Noname manuscript No.
(will be inserted by the editor)

Towards Prioritizing User-Related Issue Reports of Mobile Applications

Ehsan Noei · Feng Zhang ·
Shaohua Wang · Ying Zou

Received: date / Accepted: date

Abstract The competitive market of mobile applications (apps) has driven app developers to pay more attention to addressing the issues of mobile apps. Prior studies have shown that addressing the issues that are reported in user-reviews shares a statistically significant relationship with star-ratings. However, despite the prevalence and importance of user-reviews and issue reports prioritization, no prior research has analyzed the relationship between issue reports prioritization and star-ratings. In this paper, we integrate user-reviews into the process of issue reports prioritization. We propose an approach to map issue reports that are recorded in issue tracking systems to user-reviews. Through an empirical study of 326 open-source Android apps, our approach achieves a precision of 79% in matching user-reviews with issue reports. Moreover, we observe that prioritizing the issue reports that are related to user-reviews shares a significant positive relationship with star-ratings. Furthermore, we use the top apps, in terms of star-ratings, to train a model for prioritizing issue reports. It is a good practice to learn from the top apps as there is no well-established approach for prioritizing issue reports. The results show that mobile apps with a similar prioritization approach to our trained model achieve higher star-ratings.

Ehsan Noei and Ying Zou
Department of Electrical and Computer Engineering, Queen's University
E-mail: (e.noei, ying.zou)@queensu.ca

Feng Zhang
School of Computing, Queen's University
E-mail: feng@cs.queensu.ca

Shaohua Wang
Department of Informatics, New Jersey Institute of Technology
E-mail: davidsw@njit.edu

1 Introduction

The revenue of Android applications (apps) has increased enormously in the past few years (Statista (2017b); Stats (2016)). App markets, such as Google Play Store (Google (2017); Statista (2017a)), are very competitive for app developers. Google Play Store provides a scoring system where users can rate apps from one star (the lowest star-rating) to five stars (the highest star-rating) and post their comments (i.e., user-reviews). Star-ratings of apps can affect the number of downloads and the income of app development companies (Bavota et al (2015); Kim et al (2011)). The associated user-reviews to star-ratings can contain valuable information, such as bug reports and feature requests (Iacob and Harrison (2013); Panichella et al (2015)). Such information can be useful for app developers to manage issues and demands of users to achieve higher star-ratings.

Traditionally, issues are managed and prioritized through issue tracking systems. Many mobile apps use GitHub (GitHub (2018a)) as an issue tracking system to manage the issues that are reported by developers. It has been found that addressing the issues that are reported in user-reviews can increase the star-ratings (Palomba et al (2015)). However, there is no precise link between the issue reports in issue tracking systems and the user-reviews on Google Play Store. We propose a solution to establish a connection between issue reports and user-reviews. The benefits of having a connection between user-reviews and issue reports are twofold. First, developers can focus on the issue reports that can increase the star-ratings of their apps. However, it is still hard for app developers to decide which user-related issue report should be addressed first. For instance, an issue that is reported by an expert developer may receive high priority (Xuan et al (2012)), as well as an issue appearing in many user-reviews. A resolution of various aspects is beneficial for prioritizing issue reports. Second, app developers would identify the issues from the user-reviews that have already been reported in the issue tracking system. Hence, developers can avoid issue report duplications (Cavalcanti et al (2013)) if they plan to add the issues that are reported in the user-reviews to the issue tracking system.

Prior studies, such as Villarroel *et al.* (Villarroel et al (2016)) and Chen *et al.* (Chen et al (2014)), only focus on user-reviews, but neglect the information provided in issue tracking systems. The issue reports in issue tracking systems include the results of developers' efforts in identifying potential issues and should not be neglected. Chen *et al.* (Chen et al (2014)) propose an approach to extract the informative user-reviews using textual features of the user-reviews, and rank the informative user-reviews. Villarroel *et al.* (Villarroel et al (2016)) enhance the release planning by classifying user-reviews into meaningful groups of bug reports and feature requests. Our work aims to prioritize issue reports by leveraging both user-reviews and issue reports. Therefore, we can prioritize the issue reports by integrating both the users' feedback reflected in user-reviews and developers' experience in handling issues.

In this paper, we collect all the Android apps (i.e., $1,310$ apps) that are available on F-Droid (FDroid (2017)). F-Droid is the largest repository for open-source Android apps. We study 326 of $1,310$ apps that have a non-trivial amount of user-reviews and issue reports (Khalid et al (2016)). We address the following research questions:

RQ1) How precisely can user-reviews be mapped to issue reports?

A user-review is an unstructured piece of text (Palomba et al (2015)) that is not longer than two lines on average. We cluster the user-reviews to enhance the precision of matching user-reviews with issue reports. Each cluster contains the user-reviews that are related to the same issue. To map each cluster to its related issue report, we compute the textual similarity between issue reports and clusters of user-reviews. The results show that our approach achieves a precision of 79%.

RQ2) Does prioritizing the user-related issue reports have a relationship with star-ratings?

To explain the prioritization order of issue reports, first, we compute 59 issue report metrics and 31 user-review metrics. Then, we use the metrics to model the issue reports prioritization of each app. Our models fits well (i.e., adjusted $R^2 \geq 0.5$ (Nelder and Baker (1972))) for 37% of the apps but fails to fit for 63% of the apps. We observe that the apps which share a significant relationship between star-ratings and our metrics tend to receive higher star-ratings. The results imply that prioritizing the issue reports with respect to our suggested metrics is beneficial for achieving higher star-ratings.

RQ3) How can app developers prioritize the user-related issue reports to achieve higher star-ratings?

It is beneficial to learn from the top-rated apps for prioritizing issue reports. We use the top apps to train a prediction model using the random forest technique (Liaw and Wiener (2002)). We apply the trained model to the remaining apps. For each app, we compare the similarity score between the predicted prioritization orders and the actual prioritization orders of issue reports. We obtain two groups of apps: (i) the apps with higher similarity scores of prioritization, and (ii) the apps with lower similarity scores. We observe that the first group of apps receive higher star-ratings than the second group. Hence, our suggested method can be a helpful solution for app developers to prioritize the issue reports.

Paper Organization. Section 2 explains our experiment setup. Section 3 describes the details of the research questions and findings. Section 4 discusses the potential threats to the validity of our work. Section 5 introduces the related work. Finally, we conclude the paper in Section 6.

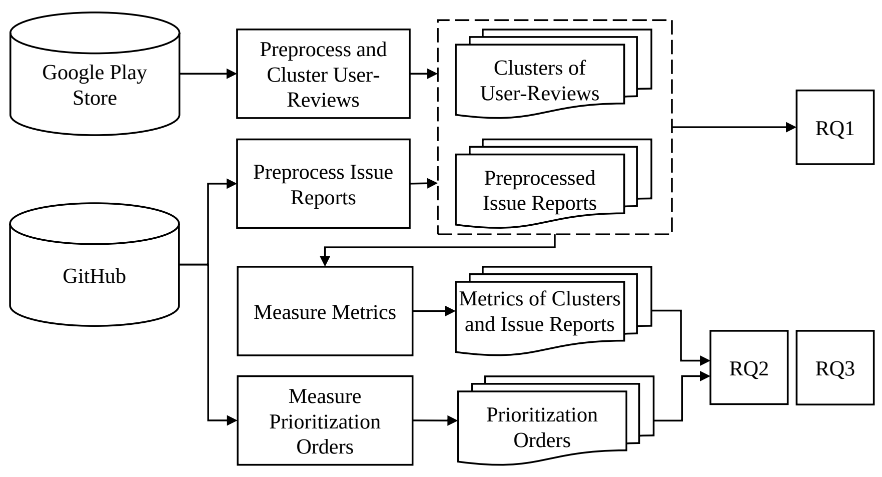

Google Play Store
Preprocess and Cluster User-Reviews
Clusters of User-Reviews
RQ1
GitHub
Preprocess Issue Reports
Preprocessed Issue Reports
Measure Metrics
Metrics of Clusters and Issue Reports
RQ2
RQ3
Measure Prioritization Orders
Prioritization Orders

Fig. 1: Overview of the experiment setup.

2 Experiment Setup

An overview of the experiment setup is depicted in Figure 1. As shown in Figure 1, our experiment setup mainly consists of the following steps: (i) preprocessing user-reviews and issue reports, (ii) clustering user-reviews, (iii) computing metrics of both user-reviews and issue reports, and (iv) measuring prioritization orders of issue reports.

We apply the vector space model (Salton et al (1975)) adopting TF-IDF (Salton and Michael (1983)) to measure the cosine similarity between issue reports and each cluster of user-reviews. We use metrics of clusters of user-reviews and issue reports to model the issue reports prioritization. Finally, we build a prediction model based on top apps to predict the issue reports prioritization of mobile apps. We observe that the apps that follow a similar prioritization order as our prediction model, receive better star-ratings.

2.1 Data Sources

We retrieved a set of open-source apps associated with their GitHub repositories from F-Droid app market (FDroid (2017)). F-Droid is an app store for open-source Android apps that provides access to source code and binary files (FDroid (2017)).

2.1.1 Apps

1,310 open-source Android apps were hosted on F-Droid app market (FDroid (2017)) as of *September* 1,2016. Not all the 1,310 apps were associated with

GitHub repositories. We obtained 1, 120 apps (i.e., 85% of the total apps) that were associated with their GitHub repositories.

We build a distinct model for each individual app (see Section 3). To avoid our findings being skewed by the apps with few numbers of user-reviews and issue reports, we filtered out the apps that have less than 10 informative user-reviews (see Section 2.2.1) and less than 10 issue reports (Khalid et al (2016)). Moreover, the number of Events Per Variable (EPV) is a metric that calculates the ratio of data points to the number of variables (Tantithamthavorn et al (2017)). To avoid the risk of over-fitting and having unstable results, having an EPV ≥ 10 is recommended (Tantithamthavorn et al (2017)). With less than 10 user-reviews or less than 10 issue reports, achieving an EPV ≥ 10 is not possible. Therefore, the apps that have received less than 10 user-reviews or issue reports should be excluded from our study. We identified 326 Android apps that meet the aforementioned criteria.

2.1.2 User-Reviews

A user-review that is posted by an individual user contains a text, star-rating, and date in which the review is posted. Figure 2 shows an overview of the process of retrieving user-reviews. We gradually retrieved the user-reviews by building a crawler on top of Selenium automation tool (Selenium (2017)). In the following paragraphs, we describe Selenium tool, the crawler, and gradual retrieving method.

