Received: 00 Month 0000 | Revised: 00 Month 0000 | Accepted: 00 Month 0000

DOI: xxx/xxxx

RESEARCH PAPER

Investigating the Criticality of User Reported Issues through their Relations with App Rating.

Andrea Di Sorbo[1] | **Giovanni Grano**[2] | **Corrado Aaron Visaggio**[1] | **Sebastiano Panichella**[3]

[1]Department of Engineering, University of Sannio, Benevento, Italy
[2]Department of Informatics, University of Zurich, Zurich, Switzerland
[3] School of Engineering, Zurich University of Applied Science, Zurich, Switzerland

Correspondence
Andrea Di Sorbo, Via Traiano 1, Benevento, Italy
Email: disorbo@unisannio.it

Summary

App quality impacts user experience and satisfaction. As a consequence, both app ratings and user feedback reported in app reviews are directly influenced by the *user-perceived* app quality. Through an empirical study involving 210,517 reviews related to 317 Android apps, in this paper, we experiment with the combined usage of app rating and user reviews analysis (i) to investigate the most important factors influencing the perceived app quality, (ii) focusing on the topics discussed in user review that most relate with app rating. Besides, we investigate whether specific code quality metrics could be monitored to prevent the rising of negative user feedback (i.e., types of user review comments), connected with low ratings. Our study demonstrates that user feedback reporting bugs are negatively correlated with the rating, while reviews reporting feature requests do not. Interestingly, depending on the app category, we observed that different kinds of issues have rather different relationships with the rating and the user-perceived quality of the app. In particular, we observe that for specific app categories (e.g., Communication) some code quality factors have significant relationships with the raising of certain types of feedback, which, in turn, are negatively connected with app ratings.

KEYWORDS:
Software Quality, Mobile Apps, App Reviews, Software Maintenance and Evolution

1 | INTRODUCTION

During the recent years, the Global App Economy (GAE)[1] experienced an unprecedented growth, driven by the increasing usage of apps and by the growing smartphone adoption around the globe[1]. According to recent market researches, in few years from now, the GAE is expected to double in size to $101 billion[2]. Android, with about 75-80% of the worldwide smartphone market, represents not only the most popular world's mobile operating system but arguably the most popular operating system nowadays[3,1,2]. In this context, the ease of distribution and the large number of potential users available in the various app stores, has made the mobile development an attractive field for software engineers. However, this led to an increasing competition between developers[4,5,6], making the GAE a *"world where everybody competes with everybody – everybody is treated equally in the app stores of the world"*[7]. Inevitably, some developers pay the price of this fierce competition: recent studies have shown that the reality for most of them is far from being rewarding, with 60% of developers gaining less than $500/month[1,5].

In such a competitive market, user experience and satisfaction play a paramount role for the success of mobile applications[4,5,8]. As a consequence, mobile developers would certainly benefit from understanding which factors most impact user experience and satisfaction (thus, app ratings) and how these factors are related to software quality[4,9].

Users mostly rely on star-ratings for choosing new apps to download[10]. Indeed, mobile apps with higher ranks are more likely to be noticed and downloaded by users[11]. To rationalize the star-ratings, users typically complement them by providing app review comments and, especially in

[1]App economy consists of everybody who makes money and has a job thanks to mobile apps.

low-rated reviews [12], significant amount of these comments include requirements-related information such as bugs or issues [13]. Thus, it is crucial to support developers in finding the key issues contained in app reviews that could be significantly related to star-ratings of a given app category, as app developers can first focus on the types of user feedback that need to be addressed to avoid receiving lots of low ratings .

A recent research [14] demonstrated that each app category has a specific set of key topics (*i.e.*, the topics appearing in app user reviews having statistically significant relationships with star-ratings). However, it is still unclear what are the types of issues that are raised in app user reviews having negative impact on the star-rating. To fill this gap, Noei *et al.* [15]. proposed to map issue reports that are recorded in issue tracking systems to user reviews, in order to prioritize the issue reports that are related to user reviews and observed that prioritizing such issue reports could positively influence the star-rating. Nevertheless, there is vocabulary mismatch between user reviews and issues reported in issue trackers and mobile app developers often rely exclusively on reviews for acknowledging the problems reported [16]. To cope with these drawbacks, similarly to what observed by Noei *et al.* [14], we conjecture that issues of the same types could have different relationships with rating, depending on the different kinds of apps. As showed in the following examples, a specific kind of bugs (*i.e.*, concerning app contents) could be more critical for users of apps belonging to certain categories (*e.g.*, Music & Audio) than others.

"The music pause is unbearable I love the web app. I stopped using other music players on PC and even paid for 1year of premium service. I simply cannot stand this app. It always pauses music which ruins songs. It also starts even when Apollo is running so that two songs are playing at the same time for no reason. No other app behaves this poorly on my S4 GE. I hate to say it but I uninstalled" (**App**: Subsonic Music Streamer, **Rating**: 1).

"Great app!! Doesn't matter if it's not working at 100% on my nexus 5 sometime the picture doesn't save properly but perfection will come with time. If you can update this project more often will be great!" (**App**: Focal, **Rating**: 5).

Thus, a user reviews prioritization mechanism based on the relationships that they exhibit with the star-rating might bring out the most critical problems, allowing developers to promptly address them, and, consequently, limiting the effects on the app rating. As a matter of fact, timely identifying the emerging issues of apps can avoid receiving massive low ratings and alleviate the losses of users [17].

Paper Contribution. The main goal of this paper is to investigate (i) the types of issues raised in user reviews that exhibit significant relationships with star-rating, within the different app categories, as well as (ii) the code quality indicators that could be connected with the rising of such kinds of issues. The purpose of the study is twofold. On the one hand, we want to provide developers with guidelines and a general approach to conduct software maintenance activities (*e.g.*, the priorities to assign to the different types of user feedback), with the aim of maximizing user satisfaction. On the other hand, our investigation is aimed at identifying the source code quality indicators that developers should monitor, in order to prevent the rising of critical issues (that, in turn, could degrade the user satisfaction). In particular, we investigate whether low quality characteristics of code are related to the rising of certain types of user feedback and the extent to which this is related to a lower app rating. Through an empirical study involving 210,517 reviews, 443 versions related to 317 Android apps, we focus our effort in analyzing more in depth the mutual connections between code metrics, rating and the kinds of user feedback reported in user review. Hence, the main research questions investigated in this paper are the following:

- **RQ_1: To what extent are specific kinds of user feedback connected with app rating?** To answer this research question we conduct two different analysis: At first, we investigate whether particular types of user feedback appear more frequently in reviews with high or low ratings. Therefore, we analyze the *correlation* between particular types of user feedback and app ratings. Specifically, feedback expressed by users in app reviews may directly reflect their satisfaction with the entire app or a specific feature [18]. Indeed, problems stated by users in reviews may often refer to specific *aspects* or *topics* of an app [8]. However, problems or shortcomings related to specific aspects of an app may exhibit a greater impact on the actual app rating (*e.g.*, depending on the app category). Thus, as second analysis, in this research question, we quantitatively observe the kinds of feedback raised by users in reviews which may be connected with their decision to rate the app with a lower (or higher) number of stars (*i.e.*, app rating). In particular, to address this aspect, we measure *the percentages* of problems or requests reported by users about a specific *aspect* of an app (*e.g.*, UI, app pricing, resource consuming, etc.). We focus on this kind of feedback since critical bugs and requests reported by users are crucial for developers aiming at maximizing user satisfaction [19].

- **RQ_2: Are there code quality indicators correlated with the rising of critical issues?** This research question is aimed at studying the code quality aspects/characteristics that developers should carefully monitor to prevent the rising of critical issues. Hence, we wonder whether poor code quality is associated with the growing of specific types of user feedback that are connected with app rating. Pragmatically speaking, we are interested in investigating the existing relationships between code quality indicators and the types of user feedback that are related to app rating (see Section 3.1.2). To answer this research question we correlate 22 code quality indicators (detailed in Section 3.1.2) with the occurrences of specific types of user feedback, to investigate whether lower (or higher) values of specific code quality indicators co-occur with the raising of certain types of feedback. Such indicators are divided into 4 categories: *Dimensional*, *Complexity*,

Object-oriented and *Android-oriented* metrics. Specifically, since mobile apps are written through Object-Oriented languages, we select some of the classic metrics used to measure object oriented design quality [20]. However, there are some peculiarities that differentiate mobile apps from traditional software products, *e.g.*, the limited computational resources, the strong usage of network and the localization services [21]. In order to fill this gap, we expand our analysis considering also Android-specific code metrics.

To provide additional insights onto the success and evolution of mobile applications, we specialize the analysis of all research questions by grouping the apps of our dataset according to the Play Store's categories.

Paper Structure. The remainder of this paper is organized as follows. Section 2 summarizes background information and presents the related literature, underlining the particular contribution of our work. Section 3 describes the collected metrics and the approach we rely on to answer our research questions. Section 4 presents the obtained results along with a discussion about our findings. The threats that could influence the study are presented in Section 5. Section 6 concludes the paper highlighting future research directions.

2 | RELATED WORK

Previous research investigated the relationship between the rating and a particular characteristic (or feature) of mobile applications [22,10,23,24,5], this because mobile apps with higher rating ranks are more likely to be noticed and downloaded by users [11]. While the research community widely investigated this topic, no prior work experimented the combined usage of app rating and user reviews analysis (i) to investigate the most important factors influencing the perceived app quality, (ii) focusing on the topics discussed in user review that most affect mobile app rating and software quality. Thus, we discuss the most close related work to our investigation, to clarify the context of our work.

