Software quality through the eyes of the end-user and static analysis tools

A study on Android OSS applications

Kamonphop Srisopha
University of Southern California
Los Angeles, California
srisopha@usc.edu

Reem Alfayez
University of Southern California
Los Angeles, California
alfayez@usc.edu

ABSTRACT

Source code analysis tools have been the vehicle for measuring and assessing the quality of a software product for decades. However, recently many studies have shown that post-deployment end-user reviews provide a wealth of insight into the quality of a software product and how it should evolve and be maintained. For example, end-user reviews help to identify missing features or inform developers about incorrect or unexpected software behavior. We believe that analyzing end-user reviews and utilizing analysis tools are a crucial step towards understanding the complete picture of the quality of a software product, as well as towards reasoning about the evolution history of it. In this paper, we investigate whether both methods correlate with one another. In other words, we explore if there exists a relationship between user satisfaction and the application's internal quality characteristics. To conduct our research, we analyze a total of 46 actual releases of three Android open source software (OSS) applications on the Google Play Store. For each release, we employ multiple static analysis tools to assess several aspects of the application's software quality. We retrieve and manually analyze the complete reviews after each release of each application from its store page, totaling 1004 reviews. Our initial results suggest that having high or low code quality does not necessary ensure user overall satisfaction.

CCS CONCEPTS

• **Software and its engineering** → *Software development process management*; *Software evolution*;

KEYWORDS

Software Engineering, Software Evolution, User Satisfaction, Static Analysis, Software Quality

1 INTRODUCTION

Software evolves over time, whether to adapt to a new hardware environment, to address the evolving needs of its users, or to repair defects. The abundance of software products in the market calls for software companies to thrive to make these changes and release new versions of software faster. This may force software developers to put less emphasis on managing software quality and more emphasis on delivering software on time. Consequently, the quality of software gradually deteriorates, which leads to higher code complexity, lower code reusability, and higher maintenance cost [3, 10].

In an effort to improve software quality, many source code analysis tools (e.g., PMD, FindBugs, and SonarQube) have been developed. Source code analysis, often referred to as static code analysis, is the type of software analysis that is performed on source code without executing the program. The analysis can reveal possible vulnerabilities, defects, or design issues at an early stage in the development phase. Static analysis tools provide numerous software metrics that can be used to give an insight into the quality of the code such as code complexity, code smells, comment density, etc. For example, software with a high number of code smells indicates low code quality [4]. Analyzing change in the software quality over time among different releases reveal how the software evolves and whether or not developers put emphasis on maintaining code quality.

While it is important for software to be developed with high code quality and within budget and schedule, successfully achieving them is still not enough to ensure a successful software product if the software does not meet user needs or expectations [1, 7]. Fortunately, application distribution platforms (e.g., Google Play Store or App Store) provide an outlet for users to share their assessment of the downloaded applications. Users can submit a written review and/or give a star rating on a scale of 1 to 5. Such reviews contain a wealth of information that can be used to help requirement engineers to better meet user needs (i.e., crowdsourcing for requirement elicitation), notify software maintainer of unexpected behavior, and inform other users about their experiences from using the application. In fact, mining app store reviews to improve the quality of software has already gained the attention of researchers [5]. An empirical study by Pagano et al. [8] found that the app store does serve as a communication channel among users and with developers. They also found that reviews have an impact on download numbers, that most reviews are provided shortly after new software releases, and that user review is triggered by new releases.

We believe that utilizing static code analysis tools and analyzing reviews are complementing steps in assessing software quality of a system since each approach reveals some unique characteristics about the software. This paper's contribution is twofold. First, it explores the extent to which user reviews in a form of text correlate

SQUADE'18:, May 28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5737-1/18/05…$15.00
https://doi.org/10.1145/3194095.3194096

with code quality. Second, it investigates whether user reviews in terms of ratings correlate with code quality. To conduct our analysis, we measured multiple aspects of software quality of 46 releases of three Android OSS applications by utilizing three static analysis tools: PMD[1], FindBugs[2], and SonarQube[3]. For each release, we retrieved the complete reviews (both texts and ratings) available from the application's store page. Each text review is tokenized into sentences. Each sentence is then manually classified based on three dimensions: Intention, Software quality, and Sentiment. We used the results from these methods in our study.

2 RESEARCH DESIGN

2.1 Research Questions

The main goal of this study is to understand whether user satisfaction relates to the application's internal quality characteristics. To guide our research, we formulate the following research questions:

- **RQ1:** To what extent do violations reported by static code analysis tools correlate with different types of review sentences?
- **RQ2:** To what extent do violations reported by static code analysis tools correlate with the average user ratings?

2.2 Data Collection

In order to answer these research questions, we analyzed a total of 46 releases of three Android OSS applications and all reviews from their store page after each release. In this section, we present our data collection method.