Selenium. Selenium provides a set of tools and APIs to automate web browsing. The chief purpose of Selenium is web testing. However, it can be used for other purposes, such as web crawling. The primary parts of Selenium are (i) an IDE, (ii) a client API, and (iii) a web driver. The Selenium IDE is implemented as a Firefox add-on that allows recording, editing, and debugging web tests (Bruns et al (2009)). The client API lets developer communicate with Selenium. Finally, the web driver sends the Selenium commands to the browser (Selenium (2017)).

Crawler. To retrieve the user-reviews, we built a crawler using Selenium (Selenium (2017)). The crawler extracts all of the app information, such as app names, and the associated user-reviews.

Gradual Retrieving. Google Play Store limits the total number of user-reviews that a user can view to 2, 400 user-reviews (Khalid et al (2014)). Therefore, one cannot access all the available user-reviews of an app at once if it comes with more than 2, 400 user-reviews (Khalid et al (2014); Google (2017)). Therefore, we run the crawler on a daily basis for five years to get the latest user-reviews of each app. Then, we merge the new user-reviews with the existing user-reviews in our database. Hence, we could capture all the user-reviews for all the subject apps. The process of getting the new user-reviews takes about one hour a day.

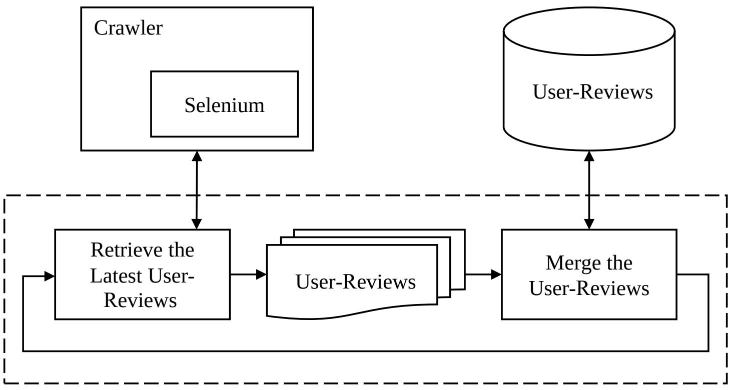


Fig. 2: Overview of the process of retrieving user-reviews.

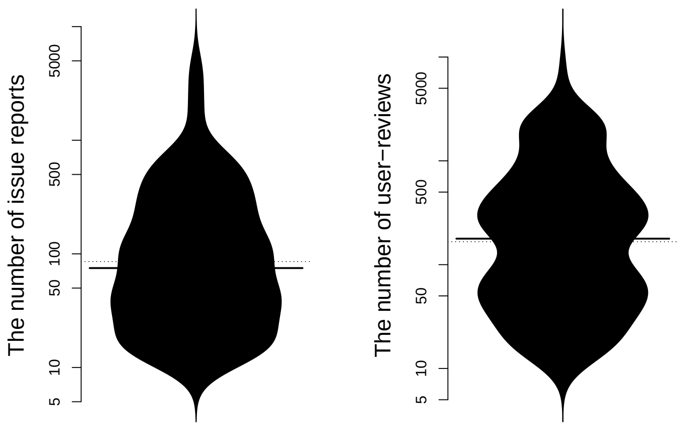
2.1.3 Issue Reports

An issue report that is posted by a user on GitHub includes a title, text, and date on which the issue is posted. We retrieved all the available issues of our subject apps using GitHub application programming interface (API) (Developer (2018)). As an alternative approach, GHTorrent (Gousios (2013)) could also have been used to measure the required metrics. As we only need the issue reports of a limited number of projects (i.e., 1,120 apps), using GHTorrent is not necessary. Figure 3a shows the number of issue reports for each of the 326 subject apps. As shown in Figure 3a, the number of issue reports varies for each app.

2.2 Preprocessing Data

A user-review is an informal piece of text (Google (2017); Palomba et al (2015)) that can potentially suffer from grammatical issues and typos. For example, a user-review, such as *"Tha pics couldnt be sentttt"*, has several typos. *"Tha"* and *"sentttt"* need to be changed to *"The"* and *"sent"*, respectively. Moreover, a user-review is usually short with few words. Furthermore, there are no consistent choices of words to describe the same issues. For instance, different users may use either the term *error* or the term *problem* to report a bug. In addition, user-reviews contain negations that confuse automatic approaches. Without considering negations, a user-review such as *"Great app! Runs with no problem!"* could have been interpreted as a user-review that reports a problem.

In the following paragraphs, we describe the taken steps for addressing the challenges that are mentioned above. In addition, we asked three non-authors to evaluate the mappings between the user-reviews and issue reports. The evaluators are graduate students in computer science and software engineering. We randomly select 384 user-reviews with the associated issue reports with a



(a) Issue Reports (b) User-Reviews

Fig. 3: The number of issue reports (from GitHub) and user-reviews (from Google Play Store) for all the 326 apps.

confidence level of 95% and the confidence interval of 5%. Each evaluator independently evaluated the mapping between the user-reviews in the sample and the issue reports. We apply the major vote rule to solve the conflicts among the evaluators. In each step, we use the above set of user-reviews as a reference to measure the improvement in the mapping precision.

2.2.1 Filtering Out Uninformative User-Reviews

In total, we collected 170, 373 user-reviews. An uninformative user-review, such as *"This app is OK"*, has no valuable information for app developers. The most recent studies (Chen et al (2014); Villarroel et al (2016)) propose different approaches to filter out uninformative user-reviews. For example, Chen *et al.* (Chen et al (2014)) employ the expectation maximization for Naïve Bayes classifier (Calders and Verwer (2010)) to identify uninformative user-reviews. We cluster the related user-reviews together. Therefore, a group of uninformative user-reviews can potentially become informative when they are grouped together. Having groups of related user-reviews (informative or uninformative) allows us to calculate the required metrics more accurately (see Section 2.4). Consequently, we only filter out the user-reviews that only praise or condemn an app. To this end, we use linguistic rules (Iacob and Harrison (2013)). The main author defined the linguistic rules by manually investigating 5, 000 randomly selected user-reviews. Although more is always better, we chose 5, 000 user-reviews because (i) a set of 5, 000 user-reviews is a representative sample of user-reviews and (ii) manually analyzing the user-reviews is a time-consuming task. On average, each user-review takes 15 seconds to analyze. Therefore, it takes about 21 hours to analyze all the user-reviews. Among

Table 1: The linguistic rules for filtering out uninformative user-reviews.

#	Rule
1	**<pronoun>? <App|Application>? <verb> <just,really,very,not>* <adjective>? <adverb>?**
	Note. In this rule, $verb \in \{work, is, run\} \cup \{describing\ verbs\}$, including all the variants of a verb. For example for *work*, we considered *works*, *does not work*, *is working*, *has worked*, *has been working*, and *has not worked.* Describing verbs are the verbs that demonstrate users' feelings, such as *rocks* and *stinks.*
2	**<just,not,article,really,very>* adjective <App|Application>?**
	Note. Articles include *a*, *an*, and *the.*
3	**<Appreciation>**
	Note. The *appreciation verbs* are *thanks*, *thank you*, *thanks a lot*, *thanks so much*, *thank you so much*, and *thank you very much*

5,000 user-reviews, we identified 3,789 user-reviews as informative and 1,211 user-reviews as uninformative ones according to the rules that are listed in Table 1. As an example, for the first rule, we match *'this app works fine'* and *'is very awful'*. For the second rule, we match *'not a good app'* and *'terrible'*. For the last rule, we match phrases like *'thank you!!!'*.

Moreover, we put aside non-English user-reviews from the collected user-reviews using Language Detector (Optimaize (2017)). The Language Detector creates a distinct profile for different languages. Then, it uses each profile to identify the language of a given text (Optimaize (2017)). We end up with 130,712 user-reviews. Figure 3b shows the number of user-reviews for each subject app.

2.2.2 Correcting Typos

Typos usually impact the results of text analysis techniques (Nord (2005)). We use Jazzy Spell Checker (Jazzy (2017)) with a dictionary of 645,289 English words to fix the typos of user-reviews and issue reports. Jazzy provides a set of Java APIs that allows us to detect misspelled words and replace them with the correct ones. Based on manually investigating 384 user-reviews with a confidence level of 95%, the Jazzy corrects 68% of the incorrect words. Correcting typos allows us to increase the mapping precision by 4%.

2.2.3 Resolving Synonyms

General-purpose thesaurus, such as *WordNet* (Miller (1995)), are not sufficient to resolve the synonyms of an informal text, such as a user-review (Noei et al (2018); Villarroel et al (2016)). Therefore, we build our own dictionary of words to resolve the synonyms. To ease the processing of building the dictionary, we applied LDA topic modeling technique (Blei et al (2003); Noei and Heydarnoori (2016)) on our data. We manually investigate each group of words that appear in the same topic and group the words that have similar meaning together accordingly. From each set of similar words, we pick one as the representative word and replace the other words with the representative word of each group. For example, *bug*, *error*, and *glitch* belong to the same group of terms.

We also replaced abbreviations and informal messaging vocabularies with formal words. We find the abbreviations and informal messaging vocabularies from the available online sources (Allacronyms (2017); Netlingo (2017)). For example, *"luv"* should be replaced with *"love"*. In our experiment, resolving synonyms let us increase the mapping precision by 3%.

2.2.4 Resolving Negations

The negations in the user-reviews can mislead the text processing techniques in getting the real meaning of user-reviews. To avoid such confusions, we use the Stanford natural language processing toolkit (Manning et al (2014)) to find and resolve the negated terms (Villarroel et al (2016)). In our experiment, resolving negations increases the precision of our mapping by 3%.

2.2.5 Removing Stop-Words

Stop-words are the most common words that exist in a language, such as *"is"* and *"the"*. We remove stop-words using Stanford CoreNLP (Manning et al (2014)). Removing stop-words allows text processing techniques to focus on the main words of user-reviews and issue reports (Rajaraman et al (2012)).

2.2.6 Stemming

Reducing inflected words to their word stem is called stemming (Lovins (1968)). By stemming user-reviews and issue reports, all forms of a word can be transformed to the same stem. For example, *"report"* and *"reporting"* have the same word stem that is *"report"*. We use the Snowball program (Snowball (2018)) to stem the words.

2.2.7 Extracting n-grams

Sometimes, words share a more concrete meaning when they come together. For example, a four-word phrase, such as *does_not_send_pictures*, shows a problem in sending pictures, while having these four words separated does not reflect its real meaning. A n-gram is a contiguous sequence of n words from a given sentence or sequence of words (Broder et al (1997)). For each user-review and issue report, we extract the n-grams with n varying from 2 to 4. Extracting the n-grams helps us to deal with the negations more effectively. Similar to Villarroel *et al.* (Villarroel et al (2016)), we extract the n-grams before the preprocessing steps to avoid losing any potential information. In our experiment, extracting the 2-grams, 3-grams, and 4-grams, increase the mapping precision by 3%, 1%, and 1% respectively. In total, extracting the n-grams ($n \in \{2, 3, 4\}$) increases the mapping precision by 5%.

