Please Please Me:*

Does the Presence of Test Cases Influence Mobile App Users' Satisfaction?

Vinicius H. S. Durelli
Federal University of São João del Rei
São João del Rei, Minas Gerais
durelli@ufsj.edu.br

Rafael S. Durelli
Federal University of Lavras
Lavras, Minas Gerais
rafael.durelli@dcc.ufla.br

Andre T. Endo
Federal University of Technology
Cornelio Procopio, Parana
andreendo@utfpr.edu.br

Elder Cirilo
Federal University of São João del Rei
São João del Rei, Minas Gerais
elder@ufsj.edu.br

Washington Luiz
Federal University of Minas Gerais
Belo Horizonte, Minas Gerais
washingtoncunha@dcc.ufmg.br

Leonardo Rocha
Federal University of São João del Rei
São João del Rei, Minas Gerais
lcrocha@ufsj.edu.br

ABSTRACT

Mobile application developers have started to realize that quality plays a vital role in increasing the popularity of mobile applications (apps), thereby directly influencing economical profit (in-app purchases revenue) and app-related success factors (i.e., number of downloads). Therefore, developers have become increasingly concerned with taking preemptive actions to ensure the quality of their apps. In general, developers have been relying on testing as their main quality assurance practice. However, little is known about how much mobile app testing contributes to increasing user level satisfaction. In this paper we investigate to what extent testing mobile apps contributes to achieving higher user satisfaction. To this end, we probed into whether there is a relation between having automated tests and overall user satisfaction. We looked into users ratings, which express their level of satisfaction with apps, and users reviews, which often include bug (i.e., fault) reports. By analyzing a quantitative indicator of user satisfaction (i.e., user rating), we found that there is no significant difference between apps with automated tests and apps that have been developed without test suites. We also applied sentiment analysis on user reviews to examine the main differences between apps with and without test suites. The results of our review-based sentiment analysis suggest that most apps with and without test suites score quite high for user satisfaction. In addition, we found that update-related problems are far more common in apps with test suites, while apps without test suites are likely to have battery-drain problems.

CCS CONCEPTS

• **Software and its engineering** → *Formal software verification*;

KEYWORDS

Software testing, mobile application, sentiment analysis

ACM Reference Format:
Vinicius H. S. Durelli, Rafael S. Durelli, Andre T. Endo, Elder Cirilo, Washington Luiz, and Leonardo Rocha. 2018. Please Please Me:: Does the Presence of Test Cases Influence Mobile App Users' Satisfaction?. In *XXXII BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING (SBES 2018), September 17–21, 2018, Sao Carlos, Brazil*. ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3266237.3266272

1 INTRODUCTION

The mobile application (usually referred to as mobile app) market has been growing steadily over the last decade. As a result, the software industry has employed thousands of developers to produce a myriad of mobile applications and generate millions of dollars in revenue [2]. Currently, mobile apps are available through many app stores (or app marketplaces), two notable examples are Apple App Store[1] and Google Play Store[2]. App stores act as online platforms where users can browse through a catalog of apps, download apps, and provide feedback concerning their user-experience using the downloaded apps (which usually takes the form of mobile app user reviews).

Mobile app development is rife with challenges. One of the major challenges is meeting the high expectations of mobile app users. To evaluate the quality of mobile apps, some app stores enforce development policies and publishing guidelines that must be observed by app developers.In addition, apps often have to go through an evaluation before being approved and showcased in the app store.

Due to the rapidly growing mobile market, unlike desktop- and web-based software users, mobile app users can choose from a wide variety of apps. Given their greater choice, mobile app users have developed a low tolerance for faulty apps. Naturally, the quality of mobile apps greatly influences how users perceive their experience using these apps. Some studies suggest that apps that fail to meet users' expectations end up deleted from users' devices [11]. Therefore, we surmise that quality issues can negatively impact an app's rating and reviews and ultimately hurt its popularity over time.

App developers have recognized that quality plays a critical role in driving popularity and have started taking preemptive actions to ensure the quality of their apps [7, 20]. Most developers have been

*In this paper we explore whether automated testing helps developers to create high quality apps that "please" the end user, so the title – as made famous by The Beatles in 1963 – "Please Please Me".

SBES'18, September 2018, São Carlos, São Paulo, Brazil

ACM ISBN 978-1-4503-6503-1/18/09…$15.00
https://doi.org/10.1145/3266237.3266272

[1] https://www.apple.com/ios/app-store/
[2] https://play.google.com/store/apps

relying on testing as their main quality assurance practice. Despite the growing interest in devising better approaches to testing mobile apps as well as building tools that can automate and improve testing of mobile apps [29], little research has been done to explore the relation between having automated tests and user satisfaction [18]. Therefore, this paper examines the contribution of mobile app testing in increasing the level of satisfaction of users with apps in the context of the Android operating system (OS) and the Google Play Store. We conjecture that apps that include some sort of automated testing (i.e., test suites) fare better in terms of quality and thus satisfy user expectations better. More specifically, we set out to examine the following research questions:

RQ$_1$ Do apps with test suites score higher in terms of user satisfaction than apps without test suites?

- **RQ$_{1.1}$** Do apps with test suites achieve higher average ratings than apps without test suites?
- **RQ$_{1.2}$** Do apps with test suites achieve more positive reviews?

We analyzed 60 mobile apps. Although the number of apps we investigated is relatively small, the analysis we carried out was sound and we believe that the results are of general interest to the software engineering community. Our experimental results on a real-world app sample would seem to indicate the following:

- Both our qualitative and quantitative results would seem to suggest that apps with and without test suites score similarly in terms of user satisfaction.
- However, these two groups of apps are prone to different problems that lead to poor reviews: *(i)* apps with test suites have a high incidence of update-related problems and *(ii)* apps without tests are fraught with battery-drain issues.

The remainder of this paper is organized as follows. Section 2 and 3 provide background on topic modeling and sentiment analysis, presenting definitions and a brief description of the review-based sentiment analysis approach we used in this paper. Section 4 presents related work. Section 5 describes the experimental design and discusses threats to validity. Section 6 presents results, statistical analysis, and compares apps with and without test suites in terms of proxies for user satisfaction. Section 7 discusses our findings in the light of previous studies. Section 8 presents concluding remarks.

