Succeeding in Mobile Application Markets

(From Development Point of View)

by

Ehsan Noei

A thesis submitted to the
Electrical and Computer Engineering Department
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
September 2018

Abstract

Mobile application (app) markets, such as Google Play Store, are immensely competitive for app developers. Receiving high star-ratings and achieving higher ranks are two important elements of success in the market. Therefore, developers continuously enhance their apps to survive and succeed in the competitive market of mobile apps.

We investigate various artifacts, such as users' mobile devices and users' feedback, to identify the factors that are statistically significantly related to star-ratings and ranks. First, we determine app metrics, such as user-interface complexity, and device metrics, such as screen size, that share a significant relationship with star-ratings. Hence, developers would be able to prioritize their testing efforts with respect to certain devices. Second, we identify the topics of user-reviews (i.e., users' feedback) that are statistically significantly related to star-ratings. Therefore, developers can narrow down their activities to the user-reviews that are significantly related to star-ratings. Third, we propose a solution for mapping user-reviews (from Google Play Store) to issue reports (from GitHub). The proposed approach achieves a precision of 79%. Fourth, we identify the metrics of issue reports prioritization, such as issue report title, that share a significant relationship with star-ratings. Finally, we determine the rank trends in the market. We investigate the most important factors that are statistically significantly related to the changes in the ranks, such as release latency.

The outcome of this thesis helps app developers to be more successful in the market. Having the knowledge of the factors that are statistically significantly related to star-ratings and ranks helps developers to make wiser decisions to develop a successful app.

Related Publications

The publications related to the early versions of this thesis are listed below:

- **A study of the relation of mobile device attributes with the user-perceived quality of Android apps (Chapter 3)**, E. Noei, M. D. Syer, Y. Zou, A. E. Hassan, and I. Keivanloo, *Empirical Software Engineering*, vol. 22, no. 6, pp. 3088 – 3116, 2017.

- **Too many user-reviews! What should app developers look first? (Chapter 4)**, E. Noei, F. Zhang, and Y. Zou, *IEEE Transactions on Software Engineering (TSE)* (to appear).

- **Towards prioritizing user-related issue reports of mobile applications (Chapter 5)**, E. Noei, F. Zhang, S.Wang, and Y. Zou, *Empirical Software Engineering*, 2018 (to appear).

- **Winning the app production rally (Chapter 6)**, E. Noei, D. da Costa, and Y. Zou, *In Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2018, Lake Buena Vista, FL, USA.

I am the primary author of all the above publications. The research papers are co-authored with Dr. Ying Zou, Dr. Ahmed E. Hassan, and my colleagues, Dr. Daniel Alencar da Costa, Dr. Feng Zhang, Dr. Iman Keivanloo, Dr. Mark D. Syer, and Dr. Shaohua Wang. Dr. Ying Zou supervised all of the related research. Dr. Ahmed E. Hassan co-supervised the publication related to Chapter 3. The co-authors participated in meetings and provided me with feedback and suggestions to improve my work.

Acknowledgments

I would like to thank my advisor, Dr. Ying (Jenny) Zou, who played great roles in my development as a researcher and as a person. I also thank Dr. Ahmed E. Hassan for his support and for sharing his experiences. I am grateful for the support of my examining committee, Dr. Scott Yam, Dr. Xiaodan Zhu, Dr. Farhana Zulkernine, and Dr. Yann-Gaël Guéhéneuc who provided me with insightful feedback and guidance.

I had the privilege of working with an amazing group of researchers in the Software Re-Engineering lab: Dr. Iman Keivanloo, Haoran Niu, Dr. Shaohua (David) Wang, Dr. Feng Zhang, Yongjian Yang, Yonghui Huang, Pradeep Venkatesh, Mariam El Mezouar, Yu Zhao, Dr. Daniel Alencar da Costa, Taher Ahmed Ghaleb, Aidan Yang, and Guoliang Zhao. Finally, I would like to thank my friends and my family whom without their support this thesis would have not been possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile application (app) markets, such as Google Play Store [69], are becoming more competitive every year [135, 153]. When it comes to users' choice when downloading an app, app ratings and app ranks play a major role [83]. In this chapter, first, background information is provided in Section 1.1. Then, the thesis statement (Section 1.2), objectives (Section 1.3), methodology (Section 1.4), and contributions of this thesis (Section 1.5) are explained.

1.1 Background

In this section, mobile apps, app markets, mobile app development process, and app ratings are explained and discussed.

1.1.1 App Markets

An app market is a platform where mobile app developers introduce and sell their apps to users (i.e., customers) [35]. After publishing an app on an app market, users can view and download the published app. While several mobile app markets exist (e.g., Apple App Store [8] and Microsoft Store [142]), Google Play Store [69] is by far the largest market [155, 185]. In this thesis, we investigate and study Google Play Store as our primary app market.