2.3 Clustering User-Reviews

We cluster the related user-reviews by customizing the Villarroel *et al.* approach (Villarroel et al (2016)), such as adding a step for correcting typos. By clustering the user-reviews, even short and uninformative user-reviews can become helpful when they are considered together. Furthermore, clustering the user-reviews is required in this study for two main reasons: (i) having the related user-reviews clustered together significantly increases the mapping precision by 45%, and (ii) computing the metrics of user-reviews requires a group of related user-reviews, such as quantifying the number of user-reviews that report the same issue.

2.3.1 Approach

We apply DBSCAN (Ester et al (1996)) on user-reviews of each app. DBSCAN is a density-based clustering algorithm that groups the elements of user-reviews (i.e., words and n-grams) together that are closely placed near each other. We compute the distance between two user-reviews by applying the vector space model (Salton et al (1975)) cosine similarity between (i) the associated star-ratings (Villarroel et al (2016)), (ii) the post-processed user-reviews, and (iii) the lists of n-grams. We adopt the term frequency-inverse document frequency (TF-IDF) (Salton and Michael (1983)) on the vectors of user-reviews. TF-IDF allows us to measure the frequency of each term and estimate how much information each term provides. DBSCAN requires two parameters: (i) the maximum distance between the user-reviews, and (ii) the minimum number of user-reviews that can be clustered together. We set the maximum distance between two user-reviews to 0.6 as it gives the best performance of DBSCAN. We set the minimum number of points of DBSCAN to 1, as one user-review may be useful in identifying a potential issue.

2.3.2 Evaluation

We followed the same approach as Villarroel *et al.* (Villarroel et al (2016)) to evaluate the clustering approach. We randomly selected 384 user-reviews with a confidence level of 95% and the confidence interval of 5%. The external evaluators clustered the related user-reviews. Then, we compared the manually clustered user-reviews with the automatically clustered user-reviews. We achieved an accuracy of 80% in clustering the user-reviews. An example of an issue report that is matched with a cluster of user-reviews is shown at Table 2. As shown in Table 2, the reported issue is about an issue in the auto-correcting module where there exists some user-reviews reporting the same issue.

2.4 Computing Metrics of User-Reviews

We follow the *Goal / Question / Metric* (GQM) paradigm (Basili (1992); Van Solingen et al (2002)) to capture the metrics of user-reviews. The GQM

Table 2: A sample issue report matched with a cluster of user-reviews.

Issue Report	**User-Reviews**
Title: Autocorrect stopped working in comment reply field **Body:** If you reply to a comment in Notifications or the Reader, you won't get any autocorrect suggestions above the keyboard. From a quick poke around in the code, it appears to be related to using a subclass of AutoCompleteTextView	(i) Autocorrect not working (ii) Fix the autocorrect (iii) When replying in nofications, autocorrect crashes (iv) Would give 5 stars, but the recent autocorrect issues... 3 stars (v) Posting a comments makes autocorrect stop

is a measurement paradigm that is based on three levels: (i) conceptual, (ii) operational, and (iii) quantitative. The conceptual level, i.e., the goal, should be defined with respect to the purpose of a given model. The operational level is a set of questions to describe the goal that is defined at the conceptual level. The quantitative level is a set of metrics that can be measured to address each question of the operational level. Also, the availability of each metric has to be considered. For instance, we cannot capture the level of expertise of a user who posts a user-review, but we can capture the number of users that post similar user-reviews.

We set our goal to quantify the user-reviews. Table 3 shows our GQM model for capturing the user-reviews. As shown in Table 3, we measure 31 metrics of user-reviews, such as the number of similar user-reviews and the proportion of negative and positive user-reviews. For *star-ratings*, *sizes of user-reviews*, and *sentiment scores*, we compute the mean, median, minimum, maximum, 1^{st} quartile, and 3^{rd} quartile of each metric. The median is a metric to measure the central tendency of the data. However, it does not reflect the distribution of data below and above the median. To reduce such a limitation, we measure the 1^{st} and 3^{rd} quartile in addition to the median. For instance, if the 1^{st} quartile is far away from the median but the 3^{rd} quartile is close to the median, we can infer that the data points that are greater than the median are closely placed together in comparison with the data points that are less than the median (Kelley (1947)).

2.5 Computing Metrics of Issue Reports

Table 4 shows the GQM model to quantify the issue reports. We compute the mean, median, minimum, maximum, 1^{st} quartile, and 3^{rd} quartile of the following metrics: (i) the sizes of comments, (ii) contribution, (iii) number of following, (iv) number of followers, (v) number of gists of the people who have involved in an issue, and (vi) the time since they have joined GitHub.

Regarding the size of an issue report, it can reflect the amount of information contained in the issue report (Kim et al (2006)). Moreover, Yu *et al.* (Yu et al (2015)) indicate that the size of a given document can be associated with the quality and the complexity of a document. For example, consider the

Table 3: GQM model to capture the metrics of user-reviews, along with a brief description.

Goal: Quantifying the user-reviews for a given issue			
Question	**Metric(s)**	**Description**	**#**
How many users reported the same issue?	Number of similar user-reviews	The number of times that an issue is reported can affect its priority. Developers may consider resolving an issue in the next release if the majority of users report the same issue.	1
How did the users reporting the same issue rate an app?	Star-ratings	To maintain the level of star-ratings, developers are more likely to prioritize the issues that are reported with low star-ratings.	6
What is the proportion of high, low, and neutral star-ratings for the user-reviews reporting the same issue?	Proportion of low, neutral, and positive star-ratings	To capture the diversity of ratings, we measure the proportion of negative, positive, and neutral user-reviews within each cluster. We consider a user-review with a star-rating equal to 3, greater than 3, or less than 3 as a neutral, high, or low user-review, respectively (Noei et al (2017)).	3
How much effort do users put to describe an issue and how much information is provided?	Sizes of user-reviews	The size of a user-review can reflect the helpfulness and the importance of the user-review (Kim et al (2006)). We use Stanford parser (De Marneffe et al (2006)) to count the number of words and sentences in the user-reviews.	12
How was the users' experience with a given issue?	Sentiment scores	The star-ratings do not always reflect the real sentiments of user-reviews. To capture the sentiment scores of user-reviews, we apply sentimental analysis on the user-reviews using the SentiStrength-SE tool (Islam and Zibran (2017)).	6
What is the proportion of user-reviews with positive, negative, and neutral sentiment scores?	Proportion of negative, neutral, and positive sentiment scores	We measure the proportion of negative, positive, and neutral user-reviews in each cluster of user-reviews.	3
			Total: 31

example issue reports that are listed in Table 5. The first issue report with a bigger size provides comprehensive details of the reported issue, including the expected behavior, the actual behavior, and the steps to reproduce the issue. However, the second issue with a smaller size report does not provide enough context to understand and resolve the issue.

Table 4: GQM model to capture the metrics of issue reports, along with a brief description.

Goal: Quantifying the issue reports			
Question	**Metric(s)**	**Description**	**#**
How many users contributed to resolving an issue?	Number of users	The number of GitHub users who have involved in the discussions can implicitly show the importance of an issue.	1
How many interactions have been happened for resolving an issue?	Number of comments	The number of comments can reflect the complexity of resolving an issue.	1
How well an issue is described?	Sizes of issue reports	To capture the size of an issue report, we count (i) the number of words of the title, and (ii) the number of words and sentences of the body.	3
How well the comments of an issue are described?	Sizes of comments	Similar to the size of an issue report, we measure the number of words and the number of sentences of the comments posted on the issue report.	12
What is the contribution of an issue reporter?	Reporter contribution	If an issue report is reported by a user with a high contribution, the issue report may be prioritized with a higher rank. For each member, GitHub (GitHub (2018a)) computes a contribution score based on the activities of the user.	1
What is the contribution of the users who have involved in resolving an issue?	Contribution of users who have involved in issues	Similar to the contribution of the reporter, we measure the contribution of the users who have involved in each issue.	6
For how long a reporter is a member of GitHub?	Time since reporter has joined GitHub	A more experienced user may be more active on GitHub. We compute the time since each reporter has joined GitHub as of *September* 1, 2016.	1
For how long the users who have involved in an issue are members of GitHub?	Time since contributors have joined GitHub	We measure the time since each distinct user who has contributed to an issue report has joined GitHub.	6
How popular is the reporter?	Number of following and followers of reporters	The number of followers and followings of a user can estimate the popularity of the user (Romero et al (2011); Bertram et al (2010)). A user with many followers could be very popular. This can result in addressing the issues that are reported by such a user earlier than other issues.	2

(continues on next page)

Table 4: GQM model to capture the metrics of issue reports (continued).

Question	**Metric(s)**	**Description**	**#**
How popular are the users who have involved in an issue?	Number of following and followers of contributors	We measure the number of followers and the number of followings of the users who have contributed to each issue report.	12
How many code snippet a reporter has shared?	Number of gists of reporter	Gist is a GitHub service that allows users to share code snippets with others (GitHub (2018a)). The number of gists can show how much a developer intends to help the development community that could be associated with the developers' activity.	1
How many code snippet the contributors have shared?	Number of gists of contributors	We measure the number of gists of the users who have contributed to an issue report.	6
What is the number of repositories of the reporter?	Number of public repositories of reporters	A user can contribute to different public repositories on GitHub. The number of repositories on which a user works can capture the level of expertise and engagement of the user in different projects.	1
What is the number of repositories of contributors?	Number of public repositories of contributors	We count the number public repositories of the contributors to an issue report.	6
		Total:	59

2.6 Measuring the Prioritization Order of Issue Reports

Developers react to some issues very fast, while they might postpone responding to some other issues for many weeks. We consider the developers' reaction attitude as an indicator of the importance of the issues. To estimate developers' reaction to each issue, we consider the following actions: (i) post comments on an issue report, (ii) submit commits for an issue report, and (iii) adding specific keywords to an issue report, including *"Fixed"*, *"Solved"*, *"Resolved"*, *"Closed"*, *"Feature added"*, and *"Finished"*. To measure the reaction time for each issue, we compute the minimum value of the intervals between each of the aforementioned actions and the time since an issue report has been posted on GitHub. We use the reaction times to measure the prioritization orders of issue reports.

However, some noises may be introduced by considering posting comments as an indicator of prioritization order of the issues. For example, a developer may immediately post a comment on an open issue to mention that they will take care of it after dealing with more important issues. We manually investigate the comments of a sample issue reports (384 issue reports) with a confidence level of 95%, confidence interval of 5, and population of $239,736$. We observe that only 1.7% of the comments are irrelevant to the associated issue reports which is a tolerable proportion of comments.

Table 5: Two sample issue reports.

