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

Hammad Khalid[1], Meiyappan Nagappan[2], Ahmed E. Hassan[1]
[1]Software Analysis and Intelligence Lab (SAIL), Queen's University, Kingston, Canada
[2]Department of Software Engineering, Rochester Institute of Technology, Rochester, USA
[1]hammad@cs.queensu.ca, [2]mei@se.rit.edu, [1]ahmed@cs.queensu.ca

ABSTRACT

In the mobile app ecosystem, end user ratings of apps (a measure of end user perception) are extremely important to study as they are highly correlated with downloads and hence revenues. In this study we examine the relationship between the app ratings (and associated review-comments) from end users with the static analysis warnings (collected using FindBugs) from 10,000 free-to-download Android apps. In our case study, we find that specific categories of FindBugs warnings such as the 'Bad Practice', 'Internationalization', and 'Performance' categories are found significantly more in low-rated apps. We also find that there exists a correspondence between these three categories of warnings and the complaints in the review-comments of end users. These findings provide evidence that certain categories of warnings from FindBugs have a strong relationship with the rating of an app and hence are closely related to the user experience. 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. INTRODUCTION

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 [1]. Thus addressing the results of static analysis can help 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 the quality of an app.

User perception in the mobile app ecosystem, represented as user ratings, exhibit a statistically significant relation to the downloads, and hence the revenue generated by an app [2]. When mobile app users are dissatisfied with the quality of an app, they often give a low rating to the app (a 1-star indicates bad quality while a 5-star indicates good quality). In addition to these ratings, app users can also leave review-comments that are a text description that explains their rating.

Problem Statement: Therefore, if bad ratings of apps are related to certain categories of static analysis warnings, then developers can use static analysis tools to potentially identify and fix the bugs that lead to poor ratings.

In this case study, we examine the different categories of static analysis warnings from FindBugs (an open source program that automatically warns about potential bugs in Java code [3]) in 10,000 free-to-download Android apps from the Google Play store. By studying a large corpus of apps, *we want to empirically examine the relationship between each of the categories of FindBugs warnings in an app, and the rating assigned to the app by the end user*. We add another dimension of evidence to the relationship, by comparing the complaints in the review-comments of the apps, and the warnings from FindBugs. More specifically, we seek to answer the following research question:

RQ. Which category of FindBugs 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.

Takeaway Message: The results from our study suggests that developers can benefit from using static analysis tools (i.e., FindBugs) on their Android apps as this can help them identify certain types of software bugs in their app that could result in bad ratings.

2. 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 [4]. 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.

Overall, FindBugs identifies warnings for over 400 possible bugs. These warnings are grouped into the following eight 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 confident FindBugs is about whether the warning 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.

3. STUDY DESIGN

Figure 1 illustrates the different steps for our study of running FindBugs on 10,000 Android apps.

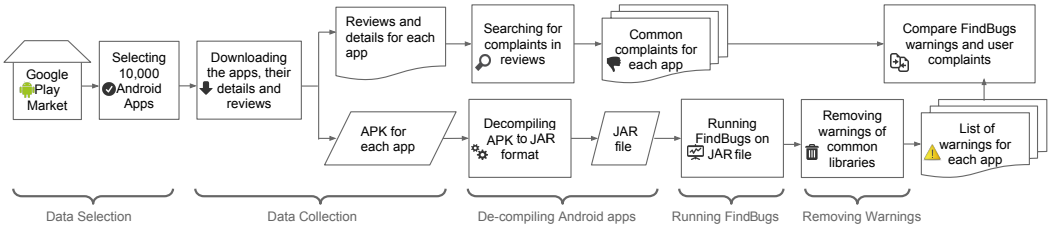

Figure 1: Overview of our process

3.1 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 from a list of apps generated by Dienst *(*et al.)[1]. The apps cover a broad range of ratings and categories. The minimum rating of our sample apps is 1.3 stars while the maximum is 5 stars. The median rating of these apps is 4.07 stars. Moreover, we only select apps, which have a minimum of 30 individual ratings. This ensures that a few users don't skew the rating of these apps. The median of the number of individual ratings in our selected apps is 181. Our sample of apps is composed of apps in all categories in the Google Play store. Collectively, game apps account for the highest number of apps, while weather apps account for the lowest.