2 TOPIC MODELING

Topic modeling (TM) is centered around the problem of discovering relations among documents and topics, as well as relations among the terms that compose the documents and topics, thus enabling the organization of textual documents according to the discovered topics. Basically, TM can be broken down into three main steps: *(i)* data representation, *(ii)* latent topic decomposition, and *(iii)* topic extraction. During data representation, textual documents are usually represented by adopting a fixed-length vector representation based on simple term occurrence information, which is encoded by term frequency - inverse document frequency (TF-IDF). The methods used in the second step (i.e., latent topic decomposition) can be divided into two main models: probabilistic and non-probabilistic models. Essentially, when using probabilistic models, data is modeled based on two concepts: *(i)* each document follows a topic distribution $\theta^{(d)}$; *(ii)* and each topic follows a term distribution $\phi^{(z)}$. $\theta^{(d)}$ and $\phi^{(z)}$ reflect to what extent a term is important to a topic and to what extent a topic is important to a document, respectively [5]. Non-probabilistic topic modeling employs strategies such as matrix factorization, wherein a dataset with n documents and m different terms is encoded as a design matrix $A \in \mathbb{R}^{n \times m}$ and the goal is to decompose A into sub-matrices that preserve some desired property or constraint. The two main non-probabilistic strategies are *(i)* singular value decomposition (SVD) [6] and *(ii)* non-negative matrix factorization (NMF) [15]. Several strategies have been proposed for extracting relevant topics from a set of previously identified topics. Bicalho et al. [1] proposed a strategy based on NMF: semantic topic combination (SToC), whose main goal is to group semantically similar topics. In this paper, to answer our research questions we consider TF-IDF, NMF, and SToC as our data representation, topic decomposition, and topic extraction strategies of choice, respectively.

3 SENTIMENT ANALYSIS

Sentiment analysis is concerned with the task of automatically detecting the polarity (i.e., positive or negative) of sentiment in textual representations [9]. Basically, there are two types of methods for carrying out sentence-level analysis: *(i)* supervised based and *(ii)* lexicon based. Since supervised methods need to process labeled data, these methods do not perform properly when complex data (i.e., real world textual data) has to be taken into account. Consequently, recent efforts have come up with sentence-level methods based on lexicons. Lexicon-based methods create lexicons and combine different natural language processing techniques to determine a sentence polarity [9]. In this paper, we adopted a lexicon based strategy [16] that outperforms several supervised strategies.

4 RELATED WORK

Several studies have been exploring user feedback to understand the most problematic features in apps [16], to probe into the relationship between bugs and end user ratings [14], or even to obtain ideas for app improvements [8]. In [14], Khalid et al. analyzed 10,000 Android mobile apps to examine whether poor ratings are related to certain categories of static analysis warnings. They observed that the "*Bad Practice*", "*Internationalization*", and "*Performance*" categories have significantly higher densities in low-rated apps than high-rated apps. Moreover, the results also show that static analysis tools (e.g., `FindBugs`) can help developers to identify certain types of bugs that have the potential to result in poor ratings.

In [8] [16], the authors were interested in extracting app features and users' sentiments associated to them in order to help developers to capture the most problematic features and understand the needs of app users. The authors introduced a method to automatically extract features from mobile app reviews. Their methods aggregate the sentiments of users for each feature by applying automated sentiment analysis techniques on app reviews. They observed that sentiment analysis can be successfully applied to requirements evolution tasks and can help developers to capture user opinions on a single feature or filter irrelevant reviews. Their method also

sheds some lights on features that more positively or negatively impact the overall evaluation of mobile apps.

In [3], Carreño and Winbladh proposed an approach to explore user feedback. Their approach automatically extracts topics representing new requirements that better represent users' needs. Their technique extracts topics prevalent in user comments and allows these topics to be analyzed by software developers. Similarly, in [10], the authors also propose MARA (`Mobile App Review Analyzer`) a prototype for automatic retrieval of mobile app feature requests from online reviews. In a recent study [21], Nayebi et al. argue that app store mining is not enough for app improvement. Nayebi et al. investigated how twitter can provide complementary information to support mobile app development. The authors applied machine learning classifiers, topic modeling, and crowd-sourcing to successfully mine 22.4% feature requests and 12.89% bug reports from Twitter. Moreover, they also found that 52.1% of all feature requests and bug reports were discussed on both tweets and reviews.

5 EXPERIMENT SETUP

As mentioned, in response to the competitive market that favors good-quality mobile apps, developers have been testing their apps prior to release. Thus, we set out to investigate whether testing mobile apps contributes to achieving higher average user satisfaction. We conjecture that mobile apps with sound test suites tend to score higher in terms of user satisfaction than poorly tested apps.

This section describes the experiment setup we used to investigate the following research questions (RQs):

RQ$_1$ Do apps with test suites score higher in terms of user satisfaction than apps without test suites?

- **RQ**$_{1.1}$ Do apps with test suites achieve higher average ratings than apps without test suites?
- **RQ**$_{1.2}$ Do apps with test suites achieve more positive reviews?

The elements that constitute *user satisfaction* (**RQ**$_{1.2}$) in the context of this experiment are defined in Subsection 5.1.2. In Subsection 5.1.2 we also set forth the hypotheses we set out to investigate in this experiment and provide the *operational definitions* [26] of user satisfaction.

5.1 Scoping

5.1.1 Experiment Goals. Defining the scope of an experiment comes down to setting its goals. We used the organization proposed by the Goal/Question/Metric (GQM) [28] template to do so. According to this goal definition template, the scope of our study can be summarized as follows.

Analyze if testing mobile apps leads to greater user satisfaction
for the purpose of evaluation
with respect to user ratings and user sentiment analysis
from the point of view of the researcher
in the context of users, i.e., customers, evaluating mobile apps.

5.1.2 Hypotheses Formulation. In the context of our study, user satisfaction is defined as the extent to which users believe that an app meets their needs and provides a frustration-free experience. To answer **RQ**$_1$ we analyzed two proxy measures of user satisfaction: *(i)* users' ratings and *(ii)* sentiment analysis of users' reviews. Thus, **RQ**$_1$ was further broken down into two RQs: **RQ**$_{1.1}$ and **RQ**$_{1.2}$. We framed our prediction for **RQ**$_{1.1}$: mobile apps with test suites achieve higher average ratings than apps without test cases. **RQ**$_{1.1}$ was turned into the following hypotheses:

> **Null hypothesis, $\mathbf{H}_{0-ratings}$:** there is no difference between mobile apps with test suites and mobile apps with no test suites in terms of average users' ratings. **Alternative hypothesis, $\mathbf{H}_{1-ratings}$:** mobile apps with test suites achieve higher average users' ratings when compared to apps without test suites.

RQ$_{1.2}$ was translated into the following hypotheses:

> **Null hypothesis, $\mathbf{H}_{0-reviews}$:** there is no difference in how users perceive mobile apps with test suites and mobile apps without test suites. **Alternative hypothesis, $\mathbf{H}_{1-reviews}$:** there is a significant difference in how users perceive apps with test suites and apps without test suites.