1.1.2 Mobile Apps

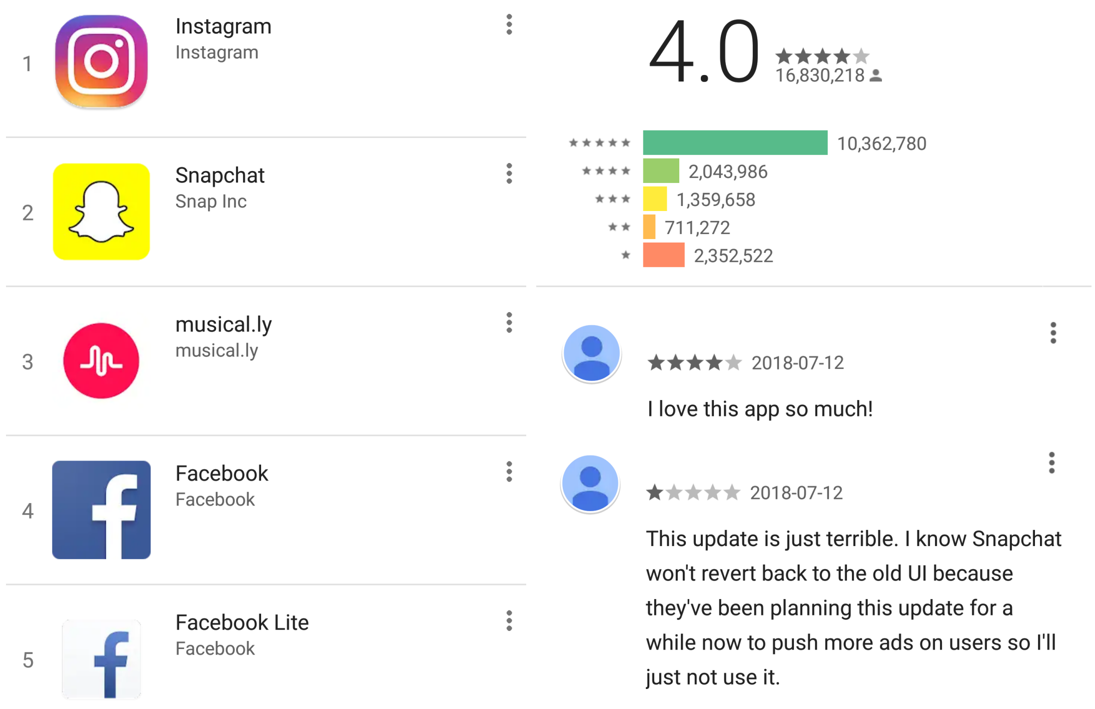
A mobile app is a computer program which is designed to run on smartphones and tablets [174]. Google Play Store hosts the mobile apps that run on top of the Android Operating System (OS) [202]. To build the Android OS, Google has modified the Linux kernel to make it compatible with mobile devices [47, 112].

The revenue of Android apps has increased enormously in the past few years [186]. Google Play Store is immensely competitive for app developers due to the recent increase in the number of mobile apps and smartphone users [188]. As of *July 1, 2018*, more than three million apps exist on Google Play Store [187].

Android apps can be published in different categories [45], such as *Finance* and *Education*. Each category holds a specific purpose. For example, an app that is published in the category of *Finance* should address or solve a financial requirement [153]. Figure 1.1 shows a sample overview of Google Play Store.

1.1.3 App Development Process

To bring a higher quality of mobile apps, developers should follow the continuous app development paradigm [1]. The continuous app development is the process of continuously releasing high-quality apps and high-quality code changes. Developers may adopt various software development methodologies, such as agile [198], for the continuous app development process. However, regardless of the development methodology, developers need to manage continuous releases to succeed in the competitive market



4.0 ★★★★☆ 16,830,218

★★★★★	10,362,780
★★★★	2,043,986
★★★	1,359,658
★★	711,272
★	2,352,522

★★★★☆ 2018-07-12

I love this app so much!

★☆☆☆☆ 2018-07-12

This update is just terrible. I know Snapchat won't revert back to the old UI because they've been planning this update for a while now to push more ads on users so I'll just not use it.

(a) App Ranks (b) Star-Ratings and User-Reviews

Figure 1.1: Overview of Google Play Store, showing app ranks, star-ratings, and user-reviews.

of mobile apps [1, 148]. As shown in Figure 1.2, the major steps that need to be continuously taken are:

(i) ***Develop.*** The first step is integrating all the new changes to the source code [19]. App developers should continuously improve their apps by adding new features, fixing bugs, and enhancing the performance of their apps [1, 37]. Bugs and new features can be identified from users' feedback or can be identified by developers (e.g., testing team).

(ii) ***Test.*** After the development step and before releasing a newer version on an app market, developers should conduct some short tests on the code [1, 99]. If

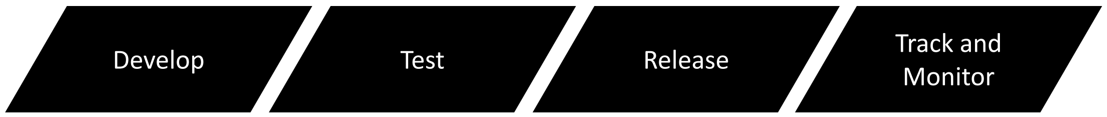


Figure 1.2: Major steps in the continuous app development process.

the code passes the test step, it would be ready to release.

(iii) ***Release.*** In the release step, developers release the tested code to the market, i.e., Google Play Store. The older versions of an app (installed on users' devices) can be updated to the newest version via Google Play Store [69].

(iv) ***Track and Monitor.*** After releasing a newer version, developers should keep track of the released version. Mobile app developers should consider (i) star-ratings, (ii) user-reviews, and (iii) ranks to evaluate the newly released version of an app (see Part 1.1.4).

Like traditional software development, different aspects of software development, such as engineering the requirements, having a correct and solid design, and intuitiveness of the initial idea, influence the success of an app [146]. Despite the similarities between mobile app development and traditional software development, there are important differences that can help developers deliver a high-quality app [33]:

(i) App developers have a direct contact with users.

(ii) App developers have a real-time access to users' feedback.

(iii) Users can rate an app and express their experience.

(iv) Users can request a feature or report a bug.

