [Online]. Available: https://appthwack.com/pricing

[64] D. Irish, *The game producer's handbook*. CT-Press, 2005.

[65] J. Clark, S. DeRose *et al.*, "Xml path language v-1.0," 1999.

[66] S. L. Lim and P. Bentley, "Investigating app store ranking algorithms using a simulation of mobile app ecosystems," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*, June 2013, pp. 2672–2679.

[67] "Sentistrength - sentiment strength detection," July 2013. [Online]. Available: http://sentistrength.wlv.ac.uk/

[68] "Android phone market share - appbrain," March 2014. [Online]. Available: http://www.appbrain.com/stats/top-android-phones

[69] "Using the device availability dialog," February 2013. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/1286017

[70] A. J. Scott and M. Knott, "A Cluster Analysis Method for Grouping Means in the Analysis of Variance," *Biometrics*, vol. 30, no. 3, pp. 507–512, 1974.

[71] "Motorola droid x2: Even raw horsepower can't save this phone," September 2013. [Online]. Available: http://www.androidpolice.com/2011/05/28/review-motorola-droid-x2-even-raw-horsepower-cant-save-this-phone-from-mediocrity/

[72] "Tap tap revenge 4 keeps closing," September 2013. [Online]. Available: https://getsatisfaction.com/tapulous/topics/tap_tap_revenge_4_keeps_closing_after_i_complete_a_song

[73] "Findbugs - find bugs in java programs," November 2013. [Online]. Available: http://findbugs.sourceforge.net

[74] S. Dienst and T. Berger, "Static analysis of app dependencies in android bytecode," *Technical Note*, 2012.

[75] "Akdeniz-google-play-crawler — github," November 2013. [Online]. Available: https://github.com/Akdeniz/google-play-crawler

[76] "Dex2jar: Tools to work with android .dex and java .class files," November 2013. [Online]. Available: https://code.google.com/p/dex2jar/

[77] H. Mann and D. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, 1947.

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

[79] I. J. Mojica Ruiz, "Large-scale empirical studies of mobile apps," *Master's Thesis, School of Computing, Faculty of Arts and Science, Queen's University*, 2013.

[80] "Ratings on the android market - appbrain," November 2013. [Online]. Available: http://www.appbrain.com/stats/android-app-rating