We selected Android applications based on the following five criteria. First, the application should be open source. Obtaining source code of a closed source application through decompiling can affect the outcome of the static analysis tools if its source code is protected by code obfuscation methods [11]. Second, the application should have adequate popularity among developers (i.e., more than 1000 stars on its Github repository). Third, since our study involves manual analysis, the application should not have more than 150 reviews between releases. Fourth, the application should have more than 100,000 downloads. Finally, the application should have at least 3,000 user ratings on its store page.

For each release of each application selected, we retrieved the source files from the application's GitHub releases page. Then we collected all user reviews of each release that we considered in our study using a web crawling tool we developed. Reviews that did not fall into the studied time span were discarded. Table 1 shows the three Android OSS applications we considered, along with their domain, number of repository stars, approximate download numbers (DL), total number of reviews used in this study (Rev.), number of releases analyzed (RL), and the time span between earliest and latest release version analyzed.

2.3 Annotation of Reviews

Pagano et al. [8] reported that popular applications, such as Facebook, can receive over 4,000 reviews per day. Studying a complete reviews of such an application after each release is therefore not a trivial task. While many researcher have proposed frameworks or tools that can classify information in reviews automatically through the use of machine learning and natural language processing (NLP) techniques, there were still several limitations that might affect the validity and outcome of our results if used. For example, ARdoc, an automated classifier by Panichella et al. [9] , incorrectly classified "*Changing account colour does not carry over to combined timeline.*" as unrelated to software developers. Nevertheless, we plan to investigate the feasibility of using such tools to help with our data collection in our future work.

Table 1: Characteristics of the studied applications

Application	Domain	Stars	DL	Rev.	RL	Time Span
Omni-Notes	Productivity	1288	100k+	295	22	08/15-01/18
Qksms	Communication	1733	500k+	507	10	12/15-07/16
Twidere	Social	1555	100k+	202	14	01/17-11/17

In this paper, text reviews were manually examined at the sentence-level granularity by two human annotators, one CS Ph.D student, the first author, and one Master's level CS student. Each sentence was classified based on three dimensions: Intention, Software quality, and Sentiment. To mitigate some threats to internal validity, the annotators were instructed to annotate at most 200 sentences per day as to avoid errors due to fatigue. We resolved category disagreements by discussion.

Intention concerns the purpose or the underlying goal of a given sentence. Each sentence was classified into four categories based on how relevant a sentence is to software developers performing software maintenance and evolution tasks. The four categories in this dimension are as follows:

- Problem Report: sentences describing issues or unexpected behaviors (e.g.,*"Pages load, display for 1 or 2 seconds then go completely white."*)
- Improvement Request: sentences suggesting or expressing needs or ways to improve or enhance the application (e.g., *"Wish it had the option to add new contact from messages."*)
- Inquiry: sentences attempting to inquire information from the developers or other users (e.g., *"How do you create tags?"*)
- Other: sentences that are irrelevant or uninformative (e.g., *"Thank you developer."*)

Software Quality: each sentence was classified with respect to the ISO/IEC 25010 software product quality definitions. ISO/IEC 25010 defines the quality of a system as the extent to which the system satisfies the stated and implied needs of its various stakeholders [6]. In our case, stakeholders refer to the end-users. The model consists of the following eight characteristics: functional suitability, performance efficiency, usability, portability, compatibility, reliability, maintainability, and security. Note that a sentence can be categorized into one or more software quality characteristics. For example, *"Fast, responsive, and highly customizable."* implies that the user was satisfied with the performance (time behavior - performance efficiency) and the set of functions the application offers (functional completeness - functional suitability).

[1] https://pmd.github.io/
[2] http://findbugs.sourceforge.net/
[3] https://www.sonarqube.org/

Sentiment concerns the tone of a sentence. Each sentence was classified into one of the three different levels of sentiment: *positive*, *negative*, or *neutral*.

Table 2 shows examples of sentences and their corresponding categories based on the three dimensions previously discussed. Table 3 lists the eleven types of review sentences used in our study. In particular, we selected "Problem report" and "Dissatisfaction" with different types of software quality characteristics because they constitute different granularity of "Negative sentiment" sentences. In addition, we also selected "Positive sentiment" because we hypothesized that it would show a negative correlation with results given by the static analysis tools.