Given that our main goal is to investigate whether testing apps leads to higher average ratings and greater user satisfaction, this experiment has one treatment and one control group: the treatment is apps with test suites and the control group includes only apps without test suites. Let μ be the average rating and r the rating. So, μ_{r_T} and $\mu_{n_{NT}}$ denote the average rating of apps with test suites (i.e, T) and apps without test suites (i.e., NT).[3] Then, the first set of hypotheses can be formally stated as:

$$\mathbf{H}_{0-ratings}: \mu_{r_T} = \mu_{n_{NT}}$$

and

$$\mathbf{H}_{1-ratings}: \mu_{r_T} > \mu_{n_{NT}}$$

Similarly, to gain a better understanding and answer **RQ**$_{1.2}$ we investigated whether apps with test suites lead to greater user satisfaction in comparison to apps that were developed without test suites. Let s be the perceived satisfaction of the app user then the second set of hypotheses can be formulated as follows:

$$\mathbf{H}_{0-reviews}: \mu_{s_T} = \mu_{e_{NT}}$$

and

$$\mathbf{H}_{1-reviews}: \mu_{s_T} \neq \mu_{e_{NT}}$$

5.2 Variables Selection

As mentioned, the purpose of this experiment is to evaluate whether the presence of automated tests leads to higher end user satisfaction. Thus, we are particularly interested in two dependent variables:

- User ratings; and
- User reviews on Google Play Store.

We believe that these two dependent variables are the main indicators of app quality available to those interested in downloading an app. The most straightforward indicator of user satisfaction in the Google Play store is user rating: apps are ranked based on a five-star rating scale. In this rating system, users can assign one

[3] Here, T stands for apps with Test suites and NT stands for apps with No Test suites, respectively.

to five stars to rate apps, with five stars representing the highest quality. In order to answer $\mathbf{RQ}_{1.1}$ we looked into user ratings.

Apart from ratings, users can also write reviews. We argue that the textual content in user reviews also represents an important dimension that can be explored from a software quality perspective. In effect, previous research [4] has shown that the information contained in user reviews can be helpful for app developers. However, due to the unstructured and somewhat informal nature of reviews [19], the quality of the feedback provided by users varies greatly, from useful fault reports and feature requests to generic praises and complaints. We believe that sentiment analysis allows for an objetive interpretation of information that might be otherwise hard to evaluate properly, i.e., user reviews. Therefore, to overcome the complexities involved in analyzing unstructured textual information (i.e., written language) and answer $\mathbf{RQ}_{1.2}$, we employed a sentiment analysis method to synthesize all useful information from user reviews. In this context, we conjecture that the sentiment expressed in informative reviews can be used to gauge the quality of mobile apps.

5.3 Sample Selection

To perform this experiment, we divided the population of open-source app projects into two groups: projects that include test suites (i.e., treatment group) and projects that were developed without any sort of automated test suite (i.e., control group). Given that our sample was randomly selected from an open source repository, we tried to include a wide range of apps that differ in size, complexity, and category. We adhered to the following criteria in constructing our sample:

- *Open-source projects:* we randomly selected open-source apps listed in F-Droid[4] and hosted in GitHub[5] repositories. We followed the guidelines proposed by Kalliamvakou et al. [13] during the construction of our sample.
- *Availability at the Google Play Store:* As rating and reviews are essential data for this study, all apps and apps-related information should be available online. We have settled on using Android apps distributed by the Google Play Store.
- *Minimum number of reviews:* in order to perform user sentiment analysis, we had to select apps with a significant number of reviews: as described in Subsection 6.3, for part of our sample we selected apps that have at least 350 reviews.
- *Presence of test suites:* for the treatment group, we focused on selecting projects that include automated test cases. In effect, the treatment group was composed of apps with automated tests and the adopted test-to-code-ratio was 1 line of test code to 10 lines of production code (i.e., 1:10): a ratio of 1:10 indicates that, for every line of test code, there are 10 lines of production code. It is worth noting that settling for the presence of automated tests is an approximation and other factors might impact the results. Furthermore, the aspects tested in the selected apps vary and that should influence the result. This is further discussed in Subsection 5.4.

We used a tool named `Prof.MApp` (*Profiling Mobile Apps*) [25] to gather data on mobile apps. Given an Android app, `Prof.MApp` inspects the source code to classify files in two categories: production and test code; from analyzing Java source files in the latter, we extracted information related to the presence of automated test.

[4]https://f-droid.org/
[5]https://github.com

5.4 Validity Evaluation

As with any experimental study, this experiment has several threats to validity. Internal validity is concerned with the confidence that can be placed in the cause-effect relationship between the treatments and the dependent variables in the experiment. External validity has to do with generalization, namely, whether or not the cause-effect relationship between the treatments and the dependent variables can be generalized outside the scope of the experiment. Conclusion validity focuses on the conclusions that can be drawn from the relationship between treatment and outcome. Finally, construct validity is about the adequacy of the treatments in reflecting the cause and the suitability of the outcomes in representing the effect. We categorized all threats to validity according to this classification.

5.4.1 Internal Validity. We mitigated the selection bias issue by using randomization. However, since we assumed that all types of mobile apps have the same characteristics, no blocking factor was applied to minimize the threat of possible variations in, for instance, the complexity of the apps, usability, and performance. Thus, we cannot rule out the possibility that some variability in how end users perceive the quality of the chosen apps stems from other quality factors as opposed to the amount of fault-related problems in the chosen apps. Also, we had to settle for apps that have any sort of automated test cases, which can be seen as an approximation that does not account for other factors that might impact the results. This is an oversimplification of an ideal scenario where mobile apps have many types of test cases (e.g., user interface test cases, functionality test cases, security test cases). In effect, our analysis shows that the majority of app developers push testing aside when trying to get their apps to the market, so we could not find apps that have been thoroughly tested. We had to compromise for apps that have a certain test-to-code ratio, thus these apps lack tests for several aspects. Our analysis is concerned with finding if the inclusion of any sort of automated test helps developers to deliver predictive business metrics as user satisfaction.

5.4.2 External Validity. The sample might not be representative of the target population. As mentioned, we randomly selected apps for the treatment and the control groups. However, we cannot rule out the threat that the results could have been different if another sample had been selected. Another potential threat to the external validity of our results is that our analysis includes only apps from Google Play Store. We did not take apps from different mobile marketplaces into account because the tool we used to extract app information is tailored towards analyzing Android programs (as mentioned in Subsection 5.3). Another reason we did not consider different mobile technologies and marketplaces also has to do with the fact that most studies involving mobile apps are centered around open source apps available on the Google Play Store. In a way, Android is seen by app developers as a more open-source-friendly platform than iOS. Although, Google Play Store is a very popular app store, further investigation including more app

stores is required before we can determine whether our findings can be generalized for different app stores and different developer populations. Consequently, we cannot be confident that the results hold for a more representative sample of the general population.

5.4.3 Conclusion Validity. The main threat to conclusion validity has to do with the quality of the data collected during the course of the experiment. More specifically, the quality of ratings and reviews is of key importance to the interpretation of our results. Given that end users are free to rate apps and write reviews that might not always reflect the attributes of a given app. Thus, it is possible that some data is not correct due to misguided or ill-informed reviews or even fabricated information. It is worth noting, however, that data inconsistencies were filtered out during the sentiment analysis process.

5.4.4 Construct Validity. The measures used in the experiment may not be appropriate to quantify the effects we intend to investigate. For instance, user-generated app reviews may neither be the only nor the most important predictor of user satisfaction. If the measures we used do not properly match the target theoretical construct, the results of the experiment might be less reliable.