(v) Mobile apps are in a tight competition with other similar apps on the market.

(vi) Different users run the apps in different geographical locations, on different mobile devices, and with different conditions and limitations.

(vii) Mobile apps can be updated automatically with minor user effort.

1.1.4 App Ratings and Users' Feedback

Developers should mine users' feedback (i.e., user-reviews) (i) to understand users' concerns and avoid declines in ratings, i.e., star-rating, and potentially increase the star-ratings and (ii) to improve app rankings. Moreover, developers should understand the factors that might affect users' experience to better succeed in the market.

Star-Ratings

Google Play Store uses a star-rating mechanism to quantify the user-perceived quality of mobile apps. A user can give one star (i.e., the lowest) to five stars (i.e., the highest) rating to each app. Figure 1.1 shows the total number of star-ratings received for a sample app. The sample app in Figure 1.1 have been rated $16,830,218$ times with the average of star-ratings of 4.0. For a given app, the received star-ratings are vital for two main reasons:

(i) Users rely on star-rating when choosing a new app to download [17]. The apps with higher star-ratings are preferred. Users usually do not install the apps with an average of star-ratings less than 3 [155].

(ii) Harman *et al.* [78] report that there is a statistically significant relationship between the number of downloads and star-ratings.

Maintaining or improving the star-ratings requires the knowledge of the factors that can impact the star-ratings. Recent studies (e.g., [59, 60, 159, 160, 163]) have shown the unavoidable importance of star-ratings and user-reviews in the success and the profitability of apps. Understanding the factors that share a significant relationship with star-ratings helps developers to succeed in the market. In addition to the app factors, such as code quality, developers should also consider the devices that users usually use. Therefore, developers can proactively quantify the expected user-perceived quality of apps prior to release. Moreover, developers can prioritize their app testing efforts by focusing on certain devices [104].

User-Reviews

A user-review is a piece of informal text without a predefined structure [160]. Each user-review is associated with a star-rating [69]. Figure 1.1 shows two sample user-reviews on Google Play Store. Studying user-reviews helps developers to improve their apps and increase the star-ratings [135] as user-reviews could contain useful information, such as bug reports, feature requests, and user experiences [59, 160]. Therefore, by studying user-reviews:

(i) The main concerns and expectations of users can be understood.

(ii) The topics of user-reviews that share a significant relationship with star-ratings can be determined.

(iii) Users' feature requests can be extracted.

(iv) Bug reports can be determined and resolved.

(v) Bug reports and feature requests can be prioritized to achieve higher star-ratings.

Ranks

When a user searches for an app on Google Play Store, the most related apps get listed in order of a rank [140] (see Figure 1.1). The apps that are not placed among the top apps have a lower chance of being noticed and downloaded by users [153]. Therefore, improving the rank of an app increases the chance of having more users. Although high star-ratings can denote an app success, star-ratings might be misleading in some cases. For example, a 5-star app could be downloaded only a couple of times, as opposed to 3-star app with thousands of downloads. Therefore, both star-ratings and ranks should be taken into the consideration.

1.2 Thesis Statement

The path to survive and succeed in the competitive markets of mobile apps is not clear for app developers. For example, whether constantly maintaining an app, such as adding new features, will improve star-ratings and ranks is a moot point. Even top apps are in danger of losing their ranks and failing in mobile app markets. App developers and development companies need to achieve (or maintain) high ranks and star-ratings in order to succeed.

The goal of this thesis is assisting app developers to achieve higher star-ratings and better ranks. Therefore, developers could survive and succeed in the competitive market of mobile apps. To help developers make an informed decision, we investigate the main factors that share statistically significant relationships with ranks and star-ratings. For example, we show that users' feedback is an important asset to consider. However, users' feedback is unstructured and informal. Thus, we propose some solutions, such as mapping user-reviews with issue reports, to simplify and enhance app maintenance. We investigate to what degree developers can succeed in the market with/without (i) considering important app and device metrics that share a significant relationship with star-ratings, (ii) understanding users' feedback (i.e., user-reviews) and identifying users' main concerns, (iii) prioritizing issue reports with respect to user-reviews, and (iv) understanding rank trends and important factor that are statistically significantly related to ranks.

1.3 Objectives

The main objectives of this thesis include:

- **Identifying the factors that share a statistically significant relationship with star-ratings.**

  The star-ratings of mobiles apps reflects users' experience using an app on a particular mobile device. Therefore, the user-perceived quality of an app is not solely determined by app metrics. Hence, studying the relation of both device metrics, such as display resolution, and app metrics, such as user-interface complexity, with the user-perceived quality can help app developers to better manage maintenance and testing efforts.

- **Finding the topics of user-reviews that share a statistically significant relationship with star-ratings.**

  Mobile apps tend to receive various user-reviews. Addressing all the users' demands and fixing all the reported issues is a time- and budget-consuming task. However, in each category of mobile apps, there are some certain topics that share a statistically significant relationship with star-ratings, so called key topics. Therefore, developers can achieve higher star-ratings by focusing on the key topics.

– **Prioritizing issues of mobile apps.**
Addressing the issues that are reported in user-reviews is important for achieving higher star-ratings [135]. 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. Therefore, we propose a solution for integrating user-reviews into the process of issue reports prioritization.

– **Studying the factors that can improve ranks of mobile apps.**
Understanding the evolution of ranks and identifying the factors that share a statistically significant relationship with ranks can help developers improve the ranks. Not all the factors that common-wisdom would deem important have a significant relationship with the ranks. In fact, only a few factors, such as release latency, can enhance a rank.