Table 2: Examples of sentences

Sentence	Intention	SQ	Sentiment
"Unbearably slow to open, deleted it to save my patience."	Problem Report	Performance efficiency	Negative
"The UI is superb, easy navigable and awesome to use."	Other	Usability	Positive
"This app is by far the biggest battery drain app on my phone."	Problem Report	Performance efficiency	Negative
"Wish we could see notifications like who faved or rted our tweets."	Improvement Request	Functional Suitability	Neutral

Table 3: Eleven types of review sentences considered

Dimension	Sentence Type
Intention	Problem report
Sentiment	Negative, Positive
Software Quality	Dissatisfaction with {functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, portability}

2.4 Static Analysis

We executed PMD, SonarQube, and FindBugs on each release to measure multiple aspects of each application. Table 4 lists the quality metrics we considered in this study. Code Smells (CS) is a maintainability-related issue in the code reported by SonarQube. PD is a set violations reported by PMD. FB is a set of violations reported by FindBugs.

Table 4: Quality Metrics

Abbr.	Tool	Metric
CS	SonarQube	Number of code smells
PD	PMD	Empty code, Naming, Braces, Import statements, Coupling, Unused Code, Unnecessary, Design, Optimization, String and StringBuffer
FB	FindBugs	Dodgy code, Bad practice, Malicious code, Performance, Correctness, Security, Multithreaded correctness, Internalization

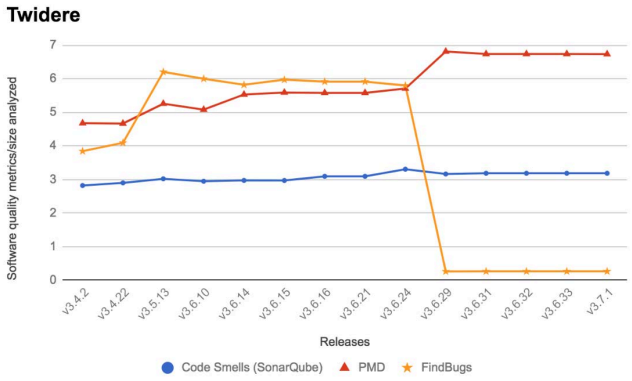

Figure 1: Results from applying the three static analysis tool on Twidere releases.

Since the size of each application was different, which would affect the total number of violations reported by the tools (i.e., the more lines of code analyzed, the more violations the tools could detect), we normalized the values of each quality metrics by the number of lines analyzed. Figure 1 shows the results of applying the three static analysis tools on Twidere application. We can observe from Figure 1 that Twidere began to incorporate FindBugs to measure its code quality starting from release v3.6.29. In fact, we found FindBugs plugin inside the dependencies block of the build file in release v3.6.29, but not in release v3.6.24.

3 ANALYSIS AND RESULTS

In this section, we explain the procedures to answer each research question and report results.

3.1 RQ1

To answer RQ1, we first paired each review sentence type with each quality metric. Then, we used Pearson correlation coefficient (r) to measure the strength of the relationship between each pair. To determine whether the correlation between them is significant, we set the significance level to 0.05. In addition, since review sentence types were collected as count (e.g., the total number of sentences with negative sentiment) and the duration between releases could increase the number of count (i.e., the longer the time between releases, the more reviews a release might get), we normalized the count of each type by the total number of review sentences for the corresponding release. We used these normalized values in our analysis.

The results reveal that only 7 out of 33 pairs showed a significantly strong correlation with one another ($p<0.05$). *"Negative sentiment"* (r=0.45, p=0.001), *"Problem report"* (r=0.36, p=0.014), *"Dissatisfaction with Functional suitability"* (r=0.35, p=0.019), *"Dissatisfaction with Performance efficiency"* (r=0.60, $p<0.01$), and *"Dissatisfaction with Reliability"* (r=0.54, $p<0.01$) had a significantly strong positive correlation with *"PD"*. This implies that the more violations reported by PMD, the more of these types of sentences the release gets. Similarly, *"Dissatisfaction with Performance efficiency"* (r=0.32, p=0.03) and *"Dissatisfaction with Reliability"* (r=0.33,

p=0.02) showed a significantly strong positive correlation with *"CS"*. Although *"Positive sentiment"* did show a negative correlation with *"CS"* and *"PD"* (i.e., less code quality violations, more positive reviews), the correlations were not statistically significance, p=0.11 and p=0.25 respectively. We found no correlation between each review sentence type and *"FB"*.

3.2 RQ2

In this question, we explore the extent to which a correlation exists between user satisfaction in a form of ratings and violations reported by static code analysis tools.

Similar to RQ1, we paired each quality metric with the average user ratings for each release. Then we used Pearson correlation coefficient (r) to measure how strong a relationship is between each test pair. We also set the significance level to 0.05.