#	Issue Report
1	***Title:*** *Refresh a post view* ***Body: Expected behavior:*** *When you are reading a post (I mean: when you have tapped in one the items in your posts list in the reader and are reading the expanded view), I expect to be able to "pull to refresh" so I can update the comments & favs of the post.* ***Actual behavior:*** *Nothing happens. Pull to refresh is not available when you are reading a post. If you want to refresh the comments you need to go pack to your posts lists, refresh there and then tap back in the post.* ***Steps to reproduce the behavior:*** *Go to the reader. Tap on any post. Try to refresh it.* ***Date:*** *Apr 12, 2016* ***Status:*** *Closed*
2	***Title:*** *after creating custom ref the spinner 'from' does not get updated (shows only after second exec or refresh).* ***Body:*** *NULL* ***Date:*** *Apr 10, 2013* ***Status:*** *Open*

3 Research Questions and Results

In this section, for each research question, we present our motivation, approach, and findings.

RQ1) How precisely can user-reviews be mapped to issue reports?

Motivation

Many apps have hundreds or even thousands of user-reviews. It is not a trivial task for app developers to manually analyze all of the user-reviews, while it is beneficial to map user-reviews to issue reports automatically. Therefore, app developers would be able to use the user-reviews to prioritize the issue reports. Moreover, having the knowledge of the issues that are also mentioned in the user-reviews can help app developers to better manage the issues and prevent issue report duplications (Cavalcanti et al (2013)).

Approach

As described in Section 2.3, first, we cluster the related user-reviews. We consider all of the user-reviews that belong to the same cluster as a single document that describes the same issue. To determine the similarity between user-reviews and issue reports, we apply the vector space model (Salton et al (1975)). First, we compute TF-IDF (Salton and Michael (1983)) to obtain the vector of each document, i.e., either a cluster of user-reviews or an issue report. Second, we calculate cosine similarities between issue reports and each cluster of user-reviews. We associate a cluster of user-reviews with an issue report if their similarity is greater than the threshold τ. We evaluate our experiment with different thresholds from 0.05 to 0.95 on five randomly selected apps.

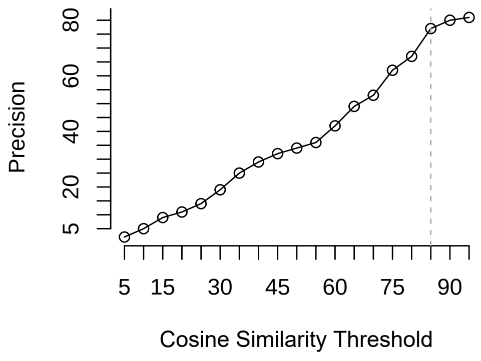
To measure the performance of our mapping approach, we adopt the precision. The precision is computed as the proportion of correctly matched pairs of clusters of user-reviews and issue reports among all the matched pairs. We asked three non-authors to manually examine the correctness of each matched pair on a statistically representative sample set of the rest of the apps, i.e., all apps excluding the five apps that we used to determine the best threshold. To obtain such a set, we randomly select 384 user-reviews with their associated issue reports from $30,520$ user-reviews with a confidence level of 95% and the confidence interval of 5%. The three evaluators independently evaluate the sample of user-reviews matched with the issue reports. We apply the major vote rule to resolve the conflicts amongst the evaluators.

Findings

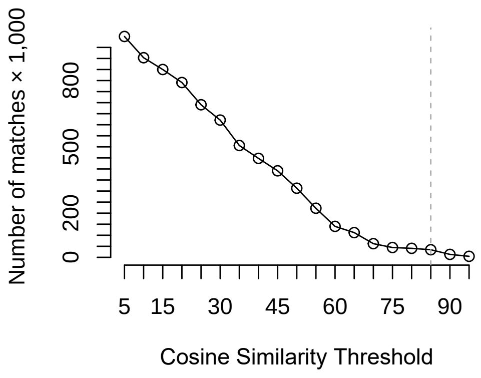
We achieve a precision of 79% in matching user-reviews with issue reports. Figure 4a shows the precision achieved by our approach with various thresholds and Figure 4b shows the number of matches between user-reviews and issue reports. We set the threshold to 0.85 with a trade-off between the precision of matches and the number of matches. Based on the manual analysis by the three evaluators, our approach achieves a precision of 79% with the threshold τ of 0.85.

We match 27% of the user-reviews with the 33% of the issue reports. The issue tracking systems are normally the working area of app developers (Janák (2009)), while user-reviews are from external users. The matches between the issue reports and the clusters of user-reviews show the issues that are reflected in both user-reviews and issue reports. In the next research questions, we show that prioritizing only the user-related issue reports shares a significant relationship with star-ratings. Therefore, our approach can help app developers to prioritize user-related issue reports to receive higher star-ratings. Having considered the user-reviews that are grouped together, developers can create new issue reports concerning the user-reviews that are left out. Therefore, with respect to our approach, developers can cover more of the user-reviews when maintaining their apps.

Using our mapping approach, user-reviews can be mapped to issue reports with a precision of 79%.



(a) The precisions of our approach.



(b) The number of matches between user-reviews and issue reports.

Fig. 4: The precisions of our approach and the number of matches between user-reviews and issue reports obtained using thresholds from 0.05 to 0.95.

RQ2) Does prioritizing the user-related issue reports have a relationship with star-ratings?

Motivation

Developers may take different priority orders when addressing the user-related issue reports. Although the lack of issue reports prioritization can negatively

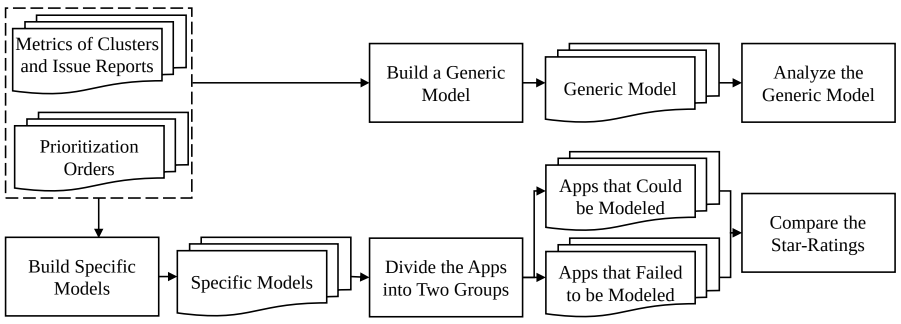

Metrics of Clusters and Issue Reports
Prioritization Orders
Build a Generic Model
Generic Model
Analyze the Generic Model
Apps that Could be Modeled
Compare the Star-Ratings
Build Specific Models
Specific Models
Divide the Apps into Two Groups
Apps that Failed to be Modeled

Fig. 5: Overview of our approach for addressing the second research question.

impact star-ratings, there is no empirical evidence to show the relation between prioritizing the user-related issue reports and star-ratings. Therefore, we investigate the relationship between star-rating and the metrics of user-reviews and issue reports.

Approach

We model the issue reports prioritization using linear regression models (Faraway (2005)). The dependent variable of the regression models is the prioritization orders of the apps. The independent variables are the metrics computed from both user-reviews and issue reports. The goodness of fitness, i.e., adjusted R^2 (Nelder and Baker (1972)), of the linear regression models shows whether issue reports prioritization has a relationship with the metrics of user-reviews and issue reports. Figure 5 shows an overall overview of our approach.

Before building the regression models, we identify the correlated variables. We apply variable clustering analysis (Hmisc (2017)) to build a hierarchical overview of the correlation between the independent metrics (Noei et al (2017)). The metrics within each sub-hierarchy of metrics with Spearman's $|\rho| > 0.7$ are considered as correlated variables (Nguyen et al (2010)). We choose one metric that is easier to comprehend for inclusion in our model from each sub-hierarchy of metrics. We build two types of regression models:

(i) *Generic Model.* We build a generic model using all of the subject apps. Building a generic regression model with a high goodness of fitness can show that different apps are following a similar strategy for prioritizing issue reports.
(ii) *Specific Models.* For each app, we build an independent regression model.

We get a goodness of fitness for each independent regression model. A higher goodness of fitness can indicate that the issue reports prioritization of an app has a significant relationship with the metrics of user-reviews and issue

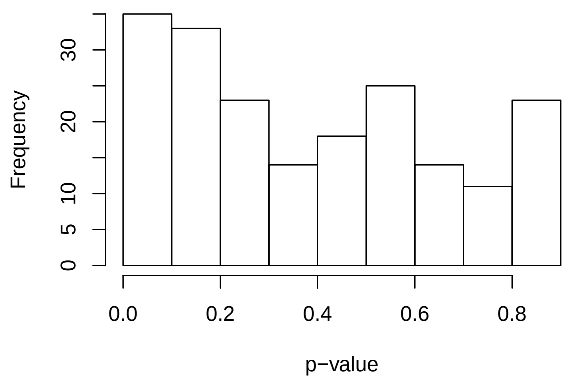


Fig. 6: Adjusted R^2s that are obtained from the regression models that are built for each app.

reports. The EPV measures the ratio of data points to the number of variables (Tantithamthavorn et al (2017)). An EPV of greater than 10 is recommended to have a low risk of over-fitting and unstable results (Tantithamthavorn et al (2017)). We did not consider 14% of the apps with EPVs less than 10. We divide the apps into two groups; one group with $R^2 \geq 0.5$ and another one with $R^2 < 0.5$ (Nelder and Baker (1972)). We compare the star-ratings of the two groups of apps using Mann–Whitney U test (Mann and Whitney (1947)). As a null hypothesis, we assume that the distributions of the star-ratings between the two groups of the apps are the same. The Mann–Whitney U test rejects this hypothesis with a $p - value$ of less than 0.05.

The Mann–Whitney U test can show a significant difference for a sufficiently large sample even if the difference is negligible. Therefore, we also measure the effect size of the differences between star-ratings by applying Cliff's δ (Cliff (1993)). Cliff's δ is a non-parametric measure without assumptions about the distribution of data (Cliff (1993)). Cliff's δ measures the degree of overlap between the two sets of star-ratings. The output of the Cliff's δ is a number between -1 and $+1$. If the distribution of star-ratings between the two sets of apps is identical, the Cliff's δ would be 0 (Cliff (1993)). If all the values of the first set are greater than the second set, it would be $+1$, and vice versa. We use Cohen's d (Cohen (2013)) to interpret the effect sizes (Zhang et al (2015)). Cliff's δ could be mapped to Cohen's standards; the values of 0.147, 0.330, and 0.474 denote small, medium, and large effect size, respectively (Zhang et al (2015)).

Findings

Developers of different apps do not follow the same strategy to prioritize the user-related issue reports. Our generic model using all of the subject apps has a very low goodness of fitness, i.e., $R^2 < 0.05$. Thus, there is no statistically significant universal relationship between the issue reports prioritization and metrics of user-reviews and issue reports for all different apps.