1.4 Methodology

Figure 1.3 shows the overall methodology of this thesis. First, we collect the required data from Google Play Store and issue tracking repositories (i.e., GitHub [63]). Then, we preprocess the retrieved data. The output of the preprocessing step is the required data for Chapters 3, 4, 5, and 6.

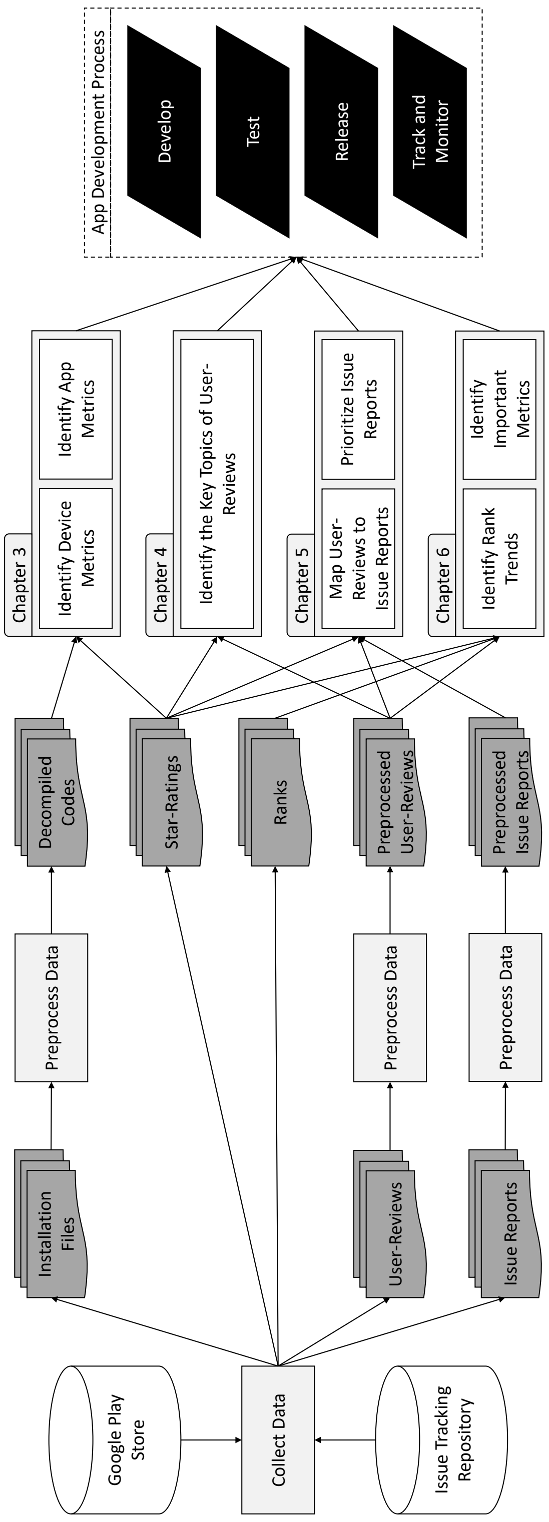


Figure 1.3: Overview of the methodology for enhancing the app development process and succeeding in the competitive markets of mobile apps.

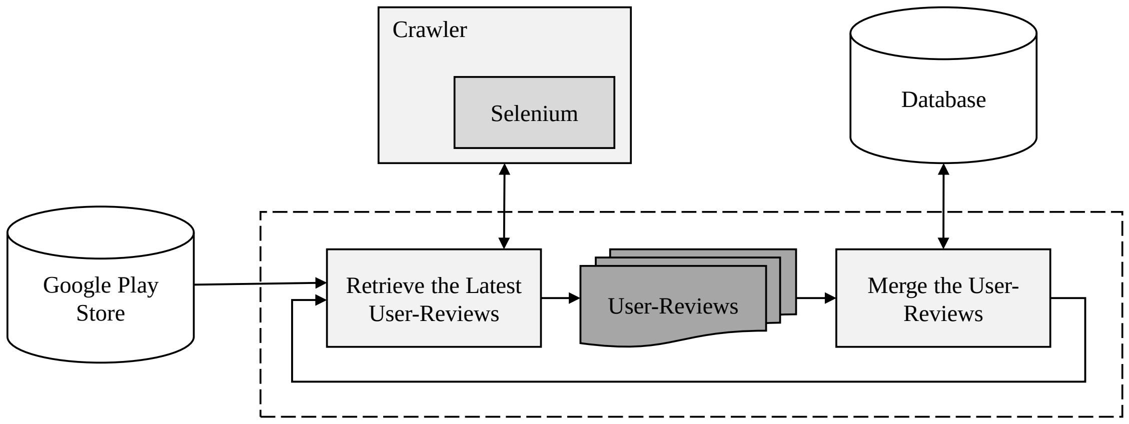


Figure 1.4: Process of gradually retrieving user-reviews from Google Play Store.

1.4.1 Data Collection

First, we explain the process of collecting star-ratings and user-reviews of mobile apps. Then, we explain how to get ranks, installation files, and issue reports.

Star-ratings and User-Reviews. Google Play Store limits the total number of user-reviews that a user can view to $2,400$ user-reviews [104]. Therefore, one cannot access all the available user-reviews of an app at once if it comes with more than $2,400$ user-reviews [69, 104].

To access all the user-reviews of an app, we need to gradually retrieve the user-reviews from Google Play Store. There are different solutions, such as utilizing Apache Nutch [154], for gradually retrieving the user-reviews. However, a simpler solution is utilizing the Selenium automation tool [181] to crawl and retrieve the user-reviews. Figure 1.4 shows an overview of the process of retrieving user-reviews using Selenium.