Corral *et al.* [22] investigated the relationship between code quality and market success, expressed in average rating and in the number of downloads. They showed how the code quality, in terms of complexity, cohesion, coupling, and size, plays a marginal role on the market success. Guerrouj *et al.* [25] studied the impact of app churn on the app rating observing that high app churn leads to lower user ratings. Furthermore, Tian *et al.* [5] investigated the most influential factors of high-rated apps. They considered also metrics about user requirements and marketing effort. The authors demonstrated that marketing effort, app size and the target SDK version are the prominent factors in recognizing successful (high-rated) apps. In the current literature, researchers also explored (i) the correlation between the number of ad libraries and rating [26], (ii) relationships between the app description, the number of downloads and average user rating, in the context of BlackBerry ecosystem [27], and (iii) the influence of app ratings and price on the amount of user feedback [28]. Compared to these work, we complement the analysis merely based on the rating, by more in-depth investigating the content of user reviews (*i.e.*, what do users discuss about) and how different topics occur in both low and high-rated reviews. Moreover, we introduce *Android-oriented* metrics that might have a more direct impact on the user experience and app success.

In recent years, researchers also demonstrated that, together with app rating users typically complement it by providing app review comments and, especially in low-rated reviews [12], significant amounts of these comments include requirements-related information such as bugs or issues [13]. This highlight the importance of supporting developers in finding the key issues contained in app reviews that could be significantly related to star-ratings of a given app category, which is the main goal of this work. A close work to our is the one by Noei *et al.* [14], which demonstrated that each app category has a specific set of key topics (*i.e.*, the topics appearing in app user reviews having statistically significant relationships with star-ratings). In a different work, Noei *et al.* [15] proposed to map issue reports that are recorded in issue tracking systems to user reviews, in order to prioritize the issue reports that are related to user reviews, as they can positively influence the star-rating. Our work builds on the the idea that there is vocabulary mismatch between user reviews and issues reported in issue trackers and mobile app developers often rely exclusively on reviews for acknowledging the problems reported [16]. To cope with these drawbacks, similarly to what observed by Noei *et al.* [14], we conjecture that issues of the same types could have different relationships with rating, depending on the different kinds of apps. In particular, we argue that users of different apps may express different judgments, based on the specific nature of the app. Thus, our work was also conducted by observing potential variation on the results/findings, specializing the analysis for the different app store's categories.

Finally, recent research efforts have been devoted to investigating the reliability of app rating when used as a proxy to represent user satisfaction. For example, Luiz *et al.* [29] proposed a framework performing sentiment analysis on a set of relevant features extracted from user reviews. Despite the star rating was considered to be a good metric for user satisfaction, their results suggest that sentiment analysis might be more accurate in capturing the sentiment transmitted by the users. Hu *et al.* [30] studied the consistency of reviews and star ratings for hybrid Android and iOS apps discovering that they are not consistent across different app markets.

TABLE 1 Intention Categories Definition

Category	Description	Example
Information Giving (IG)	Sentences that inform other users or developers about some aspect of the app.	*Great app launcher Perfect for users of Alfred or Quicksilver.*
Information Seeking (IS)	Sentences describing attempts to obtain information or help from other users or developers.	*Can anyone pls tell me the full procedure to connect this?*
Feature Request (FR)	Sentences expressing ideas, suggestions or needs for enhancing the app.	*Would love to see both an Add button & a Sync button at the top of the main page.*
Problem Discovery (FR)	Sentences reporting unexpected behavior or issues.	*Whenever I touch the Downloads option the program crashes and just does not let me use it*
Other	Sentences not belonging to any of the previous categories.	*Deserves every star.*

3 | DESIGN

In this section, for each research question, we first depict the methodology used for answering it, and then discuss the data collection and the approach used for processing it.

3.1 | Research method

In this section we describe the metrics and the approaches we rely on in order to answer the aforementioned research questions.

3.1.1 | RQ_1 - To what extent are specific kinds of user feedback connected with app rating?

In the following, we discuss the motivation driving the first research question, in addition to the metrics and the approach we rely on in order to answer it.

Motivation

Ratings have been used for long time as a main metric to measure user satisfaction. However, rating is merely a score from 1 to 5 that by design lacks of qualitative information. User reviews represent the gold mine from where such a qualitative information can be derived. This first research question is driven by the idea that user reviews analysis and app rating might be used in a complementary way to understand the kinds of issues that are more critical from the user's perspective. Therefore, here we aim at investigating which kinds of user feedback are present in the reviews in case of low or high rating. Moreover, we are interested in understanding whether issues concerning different topics discussed in the reviews relate with the rating itself.

User-oriented Metrics

To investigate the relationships existing between specific kinds of feedback and the app rating, we rely on the User Review Model (URM) proposed by Di Sorbo *et al.*which allows analyzing sentences in user review comments from a software maintenance and evolution perspective [8]. The URM framework firstly splits the reviews into atomic sentences. Then, each sentence is automatically labeled with a combination of one of the *intention* categories (detailed in Table 1) and one (or more) of the *topics* (detailed in Table 2).

In particular, given a specific apk_i we compute $P_{(I,T)}$ as the proportion of sentences $S_{(I,T)}$, belonging to the particular *intention* class, I, and treating the specific *topic*, T, with respect to the total amount of the collected reviews, S_{ALL}, related to apk_i. In particular:

$$P_{(I,T)} = \frac{S_{(I,T)}}{S_{ALL}}$$

Thus, $P_{(I,T)}$ measures the percentage of sentences having a specific *intention-topic* pair assigned. We also compute $P_{(I,all)}$ as the percentage of sentences $S_{(I,all)}$ falling in a specific *intention* category, I, and dealing with any of the topics detailed in Table 2.

$$P_{(I,all)} = \frac{S_{(I,all)}}{S_{ALL}}$$

TABLE 2 Topic Definitions

Cluster	Description	Example
App	sentences related to the entire app, *e.g.*, generic crash reports, ratings, or general feedback	*Most useful stable straightforward password management app out there.*
GUI	sentences related to the Graphical User Interface or the look and feel of the app	*The interface isn't exactly friendly but you can figure it out with some trial and error.*
Contents	sentences related to the content of the app	*I'm so glad there are songs for the church included.*
Pricing	sentences related to app pricing	*You could however pay $2.00 to play on up to 13 by 13 which was pretty reasonable.*
Feature or Functionality	sentences related to specific features or functionality of the app	*I feel so angry with this new setting of Google search box HATED IT.*
Improvement	sentences related to explicit enhancement requests	*It would be even better with a blue light filter in the night mode.*
Updates/ Versions	sentences related to specific versions or the update process of the app	*weak this version suffers from heavy lags.*
Resources	sentences dealing with device resources such as battery consumption, storage, etc.	*As an example overnight it drained 30% of the battery simply being resident in memory.*
Security	sentences related to the security of the app or to personal data privacy	*Considering privacy concerns and permissions I'm uninstalling.*
Download	sentences containing feedback about the app download	*Cannot get it installed; continuously says "downloading" but doesn't.*
Model	sentences reporting feedback about specific devices or OS versions	*Perfect under Android 4.4 but the effect doesn't persist in 5.0.*
Company	sentences containing feedback related to the company/team which develops the app	*Big thanks to the development team behind this awesome emulator.*
Other	sentences not treating any of the previous topics	*Nothing to worry about.*

It is worth to notice that in this paper we focus our analysis on user reviews belonging to the *Problem Discovery* (PD) and *Feature Requests* (FR) intention category, since they are the most occurring complaints in low rated apps[31], thus, they are the more relevant for software evolution[8]. Such two-level classification model allows us to determine, for instance, whether in a review the user has the *intention* to highlight a problem (e.g., by identifying the review comment as a *Problem Discovery*) with an aspect related to the application user interface (e.g., by categorizing the same comment as *GUI* related). An example of review discussing a *problem* with *GUI* is illustrated below.

"...My two complaints are that (at least on my phone) the screen is super small and won't enlarge easily also navigating between the words is a pain..." - App: *Shortyz Crosswords*, Rating: *4*.

Despite several other approaches have been developed with the aim to classify user reviews content[32,18,33,34,35], we rely on URM since it is able to perform a finer-grained categorization at the sentence level. From such model, we employ 2 distinct *intentions* and 13 distinct *topics*. Therefore, considering every possible combination amongst them, plus the two values of $P_{(I,all)}$ we end up with 28 different $P_{(I,T)}$ values for each apk_i.

Approach

The goal of this research question is to analyze kinds of feedback raised by users when they decide to rate an app with a lower (or higher) amount of stars. In particular, we are first interested in investigating the differences between (i) the kinds of feedback provided by users who rated an app with a high number of stars (*i.e.*, 4-5 stars), and (ii) the types of feedback appearing in reviews with a lower number of stars (*i.e.*, 1-2 stars). For this purpose, we investigate the combinations of *intention* and *topic* which occur in both low and high rated reviews. For each app category, C_i, we consider the collection of reviews R_i related to all the apks belonging to C_i. We then split R_i in two different groups: (i) $R_{i,low}$, containing the reviews in R_i with 1 or 2 stars assigned, and (ii) $R_{i,high}$, encompassing the reviews in R_i with 4 or 5 stars assigned. Afterwards we compute the aforementioned $P_{(I,T)}$ *values* for both $R_{i,low}$ and $R_{i,high}$ analyzing the similarities and differences occurring in the distributions of each metric within the two groups of reviews. The analysis is carried out for the overall dataset (*i.e.*, by considering all the apps together, *All*), as well as for the different Play Store categories present in our data collection.

Finally, to better understand the connection between topics discussed and rating, we proceed further with a second analysis. We rely on Spearman's Rank coefficient to compute the correlations between the aforementioned *user-oriented* metrics (*i.e.*, the specific pair of *intention* and *topics*) and the average rating.

Since the store rating (*i.e.*,, the rating that is displayed in the app's page of the Google Play) is very resilient to changes occurring in version-to-version ratings [36], it is worth to notice that the average rating was not statically gathered from the Play Store but it has been tailored relying on the reviews extracted. Thus, for each apk (*i.e.*, for each app version), we compute the average rating as the mean of star-rating values related to the reviews mined for such apk.