For 37% of the subject apps, issue reports prioritization can be modeled with the metrics of user-reviews and issue reports. The regression models of 37% of the subject apps achieve $R^2 s \geq 0.5$, i.e., the issue reports prioritizations share significant relationships with the metrics of user-reviews and issue reports. Figure 6 shows the obtained R^2s. Tables 6 and 7 show two sample models that are built for two distinct apps, i.e., *Indic Keyboard*[1] and *Vanilla Music*[2], sorted by $p-value$. In the last column in both tables, upward arrows indicate that when the values of the associated metrics increase, the prioritization rank is more likely to increase, while downward arrows indicate otherwise. For the remaining 63% of the apps, we could not build regression models with a high goodness of fitness.

Developers of different apps do not consider the same importance level for the metrics of issue reports and user-reviews. Among the apps having the user-related issue reports prioritized, the sets of significant metrics are different from each other. For example, some issue reports are prioritized according to the metrics that are defined in the scope of issue tracking systems, such as the contribution of a user who has reported the issue. Some other issue reports are prioritized by considering the user-reviews, such as the number of user-reviews. We count the frequencies of the metrics that share statistically significant relationships with the issue reports prioritization. Table 8 shows the top five metrics of the issue reports and the user-reviews that appear the most. In particular, the size of the title and body, the number of comments, and the contribution of the person who reported the issue are the metrics that appear the most as a significant metric. The star-ratings and the number of user-reviews are the two metrics of user-reviews that have the most relationship with the issue reports prioritization.

The title size is the most popular and the most important metric when it comes to issue reports prioritization. As shown in Table 8, for 59% of the subject apps, the title size appears as a statistically significant metric. The next important metric is the number of comments that are posted for an issue report. The issues that are associated with a higher prioritization order tend to receive more comments. Having considered the title size and the body size as two significant metrics, the contents of issue reports play an important role in issue reports prioritization. Another interesting observation is where the reporter contribution appears as a significant metric for 41% of the apps and

[1] https://play.google.com/store/apps/details?id=org.smc.inputmethod.indic
[2] https://play.google.com/store/apps/details?id=ch.blinkenlights.android.vanilla

Table 6: The model that is built for *Indic Keyboard* app, sorted by $p - value$.

Metric	**Pr(>|t|)**		**Effect**
Reporter contribution	0.001	***	↗
Minimum star-rating	0.015	*	↘
Proportion of high star-ratings	0.023	*	↘
Maximum sentiment score of user-reviews	0.051	.	↘
Number of reviews	0.053	.	↗
Maximum contribution of users who involve in issues	0.065	.	↗
Maximum number of sentences in user-reviews	0.099	.	↗
Minimum number of sentences in user-reviews	0.119		↘
Proportion of low star-ratings	0.132		↗
Number of gists of reporter	0.141		↘
Time since reporter has joined GitHub	0.172		↗
Minimum sentiment score of user-reviews	0.370		↗
Number of comments	0.452		↘
Proportion of positive sentiment scores	0.470		↘
Minimum contribution of people who involve in issues	0.631		↘
Number of followers of reporter	0.662		↗
Title size	0.776		↘
Minimum number of words in user-reviews	0.809		↘
Maximum star-rating	0.898		↗
Number of words in issue reports	0.974		↘

$p - value$ codes: '***'< 0, '**'< 0.001, '*'< 0.01, '.'< 0.05

Table 7: The model that is built for *Vanilla Music* app, sorted by $p - value$.

Metric	**Pr(>|t|)**		**Effect**
Title size	0.001	***	↘
Number of gists of reporter	0.007	**	↗
Proportion of negative star-ratings	0.017	*	↗
Number of followers of reporter	0.022	*	↗
Minimum number of words in user-reviews	0.034	*	↗
Reporter contribution	0.081	.	↗
Proportion of high star-ratings	0.096	.	↘
Number of words in issue reports	0.116		↘
Proportion of user-review with neutral sentiment scores	0.180		↘
Time since reporter has joined GitHub	0.210		↘
Minimum contribution of people who involve in issues	0.248		↗
Number of comments	0.252		↘
Minimum sentiment score of user-reviews	0.350		↘
Number of user-reviews	0.352		↗
Maximum contribution of people who involve in issues	0.369		↗
Minimum number of sentence in user-reviews	0.438		↘
Minimum star-rating	0.541		↘
Number of following of reporter	0.738		↗
Proportion of positive sentiment scores	0.956		↘

$p - value$ codes: '***'< 0, '**'< 0.001, '*'< 0.01, '.'< 0.05

Table 8: Ranking and percentages of the occurrence of the top five metrics of issue reports and user-reviews.

Context	Rank	Metric	Occurrence
Issue Reports (GitHub)	1	Title size	59%
	2	Number of comments	55%
	3	Body size	44%
	4	Reporter contribution	41%
	5	Time since reporter has joined GitHub	40%
User-reviews (Google Play Store)	1	Minimum star-rating	13%
	2	Number of user-reviews	12%
	3	Proportion of neutral star-ratings	10%
	4	Minimum sentiment score	8%
	5	Proportion of low star-ratings	8%

the time since the reporter has joined GitHub appear for 40% of the apps. This can show that an issues report that is reported by a developer with a higher reputation tend to be addressed at a faster pace.

Among the user-review metrics, the minimum star-rating, the proportion of neutral star-ratings, and the proportion of low star-ratings appear for 13%, 10%, and 8% of the subject apps as statistically significant metrics, respectively. This may be because developers would like to reduce the number of low star-ratings by addressing the user-reviews that are associated with lower star-ratings (Noei et al (2017)).

Addressing the issues in the user-reviews has a statistically significant relationship with star-ratings. The results in Figure 7 show that the apps that we could match their issues reports with the user-reviews receive higher star-ratings. The differences between the star-ratings of the apps that we could match their issues reports with the user-reviews (the first and the second boxplot in Figure 7) and the apps that we could not match their issues reports with the user-reviews (the third boxplot in Figure 7) are statistically significant with a $p-value$ of $5.37e-05$ and a medium effect size with $Cohen's\ d$ of 0.37.

Prioritizing the issue reports with respect to our metrics shares a statistically significant relationship with star-ratings. The first two boxplots in Figure 7 show the apps that we could match their user-reviews with their issue reports. The differences between the apps that (i) the issue reports prioritization is statistically significantly related to our metrics (the first boxplot in Figure 7) and (ii) the issue reports prioritization is not statistically significantly related to our metrics (the second boxplot in Figure 7) are statistically significant with a $p-value$ of $8.82e-06$ and a medium effects size with a $Cohen's\ d = 0.55$. Figure 7 shows that the star-ratings of the first group of apps are higher than the other apps.

The issue reports of different apps are prioritized differently. Higher star-ratings are recorded for apps for which a statistically significant relationship exists between our metrics and issue reports prioritization.

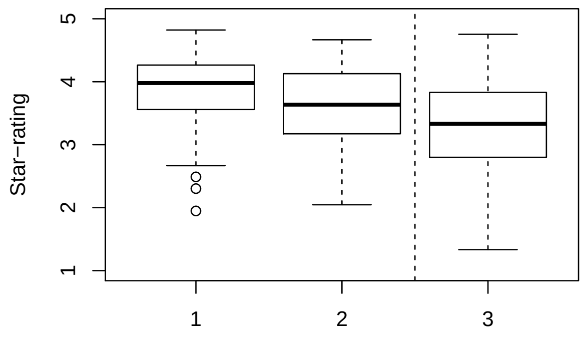


Fig. 7: The average of star-ratings of the apps that:

1) We could match their user-reviews with the issue reports and the issue reports prioritization is statistically significantly related to the metrics.
2) We could match their user-reviews with the issue reports but the issue reports prioritization does not share a statistically significant relationship with the metrics.
3) We could not match their user-reviews with the issue reports.

RQ3) How can app developers prioritize the user-related issue reports to achieve higher star-ratings?

Motivation

In **RQ2**, we observe that issue reports prioritization shares a significant relationship with star-ratings. To utilize the important findings of RQ2, we suggest a prioritization method for ranking issue reports in order to achieve higher star-ratings.

Approach

Figure 8 shows an overview of our approach. To better prioritize the issue reports, we define four levels of prioritization. Given the list of issue reports that are ranked based on the prioritization order of issue reports (see Section 2.6), we define the prioritization levels of the issue reports as follows:

(i) *High Priority*: The issue reports within the first quartile ($0-25\%$) of the prioritization orders are labeled as high priority.
(ii) *Medium Priority*: The issue reports within the second quartile ($25-50\%$) of the prioritization orders are labeled as medium priority.

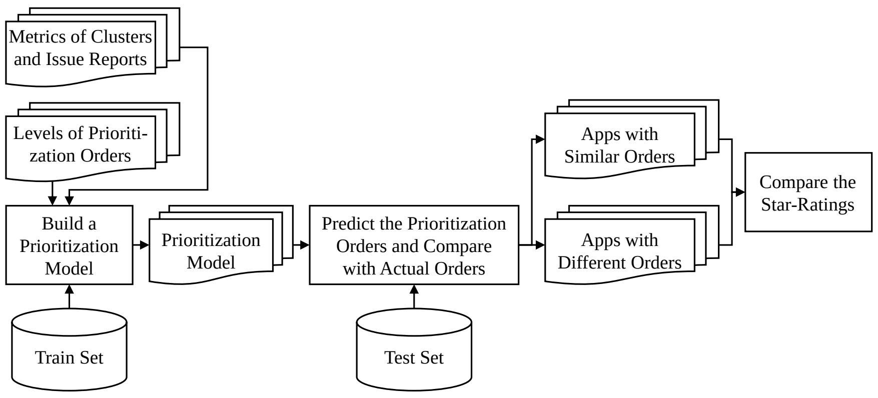

Metrics of Clusters and Issue Reports
Levels of Prioritization Orders
Build a Prioritization Model
Prioritization Model
Predict the Prioritization Orders and Compare with Actual Orders
Apps with Similar Orders
Apps with Different Orders
Compare the Star-Ratings
Train Set
Test Set

Fig. 8: Overview of our approach for addressing and evaluating the third research question.

(iii) *Low Priority*: The issue reports within the third quartile $(50 - 75\%)$ of the prioritization orders are labeled as low priority.
(iv) *Trivial Priority*: The issue reports within the last quartile $(75 - 100\%)$ of the prioritization orders are labeled as trivial priority.

Issue tracking systems usually define a limited number of prioritization orders for the issue reports (Xuan et al (2012)). For example, Bugzilla (Bugzilla (2018)) defines five orders of prioritization orders from trivial to high priority. We chose four levels of prioritization according to the distribution of the prioritization orders of our subject apps. The high priority issue reports are addressed within an hour. The medium priority issue reports are addressed within a day. The low priority issue reports are addressed within five days. The trivial priority issue reports are addressed after five days.