6 EXPERIMENTAL RESULTS

In this section we describe the experimental results for the 60 mobile apps we analyzed.[6] First we discuss some descriptive statistics, then present the hypothesis testing.

6.1 Descriptive Statistics

As shown in Table 1, the size of the apps ranges from 159,841 (141,023 lines of production code and 18,818 lines of test code) to 114 lines of code. Among the apps that include test suites, `GreenBits` is the one with most production code and test code (18,818), the app with the least lines of test code is `RPN` (180). On average, the apps in the treatment group contain around 4,436 lines of test code. As for the apps that do not include test suites (i.e., control group), `Vuze Remote` is the largest app, containing 41,209 lines of production code. The smallest app in the control group (`Simple Reboot`) has 114 lines of production code.

From observing Tables 1, 2, and 3 it can be seen that most apps in both groups have received a significant amount of feedback from users. Considering the apps with test suites, `ZXing` is the app that has received most user feedback: 624,366 user reviews (Table 1). The least reviewed app in this group is `Loyalty Card Keychain`, with only 91 users reviews (Table 1). As for the apps in the control group, the most reviewed app has received feedback from 39,731 users (`No-frills CPU Control`), the app with the least amount of feedback has been reviewed by 531 users (`Reddinator`).

Due to the outliers in our data concerning the amount of feedback received by the apps, we consider the trimmed mean and the median values in Tables 2 and 3 to be more accurate measures of central tendency than the mean. Therefore, on average (trimmed mean), the apps in the treatment group have received approximately 7,389.04 reviews, while the apps in the control group received 4,153.46 reviews. Additionally, also due to aforementioned outliers, the median absolute deviation (MAD) is a more robust measure of statistical dispersion than the standard deviation (see Tables 2 and 3).

Since the age of an app project might be an indicator of app maturity, we also looked into the lifespan of the app projects in both groups. In the context of our study, a given project's lifespan represents the time (in days) since the project's repository was created on GitHub. All apps in both groups have been under development for at least one year. In the treatment group, the longest-running project is `SMS Backup+`, which has been under development for over eight years (2,976 days). The most recent app is `Loyalty Card Keychain`, whose repository has been available for a little over than two years (759 days). On average (mean), apps on the treatment group have been developed and maintained for 1,874 days. In the control group, the longest-running project (`Sokoban`) has been around for 2,690 days, and the most recent app (`Camera Roll - Gallery`) has been available for a little over than one year (419 days). The apps in the control group have been under development for an average of 1,537 days.

Another possible indicator of project maturity is the number of commits to a project repository: this metric refers to the total number of commits that have occurred within an app project throughout its lifespan. To some extent, this quantitative metric is also reflects how actively developers have been participating in the project. In the treatment group, the app that has had most commits is `c:geo`: 11,049 (making it by far the app that has changed the most considering both groups). The app whose project has changed the least is `Speech Trainer`, with only 31 commits. In the control group, the app that has had the largest amount of commits is `Lightning` (1,698) and the one with least amount of commits is `Sokoban` (14).

Prior to an in-depth analysis of the textual content in apps reviews, we focused on a more quantitative analysis of user satisfaction. As mentioned, a proxy that provides a more quantitative indicator of user satisfaction is user rating, which is represented as a five-star rating scale. In order to answer $\mathbf{RQ}_{1.1}$ we probed into user ratings. The user ratings of the two groups is summarized in the boxplots shown in Figure 1. Overall, ratings in our sample are very positive for both groups. In Figure 1 the thick horizontal lines represent the medians, so apps with test suites achieve a slightly higher median rating (4.4 stars) than apps without test suites (4.3 stars). In Figure 1 the boxes around the medians encompass the 25th (Q1) through the 75th (Q3) percentiles. The "whiskers", i.e., the lines above and below the boxes, highlight the range of the data, excluding possible outliers. This is determined by the interquartile range (IQR), which is defined as IQR = Q3 - Q1. Values that are more than $1.5 \times$ IQR below or above the box are considered outliers. It is worth noting that we also added markers for the mean to the boxplots: in Figure 1 the mean is shown with a diamond. A more in-depth analysis of the results presented in Figure 1 is presented in the following subsection.

6.2 Hypothesis Testing

As mentioned, one of our goals is to examine whether apps that include test suites have higher average ratings, thereby indicating that apps with some sort of automated test lead to greater user satisfaction. To this end, we devised two hypotheses: the null hypothesis $\mathbf{H}_{0-ratings}$ and the alternative hypothesis $\mathbf{H}_{1-ratings}$

[6] The raw data of the experiment is available at: http://ow.ly/rP9G1013urH.

Table 1: Mobile apps we analyzed in our experiment. The entries in the table are in descending order by the number of lines of code (i.e., production code).