Selenium [181] provides a set of tools and an application programming interface (API) to automate Web browsing. The main purpose of Selenium is Web testing [181]. However, it can be used for other purposes, such as Web crawling [156]. 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 [28]. The client API lets developers communicate with Selenium. Finally, the Web driver sends the Selenium commands to the browser [181]. To retrieve the user-reviews, we use Selenium as a crawler. By using Selenium, app details, such as app names, app descriptions, and release notes, can be retrieved in addition to the user-reviews. The crawler needs to be run on a daily basis to get the latest user-reviews of each app. Then, the new user-reviews will be merged with the existing user-reviews in the database. Hence, one could capture all the user-reviews for all the required apps.

Ranks. Ranks are the order of the appearance of the apps when they are searched by a user [69]. Ranks are accessible from Google Play Store.

Installation Files. The source codes of the majority of Android app are not publicly accessible. Also, only the latest version of the installation file of an app could be downloaded from Google Play Store. To access all the versions of an app, the latest installation files need to be downloaded gradually.

Issue Reports. An issue report usually includes a title, a text, and a date on which the issue is posted. In Chapter 5, we consider the issue reports of the apps that are publicly available on GitHub [50]. We get the list of the open-source Android apps (i.e., $1,310$ apps) from F-Droid ([57]). F-Droid is the largest platform for open-source Android apps. We retrieved the issue reports of the open-source apps using GitHub API [50].

1.4.2 Preprocessing Data

In this part, we explain the data processing step for installation files, user-reviews, and issue reports.

User-Reviews and Issue Reports

The data that is retrieved from Google Play Store and code repositories contain noises that must be fixed. For example, a user-review is a piece of informal text [69, 160] that can potentially suffer from grammatical issues and typos. 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. In addition, user-reviews contain negations that can confuse automatic approaches [203]. 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. The above problems can happen with issue reports too. In the following paragraphs, we describe the steps that need to be taken for cleaning the textual data.

Step 1) Removing Non-English Sentences. We filter out non-English user-reviews and issue reports using Language Detector [158]. Language Detector is a Java library that detects the language of a piece of text [158].

Step 2) Identifying Inconsistent User-Reviews. Google Play Store faces inconsistencies among the user-reviews [59, 160]. For example, two users who perceive the same quality from the same app may give different star-ratings due to their subjective experience. Some user-reviews have negative content, but they are associated with high star-rating, and vice versa. As an example, we find *"Absolutely terrible.*

Waste of time" associated with 5 stars. The inconsistent user-reviews can affect the accuracy of the findings. To identify inconsistent user-reviews, the given star-ratings should be compared with the sentiment scores [194] of user-reviews. Different sentiment analysis tools [98], such as SentiStrength [193], SentiStrentgh-SE [92], and Natural Language Toolkit (NLTK) [24], can be used to capture the sentiment scores. SentiStrentgh-SE is a sentiment analysis tool that is trained using software engineering data [92].

Sentiment scores are normally between -5 and $+5$. The most negative user-reviews receive the score of -5 and the most positive user-reviews receive the score of $+5$. The user-reviews with the score of -1, 0, and $+1$ are considered as neutral user-reviews [193]. Any score above $+1$ is a positive sentiment score and any score below -1 is a negative sentiment score [193]. The user-reviews are associated with star-ratings between 1 and 5. The majority of users do not download the apps with the star-rating of less than three [155]. Therefore, the star-ratings of 3 should be counted as neutral ratings, any star-rating below 3 as a negative rating, and any star-rating above 3 as a positive rating. The consistent user-reviews hold a positive user-reviews with a positive sentiment score or a neutral user-reviews with a neutral sentiment score or a negative user-reviews with a negative sentiment score.

Step 3) Identifying Uninformative User-Reviews. An uninformative user-review, such as *"This app is OK"*, provides minor information for app developers [33, 203]. AR-Miner is an approach proposed by Chen *et al.* [33] to filter out uninformative user-reviews. AR-Miner applies Expectation Maximization for Naïve Bayes method [152] to build a classifier that distinguished between informative and uninformative user-reviews.

Step 4) Correcting Typos. Typos in the text would disturb the analysis techniques [157]. To reduce the risk of missing valuable information, typos should be corrected. We use Jazzy Spell Checker [94] to correct the typos. Jazzy is a tool that provides a set of Java APIs. We use Jazzy with a dictionary of $645,289$ English words.

Step 5) Coreference Resolution. Coreference happens when two or more expressions refer to the same referent [44]. For example, suppose a user-review: *"I really like the pictures in this app. They give me a more comprehensive view"*. The second sentence uses a pronoun (i.e., they). However, the machine does not understand what does "they" refer to. We use the Stanford deterministic coreference resolution [114] to resolve the coreferences in the user-reviews. The above example will be converted to *"I really like the pictures in this app. The pictures give me a more comprehensive view"*.

Step 6) Sentence Annotation. Some users write long user-reviews with several concerns, such as listing the pros and cons in details, in one single review. We use Stanford CoreNLP [132] to fragment the user-reviews with several concerns into pieces. Hence, each piece could share an independent concept. Stanford CoreNLP annotates the words in the user-reviews. It produces the base forms and the parts of speech for the words. It also identifies the structure of the sentences in terms of words and phrases dependencies. The two sentences in the example above share the same concept and refer to the same concept that is *"pictures"*. If the sentences did not share the same concern, we would have fragmented them into two pieces. For example, *"I can easily search the product names. However, the app is a battery* *killer."* is fragmented into two pieces of *"I can easily search the product names."* and *"However, the app is a battery killer"*. Therefore, we can have each fragment with an independent subject.