We build a random forest model (Liaw and Wiener (2002)) to predict the prioritization levels of issue reports. Random forest (Ho (1995)) is a classification approach that builds a number of decision trees at the training stage. Random forest runs efficiently on large databases and works accurately for predictions (Ho (1995)). First, we train a model with the issue reports (that are labeled with four levels of prioritization) of top N apps that hold the highest star-ratings. Second, we use the trained model to predict the prioritization levels of issue reports of the rest of the apps. Third, we measure the accuracy of the predicted levels with the real levels using Equation (1). In Equation (1), for an app a, $I_c(i)$ shows the number of issue reports with correct predicted levels, and the $I(i)$ shows the total number of issue reports for the app a.

$$Accuracy = \frac{I_c(i)}{I(i)} \tag{1}$$

Fourth, we divide the test apps into two groups based on the accuracy of the predicted levels. We put the apps with the pair-wise similarity of more than or equal to the threshold σ into one group. The apps with the pair-wise similarity of less than σ are placed into another group. We compare the average star-ratings of the two groups of apps using the Mann–Whitney U test (Mann and Whitney (1947)) to verify whether there is a difference between the star-ratings of the two groups of apps. If the $p-value$ is less than 0.05, it shows that the difference between the star-ratings of two groups of apps is statistically significant. We also calculate the effect size of differences between the star-ratings by measuring Cliff's δ (Cliff (1993)).

To find the best number of top apps (i.e., top N apps), we conduct a sensitivity analysis on the value of N. We did the sensitivity analysis by incrementally adding top apps; starting with the app that has received the highest star-ratings ($N = 1$), adding the second app with the highest star-ratings ($N = 2$), and continuing this process until covering all the apps. We train our prioritization model based on top N apps and test the model using the rest of the apps.

We observe that with $N = 5$, we can build up a prioritization model that can statistically significantly distinguish the star-ratings of the two groups of apps with the lowest $p-value$ (i.e., $p-value = 2.7e-2$). Starting from $N = 5$, as the value of N increases (or decreases) the $p - value$ tends to increase. We decided to consider the top 5 apps as the difference in the star-ratings of the two groups of apps is larger by having $N = 5$.

The threshold σ divides the tested apps into two groups of apps that: (i) have similar prioritization orders to our predicted levels, and (ii) have different prioritization orders from our predicted levels. To identify the best value of σ, we conduct a sensitivity analysis. In our sensitivity analysis, we repeat our experiment with different values of $0.01 <= \sigma <= 0.99$ (with the increment value of 0.01). The results of our sensitivity analysis show that any values of σ between 0.43 and 0.55 cause a statistically significant difference between the two groups of the tested apps. 48 apps have a similar prioritization approach to the top 5 apps.

We calculate the mean decrease in Gini, i.e., Gini importance (Archer and Kimes (2008)), to sort the metrics of the ranking model with respect to their importance. The mean decrease in Gini measures the contribution of each metric to the homogeneity of the nodes and the leaves in the model (Biau and Scornet (2016)). As all the metrics are numerical, the mean decrease in Gini is a proper approach to identify the important metrics (Strobl et al (2007); Archer and Kimes (2008)).

Findings

Top apps can provide good patterns for other apps to follow for prioritizing the issue reports. The apps that have similar prioritizations to our predicted prioritizations receive higher star-ratings than other apps. Figure 9 shows the distribution of star-ratings for the apps that: (i) the prioritizations of the issue reports are similar to our predicted prioritizations,

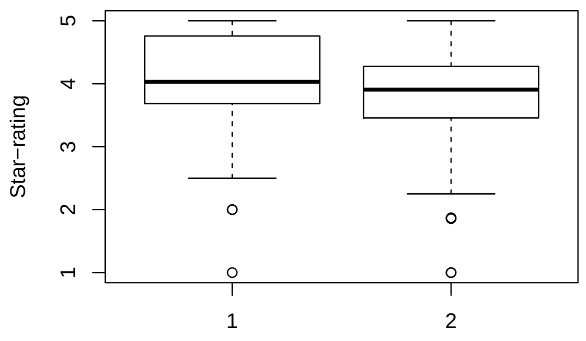


Fig. 9: The average of star-ratings of the apps that: 1) have similar prioritizations as our predicted prioritizations, 2) have different prioritizations from our predicted prioritizations.

and (ii) the prioritizations of the issue reports are different from our predicted prioritizations. As shown in Figure 9, the star-ratings are higher for the apps that have similar prioritizations to our predicted prioritizations. The difference in star-ratings between the two groups of apps is statistically significant. The $p - value$ is $3.4e - 2$ and the effect size is medium with a $Cohen's\ d = 0.30$, indicating that the difference is observable and cannot be neglected.

Table 9 shows the metrics that are used to build the ranking model with respect to the top five apps. As shown in Table 9, the time since a reporter has joined GitHub holds the highest Gini importance. The appearance of the time since a reporter has joined GitHub as a statistically significant metric could be due to the lack of knowledge and experience of the newer members (Steinmacher et al (2018); Vasilescu et al (2015)). Vasilescu *et al.* (Vasilescu et al (2015)) investigated different aspects of having a diverse (e.g., new and experienced) team members on GitHub. Having diverse developers has some advantages, such as providing a wider range of new ideas (Vasilescu et al (2015)). However, it takes some time for new developers to fit in a project (Zanatta et al (2017)). Improper contributions by newcomers require more efforts for integrations which makes the development process harder (Vasilescu et al (2015)). Also, sometimes new developers are not fully familiar with GitHub (Vasilescu et al (2015)). As a result, new developers face various barriers when contributing to an open source project (Steinmacher et al (2018); Zanatta et al (2017)), such as lack of experience and communication issues (Steinmacher et al (2018)). Different strategies, such as recruiting mentors, have been suggested in the literature to let new developers better contribute to open source projects (Steinmacher et al (2012)).

Table 9: The metrics of the ranking model based on top five apps, sorted by mean decrease in Gini.

Metric	**Description**	**Gini**
Time since reporter has joined GitHub	Estimates the reporter experience by measuring the time since the reporter is on GitHub.	89.94
Number of words in body	The number of words that is used to report an issue.	73.43
Reporter contribution	The contribution of the reporter is based on the contribution score that is reported by GitHub. Github calculates the contribution scores with respect to the following actions: committing, opening an issue, proposing a pull request, submitting a pull request, and co-authoring commits in a repository (GitHub (2018b)).	54.16
Title size	The number of words in the title of an issue report.	51.60
Mean of time since users who have involved have joined GitHub	The mean value is calculated by considering all the users who have involved in an issue report, including posting a comment and resolving the issue report.	49.70
Number of sentences in body	The number of sentences in the body of an issue report.	45.09
Mean of number of words in user-reviews	The mean value is calculated by considering the number of words in each user-review that is related to the issue report.	44.55
Mean of contribution of users who have involved in an issue	The mean value is calculated by considering all the users who have involved in an issue report, including posting a comment and resolving the issue report. The contribution of each person is calculated by GitHub.	40.46
Number of comments	The number of comments that are posted on an issue report.	39.31
Mean of number of sentences in user-reviews	The mean value of the number of sentences in the user-reviews that are related to the issue report.	39.10
Mean of sentiment scores of user-reviews	The mean value of the sentiment scores of every user-review that is related to the issue report.	32.65
Mean of numbers of followers of users who have involved in an issue	The mean value of the number of followers of the users who have involved in resolving an issue report, including posting a comment and resolving the issue report.	29.92
Number of gists of reporter	The number of gists that the reporter has shared.	28.43
Mean of star-ratings	Average of star-ratings that are associated with an issue report.	26.63
Median of star-ratings	Median of star-ratings that are associated with an issue report.	24.32
Number of followings	The number of users that the reporter follows them on GitHub.	21.02
Number of repositories of reporter	The number of repositories which to the reporter contributes.	16.63
Proportion of user-reviews with positive sentiment scores	The user-reviews with positive sentiment scores associated with an issue report.	16.13

(continues on next page)

Table 9: The metrics of the ranking model based on top five apps (continued).

Metric	Description	Gini
Proportion of user-reviews with neutral sentiment scores	The user-reviews with neutral sentiment scores associated with an issue report.	8.34
Proportion of user-reviews with negative sentiment scores	The user-reviews with negative sentiment scores associated with an issue report.	7.54
Number of user-reviews	The number of user-reviews associated with an issue report.	7.24
Proportion of user-reviews with high star-ratings	The user-reviews with high star-ratings associated with an issue report.	6.76
Proportion of user-reviews with low star-ratings	The user-reviews with low star-ratings associated with an issue report.	6.19
Number of followers of reporter	The number of users that follow the reporter on GitHub.	3.58
Mean of numbers of followings of users who have involved in an issue	The average of the numbers of followings of users who have involved in an issue, including posting a comment and resolving the issue report.	3.27
Mean of numbers of repositories of users who have involved in an issue	The average of the numbers of repositories of users who have involved in an issue, including posting a comment and resolving the issue report.	3.00
Proportion of user-reviews with neutral star-ratings	The user-reviews with neutral star-ratings associated with an issue report.	1.84
Mean of numbers of gists of users who have involved in an issue	The average of the numbers of gist of users who have involved in an issue, including posting a comment and resolving the issue report.	1.82

The size of an issue is the second metric with the highest Gini importance. Similarly, the size of the user-reviews holds the highest Gini importance among the user-related metrics. This can denote that developers of top five apps normally tend to address the user-reviews that have described an issue more in details.

For the top apps, we could match a higher proportion of user-reviews to issue reports in comparison with the rest of the apps. As reported in the first research question (see Section 3), we could match 27% of the user-reviews with 33% of the issue reports. However, for the top five apps, 52% of the user-reviews are matched with 29% of the issue reports. There is a notable increase in the proportion of the matches in the user-reviews. However, there is a small decrease in the proportion of matches in the issue reports, i.e., 33% to 29%. The top five apps address more issues that are reported in the user-reviews, while developers keep reporting other issues that may not be reflected in the user-reviews.

Building a prediction model based on the prioritization strategy of the top-rated apps can help developers to better prioritize the issue reports. Apps that have similar prioritizations to the prioritizations recommended by our approach receive higher star-ratings.

4 Threats to Validity

In this section, we discuss the threats to the validity of our study (Yin (2013)).