App Name	Production Code	Test Code	Google Play Store and GitHub Information			
			Google Play Reviews	Rating	Project Lifespan	#Commits
Treatment Group (Apps with Test Suites)						
GreenBits	141,023	18,818	289	3.5	1,133	1,775
K-9 Mail	68,249	13,272	87,972	4.3	2,581	7,606
c:geo	64,469	8,432	57,427	4.4	2,424	11,049
OpenKeychain	50,669	6,723	2,235	4.5	2,182	6,672
Mirakel	42,109	5,444	280	4.1	1,842	3,722
AntennaPod	38,183	6,549	17,058	4.6	2,038	4,165
ownCloud	37,104	4,003	6,710	3.0	2,013	6,374
My Expenses	32,633	4,426	7,117	4.3	2,191	6,448
Etar	31,991	3,958	196	4.6	908	5,589
And Bible	31,401	4,816	5,648	4.6	2,346	2,634
ZXing	31,323	7,706	624,366	4.1	2,331	3,421
Kore	27,515	4,659	13,956	4.4	1,141	670
AnySoftKeyboard	22,607	3,689	22,139	4.4	2,142	4,041
Mozilla Stumbler	21,350	2,578	889	4.5	1,700	2,698
GnuCash	17,891	2,531	4,203	4.3	2,110	1,679
Materialistic	14,131	11,220	2,224	4.7	1,133	1,686
Wikimedia Commons	11,034	1,593	242	4.2	905	3,231
GPS Logger for Android	9,743	1,014	4,199	4.0	2,445	1,364
Dir	8,786	1,775	239	4.4	2,119	814
Agit: Git client	8,651	1,290	142	3.2	2,741	894
SMS Backup+	7,165	2,558	59,648	4.4	2,976	1,554
Termux	6,954	2,018	10,920	4.7	859	409
WiFiAnalyzer	6,946	9,262	7,508	4.4	811	1,049
Equate	5,547	1,045	211	4.8	1,405	390
Ministocks	3,972	410	2,847	4.1	1,782	249
Calendar Widget	2,438	850	8,473	4.5	2,117	414
Loyalty Card Keychain	2,124	965	91	4.5	759	378
Blokish	2,082	246	2,082	4.3	2,506	97
Speech Trainer	1,261	1,057	241	3.9	2,306	31
RPN	832	180	200	4.5	2,268	33
Control Group (Apps without Test Suites)						
Vuze Remote	41,209	0	3,376	4.3	1,146	41
Transdrone	23,969	0	3,556	4.3	1,705	560
Linux CLI Launcher	21,721	0	8,183	4.7	657	165
LibreTorrent	16,919	0	778	4.3	497	317
Camera Roll - Gallery	16,373	0	1,083	4.5	419	313
Lightning	14,748	0	1,343	3.5	1,853	1,698
Pathfinder Open Reference	12,758	0	7,781	4.7	2,262	249
Reddinator	11,812	0	531	4.3	1,811	357
WhereYouGo	11,290	0	1,864	3.7	1,282	224
Sokoban	9,816	0	893	4.4	2,690	14
OpenVPN for Android	9,457	0	29,637	4.4	1,552	1,004
Taskbar	8,981	0	2,284	4.4	572	659
OpenSudoku	6,079	0	15,704	4.6	2,197	241
SealNote	5,531	0	2,562	4.5	1,121	337
Night Screen	5,399	0	6,026	4.5	758	136
Binaural Beats Therapy	4,630	0	13,663	4.2	2,628	104
OONI Probe	4,529	0	1,028	4.6	1,078	415
HN - Hacker News Reader	4,035	0	1,362	4.4	2,011	306
Torchie Torch	3,408	0	2,265	4.2	758	135
Ted	2,300	0	2,100	4.6	2,192	41
Pedometer	2,249	0	2,663	4.1	1,557	438
No-frills CPU Control	1,997	0	39,731	4.3	1,848	38
Tinfoil for FB	1,560	0	10,109	4.1	2,319	308
Activity Launcher	1,168	0	3,664	4.3	1,471	49
Counter	946	0	1,791	4.5	2,220	206
OI Flashlight	750	0	1,840	4.1	2,113	73
Locale	668	0	7,729	3.7	461	93
Ridmik Bangla Dictionary	563	0	8,421	4.6	1,891	36
Wikivoyage	215	0	570	2.7	1,872	23
Simple Reboot	114	0	4,097	4.3	1,164	16

(described in Subsection 5.1.2). To test these hypothesis we applied a well-known approach to comparing two independent groups: Wilcoxon's rank-sum test [27]. According to the results of this non-parametric test, apps with automated tests (median = 4.4, Table 2) do not differ significantly from apps without tests (median = 4.3, Table 3), $W = 470$, $p = 0.771$, $r = -0.04$.[7]

[7] We used the following equation to calculate the effect size: $r = \frac{z}{\sqrt{N}}$ in which z is the z-score and N is the size of the study (i.e., number of apps) in which z is based.

Table 2: Descriptive statistics summarizing the characteristics of the treatment group (i.e., group of apps that include test suites).

	Overview of the Treatment Group			
	Google Play Reviews	Rating	Lifespan	Commits
Max	624,366	4.8	2,976	11,049
Min	91	3	759	31
Mean	31,658.4	4.27	1,873.80	2,704.53
Trimmed	7,389.04	4.35	1,895.29	2,318.67
Median	3,523.0	4.4	2,113.5	1,682.5
Std Dev	113,831.38	0.42	641.29	2,748.98
MAD‡	4,918.53	0.22	491.48	2,029.68

‡MAD stands for median absolute deviation.

Table 3: Descriptive statistics summarizing the characteristics of the control group (i.e., group of apps that do not include test suites).

	Overview of the Control Group			
	Google Play Reviews	Rating	Lifespan	Commits
Max	39,731	4.7	2,690	1,698
Min	531	2.7	419	14
Mean	6,221.13	4.26	1,536.83	286.53
Trimmed	4,153.46	4.33	1,545.46	215.92
Median	2,612.5	4.3	1,631.0	215.0
Std Dev	8,762.01	0.41	675.59	346.36
MAD‡	2,308.41	0.30	788.0	210.53

‡MAD stands for median absolute deviation.

6.3 Review-based Sentiment Analysis

As mentioned, we surmise that some of the sentiment expressed in written reviews can be used to shed some light on the quality of mobile apps. Several studies show that reviews include valuable information about user experience and that such information can be used as a proxy to evaluate quality [8, 22]. Furthermore, according to Rodrigues et al. [24], it is common that the star rating assigned to an app does not reflect the contents of the associated written review: that is, the information in reviews often does not match with the star rating given to the app. We believe that a sentiment-based method is an effective way to probe into a corpora of written accounts of user experience. Thus, to further investigate the benefits of automated tests, we employed a sentiment analysis method to synthesize useful information from user reviews. More specifically, we adopted a lexicon based method [16] that allowed us to filter, summarize, and analyze app reviews. The method we employed is able to automatically extract relevant information from reviews and analyze the sentiment (strength) associated with such information.

In hopes of obtaining more meaningful results, we sorted through all the apps in both groups and selected two subsets comprised of the 15 apps with the most amount of reviews in each group. It is worth mentioning that all 30 apps we selected have at least 350 written reviews. Table 4 gives an overview of the subsets of apps we selected to perform sentiment analysis.

Figure 2 gives an example of the output yielded by our sentiment analysis method for each app. As shown in Figure 2, the overall sentiment strength (in a star rating scale from one to five) appears in the top part of the example output. Additionally, both the topics extracted by the topic modeling method and their respective sentiment score are shown in the example output in Figure 2. Similarly to

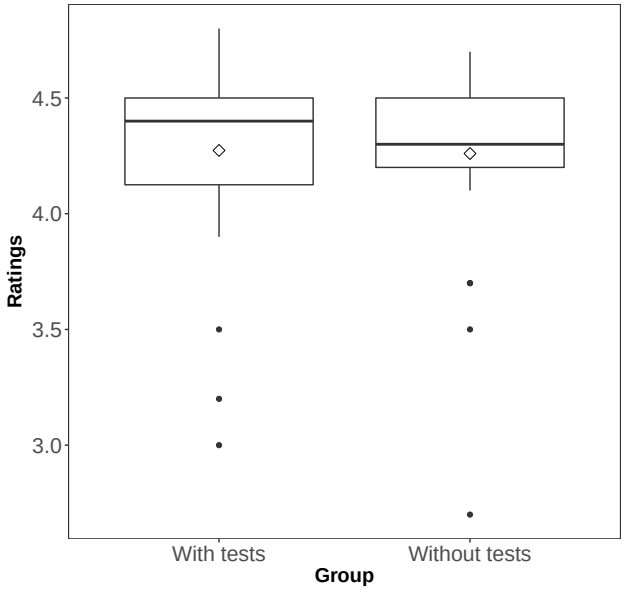


Figure 1: Overview of the ratings of the two app groups.