Spearman's rank is an index (ρ) to calculate the correlation between two non-parametric variables: the closer is ρ to ± 1, the stronger it is the relationship [37]. In our case, if a specific $P_{(I,T)}$ value tends to increase when the rating does the same, the Spearman correlation ρ coefficient is positive, while in the opposite case it is negative. We interpret the strength of the correlation as *very weak* for $0.00 < |\rho| < 0.19$, *weak* for $0.20 < |\rho| < 0.39$, *moderate* for $0.40 < |\rho| < 0.59$, *strong* for $0.60 < |\rho| < 0.79$ and *very strong* for $|\rho| > 0.8$. Since some correlations could be altogether casual, we only reported the most significative ones (*p-value* ≤ 0.05) along with the marginally significative ones ($0.05 <$ *p-value* ≤ 0.10, marked with the symbol * in all the figures).

3.1.2 | RQ_2 - Are there code quality indicators correlated with the rising of critical issues?

As done for the RQ1, we present the motivations for the second research question. Thus, we describe the metrics and the employed approach.

Motivation

The main goal of the second research question is to investigate whether different code quality aspects/characteristics — that can be observed through code metrics — can be monitored in order to prevent the raising of issues that are correlated with the rating, and, thus, can be critical from the users' perspective. In other words, we want to establish whether some code quality indicators are connected with a larger presence of certain types of issues, that in turn exhibit negative relationships with the rating. Thus, timely interventions on such code aspects might limit the growing of users' complaints related to critical issues, and consequently the effects on app rating.

Code quality Metrics

To asses and measure the code quality of an Android application, given an apk_i available in our dataset we compute 22 code quality indicators, divided into 4 categories: *Dimensional*, *Complexity*, *Object-oriented and Android-oriented* metrics. Most of these indicators have been previously investigated in recent work [5]. However, as already reported, we also investigate *Android-oriented* metrics, since they model factors which might have a direct influence on the user experience of Android applications [38,39,40,41,42]. In the following we describe the meaning and the rationale of each metric. In our replication package we further include a description of the collection process[2].

Dimensional Metrics (described in Table 3) provide a quantitative measure of the app sizes in terms of code size and modularity. Apps with larger dimensional metrics might contain more features and better functionalities leading to higher ratings and better users' experience. However, apps with larger code size might have an higher probability to contain bugs [5,43], leading to lower ratings, negative users' experience and an higher number of complains or negative comments in user reviews.

TABLE 3 Description of the dimensional metrics

Metrics	Description
Number of Byte- code Instructions (**NBI**)	It counts the total number of `smali` byte-code instructions, ignoring comment lines and blank lines (i.e., NCLOC)
Number of Classes (**NOC**)	It computes the number of classes within the app package
Number of Methods (**NOM**)	It estimates the amount of methods within the app package
Instructions per Method (**IPM**)	Is computed by the proportion between the total number of instructions (i.e., NBI) and the total number of methods (i.e., NOM)

[2]https://github.com/sealuzh/user-satisfaction/wiki/Code-Quality-Metrics

TABLE 4 Description of the complexity metrics

Metrics	Description
Cyclomatic Complexity (**CC**)	This is a complexity metric introduced by Thomas McCabe. This metric measures the number of linearly independent paths contained in the control flow of the program
Weighted Methods per Class (**WMC**)	This is a complexity metric introduced by Chidamber and Kemerer. The WMC metric is the sum of the complexities of all class methods

TABLE 5 Description of the Object-Oriented metrics

Metrics	Description
Number of Children (**NOCH**)	It indicates the number of immediate subclasses subordinated to a class in the class hierarchy. Greater is the number of children, greater is the reuse, but if a class has a large number of children, it may require more testing of the methods in that class
Depth of Inheritance Tree (**DIT**)	It indicates the depth (i.e., the length of the maximal path from the node representing the class to the root of the tree) of the class in the inheritance tree. Deeper trees constitute greater design complexity, since more methods and classes are involved
Lack of Cohesion in Methods (**LCOM**)	It indicates the level of cohesion between methods and attributes of a class
Coupling Between Objects (**CBO**)	It indicates the dependency degree of a class by another one
Percent Public Instance Variables (**PPIV**)	It indicates the ratio of variables introduced by a `public` modifier
Access to Public Data (**APD**)	This metric counts the number of accesses to `public` or `protected` attributes of each class

Complexity Metrics (described in Table 4) and **Object-Oriented Metrics** (described in Table 5 can be used to estimate the apps code complexity, cohesion and coupling. We rely on two different complexity metrics, *i.e.*, cyclomatic complexity (CC) and weighted method per class (WMC). Since mobile apps are implemented with object-oriented programming languages we can assess their code quality by using the metrics suite by Chidamber and Kemerer[20]. Similarly to dimensional factors, apps with high complexity metrics or high code modularity might contain more complex features and better functionalities leading to higher ratings and better user experience. However, also in this case, apps with higher code complexity or low code modularity might have a higher probability to contain bugs[44,45], leading to lower ratings and negative users' experience. Example of metrics that fall in such a category are number of children (NOCH) and depth of inheritance tree (DIT).

Android-Oriented Metrics (described in Table 6) are aimed at assessing aspects influencing the users' experience, such as the management of resources or the handling of possible error conditions. Some of the factors that may influence the users' experience in Android applications are related to (i) the incorrect use of specific methods that may cause crashes[38], (ii) the high resources consumption[46,47,48,49], and (iii) the poor responsiveness[50,51].

Approach

Given an app category C_i for which we observed meaningful correlations between specific kinds of user feedback (*i.e.*, $P_{(I,T)}$ values) and app rating (i.e., the one concerning problem discovery intention category, as emerged from Section 4.1.1), we observe whether some of those kinds of feedback influencing the app rating also exhibit meaningful correlations with any of the code quality metrics collected. In order to perform this task, we rely on Spearman's Rank coefficient ρ to calculate the correlations between the $P_{(I,T)}$ values and the collected *code quality* indicators. Closer is ρ to ± 1, stronger is the relationship between (i) the particular aspect of the code structural quality (*i.e.*, complexity, coupling, cohesion, etc.) encompassed by the metric and (ii) the frequency of certain kinds of feedback[37].

3.2 | Data Extraction and Processing

In this subsection, we report the methodology adopted to (i) collect our dataset, and (ii) extract the metrics (described in Sections 3.1.1 and 3.1.2) employed for carrying out our study. In particular, in order to collect and extract the data, we applied the same approach and methodology used

TABLE 6 Description of the Android-Oriented metrics

Metrics	Description
Bad Smell Method Calls (**BSMC**)	Kechagia and Spinellis individuated 10 Android methods throwing exceptions that can cause app crashes. These methods have to be invoked in a `try- catch` block
WakeLocks with no timeout (**WKL**)	A WakeLock allows to keep the device in an active state, avoiding the switch- off of the display. On a WakeLock object the following methods could be invoked: (i) the `acquire()` method to keep active the display, and (ii) the `release()` method to allow the display switch-off
Number of Location Listeners (**LOCL**)	Through the class `LocationListener` an Android application can keep track of the user's position. However this functionality reduces the battery power
Number of GPS Uses (**GPS**)	Location-aware applications can use GPS to acquire the user location. Although GPS is more accurate, it quickly consumes battery power
XML Parsers (**XML**)	In Android applications, event-based parsers have to be preferred because this kind of parsers can save the battery power
Network Timeouts (**NTO**)	Network timeouts are mechanisms which allow app developers to set time limits for establishing a TCP connection. Without setting a timeout can produce ANR messages
Networking (**NET**)	In Android applications all the networking operations could introduce latency and consequently cause an Application Not Responding(ANR) message
File I/O (**I/O**)	I/O operations could cause ANR messages, since even simple disk operations could exhibit significant and unexpected latencies
SQLite (**SQL**)	The database accesses can generate substantial amount of expensive write operations to the flash storage, with obvious latency implications
Bitmaps (**BMAP**)	Processing of large bitmaps could be computationally expensive and produce ANR messages

in previous work [52]. As shown in Figure 1, such approach comprises two major steps: data collection and analysis. In the following, we detail each of these steps.

For facilitating the replication of our work, we make available a **replication package**[3] including (i) the datasets, (ii) the tool adopted for extracting the data, (iii) the raw data and results of our study.

3.2.1 | Data Collection Phase

Gathering App Metadata. To evaluate user satisfaction before and after releasing a new app version (as recommended by Li *et al.* [53]), we mined subsequent releases of a given app. Specifically, we first developed a web crawler able to mine from the entire F-Droid[4] repository several metadata (e.g. package name, available versions, the release date of each version) along with the correspondent apk for each available version of F-Droid apps. We then discarded the apps (i) not yet available on the Google Play Store, or (ii) whose latest version was released before 2014, with the assumption that apps not updated for such a long time are no longer maintained.