3.2 Data Collection

For each of these apps, we download the APK (Android application package) file, the overall rating of the app, the number of people rating the app, and their review-comments. 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 rating assigned to the app by the user, and the text of the review-comment that the users enter.

3.3 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 [2].

3.4 Running FindBugs on Android apps

We run FindBugs using its recommended settings that detects high and medium priority warnings, but ignores low priority warnings – such low priority warnings often include false positives and are thus not a part of the recommended configuration [3]. In addition, we ignore all style and naming convention warnings (since we are looking at the decompiled binary of the original code).

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.

3.5 Removing warnings of common libraries

Android apps, like all software, are built with numerous external libraries. Since we want to examine the relationship between ratings and warnings within each app, we cannot have interference in our analysis because of warnings from common 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.

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. We found 4,049 shared packages, with a few packages that are found in many of the apps. After the first few hundred popular packages, the frequency quickly declines. The skew in the data is very high with *com.google* packages, included in 5,611 apps.

For this study, we manually examine 766 packages that are shared in 10 or more apps and the libraries that the packages are a part of. We examine the packages 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 package name. After examining hundreds of potential libraries, we end up flagging a total of 329 common libraries and we ignore the warnings from the packages of these libraries in our analysis.

4. RESULTS

In this section, we discuss the motivation, approach and findings for our research question. We then present a more detailed discussion of our results.

RQ: Which category of FindBugs 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 [1]. This means that even if developers incorporate static analysis tools into their workflow, developers might end up wasting their time solving warnings that don't have an impact on the quality of 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 low rated apps. Identifying such warnings will help developers prioritize their efforts towards such warnings, which could be the culprit behind the issues about which users complain.

Approach: To answer this RQ, we want to check if densities of a particular category of FindBugs warnings are different between high and low-rated apps. To identify high and low-rated apps, we first sort the 10,000 apps by their ratings. We identify the 25%(2,500) apps with the best ratings as high-rated apps, and 25% with the worst ratings as low-rated apps. The high-rated apps range from a rating of 4.3 to 5 stars, while the low-rated apps range from 1.29 to 3.6 stars.

We compare the warning densities (to control for the size of

[1]http://www.informatik.uni-leipzig.de/ berger/tr/2012-dienst.pdf
[2]https://code.google.com/p/dex2jar/

the apps) for each of the eight categories (as mentioned in Section 2) of FindBugs warnings, for the high and low-rated apps, using a one-tailed *Mann-Whitney U-test* (MWU) with $\alpha < 0.05$ [5]. The dependent variable is the warning density of each of the eight categories of FindBugs warnings (which is continuos in nature). The independent variables are the two independent groups of apps: high-rated apps and low-rated apps. The *null hypothesis* of the MWU test is: The warning densities for a category of FindBugs warning is the same for two populations, namely the high-rated apps and low-rated apps. The *alternate hypothesis* is: The warning density for a category of FindBugs warnings is larger in low-rated apps when compared to high-rated apps. We use the *Mann-Whitney U-test*, since it does not assume that the data is normally distributed, unlike the *Student T-test*. In the next subsection, we present the categories of 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 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 encoding in characters. 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 [6]. We filter away the apps that have less than 10 review-comments (so that a few review-comments will not skew the overall complaints) [7]. 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 [8], we select the keywords shown in Table 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.

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 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 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. RELATED WORK

Hovemeyer *et al.* introduced FindBugs as a tool that automatically warns about a variety of bugs within Java programs [4]. In this section we survey the work most related to static analysis tools on Android, and work related to mobile apps. In this study, we examine the relationship between the warnings from FindBugs, and the user ratings in the Google Play market.