Step 7) Resolving Synonyms. General-purpose thesaurus, such as *WordNet* [143], are not sufficient to resolve the synonyms of an informal text, such as a user-review [203]. 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 [26, 154] on the 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 [6, 150]. For example, *"luv"* should be replaced with *"love"*.

Step 8) Resolving Negations. The negations in a text can mislead the text processing techniques in getting the real meaning of the text. To avoid such confusions, we use the Stanford natural language processing toolkit [132] to find and resolve the negated terms [203].

Installation Files

Android apps on Google Play store are compiled into APK files. An APK file contains all the content that is required for both the installation and the execution of an app. An APK file contains a manifest file, a meta-information directory, a Dalvik EXecutable (DEX) file, and a resource directory. The manifest file describes app-level meta-information, such as the title. The compiled source code is stored in a DEX file. A DEX file represents the compiled code of Java classes using an intermediate language similar to Java byte code. Unlike Java byte code, DEX files are compiled to be consumed by Dalvik, which is a register-based virtual machine that is provided by the Android platform. We use Dex2jar [51] to decompile the DEX files into Java classes. Then, we use Java Decompiler [95] to convert the Java classes to Java source codes.

1.4.3 Approach

Figure 1.3 shows an overall view of our approach to enhance the app development process and succeed in the competitive markets of mobile apps. The details of the approach are thoroughly described and discussed in the next chapters.

Identifying the app and the device metrics share a significant relationship with star-ratings (Chapter 3) helps developers as follows:

– *Develop.* When developing an app, developers should carefully take care of the app and the device metrics that will be potentially related to the star-ratings of their app.

– *Test.* Testing team should prioritize the testing efforts on certain devices that might harm the star-ratings.

– *Release.* Developers can limit the availability of their apps to specific devices to reduce the risk of receiving low star-ratings from the users of specific devices.

- *Track and Monitor.* By taking care of the devices that can potentially harm the star-ratings, the number of negative user-reviews should decrease. Therefore, developers could focus on other important issues of users, such as bug reports.

Identifying the key topics of user-reviews (Chapter 4) helps developers as follows:

- *Develop.* Identifying the topic of user-reviews that share a statistically significant relationship with star-ratings will help app developers to prioritize their development process on important issues.

- *Test.* The testing efforts should be prioritized on the key topics of user-reviews because a flaw in a key topic (e.g., *messaging* in a *social* app) can significantly harm the star-ratings.

- *Release.* When releasing an app, developers should mention the fixes and the changes that have been committed with respect to the key topics of user-reviews. We observe that having more similar release notes to user-reviews share a positive relationship with star-ratings.

- *Track and Monitor.* Developers can consider user-reviews (e.g., bug reports and feature requests) that are related to the key topics carefully. For example, due to the vital importance of the key topics, user-reviews related to the key topics should be considered prior to other user-reviews.

Matching user-reviews with issue reports and prioritizing the user-related issue reports (Chapter 5) helps developers as follows:

- *Develop.* Developers could prioritize addressing the issue reports to achieve higher star-ratings and succeed in the competitive market of mobile apps.

– *Test.* Testing team should consider and test the changes that are conducted with respect to the prioritized issue reports.

– *Release.* When releasing a newer version, the new changes that are done with respect to user-reviews should be mentioned in the release notes that are viewable by users on Google Play Store.

– *Track and Monitor.* By mapping user-reviews to issue reports, developers could identify (i) the issue reports that already exist in the issue tracking repository and (ii) the issue reports that does not exist in the issue tracking repository. Therefore, developers would be able to narrow down the number of issues that are not identified yet.

Identifying the rank trend and the metrics that share a significant relationship with the changes in the ranks (Chapter 6) helps developers as follows:

– *Develop.* Identifying the factors that are significantly related to the ranks, such as app description, installation file size, and app features, can help developers to maintain an app better.

– *Test.* The testing team could consider the important factors that could potentially improve or harm the ranks very carefully.

– *Release.* We observe that the release latency and the number of releases are two important factors that share statistically significant relationships with ranks.

– *Track and Monitor.* User-reviews are important artifacts for improving a rank. However, we observe that user-reviews, app descriptions, and release notes of other competitors are other important artifacts that should be tracked and monitored for improving the rank of a given app.

1.5 Contributions

The major contributions of this thesis include:

(i) We identify both app metrics, such as code complexity, and device metrics, such as screen resolution, that share a statistically significant relationship with star-ratings.

(ii) We determine the topics of user-reviews that are statistically significantly related to star-ratings. We find that each category has a specific set of topics that share a significant relationship with star-ratings (i.e., key topics). The key topics are not necessarily the most frequent topics of user-reviews.

(iii) We propose a solution to map user-reviews (from Google Play Store) to issue reports (from GitHub) for prioritizing user-related issue reports.

(iv) We identify the evolution of ranks (i.e., rank trends) over time. Also, we empirically determine the factors that share a statistically significant relationship with ranks, such as the release latency and the name similarity.

1.6 Organization of Thesis

We proceed by introducing related work in **Chapter 2**.

Chapter 3 describes our study for identifying app and device metrics that share a significant relationship with star-ratings. In this chapter, first, the study setup is explained (e.g., data collection). Then, we compare star-ratings of different mobile apps to show that there is a significant difference in the star-ratings given by users of different mobile devices. Finally, we identify the app and the device metrics that share a significant relationship with star-ratings.