Gathering User Reviews. With the purpose of gathering the reviews related to the collected apps, we built a scraper tool able to automatically download them from the Google Play Store. Such a tool was entirely written in Java, relying on Phantom JS[5] and Selenium[6]. Furthermore, it uses MongoDB as database for the reviews storage. The mining process of our tool can be summarized as follows:

1. the tool reads from an input file the list of the crawled apps, each one identified by the unique package name that is showed in the corresponding Google Play link;

[3] https://github.com/sealuzh/user-satisfaction
[4] https://f-droid.org/
[5] http://phantomjs.org/
[6] http://www.seleniumhq.org/

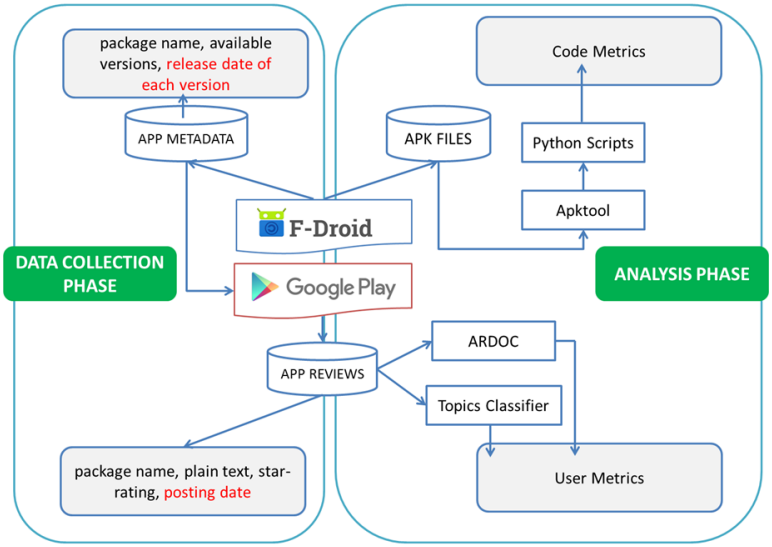

FIGURE 1 Data Collection Approach

2. it queries the existing crawled reviews database, looking for the posting date of the chronologically last mined one for the interested app; if there are no reviews for that specific package name, a default date (*i.e.*, 31 December 2013) is considered;

3. it uses the Phantom JS browser, manipulated through Selenium, to navigate the app's Play Store page, order the available reviews from the newest to the oldest and scraping them from the HTML.

We defined two different stopping criteria for the reviews extraction for a given app: (i) the posting date of the last mined review is older than the latest review already stored in the DB, (ii) a total of 2,000 reviews have been extracted. We had to set this last limit in order to avoid the page overload and the consequential crash. We deployed the crawling tool on a server setting up a *cronjob* that automatically repeats the process 4 times a week.

For each review, we were able to store (i) the package name of the corresponding app, (ii) the review text, (iii) the rating, expressed in the number of stars, provided by the user while posting the review, and (iv) the posting date. Relying on this last information (*i.e.*, posting date) we assign each review to one of the available app versions as described below. Given two consecutive version of a given app, V_i and V_{i+1}, we assigned to the version V_i all the reviews whose posting date occurs after the release date of V_i and before the one of V_{i+1}. We are aware that this assumption may produce for some of the reviews an assignment to a wrong version of the app. However, Pagano and Maalej [13] empirically demonstrated that user feedback is mostly triggered by new releases. This means that users provide most feedback in the first few days after downloading the latest version of an app. As a consequence of this assignment, we discarded all the reviews that were too old to be matched with any of the available versions. Furthermore, in order to ensure a meaningful set of reviews, for each app version, we discarded from the collected data all the versions having less than 30 reviews assigned. Through this process, we collected 104,975 reviews related to 350 versions of 243 apps.

Despite such a remarkable amount of data, we observed that some particular app categories (*e.g.*, *Lifestyle* or *Travel & Local*) were poorly represented (less than 15 apps) in our initial dataset. Therefore, we decided to manually select from Google Play Store a set of popular apps (all appearing in the top 100 popular apps of Google Play) for populating these categories. We used a tool able to download the needed apks from the Google marketplace[7]. Afterwards, we crawled the user reviews related to these new apps. It is worth to notice that we used the same filtering criteria discussed above (*i.e.*, at least 30 reviews for each version). At the end of this new extraction process, a total of 210,517 reviews for 443 versions related to 317 apps have been collected. In particular, 100 apps in our dataset (about 1/3 of the overall dataset) present at least 2 different versions and each of these versions has at least 30 user reviews assigned. The final dataset encompasses at least 15 different apks for 15 distinct Play Store categories.

[7]https://github.com/liato/android-market-api-py

3.2.2 | Analysis Phase

In this phase, we computed the *user-oriented* (described in Section 3.1.1) and the *code quality* (described in Section 3.1.2) metrics for each version of the mined apps. In the following, we briefly explain how we performed the two tasks.

User-oriented metrics computation. All the collected reviews have been labeled using automated classifiers based on machine learning and text-analysis approaches [54,55], that, in previous work, exhibited an accuracy above 80%. To this aim, we first split each review into a set of atomic sentences. Afterwards, we performed a two-dimension classification using as a conceptual framework the URM taxonomy proposed by Di Sorbo *et al.* [8]. In particular, to each resulting sentence has been assigned:

- an *intention* category (detailed in Table 1), by using the ARDOC [54] tool. which is able to automatically detect intentions in app reviews with high accuracy, and
- one (or more) topics (detailed in Table 2), by using the classifier module proposed by the SURF summarizer tool [55], which leverages keywords and n-grams for automatically detecting the topics discussed in app reviews.

Code quality metrics computation. In order to compute the set of code-quality factors we developed several Python scripts. We rely on such quality factors to answer our second research question. In detail, we first disassembled the apks of all the mined versions with at least 30 reviews assigned, with the aim of obtaining a set of human readable Dalvik bytecode `.smali` files from the binary Dalvik bytecode format `.dex` ones. To accomplish this task we used apktool[8], a tool for reverse engineering that allows to decompile and recompile Android applications. The `.smali` files were the targets of our measurements. Thus, for every application version in our dataset, we parsed these files with Python scripts in order to automatically compute the 22 code quality metrics.

4 | RESULTS

In this section, we report the results of our investigation, discussing the main findings for each research question.

4.1 | *RQ_1: To what extent are specific kinds of user feedback connected with app rating?*

In this research question we analyze the types of comments reported in user reviews along with the associated rating. We believe that the combination of star rating and user review analysis can provide useful information for helping developers prioritize the maintenance and evolution tasks. More specifically, we analyze the comments reported in user reviews which may be connected with the decision to rate the app with a lower (or higher) number of stars, with the aim of more in depth understanding mobile users expectations and behaviors.

We apply two different analysis, as detailed in Section 3: a first one observes the distribution of the various kind of user feedback in respectively low and high rated reviews; a second analysis correlates —on both the aggregate and the single categories— the user feedback metrics with the app rating.

4.1.1 | User-oriented metrics distributions

In Table 7 we report the most significant values of the *user oriented metrics* (*i.e.*, $P_{(PD,all)}$, $P_{(FR,all)}$, $P_{(FR,Feature/functionality)}$, $P_{(FR,Feature/functionality)}$) obtained for both the reviews with low rating (R_{low}) and the reviews with high rating (R_{high}). Specifically, such values are shown for the overall dataset (*i.e.*, by considering all the apps together, *All*), as well as for the different Play Store categories (on the rows). It is worth noticing that Table 7 only reports the values obtained for the most recurring user oriented metrics. For complete results, please refer to our replication package[9].

In general, when looking at the whole app dataset (*i.e.*, All), we observe a high percentage (40.2%) of sentences classified as *Problem Discovery* (*PD*) contained in reviews with low rating (*i.e.*, 1-2 stars). On the contrary, we find that in reviews with high rating (*i.e.*, 4-5 stars) only the 6.2% of sentences are recognized as *PD*. Vice versa, the percentage of *Feature Requests* (*FR*) discovered in reviews with low rating is very similar (i.e., around 6%) to the one observed in comments with high rating. Specifically, results show that problem discoveries dealing with the *Feature/Functionality* (*PD+Feature/Functionality*) topic are the most frequent (*i.e.*, 34%) complaints contained in reviews with low rating, while requests or problems dealing with a specific functionality (*FR+Feature/Functionality* and *PD+Feature/Functionality*) are the user-oriented metrics with the highest percentages (*i.e.*, 5.5% and 5.1%, respectively) found in reviews with high rating.

[8] http://ibotpeaches.github.io/Apktool/
[9] https://github.com/sealuzh/user-satisfaction/blob/master/rqs/topic.md

TABLE 7 Percentages of user-oriented metrics in reviews with low and high rating

Category	PD		FR		PD+Feature		FR+Feature	
	R_{low}	R_{high}	R_{low}	R_{high}	R_{low}	R_{high}	R_{low}	R_{high}
Books & Reference	0.397	0.063	0.067	0.090	0.342	0.052	0.058	0.082
Communication	0.499	0.088	0.067	0.071	0.424	0.073	0.060	0.065
Education	0.173	0.027	0.064	0.036	0.138	0.019	0.055	0.031
Entertainment	0.321	0.046	0.057	0.023	0.273	0.036	0.051	0.020
Finance	0.411	0.034	0.087	0.046	0.324	0.028	0.080	0.043
Games	0.409	0.118	0.062	0.084	0.331	0.093	0.054	0.075
Lifestyle	0.409	0.050	0.054	0.031	0.348	0.040	0.048	0.026
Music & Audio	0.513	**0.138**	**0.124**	**0.167**	0.447	**0.119**	**0.113**	**0.160**
Personalization	**0.526**	0.119	0.076	0.140	**0.457**	0.105	0.073	0.130
Photography	0.271	0.044	0.051	0.043	0.222	0.036	0.047	0.040
Productivity	0.494	0.081	0.108	0.140	0.434	0.069	0.094	0.130
Social	0.474	0.062	0.058	0.043	0.398	0.050	0.055	0.036
Tools	0.417	0.067	0.072	0.076	0.361	0.057	0.066	0.070
Travel & Local	0.274	0.027	0.060	0.026	0.218	0.022	0.052	0.024
Video Players & Editors	0.463	0.072	0.037	0.077	0.413	0.059	0.034	0.071
All	0.402	0.062	0.063	0.060	0.340	0.051	0.057	0.055

Considering the results achieved by the different app categories, we can notice that bug descriptions (*PD*) exhibit a massive presence in reviews with low rating. Contrariwise, we have a higher presence of feature requests (*FR*) within reviews with high rating than reviews with low scores, for 8 out of 15 analyzed categories (*i.e.*, *Books & Reference*, *Communication*, *Games*, *Music & Audio*, *Personalization*, *Productivity*, *Tools* and *Video Players & Editors*). Moreover, for all the analyzed categories, *PD+Feature/Functionality* is the most frequent user-oriented metrics observed in reviews with low rating.