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 [9]. 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 [10]. Their work also demonstrated the usefulness and versatility of static analysis tools in Android apps. Maji *et al.*, characterize the failures in the Android and Symbian OSes [11]. Our work complements these studies as we show that there is a value to developers in using automated static analysis on mobile apps (and not the OS on which the apps are installed) to check for issues that could lead to complaints in user reviews.

Table 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

Table 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

Mobile App Ratings: Harman *et al.* found that there is a strong correlation between app rating and number of downloads, indicating that ratings are a strong indicator of the users' opinion of apps [2]. Dennis *et al.* analyzed app review-comments and found that while the quality of the feedback varies, review-comments often contain useful feedback, bug reports and user experience [12]. Our case study uses the ratings of apps to identify high and low-rated Android apps and compare FindBugs warnings within the apps. We also examine the complaints in the review-comments of apps.

Quality in mobile apps: In our previous study we examined the reviews of apps and showed that complaints about functional errors (i.e., bugs) and crashes are the two most common complaints that are received by apps [8]. This highlights the importance of improving the quality of apps. 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 [13]. 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 ratings of apps [14].

Previous research has shown the usefulness of FindBugs and the importance of studying ratings and review-comments of mobile apps. In this study, unlike the past research, we examine the relationship between FindBugs and the user perceived quality of mobile apps.

6. THREATS TO VALIDITY

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

Construct Validity: In this study, we did not analyze the code of the apps in their original form. We ran FindBugs on the decompiled 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 [15].

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 the libraries are 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. However, we verified that the relationship between FindBugs warnings and ratings holds even when all libraries are included.

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 that have been used in past studies for reverse engineering Android apps [7, 14, 15]. 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 RQ, 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 [8]. By focusing on the reviews which gave a rating of 3 or less, we made sure that we focus on the complaints [6].

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 ratings that is similar to all apps in the Google play market[3]. Additionally, in our paper, we only consider one static analysis tool - FindBugs. Other static analysis tools like Coverity [4] or IBM Security AppScan [5] might yield different set

[3]http://www.appbrain.com/stats/android-app-ratings

[4]http://www.coverity.com/

[5]http://www-03.ibm.com/software/products/en/appscan

of warnings. Therefore such tools could be used to identify other warnings that could impact the user ratings as well. In order to control for the threat that arises due to using only FindBugs, we refrain to only making conclusions about FindBugs warnings throughout the paper. Future work could use the same analysis technique as our paper to examine the use of other static analysis tools in order to identify other types of warnings that could have a relationship with the user ratings.

Conclusion Validity: In our paper we examined a very large set of apps (10K) to identify if there was a relationship between specific types of FindBugs warnings and user ratings. We identified three categories of FindBugs warnings. Note that our claim is not that removing the FindBugs warning will increase the rating, rather only that they appear more significantly in low-rated apps. However, we did not carry out a controlled study to check if removing the FindBugs warnings would in fact increase the rating. While an interesting question, such a controlled study is out of the scope of this paper, and we would like to address this in future work.

7. CONCLUSION AND RECOMMENDATIONS

The most important take away for app developers is that there are three categories of FindBugs warnings that appear significantly more in low-rated apps. This means that these FindBugs 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 study provides a direct link between static analysis warnings from one tool and software quality (expressed as user ratings). In the future, we plan to examine other static analysis tools and how they could help developers improve the quality of their apps. We also intend to apply the same experimental process to the other steps in the quality assurance cycle, so that we can recommend a more complete set of quality assurance practices to mobile app developers.

References

[1] A. Vetro', M. Morisio, and M. Torchiano, "An empirical validation of FindBugs issues related to defects," in *15th Annual Conference on Evaluation Assessment in Software Engineering (EASE 2011)*, 2011, pp. 144–153.

[2] 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, pp. 108–111.

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

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

[5] 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.

[6] H. Khalid, M. Nagappan, E. Shihab, and A. 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.

[7] 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.

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

[9] 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.

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

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

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

[13] 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.

[14] 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.

[15] 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, pp. 113–122.