4.1 Conclusion Validity

Threats to conclusion validity concern the relationship between the treatment and the outcome. Martin *et al.* (Martin et al (2015a)) report that using an incomplete set of user-reviews can introduce bias to the findings of an empirical study. To eliminate this threat, we take all the user-reviews of our apps into consideration. The choice of modeling technique is another threat to conclusion validity. We use the linear regression model in the second research question. To evaluate the possible threat from the choice of modeling technique, we repeat our experiment using a multinomial regression model and find that our conclusion is not affected (i.e., no generic model can be built and the same trend as in Figure 7 is obtained).

4.2 Internal Validity

Threats to internal validity concern the analysis methods and selection of subject systems. We select the mobile apps that have more than N matches between the user-reviews and issue reports. The underlying assumption is that the apps with fewer matches do not (or rarely) address the issues described in user-reviews. With our setting (i.e., $N = 10$) (Khalid et al (2016)), only two of our apps are outliers, which have a small number of issue reports (i.e., 10 and 11) while more than 50% of their issue reports mapped to the user-reviews. Our conclusion remains the same with or without the two outlier apps. The number of comments is one of the metrics that we measured for the issue reports. Only 1.5% of the comments are "+1 comments" in our study. We use the *reaction time* to estimate the prioritization orders. However, such an estimation may introduce some noises. Nonetheless, unfortunately, there is no specific indicator that shows the exact priority of an issue report, i.e., the issue reports are not tagged with specific priority levels. As described in the paper, we carefully measured the reaction times to mitigate this threat. Finally, we did not report the recall of our approach as it requires manually matching the user-reviews with all the issue reports. Instead, we reported the precision.

4.3 External Validity

Threats to external validity concern the possibility to generalize our findings. Although we only study the open-source apps, the subject apps are from diverse categories such as *Tools*, *Video Players & Editors*, and *Shopping*. Therefore, our subject apps can represent a considerable amount of mobile apps.

Nonetheless, future work is welcome to examine our findings on proprietary mobile apps.

4.4 Reliability Validity

Threats to reliability validity concern the possibilities of replicating this study. The user-reviews and issue reports of all our subject apps are publicly accessible.

5 Related Work

In this section, we summarize the related work from three aspects: (i) user-reviews, (ii) issue tracking systems and bug reports, and (iii) issue reports prioritization.

5.1 User-Reviews

Recent work investigates the user-reviews that are posted in Google Play Store (Google (2017)) to ease the process of app development and app maintenance (Galvis Carreño and Winbladh (2013); Guzman and Maalej (2014); Iacob and Harrison (2013)).

Chen *et al.* (Chen et al (2014)) proposed a tool to identify the uninformative user-reviews and rank the informative ones. Chen *et al.* (Chen et al (2014)) employed textual analysis to detect the informative user-reviews. Villarroel *et al.* (Villarroel et al (2016)) classified and ranked the issues that are reported in the user-reviews to help app developers in planning for the next releases of their app. Villarroel *et al.* (Villarroel et al (2016)) proposed a tool to classify the user-reviews into the groups of bug reports and feature requests. However, neither work considers the issue tracking systems in the ranking process. Some of the issues that are reported in the user-reviews could already have been reported in the issue tracking systems. Therefore, adding such issues to the issue tracking system can introduce issue report duplication (Cavalcanti et al (2013)).

Ciurumelea *et al.* (Ciurumelea et al (2017)) proposed an approach to organize user-reviews with respect to different topics, such as performance and memory. Having the user-reviews organized, they recommend source-code using code localization. Sorbo *et al.* (Di Sorbo et al (2016)) presented an approach to summarize the user-reviews. Iacob and Harrison (Iacob and Harrison (2013)) employed LDA (Blei et al (2003)) to extract the feature requests from user-reviews. They applied LDA on user-reviews and looked for linguistic rules. Guzman and Maalej (Guzman and Maalej (2014)) presented an approach to assist developers in analyzing user-reviews. Guzman and Maalej (Guzman and Maalej (2014)) applied topic modeling techniques on the user-reviews for extracting features.

Moran *et al.* (Moran et al (2015)) introduced a tool, called FUSION, to auto-complete bug reports. Moran *et al.* (Moran et al (2015)) applied statistic and dynamic analysis on the decompiled code of Android apps. FUSION helps developers in reproducing bugs and auto-completing bug reports. Martin *et al.* (Martin et al (2015b)) studied the impact of app release in mobile app stores. Martin *et al.* (Martin et al (2015b)) observed that 40% of app releases impact performance in Google Play Store. Galvis and Winbladh (Galvis Carreño and Winbladh (2013)) applied textual analysis on users' feedback. They applied topic modeling and sentimental analysis on user-reviews to assist app developer in the revision of requirements for the next releases of their apps.

Earlier studies attempted to extract knowledge from user-reviews and ease the development process. None of the earlier work has integrated the user-reviews into the process of prioritizing the issue reports of the Android apps and have not studied the relationship between prioritizing issue reports and star-ratings.

5.2 Bug Reports and Issue Tracking Systems

The source code and the issue tracking system of the majority of the Android apps are not publicly available. Thus, the number of papers that study bug repositories and issue tracking systems of Android apps is not comparable to the number of papers that study user-reviews.

Some papers are based on empirical studies on characteristics of issue tracking systems. Bhattacharya *et al.* (Bhattacharya et al (2013)) conducted an empirical study on 24 open-source Android apps. They defined some metrics of bug report quality and developer involvement. Bhattacharya *et al.* (Bhattacharya et al (2013)) observed that bug reports are of high quality, especially the security bug reports have the highest quality among the bug reports. Palomba *et al.* (Palomba et al (2015)) studied 100 Android apps. They compared user-reviews with the change log of open-source apps that are available on GitHub. They reported that implementing the users' feature requests in the next releases can increase the star-ratings.

Some recent researches are based on the source-code of the open-source Android app. Mcdonnell *et al.* (McDonnell et al (2013)) studied the API update adoption by Android apps. They noticed that about 28% of references are not up-to-date. They reported that the propagation time of the API references to be updated is around 14 months. Maji *et al.* (Maji et al (2010)) applied a failure characterization study on Android and Symbian apps. They investigate the relationship between bugs, locations of the bugs in the code and code changes. Linares-Vásquez *et al.* (Linares-Vásquez et al (2015)) studied open-source apps. They noticed that app developers rely on manual execution of apps and user-reviews to identify performance bugs.

None of the recent work has studied the relationship between prioritization of issue reports and star-ratings of mobile apps.

5.3 Issue Reports Prioritization

In this section, we introduce the related work that concerns the issue reports prioritization which is mainly done on other eco-systems rather than mobile apps.

Lamkanfi *et al.* (Lamkanfi et al (2010)) proposed a severity prediction approach by analyzing textual description of issue reports of three open-source communities, including *Mozilla*, *Eclipse* and *GNOME*. They used Naïve Bayes to label the issue reports as severe or non-severe. Alenezi and Banitaan in (Alenezi and Banitaan (2013)) employed Naïve Bayes, decision trees, and random forest to predict the priority of issue reports in *Bugzilla* (Bugzilla (2018)). They observed that random forest and decision tree outperform Naïve Bayes. Yu *et al.* (Yu et al (2010)) used neural network to prioritize the issue reports. They showed their approach works better than Naïve Bayes. Kanwal and Maqbool (Kanwal and Maqbool (2012)) applied Naïve Bayes and SVM to predict the issue reports prioritization. They observed that SVM is better than Naïve Bayes when adopting textual metrics, such as issue report descriptions. However, they observed that when considering the categorical metrics, such as platform, Naïve Bayes performs better than SVM. Menzies and Marcus (Menzies and Marcus (2008)) ranked the terms that appear in the issue reports by adopting TF-IDF. They used top terms to predict the priority of issue reports. Tian *et al.* (Tian et al (2012)) used the similarity between the current issue reports and the issue reports in the past to estimate the priority of the new issue reports.

None of the above work incorporates the user-reviews with the issue report for prioritization. We take the metrics of both issue reports and user-reviews to prioritize the issue reports.

6 Conclusion

In this paper, we investigate the prioritizations of user-related issue reports and their relationship with star-ratings. First, we introduce an approach for mapping user-reviews to issue reports. We perform an empirical study of 326 open-source Android apps that have both user-reviews and issue tracking systems publicly available. Our approach achieves a precision of 79%. Second, we observe that prioritizing issue reports is positively related to increases in star-ratings. Finally, we propose a prioritization prediction method using the top apps. The prioritization model can be applied to each app to identify the prioritization orders of issue reports. Our results show that the apps with similar prioritizations to our recommended ones receive higher star-ratings.

In the future, we plan to study the generalizability of our approach more deeply. For example, we will apply our approach to user-reviews and issue reports from other app markets and issue tracking systems.

Acknowledgments

We thank the anonymous reviewers who reviewed our paper and the associated editor for their valuable feedback that helped us improve this work.

References

Alenezi M, Banitaan S (2013) Bug reports prioritization: Which features and classifier to use? In: 12th International Conference on Machine Learning and Applications (ICMLA), IEEE, vol 2, pp 112–116

Allacronyms (2017) Acronyms and abbreviations related to computer science. [Online]. Available: https://www.allacronyms.com/computer-science/abbreviations

Archer KJ, Kimes RV (2008) Empirical characterization of random forest variable importance measures. Computational Statistics & Data Analysis 52(4):2249–2260

Basili VR (1992) Software modeling and measurement: the goal/question/metric paradigm. Tech. rep., Institute for advanced computer studies

Bavota G, Linares-Vasquez M, Bernal-Cardenas CE, Penta MD, Oliveto R, Poshyvanyk D (2015) The impact of api change-and fault-proneness on the user ratings of android apps. IEEE Transactions on Software Engineering 41(4):384–407

Bertram D, Voida A, Greenberg S, Walker R (2010) Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In: 2010 ACM conference on Computer supported cooperative work, ACM, pp 29–300

Bhattacharya P, Ulanova L, Neamtiu I, Koduru SC (2013) An empirical analysis of bug reports and bug fixing in open source android apps. In: 17th European Conference on Software Maintenance and Reengineering (CSMR), IEEE, pp 133–143

Biau G, Scornet E (2016) A random forest guided tour. Test 25(2):197–227

Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. The Journal of Machine Learning Research 3:993–1022

Broder AZ, Glassman SC, Manasse MS, Zweig G (1997) Syntactic clustering of the web. Computer Networks and ISDN Systems 29(8-13):1157–1166

Bruns A, Kornstadt A, Wichmann D (2009) Web application tests with selenium. IEEE software 26(5)

Bugzilla (2018) Bugzilla. [Online]. Available: https://www.bugzilla.org/

Calders T, Verwer S (2010) Three naive bayes approaches for discrimination-free classification. Data Mining and Knowledge Discovery 21(2):277–292

Cavalcanti YC, Neto PAdMS, Lucrédio D, Vale T, de Almeida ES, de Lemos Meira SR (2013) The bug report duplication problem: an exploratory study. Software Quality Journal 21(1):39–66