Looking at feature requests (*FR*) raised in reviews with high rating, we can notice that these types of reviews may be useful for collecting ideas and suggestions in order to make changes that can lead to higher user rating, for example:

"The only reason I'm not giving it a full five stars is that I wish it gave me the option to share a good book with friends by being able to send information about the book to my friends via some form of media" - App: *Smashwords Access*, Rating: *4*.

'I would give 5 stars if there was a way to move emails from the delete folder back into the inbox folder" - App: *K-9 Mail*, Rating: *4*.

This confirms our conjecture that combining app rating and user review analysis can be very useful for mobile developers to stay ahead of their competitors and respond to the needs of their users[5,33,8,6]. In Table 8 we report some examples of requests of features (*i.e.*, FR) stated in reviews with high and low ratings for the app *GPS Logger for Android*. Feature requests raised in reviews with high rating (*i.e.*, 4-5 stars) appear more specific and less vague than feature requests stated in reviews with low rating (*i.e.*, 1-2 stars). As a matter of fact, some of the suggestions contained in reviews with high rating have been (i) effectively considered for implementation (*e.g.*, send the location automatically via sms[10]), (ii) actually implemented (*e.g.*, POST method when logging via URL[11]), or (iii) enabled through workarounds (*e.g.*, SSH upload[12]).

[10]https://github.com/mendhak/gpslogger/issues/526
[11]https://github.com/mendhak/gpslogger/pull/574
[12]https://github.com/mendhak/gpslogger/issues/523

TABLE 8 Examples of feature requests raised in reviews with low and high rating for the app GPS Logger for Android

Low-Rated Reviews	High-Rated Reviews
Loaded it at the hotel and it worked thinking it would work great on the road left for our tour when out of the country and needed it and did not work because it keep looking for an internet connection	Good job Would be nice to have also acceleration
What I think should be changed is that when logging it started it should not start until GPS gets a lock.	So there should be an option to sent the location automatically via sms to a predefined mobile number at predefined time intervals.
Unless you have it turned up crazy high (uploading every minute) you will miss data	Would have been better if the path could be seen live on a map.
The settings warn against setting the log filename to a frequent interval but event with the defaults you will miss 1 hour of stats every day.	Great app I would really like to be able to autosend co-ords via sms preferably with customizable text
I need to export them to a PC and cannot.	Perfect gps weird sharing Maybe it'a android lacking the right activities but this can't use the same apps you get for ""sharing""
It would be nice if the app could reactivate when it turns back on again?	Please add an option to turn off notifications
	Ui need to be improved
	And could you add SSH upload?
	will be good to have POST method when logging via URL
	Wish it wold stack data in memory when call is on and later push data on network

4.1.2 | User-oriented metrics and relationships with app rating

Our analysis is aimed at identifying the key user review feedback that exhibits significant relationships with rating (a proxy of the success of apps) for the different app categories. Specifically, to study the mutual connections between *specific kinds of user feedback* and star rating, from a more rigorous statistical perspective, we (i) compute the correlations between the defined user-oriented metrics (see Section 3.1.1) and app rating, and (ii) illustrate, through heat-map representations, the obtained results. Interestingly, our previous results demonstrated that FR-related metrics exhibit similar distributions both in reviews with high rating and in reviews with low rating. Thus, we decided to conduct further analysis by exclusively focusing on PD-related metrics, as we argue that reviews of these types may exhibit stronger relationships with app rating.

Figure 2 shows the correlations occurring between the *percentage of problem discoveries (PD)*, along with the percentages obtained for each *PD-topic* pair, and the average rating. In Figure 2, we annotate only the cells (*i.e.*, each cell represents the relation between a given metric and the rating) with at least a *moderate* strength (i.e. $|\rho| \geq 0.39$) and a significant or marginally significant (*i.e.*, marked with the * symbol) p-value. The outcomes are presented for both the entire dataset of apps (*i.e.*, the *All* column) and each app category. As illustrated in Figure 2, percentages of *PD-topic* pairs frequently (*i.e.*, in many app categories) report a statistically significant relationship with app rating.

When looking at the whole app dataset (*i.e.*, *All*), we have a *moderate* negative correlation between the average rating and the percentage of reviews that claim a problem discovery (*i.e.*, the *PD* row). Specifically, results show that *problem discoveries* dealing with the *App* and *Feature/Functionality* topics may widely affect in a negative way the app rating — confirming the finding we observe in Section 4.1.1. This means that developers of all kinds of apps should care of comments reporting *App* and *Feature/Functionality* related problems, since a quick resolution of them can lead to an improvement of overall user satisfaction and a higher app rating. However, by specializing the analysis trough the different app categories,

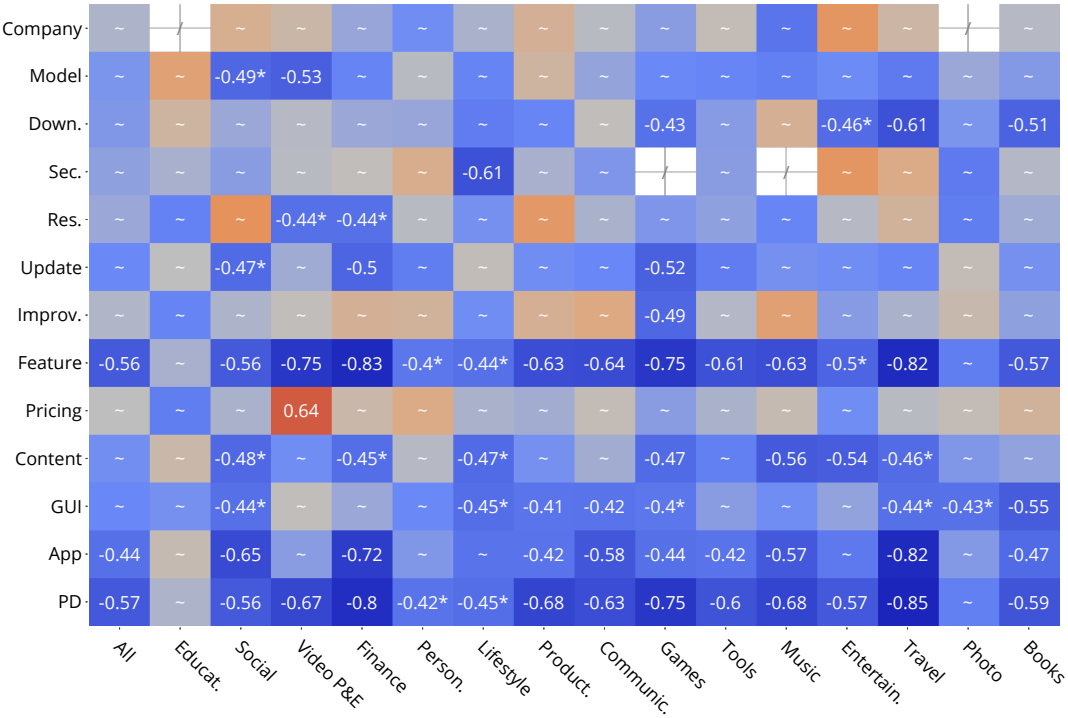

FIGURE 2 Correlations between Problem Discovery's topics and rating

we can notice that the correlations between different kinds of feedback and the average rating significantly differ among the various Play Store's categories.

Almost all the categories (except for *Education*, and *Photography*) report meaningfully correlations between the *PD* factor and the rating (*i.e.*, 8 out of 15 categories present *strong* or *very strong* negative correlations). More in-depth, problem discoveries dealing with the *Feature/Functionality* topic (*PD+Feature/Functionality*) exhibit strong negative correlations with average rating for apps belonging to several categories (*i.e.*, 2 *very strong* and 5 *strong* negative correlations). Moreover, noticeably ρ values can also be observed (i) between the pair *PD + App* and the average rating (*i.e.*, 6 *moderate*, 2 *strong* and 1 *very strong* negative correlations), and (ii) between the *PD + GUI* pair and the rating for the *Social*, *Lifestyle*, *Productivity*, *Communication*, *Games*, *Travel & Local*, *Photography* and *Books & Reference* categories (*i.e.*, 8 *moderate* negative correlations, but only 3 of them are fully significant from a statistical point of view). Furthermore, for some categories (*i.e.*, *Social*, *Finance*, *Lifestyle*, *Games*, *Music & Audio*, *Entertainment* and *Travel & Local*) the users are particularly sensitive about the *Contents* topic. Indeed, for these categories, the *PD + Contents* factor *moderately* binds with the rating in a negative way, but for only 3 of them the correlations are fully significant. As a practical example, according to our results, developers of apps belonging to the *Communication* category should pose particular attention on issues dealing with the *App*, *GUI* and *Feature/Functionality* topics. For instance, for the apps *Fon for Members* and *Yaaic* we observe a higher presence of bugs dealing with the *App* and Feature/Functionality topics. Indeed, these apps exhibit values for the metrics $P_{(PD,App)}$ (0.12 and 0.14, respectively), and $P_{(PD,Feature/Functionality)}$ (0.41 and 0.39, respectively) that are higher than the median values (*i.e.*, 0.07 for $P_{(PD,App)}$ and 0.21 for $P_{(PD,Feature/Functionality)}$) computed on the same metrics and considering all the apps of the *Communication* category present in our dataset. Interestingly, the average rating values of both apps (*i.e.*, 2.12 and 3.04, respectively) are strongly lower than the median average rating (3.82) obtained by apps in our dataset belonging to the *Communication* category, suggesting a lower level of user satisfaction. Looking deeper at the reviews with low rating related to *Fon for Members* and *Yaaic* apps that are marked as *Problem Discovery* and discussing the *App* and *Feature/Functionality* topics, as shown in Table 9, we can notice that while the users of the *Fon for Members* app mostly complain about *login* and *purchase* functionalities, the users of the *Yaaic* app recurrently complain about the *IRC commands* and *SSL identification* functionalities.