Luiz et al. [16], we normalized the sentiment strength to values that range from 1 to 5. For instance, an app whose sentiment inferred from our analysis was 80% positive and 20% negative has a normalized sentiment score of 4 stars. The arrows indicate whether the set of topics in a given row are below average (in which case the arrow appears in red) or above average (green arrow pointing upwards) when compared to the overall sentiment strength (inferred from all reviews of an app). We used these arrows to pick out topics that negatively influence the overall rating of an app.

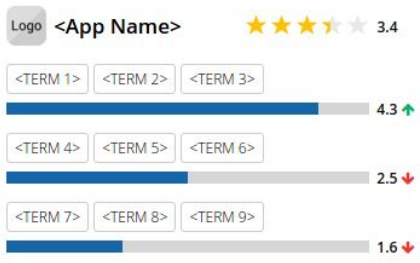


Figure 2: An example of output from the sentiment analysis method we used in our study.

As shown in Table 4, the results of the review-based sentiment analysis indicate that most apps in the treatment group scored quite high for user satisfaction. The smallest score was 3.2 (`ownCloud`) and the highest score was 4.7 (`Materialistic`). On average (mean), the score of the apps in this group was 3.93.

As mentioned, the approach we used for topic modeling and sentiment analysis also yields a high-level summary of the topics that tend to co-occur in the same reviews as well as the general sentiment associated with these topics. By analyzing these topics we found that apps with test suites are adversely affected by updates.

Table 4: Subsets of apps with most reviews.

App Name	#Reviews	SA‡	Rating
Treatment Group (Apps with Test Suites)			
c:geo	4,480	3.8	4.4
K-9 Mail	4,480	3.3	4.3
SMS Backup+	4,480	3.8	4.4
AntennaPod	3,198	4.2	4.6
AnySoftKeyboard	2,966	4.0	4.4
Termux	2,295	4.3	4.7
Kore	1,890	3.9	4.4
Calendar Widget	1,623	4.2	4.5
And Bible	1,327	4.5	4.6
ownCloud	1,035	3.2	3.0
Ministocks	760	3.9	4.1
ZXing	560	3.4	4.1
Blokish	491	4.0	4.3
OpenKeychain	402	3.8	4.5
Materialistic	396	4.7	4.7
Control Group (Apps without Test Suites)			
OpenVPN for Android	2,629	4.0	4.4
Binaural Beats Therapy	2,597	4.1	4.2
Linux CLI Launcher	2,584	4.2	4.7
No-frills CPU Control	2,577	4.2	4.3
Locale	2,367	3.8	3.7
Tinfoil for Facebook	2,312	3.8	4.1
Pathfinder Open Reference	1,636	4.4	4.7
Ridmik Bangla Dictionary	1,406	4.4	4.6
Night Screen	1,053	3.7	4.5
SealNote	821	4.1	4.5
Torchie	618	3.8	4.2
OpenSudoku	600	4.5	4.6
Transdrone	587	4.0	4.3
Taskbar	520	4.3	4.4
Vuze Remote	516	4.0	4.3

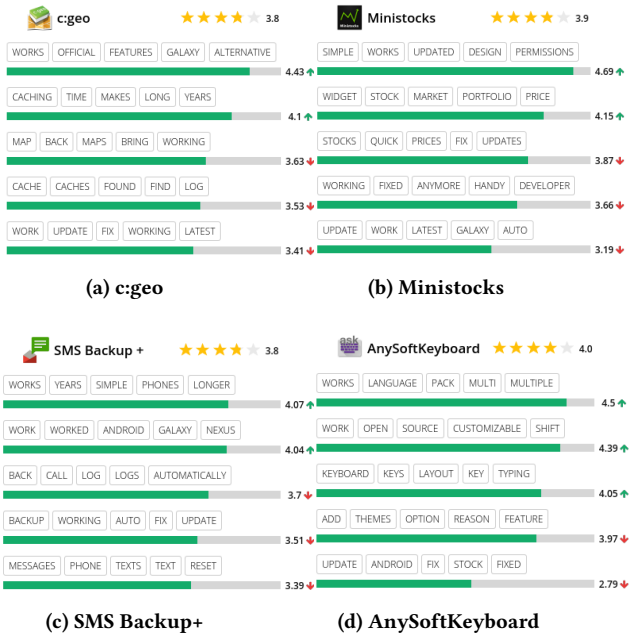
‡SA stands for sentiment analysis.

As shown in Figure 3, one topic that often negatively influenced user level satisfaction was related to updates. Additionally, since the topic *fix* often appears along with *update*, we surmise that frequent updates might actually lead to more problems than they fix.

Considering the results of our sentiment analysis, the apps in the control group also scored quite high for user satisfaction. The app with the smallest score was `Night Screen` (3.7) and the one with the highest score was `OpenSudoku` (4.5). On average (mean), the apps in this group scored slightly higher than the apps in the treatment group: 4.09. The results of the sentiment analysis would seem to suggest that energy inefficiencies are the main reason behind the negative reviews of the apps in this group. As highlighted in Figure 4, battery-drain issues are common in the control group: the topics *battery*, *life*, *drain*, *drains*, and *consuming* appear in many negative reviews. It is worth mentioning that the results of our review-based sentiment analysis suggest that apps with tests are not completely devoid of battery-drain issues. More precisely, our results indicate that apps with tests are less prone to deplete batteries faster than usual.

6.4 Bug Reports and Updates

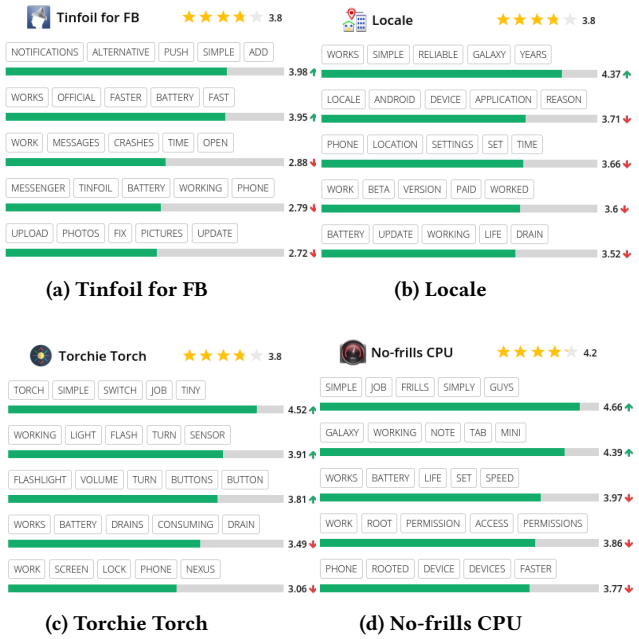
Most apps are updated very frequently in order to create a better experience for users and fix faults. Therefore, we conjectured that users often leave bug-related (i.e., fault-related) and update-related feedback in their written reviews. To take this into account, we decided to count the number of one-star and two-star reviews with keywords related to bug reports. Previous work has shown that one-star and two-star reviews often highlight negative issues [19]. The bug-related keywords we used in our experiment were extracted from Maalej and Nabil [17].