The results show that all test pairs exhibited a negative correlation coefficient (r<0). However, only the pair of *"PD"* and "user ratings" showed a significantly strong negative correlation (r=-0.454, p=0.0019). The correlation coefficient (r) between *"CS"* and "user ratings" was -0.2315 and p-value was 0.13, while the correlation coefficient (r) between *"FB"* and "user ratings" was -0.0062 and p-value was > 0.1. In this case, these two test pairs failed to reject the null hypothesis. The results indicate that, to some extent, user ratings correlate with the application's internal characteristics. Our results support the findings of Stamelos et al. [12], who found that, to some extent, software internal characteristics negatively correlate to user satisfaction.

4 DISCUSSION

In this section, we provide an insight to explain the results in RQ1 and RQ2.

First, some of the eight software quality characteristics, such as Usability, can be subjective. For example, one user may feel that the application is easy to use or aesthetically pleasing, while others may not. In v2.7.0 of QKSMS, the developers changed the look of the application (e.g., icon), and this resulted in mixed reviews between those who liked the new design and those who did not. Conversely, we observed less diverse opinions when users experienced response-time degradation, crash, or unexpected behavior after updating the application. Chulani et al. [2] also found a strong correlation between user satisfaction with reliability and some types of defects.

Second, static analysis tools do not have the capability to measure, for example, whether the application "can replace another specified software product for the same purpose in the same environment." - Portability or whether the application has "a user interface that enables pleasing and satisfying interaction for the user." - Usability [6]. As a result, no correlation was found.

Third, we observed that users seldom discussed Security, Maintainability, Compatibility, and Portability aspect of the applications. This indicates that these types of software quality are difficult to be assessed by the end-users. Hence, no correlation was found.

Fourth, Pagano et al. [8] found that bug reports and negative messages have high influence on user ratings. Recall that in RQ1, *"PD"* showed a significantly strong correlation with *"Problem report"* and *"Negative sentiment"*. Therefore, strong correlation was found in the pair of *"PD"* and "user ratings" in RQ2.

Nonetheless, we will further investigate why we could observe a strong correlation in some test pairs in our future work. Nevertheless, the results of our analysis do confirm that, to some extent, having high or low code quality does not necessary ensure user satisfaction.

5 CONCLUSION AND FUTURE WORK

In this paper, we investigated the extent to which code quality relates to user satisfaction by utilizing static code analysis and end-user reviews in 46 actual releases of three Android OSS applications. Our results do confirm that code quality represents only one side of the coin. This implies that having high or low code quality, as reported by the tools, does not guarantee user overall satisfaction. This study provides an initial proof of technical value of analyzing user reviews, in a form of passive crowdsourcing, to gain an insight into the quality of software.

This work can be extended to several directions. For instance, we plan to conduct a large scale analysis to test if our preliminary findings hold, to investigate whether a specific type of review influences software release frequency, to incorporate analysis tools that are specifically tailored for Android applications, and finally to correlate user satisfaction to more software internal quality characteristics, such as technical debt.

ACKNOWLEDGMENTS

This material is based upon work supported in part by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract H98230-08-D-0171. It was also supported by the National Science Foundation grant CMMI-1408909.

REFERENCES

[1] Barry Boehm, J Lane, Supannika Koolmanojwong, and Richard Turner. 2014. The Incremental Commitment Spiral Model. (2014).
[2] Sunita Chulani, P Santhanam, Darrell Moore, Bob Leszkowicz, and Gary Davidson. 2001. Deriving a software quality view from customer satisfaction and service data. In *European Conference on Metrics and Measurement*. Citeseer.
[3] Stephen G Eick, Todd L Graves, Alan F Karr, J Steve Marron, and Audris Mockus. 2001. Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering* 27, 1 (2001), 1–12.
[4] Martin Fowler and Kent Beck. 1999. *Refactoring: improving the design of existing code.* Addison-Wesley Professional.
[5] Necmiye Genc-Nayebi and Alain Abran. 2017. A systematic literature review: Opinion mining studies from mobile app store user reviews. *Journal of Systems and Software* 125 (2017), 207–219.
[6] ISO/IEC 25010:2011 2011. *ISO/IEC 25010:2011: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.* Standard. ISO.
[7] Capers Jones. 2008. *Applied software measurement: global analysis of productivity and quality.* McGraw-Hill Education Group.
[8] Dennis Pagano and Walid Maalej. 2013. User feedback in the appstore: An empirical study. In *Requirements Engineering Conference (RE), 2013 21st IEEE International*. IEEE, 125–134.
[9] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. 2016. Ardoc: App reviews development oriented classifier. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 1023–1027.
[10] David Lorge Parnas. 1994. Software aging. In *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on*. IEEE, 279–287.
[11] Patrick Schulz. 2012. Code protection in android. *Insititute of Computer Science, Rheinische Friedrich-Wilhelms-Universitgt Bonn, Germany* 110 (2012).
[12] Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou, and Georgios L Bleris. 2002. Code quality analysis in open source software development. *Information Systems Journal* 12, 1 (2002), 43–60.