These results suggest that it is of primary importance to ensure the correct behavior of the apps' existing functionalities. As a consequence, developers should first look at (and timely address) the kinds of bugs exhibiting stronger relationships with star rating, in order to acquire new users without losing market slices.

In conclusion, from this analysis we can observe that users experiencing functional bugs are more likely to rate the whole app with a lower score, while users requiring features implementation/enhancement present less predictable behaviors in assigning star rating: they can express a

TABLE 9 Examples of problems discussed in reviews with low rating for the apps Fon for Members and Yaaic

Fon for Members	Yaaic
Unable to login with my FON enabled Mweb account.	*basic IRC commands don't even work!*
Unable to login with Netia account. Appending #netia doesn't help.	*Crashes using /list.*
The login does not work..	*Can't even /list on many servers.*
Cannot log in. Username and password invalid.	*Channels bug /part /close or close button does not work.*
Cant log in to use credit..	*Missing some key things: SSL identification doesn't work.*
Hmm It gives an option to purchase credit but this feature doesn't work.	*Other users are saying this is an SSL bug which is possible (all my servers use SSL).*
I can't purchase credits If I attempt to try and purchase credits to use the FON networks my options are disabled and I can't use play store or pay pal.	*I'm not sure if this is just an issue with SSL servers only as I only use the app for one server.*

high level of satisfaction with the app (4 & 5 stars) and, in the same time, stimulate its evolution by asking developers for enhancements that could satisfy their needs.

In order to verify whether apps with more than one release could affect our results, we also conducted the analysis by considering only the latest version (available in our dataset) of each of the 317 apps. Such analysis produced similar results to the ones obtained when considering all the available versions. This suggests that problems or requests dealing with specific topics have different relations with app rating and these relations are mainly influenced by the app categories. Moreover, for specific app categories, some types of comments positively/negatively correlate with the rating, even though they do not cover a high percentage in reviews with high/low ratings.

RQ*$_1$ *summary. Comments belonging to the *Problem Discovery* category are strongly connected with the rating of almost all apps. However, the relations between the different kinds of issues and the app rating significantly differ among the various Play Store's categories. Consequently, developers should timely address the kinds of bugs exhibiting stronger relationships with app rating, in order to acquire new users without losing market slices. On the contrary, feature-request metrics are rarely connected with the rating

4.2 | *RQ*$_2$*: Are there code quality indicators correlated with the rising of critical issues?*

With this research question, our aim is to complement the analysis performed in **RQ**$_1$. Since in **RQ**$_1$ we found that app rating degradation might be related with the rising of some *user-oriented metrics* (*e.g.*, PD), the goal of **RQ**$_2$ is to better understand whether *code quality indicators* (*i.e.*, that developers can directly monitor) exhibiting meaningful relationships with these *user-oriented metrics* could be identified in order to prevent such growth.

In order to perform this analysis, we correlate the *code quality* indicators (see Section 3.1.2) and the *user-oriented* metrics (detailed in Section 3.1.1). We use the Spearman's Rank coefficient and interpret it as described in Section 3.1.1.

The first immediate result we can observe is that, when looking at the overall dataset —*i.e.*, all the apps together— there is not even a singular correlation that is statistically significant ($p \leq 0.05$). Therefore, we can conclude that, in general, the frequency of user reviews reporting a *Problem Discovery* is not correlated with code quality aspects. This result highlights that users are not able to directly perceive the internal quality of the applications they are using and therefore the content of their comments does not vary accordingly. This is a result aligned with previous findings from the literature [22].

Similarly to the analysis carried out for **RQ**$_1$, we deepen our investigation by analyzing each Google Play Store category separately, looking for cases in which the specific nature of the apps could lead to peculiar results. In particular, we specify the analysis to the categories reporting meaningful correlations between user feedback (*i.e.*, *PD-oriented* metrics) and app rating (see Section 4.1). For the sake of space, we report all the

corresponding heat-maps in our online appendix[13], limiting this section to the discussion of most interesting results. Again, we are not able to find any strong pattern in our correlations; however, it is worth to describe some interesting arose results. For instance, in the *Communication* category, we observe a *moderate* negative correlation between the NTO (*Network Timeouts*) metric and the percentage of *PD*, specifically with the PD regarding the *Feature/Functionality* topic. This means that for this specific category, when the amount of *Network Timeouts* decreases, the frequency of reviews reporting *Problem Discoveries* about the *Feature/Functionality* topic increases. Moreover, for the apps of the same category (*i.e.*, *Communication*), as reported in Section 4.1, when the percentage of reviews of the *PD+Feature/Functionality* pair is higher the average rating decreases. This result could be due to the specific characteristics exhibited by apps in this category. Indeed, the *Communication* category encompasses messaging, chat and call management apps which make a strong use of networking capabilities. Consequently, developers of apps belonging to the *Communication* category should pose particular attention on network timeouts management, as an inadequate amount of network timeouts could degrade the user experience (*e.g.*, the app is always in fetching mode with consequences on the power consumption) and lead to a lower app rating. An example of review related to an app of the *Communication* category and complaining about this particular aspect is illustrated below.

"..But few things needs to be fixed: timeout when fetching emails sometimes it keeps fetching" - App: *K-9 Mail*, Rating: *2*.

Another interesting result is observed for the category *Music & Audio*. Here, high values of NTO (*Network Timeouts*) and NET (*Networking*) co-occur (exhibiting *moderate* ρ values) with a higher percentage of reviews belonging to the *PD* category and treating the *App* topic. Again, we argue that this could be related to the key features that applications in this category provide. Indeed, apps in thr category *Music & Audio* usually get and reproduce contents in streaming. An adequate use of networking operations and timeouts in applications belonging to the *Music & Audio* category could lead to a higher user satisfaction, since a higher frequency of reviews of the *PD+App* type might negatively affect the rating (moderate negative correlation, as showed in Section 4.1). Below, some examples of user reviews complaining about the apps belonging to the *Music & Audio* category are reported.

"Uses 30-35% of my battery even if I don't open the app at all." - App: *Subsonic Music Streamer*, Rating: *2*.

"... occasional connection problems are sometimes tedious to resolve..." - App: *DSub for Subsonic*, Rating: *3*.

"... I often find that in the middle of paging through music the connection to the server is lost and I have to reconnect and start from the beginning again. After 4 or 5 times it can get pretty frustrating... " - App: *Squeezer*, Rating: *3*.

Thus, it is convenient that developers of apps belonging to the *Music & Audio* category carefully monitor the NTO and NET metrics, as fluctuations in these metrics, could lead to higher numbers of user complaints, and, consequently, fluctuations in app rating. It is also worth highlighting how users' perception about timeouts is different for apps belonging to *Communication* and *Music & Audio* categories. Indeed, while for the *Communication* category an insufficient amount of timeouts may degrade the user satisfaction, for the apps in the *Music & Audio* category excessive amounts of timeouts could lead to a lower app rating. As a matter of fact, in Communication apps, timeouts are very relevant for users and an ineffective use of timeouts could have negative effects on user experience, as they (i) could have impact on power consumption[14], (ii) make it appear that the app has frozen[15], or (iii) may cause synchronization problems[16]. On the other hand, in apps belonging to the Music & Audio category, timeouts could cause problem in downloading data[17], and thus timeouts should be appropriately managed[18] .

This suggests that also code quality standards should vary according to app categories, in order to more precisely reflect user expectations.

Our results demonstrate that an investigation merely based on the app rating can be fruitfully enriched by a context-based analysis (*i.e.*, taking into account the specific category of the app) in order to better understand user needs and expectations . In particular, our investigation demonstrated that for apps belonging to specific categories some software quality indicators are connected with the raising of certain types of issues, that in turn exhibit negative relationships with app rating. Thus, such code metrics should be carefully monitored by developers with the aim of preventing the raising of such kinds of issues.

RQ*$_2$ *Summary. In specific app categories *inadequate* values of some Android-related metrics might lead to higher amounts of user complaints, and, consequently, to a lower app rating. Thus, developers of apps belonging to these categories could monitor these metrics to prevent the raising of such kinds of issues.

[13] https://github.com/sealuzh/user-satisfaction/blob/master/rqs/cat.md
[14] https://github.com/k9mail/k-9/issues/1066
[15] https://github.com/k9mail/k-9/issues/2652
[16] https://github.com/k9mail/k-9/issues/1267
[17] https://github.com/clementine-player/Android-Remote/issues/13
[18] https://github.com/segler-alex/RadioDroid/commit/0222904

5 | THREATS TO VALIDITY

This section discusses the main threats that could influence our study.

5.1 | Threats to Internal Validity

Threats to internal validity concern any confounding factor that could influence our results. It is worth to notice that this is only an observational study. We rely on the quantitative and qualitative information collected to answer the various research questions and try to provide plausible explanations for the obtained results. However, relationships between specific *changes* occurred on *code quality metrics*, *rating* (*i.e.*, the main proxy used in literature to measure the success) of apps, and higher amounts of specific *types of user feedback* in reviews, could be due to several other internal (*e.g.*, changes in the app features and usability) and/or external (*e.g.*, availability of alternative similar apps) factors, which have not been considered in our study. This could represent a threat to internal validity. To mitigate this issue and verify the reliability of obtained results, we plan in the future to consider a wider set of internal/external factors that could influence user decisions to rate an app with lower or higher amounts of stars.

A further threat to internal validity could involve the criterion we used for assigning reviews to the different versions of an app. In particular, we rely on the publication date of the user review for assigning it to one of the app releases. This criterion may generate wrong assignments. However, user feedback is mostly triggered by new releases and users give most feedback in the first few days after a new version of the app is released [13].