Chen N, Lin J, Hoi SC, Xiao X, Zhang B (2014) Ar-miner: mining informative reviews for developers from mobile app marketplace. In: 36th International Conference on Software Engineering, ACM, pp 767–778

Ciurumelea A, Schaufelbhl A, Panichella S, Gall H (2017) Analyzing reviews and code of mobile apps for better release planning. In: 24th International Conference on Software Analysis Evolution and Reengineering, IEEE

Cliff N (1993) Dominance statistics: Ordinal analyses to answer ordinal questions. Psychological Bulletin 114(3):494

Cohen J (2013) Statistical power analysis for the behavioral sciences. Academic press

De Marneffe MC, MacCartney B, Manning CD, et al (2006) Generating typed dependency parses from phrase structure parses. In: 5th International Conference on Language Resources and Evaluation, vol 6, pp 449–454

Developer G (2018) Github developer. [Online]. Available: https://developer.github.com/v3/

Di Sorbo A, Panichella S, Alexandru CV, Shimagaki J, Visaggio CA, Canfora G, Gall HC (2016) What would users change in my app? summarizing app reviews for recommending software changes. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, pp 499–510

Ester M, Kriegel HP, Sander J, Xu X, et al (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: 2nd International Conference on Knowledge Discovery and Data Mining, vol 96, pp 226–231

Faraway JJ (2005) Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models. CRC press

FDroid (2017) F-droid. [Online]. Available: http://www.f-droid.org/

Galvis Carreño LV, Winbladh K (2013) Analysis of user comments: an approach for software requirements evolution. In: 35th International Conference on Software Engineering, IEEE, pp 582–591

GitHub (2018a) Github. [Online]. Available: http://www.github.com/

GitHub (2018b) Github help. [Online]. Available: https://help.github.com/articles/viewing-contributions-on-your-profile/

Google (2017) Google play store. [Online]. Available: http://play.google.com/

Gousios G (2013) The ghtorrent dataset and tool suite. In: 10th Working Conference on Mining Software Repositories, IEEE Press, Piscataway, NJ, USA, MSR '13, pp 233–236

Guzman E, Maalej W (2014) How do users like this feature? a fine grained sentiment analysis of app reviews. In: 22nd International Conference on Requirements Engineering, IEEE, pp 153–162

Hmisc (2017) Harrell miscellaneous. [Online]. Available: http://cran.r-project.org/web/packages/Hmisc/index.html

Ho TK (1995) Random decision forests. In: Third International Conference on Document Analysis and Recognition, IEEE, vol 1, pp 278–282

Iacob C, Harrison R (2013) Retrieving and analyzing mobile apps feature requests from online reviews. In: 10th Working Conference on Mining Software

Repositories, IEEE, MSR '13, pp 41–44

Islam MR, Zibran MF (2017) Leveraging automated sentiment analysis in software engineering. In: 14th International Conference on Mining Software Repositories, IEEE Press, pp 203–214

Janák J (2009) Issue tracking systems. Brno, spring

Jazzy (2017) Jazzy spell checker. [Online]. Available: http://jazzy.sourceforge.net/

Kanwal J, Maqbool O (2012) Bug prioritization to facilitate bug report triage. Journal of Computer Science and Technology 27(2):397–412

Kelley TL (1947) Fundamentals of statistics. Harvard University Press

Khalid H, Nagappan M, Shihab E, Hassan AE (2014) Prioritizing the devices to test your app on: A case study of android game apps. In: 22nd International Symposium on the Foundations of Software Engineering, pp 370–379

Khalid H, Nagappan M, Hassan AE (2016) Examining the relationship between findbugs warnings and app ratings. IEEE Software 33(4):34–39

Kim HW, Lee H, Son J (2011) An exploratory study on the determinants of smartphone app purchase. In: 11th International DSI and the 16th APDSI Joint Meeting

Kim SM, Pantel P, Chklovski T, Pennacchiotti M (2006) Automatically assessing review helpfulness. In: 2006 Conference on empirical methods in natural language processing, Association for Computational Linguistics, pp 423–430

Lamkanfi A, Demeyer S, Giger E, Goethals B (2010) Predicting the severity of a reported bug. In: 7th IEEE Working Conference on Mining Software Repositories (MSR), IEEE, pp 1–10

Liaw A, Wiener M (2002) Classification and regression by randomforest. R news 2(3):18–22

Linares-Vásquez M, Vendome C, Luo Q, Poshyvanyk D (2015) How developers detect and fix performance bottlenecks in android apps. In: 31st Conference on Software Maintenance and Evolution, IEEE, pp 352–361

Lovins JB (1968) Development of a stemming algorithm. MIT Information Processing Group, Electronic Systems Laboratory

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

Mann HB, Whitney DR (1947) On a test of whether one of two random variables is stochastically larger than the other. The annals of mathematical statistics pp 50–60

Manning CD, Surdeanu M, Bauer J, Finkel J, Bethard SJ, McClosky D (2014) The stanford corenlp natural language processing toolkit. In: 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp 55–60

Martin W, Harman M, Jia Y, Sarro F, Zhang Y (2015a) The app sampling problem for app store mining. In: 12th Working Conference on Mining Software Repositories, IEEE, pp 123–133

Martin W, Sarro F, Harman M (2015b) Causal impact analysis applied to app releases in google play and windows phone store. RN 15:07

McDonnell T, Ray B, Kim M (2013) An empirical study of api stability and adoption in the android ecosystem. In: 29th International Conference on Software Maintenance, IEEE, pp 70–79

Menzies T, Marcus A (2008) Automated severity assessment of software defect reports. In: International Conference on Software Maintenance, IEEE, pp 346–355

Miller GA (1995) Wordnet: a lexical database for english. Communications of the ACM 38(11):39–41

Moran K, Linares-Vásquez M, Bernal-Cárdenas C, Poshyvanyk D (2015) Auto-completing bug reports for android applications. In: 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 673–686

Nelder JA, Baker RJ (1972) Generalized linear models. Encyclopedia of statistical sciences

Netlingo (2017) Top 50 most popular text terms. [Online]. Available: http://www.netlingo.com/top50/popular-text-terms.php

Nguyen TH, Adams B, Hassan AE (2010) Studying the impact of dependency network measures on software quality. In: 26th International Conference on Software Maintenance, IEEE, pp 1–10

Noei E, Heydarnoori A (2016) Exaf: A search engine for sample applications of object-oriented framework-provided concepts. Information and Software Technology 75:135–147

Noei E, Syer MD, Zou Y, Hassan AE, Keivanloo I (2017) A study of the relation of mobile device attributes with the user-perceived quality of android apps. Empirical Software Engineering 22(6):3088–3116

Noei E, Da Costa DA, Zou Y (2018) Winning the app production rally. In: 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ACM, New York, NY, USA, ESEC/FSE 2018, pp 283–294

Nord C (2005) Text analysis in translation: Theory, methodology, and didactic application of a model for translation-oriented text analysis. 94, Rodopi

Optimaize (2017) Language detection library for java. [Online]. Available: https://github.com/optimaize/language-detector/

Palomba F, Linares-Vásquez M, Bavota G, Oliveto R, Di Penta M, Poshyvanyk D, De Lucia A (2015) User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In: 31st International Conference on Software Maintenance and Evolution, IEEE, pp 291–300

Panichella S, Di Sorbo A, Guzman E, Visaggio C, Canfora G, Gall H (2015) How can i improve my app? classifying user reviews for software maintenance and evolution. In: 31st International Conference on Software Maintenance and Evolution

Rajaraman A, Ullman JD, Ullman JD, Ullman JD (2012) Mining of massive datasets, vol 77. Cambridge University Press Cambridge

Romero DM, Galuba W, Asur S, Huberman BA (2011) Influence and passivity in social media. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, pp 18–33

Salton G, Michael J (1983) Mcgill. Introduction to modern information retrieval pp 24–51

Salton G, Wong A, Yang CS (1975) A vector space model for automatic indexing. Communications of the ACM 18(11):613–620

Selenium (2017) Selenium - web browser automation. [Online]. Available: http://seleniumhq.org/

Snowball (2018) Snowball. [Online]. Available: http://snowballstem.org/

Statista (2017a) Number of apps available in leading app stores as of march 2017. [Online]. Available: http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores

Statista (2017b) Number of smartphone users worldwide from 2014 to 2020 (in billions). [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide

Stats A (2016) Number of android applications. [Online]. Available: http://www.appbrain.com/stats/number-of-android-apps

Steinmacher I, Wiese IS, Gerosa MA (2012) Recommending mentors to software project newcomers. In: Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering, IEEE Press, pp 63–67

Steinmacher I, Treude C, Gerosa M (2018) Let me in: Guidelines for the successful onboarding of newcomers to open source projects. IEEE Software

Strobl C, Boulesteix AL, Zeileis A, Hothorn T (2007) Bias in random forest variable importance measures: Illustrations, sources and a solution. BMC bioinformatics 8(1):25

Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2017) An empirical comparison of model validation techniques for defect prediction models. IEEE Transactions on Software Engineering 43(1):1–18

Tian Y, Lo D, Sun C (2012) Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In: Reverse Engineering (WCRE), 2012 19th Working Conference on, IEEE, pp 215–224

Van Solingen R, Basili V, Caldiera G, Rombach HD (2002) Goal question metric (gqm) approach. Encyclopedia of software engineering

Vasilescu B, Filkov V, Serebrenik A (2015) Perceptions of diversity on github: A user survey. In: Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering, IEEE Press, pp 50–56

Villarroel L, Bavota G, Russo B, Oliveto R, Di Penta M (2016) Release planning of mobile apps based on user reviews. In: 38th International Conference on Software Engineering, ACM, pp 14–24

Xuan J, Jiang H, Ren Z, Zou W (2012) Developer prioritization in bug repositories. In: Software Engineering (ICSE), 2012 34th International Conference on, IEEE, pp 25–35

Yin RK (2013) Case study research: Design and methods. Sage publications

Yu L, Tsai WT, Zhao W, Wu F (2010) Predicting defect priority based on neural networks. In: International Conference on Advanced Data Mining and Applications, Springer, pp 356–367

Yu Y, Wang H, Filkov V, Devanbu P, Vasilescu B (2015) Wait for it: determinants of pull request evaluation latency on github. In: Mining software repositories (MSR), 2015 IEEE/ACM 12th working conference on, IEEE, pp 367–371

Zanatta AL, Steinmacher I, Machado LS, de Souza CR, Prikladnicki R (2017) Barriers faced by newcomers to software-crowdsourcing projects. IEEE Software 34(2):37–43

Zhang F, Mockus A, Keivanloo I, Zou Y (2015) Towards building a universal defect prediction model with rank transformed predictors. Empirical Software Engineering pp 1–39