Chapter 4 explains our study to identify the topics of user-reviews that share a significant relationship with star-ratings (i.e., key topics). After setting up the study, we apply topic modeling to identify the topics of user-reviews. Then, we determine the key topics with respect to the star-ratings. Finally, we evaluate the key topics.

Chapter 5 explains our solution to map user-reviews to issue reports. **Chapter 5** also includes our proposed approach for prioritizing the user-related issue reports. In this chapter, we highlight the important factors for prioritizing issue reports of mobile apps. We also evaluate the suggested prioritizing solution.

Chapter 6 discusses our study on the app ranks. In this chapter, first, we determine the rank trends of mobile apps. Then, we highlight the factors that are statistically significantly related to the ranks trends and the changes in the ranks.

Finally, **Chapter 7** concludes and outlines future work.

Chapter 2

Related Work

In this chapter, related work is introduced along five directions: (i) user-perceived quality analysis (Section 2.1), (ii) summarizing user-reviews (Section 2.2), (iii) source codes and issue reports analysis (Section 2.3), (iv) issue reports prioritization analysis (Section 2.4), and (v) other aspects (Section 2.5).

2.1 User-Perceived Quality Analysis

The majority of the existing work in the literature rely on star-ratings to measure the user-perceived quality of Android apps. Harman *et al.* [78] report that there is a statistically significant relationship between the number of downloads and star-ratings. Kim *et al.* [106] discuss how star-ratings influence the users' decision to download an app.

Some empirical studies examine app metrics that share a relationship with the user-perceived quality [135]. Taba *et al.* [190] studied the relationship between the UI complexity and the user-perceived quality of Android apps. They observed that the UI complexity has a significant relationship with the user-perceived quality.

Linares-Vasquez *et al.* [123] found that low-quality APIs affects the star-ratings of apps. In another work, They noticed that anti-patterns in source code may negatively affect app quality [124]. Bavota *et al.* [17] indicated that the user-perceived quality of an app is related to the fault- and change-proneness of the APIs used by such an app. Tian *et al.* [196] measured 28 metrics of apps and found that highly rated apps were different from low rated apps in 17 metrics, such as the source code complexity and marketing. Ruiz *et al.* [177] examined the rating system of app stores to see how the ratings are demonstrated to the users and how the ratings affect the number of users of each app. Bakar and Mahmud [12] found that there is no correlation between star-ratings and the number of permissions which for an app asks.

In this thesis, we further investigate the user-perceived quality of Android app in terms of star-ratings. We study the relation of mobile device metrics, the topics of user-reviews, and issue reports prioritization with star-ratings.

2.2 Summarizing User-Reviews

User-reviews contain users' feedback running an app in different situations and conditions. User-reviews are important as they can contain the concerns of users regarding an app. Due to the importance of user-reviews, recent work proposed some solutions to summarize user-reviews and ease the process of app development accordingly [60, 75, 89].

Goul *et al.* [70] published the earliest work on user-reviews in 2012 [135] in the context of requirements engineering. They applied sentiment analysis on $5,000$ user-reviews that were retrieved from Apple App Store [8].

In 2014, Chen *et al.* [33] proposed a framework, called AR-Miner, to identify and filter out uninformative user-reviews. They employed textual analysis techniques to detect and rank informative user-reviews. In Chapters 4 and 6, we use AR-Miner to filter out uninformative user-reviews.

Classifying user-reviews into meaningful groups is an important issue in the mobile app development research. Gu and Kim [74] classified user-reviews in five groups of evaluation, praises, feature requests, bug reports, and others. They applied aspect opinion mining [109] and sentiment analysis to find the most popular features of the apps. However, their work did not reveal relation of such features with star-ratings. Panichella *et al.* [164] applied natural language processing, text analysis, and sentiment analysis to classify user-reviews into five main intentions of users, including information giving, information seeking, feature requests, problem discovery, and others. Di Sorbo *et al.* [52] presented an approach, called SURF, and suggested two levels of classification: (i) intention classification [164], and (ii) topic classification. In the topic classification level, Di Sorbo *et al.* [52] manually train a set of user-reviews to find the topics. Villarroel *et al.* [203] proposed a tool, called CLAP, to classify user-reviews into bug reports and feature requests. Then, they classified and ranked the bug reports and feature requests that were reported in user-reviews to help app developers in planning for the next releases of their app. We adopt the approach of Villarroel *et al.* [203] as a part of our approach presented in Chapter 5. Ciurumelea *et al.* [37] proposed an approach to organize user-reviews with respect to different topics, such as performance and memory. Having the user-reviews organized, they could recommend source-code pertaining to these topics using code localization.

Topic modeling is widely used in summarizing user-reviews [26, 86]. For example, Iacob and Harrison [89] and Guzman and Maalej [75] applied LDA on user-reviews to extract the feature requests. Iacob and Harrison [89] also looked for linguistic rules to detect feature requests. Guzman and Maalej [75] employed topic modeling techniques to extract app features from user-reviews. Galvis and Winbladh [60] benefited from topic modeling techniques and sentimental analysis to help developers in the revision of requirements for the next releases of their apps. Fu *et al.* [59] proposed an approach to analyze star-ratings and user-reviews. Fu *et al.* [59] also listed the most frequent keywords in user-reviews by applying topic modeling techniques.

In this thesis, we identify the topics of user-reviews that share a significant relationship with star-ratings, i.e., key topics, so that app developers can address the issues that are related to the key topics prior to other issues that are reported in the user-reviews. Moreover, we propose a solution to map user-reviews to issue reports to prioritize the user-related issue reports.