(a) c:geo (b) Ministocks

(c) SMS Backup+ (d) AnySoftKeyboard

Figure 3: Topics extract from user reviews. The results of our sentiment analysis method would seem to indicate that apps with tests have update-related issues.



(a) Tinfoil for FB (b) Locale

(c) Torchie Torch (d) No-frills CPU

Figure 4: Topics extracted from user reviews. According to the results of our sentiment analysis method, some apps without test suites are fraught with battery-drain issues.

Figure 5a compares the control and treatment groups with respect to the number of one-two star reviews with bug related keywords; given that we examined apps with varying number of reviews, each data point is a proportion (in %) to the total number of reviews. As shown in Figure 5a, there is no major difference between the two groups: the interquartile ranges are quite similar and the treatment group (i.e., apps with tests) has the greatest percentage of reviews with bug-related information.

Since regression testing can be used to expose faults caused by updates and/or changes in apps, we checked whether the treatment group has fewer reviews whose main issue brought up by end users is update related. To look for reviews related to update issues, we adopted the following terms: `updat`, `upgrad`, and `chang`. Figure 5b compares the control and treatment groups with respect to the number of reviews with update related keywords; each data point is a proportion (in %) to the total number of reviews. Considering the median, the treatment group is slightly better. That is, interestingly, apps with test suites have less reviews with update-related terms – though there exists a reasonable intersection between the interquartile ranges. However, as mentioned, update-related problems are the main source of negative reviews considering apps with test suites: although update-related keywords do not appear often in the reviews for apps with test suites, when they appear they are usually accompanied by keywords that carry negative sentiments.

7 DISCUSSION AND COMPARISON WITH RELATED WORK

Since our analysis was not able to evince any meaningful impact of having automated test suites, in this section we lay out some of the issues that we believe need to be addressed in future research.

Concerning the amount of tests. In our study the treatment group was composed of apps with automated tests and the adopted test-to-code-ratio was 1:10. Since the presence of automated tests in mobile apps is lacking[8], we initially assumed that it was a reasonable threshold. However, this might not be the case, as observed in Figures 6a and 6b. We relate the ratio test/production with bug related reviews (Figure 6a) and with update related reviews (Figure 6b). It is worth noting that most of our apps have a ratio below 0.3. Thus, one can argue that this amount of automated tests is not able to bring about the benefits of a sound and systematic practice of software testing.

User reviews. The amount of reviews most apps in our sample provides an indication that users are willing to spend time on writing review and share their experiences. Nevertheless, we found that a considerable number of reviews is not very informative: generally, the written reviews are brief, similar to tweets and thus do not provide much insight into the pros and cons of apps. As remarked by Pinto and Castor [23], the feedback left by customers/users in other domains (e.g., movies or hotels) tends to be 3-4 times longer and these customers seem to be willing to elaborate more on their written reviews. Consequently, we found that the user experience reported in reviews often lacks much in the way of providing informative feedback concerning bugs (i.e., faults) and similar issues.

[8]This is based on open source apps, though there exists evidence of such observation in industry [12].

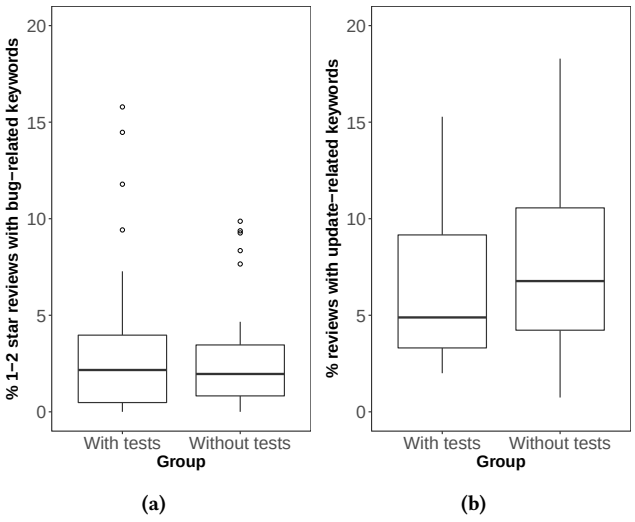


Figure 5: Boxplots outlining the (a) percentage of one-star and two-star reviews with bug related keywords and the (b) percentage of reviews with update related (i.e., `updat`, `upgrad`, and `chang`) keywords.

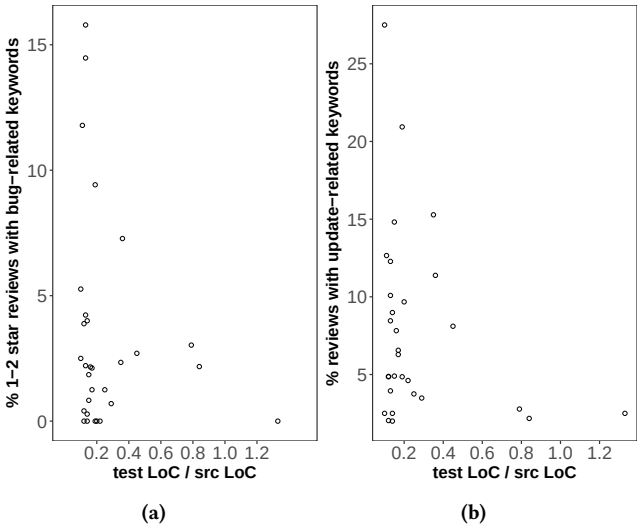


Figure 6: Dot plots showing the (a) test-to-code-ratio and percentage of bug related reviews. the (b) test-to-code-ratio and percentage of update related reviews.

Sample size. We considered 30 apps for each group. Although 30 is a common choice for experiments, we might need a slightly larger sample size to confirm our results and answer the RQs we posed. However, as mentioned, we believe that some insights gained from this study can still be generalized to similar settings.

User perception. Given that the apps we analyzed are adopted in very different contexts by a diverse group of people, the answers to the RQs we posed might not be evinced by observing direct feedback from users (i.e., star rating and reviews). Yet, similar studies (viz., [14]) have shown that there is a correlation between some of the proxies we used to evaluate quality and user ratings.

Energy efficiency is a main concern. The results of our study back up the conclusions made by Pinto and Castor [23]. Our review-based sentiment analysis showed that apps without tests are fraught with battery-drain issues, which leads to poor apps reviews. As pointed out by Pinto and Castor, the creation of techniques and tools (e.g., software energy profilers) to allow developers to create, maintain, and evolve energy-efficient software has received little attention. Instead, most research on computing and energy efficiency has been centered around low level hardware and software issues. Therefore, we also believe that energy consumption has become an ubiquitous problem largely due to the lack of specialized tools and knowledge.