Another threat may concern the way we classify reviews. Specifically, we use automated classifiers in order to assign each review to an intention and one or more topics categories. This could lead to misclassifications that might affect our results. In particular, the intention classifier we used demonstrated to achieve an accuracy above the 80% when used in a real scenario [54], while the topics classifier achieved an accuracy of about 75% [8]. It is worth to point out that, in order to mitigate the effects related to information bias, we filtered out (i) apks having less than 30 reviews, and (ii) app categories encompassing less than 15 different apks. While the former criterion is aimed at augmenting the variability of topics discussed in user reviews related to a specific apk, the latter has the purpose of avoiding to polarize the analysis around too specific apps.

5.2 | Threats to External Validity

Threats to external validity are related to the generalizability of our findings. The work by Martin *et al.* [56] highlights that mobile data are susceptible to the app sampling problem. This means that results only based on a subset of apps may be affect by potential sampling bias. Thus, our results may not be generalized to all apps on the Android platform. A potential limitation of our study could also be represented by the fact that each apk has a different number of related comments (*i.e.*, there are several apps that are less popular than others). However, to mitigate these issues, our dataset is composed by a representative number of app belonging to the most relevant Play Store categories and each considered apk (*i.e.*, version) in our study has at least 30 reviews assigned.

Another problem affecting our research is that we analyzed only free apps. However, it is not the purpose of this paper establishing if our observations are still valid for paid apps.

As reported by Martin *et al.* [56], in the Google Play store the full set of reviews is available only for apps with fewer than 481 reviews under each star rating (2,400 total). This could be a threat to external validity, due to the fact that we only considered the reviews that the scraping tool was able to retrieve. Nevertheless, to avoid missing a high number of reviews, we set a cronjob which repeated the reviews mining process 4 times a week (see Section 3.2). Furthermore, for each apk (*i.e.*,, each app version), all the *user-oriented* metrics, as well as the average rating value used in our analysis are computed relying exclusively on the reviews we were able to gather for such apk (as stated in Sections 3.1.1 and 3.1.2).

6 | CONCLUSIONS

In recent years, researchers proposed approaches based on app rating to measure user satisfaction, thus, correlating app rating with specific software evolution and quality indicators. This paper reports an empirical study aimed at investigating which types of feedback are directly related to the app rating. More specifically, we conducted an empirical study involving 210,517 reviews, 443 versions related to 317 Android apps, and investigated whether (i) specific types of user feedback are associated with reviews with high/low rates; and (ii) specific kind of code quality indicators correlate with the increase of user reviews mainly reporting bugs. While in general users do not directly perceive the internal quality of applications they use, our results show that some code metrics exhibit meaningful correlation with the rating, and this strictly depends on the type of the considered apps.

Findings and practical implications for researchers and developers. In the following we summarize our main findings and the practical implications relevant for developers and the research community:

- **On the impact of bugs and feature requests on the app rating.** User comments belonging to the *Problem Discovery* category are negatively connected with the rating of almost all apps, which implies that users experiencing bugs are more likely to rate the app with a lower score. Surprisingly, no noticeable relationships between the reviews asking for new features — or feature enhancement — and the app rating are observed. For those reasons, future strategies, aimed at achieving a higher user satisfaction, should first prioritize user feedback concerning bug fixing activities, while giving lower priorities to other software evolution activities.

- **App category matters.** Depending on the app category, different kinds of user feedback exhibit stronger/weaker relationships with the app rating. Indeed, as observed in Section 4.1.2, for specific app categories (*e.g.*, Communication), a lower user satisfaction is strongly connected with different kinds of user feedback (*e.g.*, PD+App and PD+Feature/Functionality), and, in this context, the analysis of such user feedback types allows to identify the specific mulfunctions that are recurrently experienced by users (*e.g.*, the *login* and purchase features), and that lead to lower ratings. As consequence, better prioritization strategies should consider the app category as a criterion to design rewarding/penalizing mechanisms able to rank the most relevant user feedback types that exhibit the stronger relationships with the app rating. More in general, future research on the evolution of mobile apps should focus on leveraging both app rating and user reviews analysis to conceive automated approaches able to identify the specific issues that developers should pose particular attention on, depending on the category of the app.

- **On the relevance of code quality on user feedback and app rating.** We observed that types of issues co-occuring with lower app rating are connected with specific code quality indicators (*e.g.*, NTO). Interestingly, the strength of this relationships strictly depends on the app category. This suggests that the development and evolution of apps of specific categories should integrate mechanisms that monitor the source code quality indicators associated with the raising of critical issues.

- **Code quality perception implications.** User perception of code quality varies depending on the app (categories) they are using. As a concrete example, an higher amount of timeouts is perceived (i) more positively by users of the *Communication* category, and (ii) more negatively by users of the *Music & Audio* category. This suggests, for future research on mobile apps evolution, that code quality standards should be adapted considering the different app categories, in order to more precisely reflect user expectations.

From a practical perspective, we can envision two different *development scenarios* where our results could be adopted:

- **User issue criticality for developer-centric app marketplaces**. App marketplaces do not provide information about the general quality levels of the available apps [57]. Software quality strongly depends on the ability to collect feedback from end-users efficiently and dynamically [58]. To support app developers in performing more informed software maintenance and evolution activities, we argue that app marketplaces can provide mechanisms aimed at fostering the perceived quality of the apps hosted. As proposed in our work, such mechanisms could leverage text- and data-mining techniques for profitably linking the feedback types, the user ratings, and the code quality metrics, over each app category. In a similar context, to plan and prioritize the changes to apply, the developers of an app could easily retrieve the specific feedback types that exhibit the stronger relations with user-perceived quality (according to the app category) and explore the recurrent topics in such reviews (as exemplified in Section 4.1.2).

- **Criticality-driven issue prioritization in RE sessions**. Requirements prioritization is an important activity of the requirement engineering (RE) process [59]. In RE sessions, developers are tasked to determine and prioritize the requirements of the software system, trying to integrate end-users needs [58]. Our results can be used to support collaborative decision-making processes in requirements prioritization [60]. In particular, information about the criticality of user-reported issues can provide a better picture of the specific needs of the users of a particular app category, This information can be exploited during the requirements elicitation and prioritization phases for ensuring a higher perceived quality from the early stages of software development.

Potential future work. Our study demonstrates that exploiting the huge amount of knowledge that can be collected through user reviews allows to enrich the typical analysis merely based on app score with a context-based one. Thus, combining user rating with user review analysis might be very useful to determine which kind of feedback address at first in order to maximize user satisfaction (*i.e.*, the associated rating [61]) as well as the source code factors to keep under control to avoid a degradation in such a satisfaction. Tools and techniques can be built on top of our results to help developers to monitor those factors and achieve higher success for their apps. As future work, we also plan to use the data available for different version of the apps in our dataset, to more in-depth investigate, from a more evolutionary perspective, if best practices can

be identified in order to suggest the specific actions that developers should undertake to obtain variations in ratings. Moreover, previous research demonstrated that information sources different from app stores (*e.g.*, Twitter) provide relevant feedback for app developers [62,63]. In the future, we aim at investigating the extent to which the feedback gathered from these information sources can be used to more precisely estimate the criticality of specific issues. We believe that this work might help to develop tools able to (i) guide developers distilling both the user feedback and quality indicators that are related to the app rating and success, and, consequently, (ii) help developers assigning the right priorities to the different software maintenance and evolution activities.

ACKNOWLEDGMENTS

We gratefully thank dr. Francesco Mercaldo for his help during the data extraction and collection steps of our study.

References

1. Online . *VisionMobile. The new mobile app economy.* 2015.
2. Online . *App Annie Reveals Future of the App Economy: $101 Billion by 2020; China to Surpass U.S. This Year.* 2016.
3. Online . Android (operating system). *Wikipedia - Android (operating system).* 2019;.
4. Martin W., Sarro F., Jia Y., Zhang Y., Harman M.. A Survey of App Store Analysis for Software Engineering. *IEEE Transactions on Software Engineering.* 2016;PP(99):1-1.
5. Tian Yuan, Nagappan Meiyappan, Lo David, Hassan Ahmed E.. What Are the Characteristics of High-rated Apps? A Case Study on Free Android Applications. In: ICSME '15:301–310; 2015; Washington, DC, USA.
6. Ciurumelea Adelina, Schaufelbuhl Andreas, Panichella Sebastiano, Gall Harald C.. Analyzing reviews and code of mobile apps for better release planning. In: SANER '17:91–102; 2017.
7. Online . *How Facebook Is Fuelling the Growth of the Super Start-Up.* 2015.
8. Di Sorbo A., Panichella S., Alexandru C., et al. What Would Users Change in My App? Summarizing App Reviews for Recommending Software Changes. In: FSE 2016:499-510; 2016.
9. Seyff Norbert, Stade Melanie J. C., Fotrousi Farnaz, et al. End-user Driven Feedback Prioritization. In: REFSQ '17; 2017.
10. Bavota G., Linares-Vasquez M., Bernal-Cardenas C.E., Di Penta M., Oliveto R., Poshyvanyk D.. The Impact of API Change- and Fault-Proneness on the User Ratings of Android Apps. *Software Engineering, IEEE Transactions on.* 2015;41(4):384-407.
11. Noei Ehsan, Costa Daniel Alencar, Zou Ying. Winning the app production rally. In: FSE 2018:283–294; 2018.
12. Khalid Hammad, Shihab Emad, Nagappan Meiyappan, Hassan Ahmed E.. What Do Mobile App Users Complain About?. *IEEE Software.* 2015;32(3):70–77.
13. Pagano Dennis, Maalej Walid. User feedback in the appstore: An empirical study.. In: RE 2013; 2013.
14. Noei E., Zhang F., Zou Y.. Too Many User-Reviews, What Should App Developers Look at First?. *IEEE Transactions on Software Engineering.* 2019;:1-1.
15. Noei Ehsan, Zhang Feng, Wang Shaohua, Zou Ying. Towards prioritizing user-related issue reports of mobile applications. *Empirical Software Engineering.* 2019;.
16. Palomba Fabio, Vásquez Mario Linares, Bavota Gabriele, et al. Crowdsourcing user reviews to support the evolution of mobile apps. *Journal of Systems and Software.* 2018;137:143–162.
17. Gao Cuiyun, Zeng Jichuan, Lyu Michael R., King Irwin. Online App Review Analysis for Identifying Emerging Issues. In: ICSE '18:48–58; 2018; New York, NY, USA.