8 CONCLUDING REMARKS

Software testing has proven to be central to quality improvement efforts. Moreover, since user feedback has become crucial for modern software development, in this paper, we study the extent to which software testing contributes to the quality of apps in terms of user satisfaction. To this end, we analyzed a total of 60 mobile apps taking into account two proxy measures of user satisfaction: users' ratings and users' reviews. While there is an increasing number of studies based on analyzing/mining app reviews and quantitatively analyzing the reasons behind app users' dissatisfaction, there is little empirical evidence on the benefits of software testing in terms of end user satisfaction and the success of mobile apps.

We found that there is no significant difference between apps with test suites and apps that have been developed without test suites in terms of their user rating. The key value to this research is that we looked further into user feedback in the form of written user reviews in hopes of characterizing the main problems of apps with and without test suites. The results of our review-based sentiment analysis indicate that the vast majority of the apps with and without test suites score quite high for user satisfaction. We also found that apps with test have a high incidence of update-related problems, while apps without test suites are prone to deplete a device's battery faster than usual. We believe that our study can be seen as a step towards characterizing the different problems that occur in apps that have automated tests and apps developed without test suites.

There are several interesting avenues for future exploration. First, as we pointed out in our threats to validity section, all apps we analyzed are from the Google Play Store. Thus, as future work, it would be interesting to analyze apps from different app marketplaces and increase the size of our sample. Second, although several studies have taken into account app store reviews to analyze different aspects of app development, app users' feedback is also widely available on social media. We believe that in a future study it would be important to look at how users' feedback in social media compares to users' feedback present in app stores. Third, another important point is that we should look at more sophisticated ways to identify bug- (i.e., fault-related) and update-related issues in reviews. Finally, we plan to look into how developers have been testing different aspects of mobile apps (e.g., usability and functionality). More specifically, we plan to go over information available in bug trackers, use more sophisticated methods to examine the types of automated tests included in our sample, and carry out surveys with app committers/developers.

REFERENCES

[1] P. V. Bicalho, T. Cunha, F. Mourão, G. L. Pappa, and Wagner M. Jr. 2014. Generating Cohesive Semantic Topics from Latent Factors. In *Brazilian Conference on Intelligent Systems (BRACIS)*. 271–276.

[2] M. Butler. 2011. Android: Changing the Mobile Landscape. *IEEE Pervasive Computing* 10, 1 (2011), 4–7.

[3] Laura V. Carreño and Kristina Winbladh. 2013. Analysis of User Comments: An Approach for Software Requirements Evolution. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. IEEE, 582–591.

[4] N. Chen, J. Lin, S.C. Hoi, X. Xiao, and B. Zhang. 2014. AR-miner: Mining Informative Reviews for Developers from Mobile App Marketplace. In *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 767–778.

[5] Z. Chen and B. Liu. 2014. Topic Modeling Using Topics from Many Domains, Lifelong Learning and Big Data. In *Proceedings of the International Conference on International Conference on Machine Learning*. II–703–II–711.

[6] G.H. Golub and C. Reinsch. 1970. Singular Value Decomposition and Least Squares Solutions. *Numerische mathematik* 14, 5 (1970), 403–420.

[7] G. Grano, A. Ciurumelea, S. Panichella, F. Palomba, and H. C. Gall. 2018. Exploring the Integration of User Feedback in Automated Testing of Android Applications. In *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 72–83.

[8] E. Guzman and W. Maalej. 2014. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In *IEEE 22nd International Requirements Engineering Conference (RE)*. 153–162.

[9] C. J. Hutto and E. Gilbert. 2014. VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text.. In *International Conference on Weblogs and Social Media (ICWSM)*. The AAAI Press.

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

[11] S. Ickin, K. Petersen, and J. Gonzalez-Huerta. 2017. Why Do Users Install and Delete Apps? A Survey Study. In *Software Business*. Springer, 186–191.

[12] M.E. Joorabchi, A. Mesbah, and P. Kruchten. 2013. Real Challenges in Mobile App Development. In *The ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 15–24.

[13] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D.M. German, and D. Damian. 2014. The Promises and Perils of Mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositoriesa (MSR)*. ACM, 92–101.

[14] H. Khalid, M. Nagappan, and A. E. Hassan. 2016. Examining the Relationship between FindBugs Warnings and App Ratings. *IEEE Software* 33, 4 (2016), 34–39.

[15] Daniel D. Lee and H. Sebastian Seung. 1999. Learning the Parts of Objects by Non-negative Matrix Factorization. *Nature* 401, 6755 (1999), 788–791.

[16] W. Luiz, F. Viegas, R. Alencar, F. Mour ao, T. Salles, D. Carvalho, M. Gonçalves, and L. Rocha. 2018. A Feature-Oriented Sentiment Rating for Mobile App Reviews. In *Proceedings of the 2018 World Wide Web Conference*. 1909–1918.

[17] W. Maalej and H. Nabil. 2015. Bug Report, Feature Request, or Simply Praise? On Automatically Classifying App Reviews. In *IEEE International Requirements Engineering Conference*. 116–125.

[18] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman. 2017. A Survey of App Store Analysis for Software Engineering. *IEEE Transactions on Software Engineering* 43, 9 (2017), 817–847.

[19] S. McIlroy, N. Ali, H. Khalid, and A. E. Hassan. 2016. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Software Engineering* 21, 3 (2016), 1067–1106.

[20] K. Moran, M. L. Vásquez, and D. Poshyvanyk. 2017. Automated GUI Testing of Android Apps: From Research to Practice. In *IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 505–506.

[21] M. Nayebi, H. Cho, and G. Ruhe. 2018. App Store Mining is not Enough for App Improvement. *Empirical Software Engineering* (2018).

[22] D. Pagano and W. Maalej. 2013. User Feedback in the Appstore: An Empirical Study. In *IEEE International Requirements Engineering Conference*. 125–134.

[23] G. Pinto and F. Castor. 2017. Energy Efficiency: A New Concern for Application Software Developers. *Communications of the ACM* 60, 12 (2017), 68–75.

[24] P. Rodrigues, S. Silva, G. Barbosa, F. Coutinho, and F. Mourao. 2017. Beyond the Stars: Towards a Novel to Evaluate Applications in Web Stores of Mobile Apps. In *Proceedings of the 2017 World Wide Web Conference*. 109–117.

[25] D.B. Silva, M.M. Eler, V.H.S. Durelli, and A.T. Endo. 2018. Characterizing Mobile Apps from a Source and Test Code Viewpoint. *Information and Software Technology* 101 (2018), 32–50.

[26] S.W. VanderStoep and D.D. Johnson. 2008. *Research Methods for Everyday Life: Blending Qualitative and Quantitative Approaches.* Jossey-Bass. 352 pages.

[27] R. R. Wilcox. 2016. *Introduction to Robust Estimation and Hypothesis Testing (Statistical Modeling and Decision Science)* (4th ed.). Academic Press. 810 pages.

[28] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering.* Springer. 236 pages.

[29] Samer Zein, Norsaremah Salleh, and John Grundy. 2016. A Systematic Mapping study of Mobile Application Testing Techniques. *Journal of Systems and Software* 117 (2016), 334–356.