18. Guzman E., Maalej W.. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In: RE 2014:153-162; 2014.

19. Martin William, Sarro Federica, Harman Mark. Causal Impact Analysis for App Releases in Google Play. In: FSE 2016:435–446; 2016; New York, NY, USA.

20. Chidamber S. R., Kemerer C. F.. A Metrics Suite for Object Oriented Design. *IEEE Trans. Softw. Eng.*. 1994;20(6):476–493.

21. Fuggetta Alfonso, Di Nitto Elisabetta. Software Process. In: FOSE 2014:1–12; 2014; New York, NY, USA.

22. Corral Luis, Fronza Ilenia. Better Code for Better Apps: A Study on Source Code Quality and Market Success of Android Applications. In: MOBILESoft '15:22–32; 2015; Piscataway, NJ, USA.

23. Linares-Vásquez Mario, Bavota Gabriele, Bernal-Cárdenas Carlos, Di Penta Massimiliano, Oliveto Rocco, Poshyvanyk Denys. API Change and Fault Proneness: A Threat to the Success of Android Apps. In: ESEC/FSE 2013:477–487; 2013; New York, NY, USA.

24. Taba Seyyed Ehsan Salamati, Keivanloo Iman, Zou Ying, Ng Joanna, Ng Tinny. An Exploratory Study on the Relation between User Interface Complexity and the Perceived Quality:370–379. ICWE 20142014.

25. Guerrouj Latifa, Azad Shams, Rigby Peter C.. The influence of App churn on App success and StackOverflow discussions. In: SANER '15:321–330; 2015.

26. Ruiz I Mojica, Nagappan Meiyappan, Adams Bram, Berger Thorsten, Dienst Steffen, Hassan Ahmed. On the relationship between the number of ad libraries in an android app and its rating. *IEEE Software.* 2014;99(1).

27. Harman Mark, Jia Yue, Zhang Yuanyuan. App store mining and analysis: MSR for app stores. In: MSR '12:108–111; 2012.

28. Iacob Claudia, Veerappa Varsha, Harrison Rachel. What Are You Complaining About?: A Study of Online Reviews of Mobile Applications. In: BCS-HCI '13:29:1–29:6; 2013; Swinton, UK, UK.

29. Luiz Washington, Viegas Felipe, Alencar Rafael, et al. A Feature-Oriented Sentiment Rating for Mobile App Reviews. In: WWW '18:1909–1918; 2018; Republic and Canton of Geneva, Switzerland.

30. Hu Hanyang, Wang Shaowei, Bezemer Cor-Paul, Hassan Ahmed E.. Studying the consistency of star ratings and reviews of popular free hybrid Android and iOS apps. *Empirical Software Engineering.* 2018;.

31. Gui Jiaping, Nagappan Meiyappan, Halfond William G. J.. What Aspects of Mobile Ads Do Users Care About? An Empirical Study of Mobile In-app Ad Reviews. *CoRR.* 2017;abs/1702.07681.

32. Chen Ning, Lin Jialiu, Hoi Steven C. H., Xiao Xiaokui, Zhang Boshen. AR-Miner: Mining Informative Reviews for Developers from Mobile App Marketplace. In: ICSE 2014:767–778; 2014; New York, NY, USA.

33. Panichella S., Di Sorbo A., Guzman E., Visaggio C.A., Canfora G., Gall H.C.. How can I improve my app? Classifying user reviews for software maintenance and evolution. In: ICSME 2015:281-290; 2015.

34. Guzman Emitza, El-Haliby Muhammad, Bruegge Bernd. Ensemble Methods for App Review Classification: An Approach for Software Evolution (N). In: ASE 2015:771–776; 2015.

35. Palomba Fabio, Salza Pasquale, Ciurumelea Adelina, et al. Recommending and localizing change requests for mobile apps based on user reviews. In: ICSE 2017:106–117; 2017.

36. Ruiz I. Mojica, Nagappan M., Adams B., Berger T., Dienst S., Hassan A.. An Examination of the Current Rating System used in Mobile App Stores. *IEEE Software.* 2017;PP(99):1-1.

37. Daniel Wayne W. Spearman rank correlation coefficient. *Applied nonparametric statistics, 2nd ed. PWS-Kent, Boston.* 1990;:358–365.

38. Kechagia Maria, Spinellis Diomidis. Undocumented and Unchecked: Exceptions That Spell Trouble. In: MSR 2014:312–315; 2014; New York, NY, USA.

39. Yang Shengqian, Yan Dacong, Rountev A.. Testing for poor responsiveness in android applications. In: MOBS 2013:1-6; 2013.

40. Ickin S., Wac K., Fiedler M., Janowski L., Hong J. H., Dey A. K.. Factors influencing quality of experience of commonly used mobile applications. *IEEE Communications Magazine*. 2012;50(4):48-56.

41. Sharkley Jeff. *Coding for Life - Battery Life, That Is.* 2009.

42. Mercaldo Francesco, Sorbo Andrea Di, Visaggio Corrado Aaron, Cimitile Aniello, Martinelli Fabio. An exploratory study on the evolution of Android malware quality. *Journal of Software: Evolution and Process*. 2018;30(11).

43. Zimmermann Thomas, Premraj Rahul, Zeller Andreas. Predicting Defects for Eclipse. In: PROMISE '07:9–IEEE Computer Society; 2007; Washington, DC, USA.

44. Marcus Andrian, Poshyvanyk Denys, Ferenc Rudolf. Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems. *IEEE Trans. Software Eng.*. 2008;34(2):287–300.

45. D'Ambros Marco, Lanza Michele, Robbes Romain. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*. 2012;17(4):531–577.

46. Hao Shuai, Li Ding, Halfond William G. J., Govindan Ramesh. Estimating Mobile Application Energy Consumption Using Program Analysis. In: ICSE '13:92–101; 2013; Piscataway, NJ, USA.

47. Kim Sang-Hoon, Kwon Sejun, Kim Jin-Soo, Jeong Jinkyu. Controlling Physical Memory Fragmentation in Mobile Systems. *SIGPLAN Not.*. 2015;50(11):1–14.

48. Carroll Aaron, Heiser Gernot. An Analysis of Power Consumption in a Smartphone. In: USENIXATC'10:21–21; 2010; Berkeley, CA, USA.

49. Vásquez Mario Linares, Bavota Gabriele, Bernal-Cárdenas Carlos, Oliveto Rocco, Penta Massimiliano Di, Poshyvanyk Denys. Mining energy-greedy API usage patterns in Android apps: an empirical study. In: MSR '14:2–11; 2014.

50. Ongkosit Thanaporn, Takada Shingo. Responsiveness Analysis Tool for Android Application. In: DeMobile 2014:1–4; 2014; New York, NY, USA.

51. Kang Yu, Zhou Yangfan, Gao Min, Sun Yixia, Lyu Michael R.. Experience Report: Detecting Poor-Responsive UI in Android Applications. In: ISSRE 2016:490–501; 2016.

52. Grano Giovanni, Di Sorbo Andrea, Mercaldo Francesco, Visaggio Corrado A., Canfora Gerardo, Panichella Sebastiano. Android Apps and User Feedback: A Dataset for Software Evolution and Quality Improvement. In: WAMA 2017:8–11; 2017; New York, NY, USA.

53. Li Huiying, Zhang Li, Zhang Lin, Shen Jufang. A user satisfaction analysis approach for software evolution. In: PIC 2010, vol. 2: :1093-1097; 2010.

54. Panichella S., Di Sorbo A., Guzman E., Visaggio C.A., Canfora G., Gall H.C.. ARDOC: App Reviews Development Oriented Classifier. In: FSE 2016:1023-1027; 2016.

55. Di Sorbo Andrea, Panichella Sebastiano, Alexandru Carol V., Visaggio Corrado A., Canfora Gerardo. SURF: Summarizer of User Reviews Feedback. In: ICSE-C '17:55–58; 2017; Piscataway, NJ, USA.

56. Martin William, Harman Mark, Jia Yue, Sarro Federica, Zhang Yuanyuan. The app sampling problem for app store mining. In: MSR '15:123–133; 2015.

57. Canfora Gerardo, Di Sorbo Andrea, Mercaldo Francesco, Visaggio Corrado Aaron. Exploring Mobile User Experience Through Code Quality Metrics. In: PROFES 2016:705–712; 2016.

58. Panichella Sebastiano, Ruiz Marcela. Requirements-Collector: Automating Requirements Specification from Elicitation Sessions and User Feedback. In: RE 2020; 2020.

59. Qaddoura Raneem, Abu-Srhan Alaa, Qasem Mais Haj, Hudaib Amjad. Requirements prioritization techniques review and analysis. In: :258–263IEEE; 2017.

60. Stettinger Martin, Felfernig Alexander, Leitner Gerhard, Reiterer Stefan. Counteracting Anchoring Effects in Group Decision Making. In: :118–130; 2015.

61. Fu Bin, Lin Jialiu, Li Lei, Faloutsos Christos, Hong Jason, Sadeh Norman. Why People Hate Your App: Making Sense of User Feedback in a Mobile App Store. In: KDD '13:1276–1284; 2013.

62. Nayebi Maleknaz, Cho Henry, Ruhe Guenther. App store mining is not enough for app improvement. *Empirical Software Engineering*. 2018;23(5):2764–2794.

63. Deshpande Gouri, Rokne Jon G.. User Feedback from Tweets vs App Store Reviews: An Exploratory Study of Frequency, Timing and Content. In: :15–21; 2018.