2.3 Source Code and Issue Reports Analysis

One of the biggest challenges of studying source codes and issue reports of mobile apps is that for the majority of mobile apps, such data is not publicly available. Thus, the number of papers that study the bug repositories and source code of Android apps is smaller than the number of papers that study star-ratings and user-reviews. To gain access to source code and issue reports, researchers usually follow two approaches: (i) studying only open-source apps (e.g., Ciurumelea *et al.* [37]) and (ii) decompiling the installation files of android apps (e.g., Moran *et al.* [145]).

Ciurumelea *et al.* [37] suggested an approach to recommend code using code localization [127] having the user-reviews organized. Palomba *et al.* [161] followed the approach proposed by Panichella *et al.* [163] to classify user-reviews and map the classified user-reviews to source code. Palomba *et al.* [161] recommended the source code changes that are required to address the user-reviews by measuring the asymmetric Dice similarity coefficient [11] between the words in each cluster of user-reviews and the words in each class of source code.

Moran *et al.* [145] introduced a tool, called FUSION, to manage auto-completion of bug reports of mobile apps. They applied statistic and dynamic analysis on the decompiled code of Android apps. FUSION helps developers in reproducing bugs and auto-completing bug reports. With respect to the findings of Chapter 4, developers should pay special attention to bug reports related to the key topics of user-reviews.

Some papers are based on empirical studies of the characteristics of issue tracking systems. Bhattacharya *et al.* [23] conducted an empirical study on 24 open-source Android apps. They defined some metrics of bug report quality and developer involvement. They observed that bug reports are of high quality, especially security bug reports have the highest quality amongst all bug reports. Palomba *et al.* [160] studied 100 Android apps. They compared user-reviews with the changelogs of open-source apps available on GitHub. They reported that implementing the users' feature requests in the next releases increases the star-ratings.

Mcdonnell *et al.* [138] studied the adoption of API updates by Android apps. They noticed that, on average, 28% of Android API calls are not up-to-date. They reported that the propagation time of API references to be updated is around 14 months. Maji *et al.* [130] applied a failure characterization study on mobile apps. They investigate the relationship between bugs, locations of the bugs in the code and the code changes. Linares-Vasquez *et al.* [125] studied open-source apps. They noticed that app developers rely on the manual execution of apps and user-reviews to identify performance bugs.

None of the previous work studied the relationship between prioritization of issue reports and star-ratings of mobile apps. We integrate the user-reviews into the process of prioritizing issue reports of Android apps and investigate the relationship between prioritizing issue reports and star-ratings. Furthermore, we identify code metrics that are statistically significantly related to star-ratings, such as user-interface complexity.

2.4 Issue Reports Prioritization Analysis

In this section, we introduce the related work that concerns the issue reports prioritization. The majority of the work presented in the section is done on other eco-systems rather than mobile apps.

Lamkanfi *et al.* [113] proposed a severity prediction approach by analyzing the textual description of issue reports of three open-source communities: *Mozilla*, *Eclipse* and *GNOME*. They applied Naïve Bayes to label the issue reports as severe or non-severe. Alenezi and Banitaan [5] employed Naïve Bayes, decision trees, and random forests to predict the priority of issue reports in *Bugzilla* [29]. They observed that random forest and decision tree outperform Naïve Bayes. Yu *et al.* [211] used neural network to prioritize issue reports. They showed that their approach works better than Naïve Bayes. Kanwal and Maqbool [100] applied Naïve Bayes and SVM to predict issue reports prioritization. They observed that SVM is better than Naïve Bayes when adopting textual metrics, such as issue report descriptions. Menzies and Marcus [141] ranked the terms that appear in issue reports using TF-IDF. They used top terms to predict the priority of issue reports. Tian *et al.* [195] used the similarity between current issue reports and issue reports in the past to estimate the priority of the new issue reports.

None of the above work incorporates user-reviews with issue report for prioritization. In Chapter 5, we take metrics of both issue reports and user-reviews to prioritize user-related issue reports.

2.5 Other Aspects

Some work is conducted to investigate the relationship between app releases and the successes of apps. Henze and Boll [83] studied the relationship between the release time and user activities in Apple App Store [8]. They observed that releasing new updates can improve app ranks. Zhu *et al.* [215] observed that some developers release updates with minor changes just to boost the rank of their app in Apple App Store. To avoid receiving low star-ratings, Ruiz *et al.* [177] recommend developers not to release incomplete apps early.

Some studies explored user-perceived quality based on user studies. Ickin *et al.* [91] focused on measuring user experience for a set of mobile apps that are used on a daily basis. They [91] observed that some factors, such as interface design, app performance, and user lifestyle, influence the user-perceived quality. Hassenzahl and Tractinsky [80] reported that the context within which an interaction occurs influence the user-perceived quality. Korhonen *et al.* [110] surveyed the literature and concluded that user's task is an important attribute that can affect the user-perceived quality.

2.6 Summary

In this chapter, we introduced the related work. In Section 2.1, we presented related work that investigated the user-perceived quality of apps. Despite the importance of device attributes on the performance of Android apps, no prior work investigates the relationship between mobile device attributes (see Chapter 3) and the user-perceived quality of Android apps.

In Section 2.2, we introduced research papers that summarized and clustered user-reviews. However, none of the earlier papers has specified a set of topics with a broader view, i.e., category, that are statistically significantly related to star-ratings, while every category of mobile apps shares a specific set of characteristics and requirements [128].

Section 2.3 summarized the studies on source code and issue reports. Section 2.4 introduced the related work on issue reports prioritization. None of the earlier works has integrated user-reviews into the process of issue reports prioritization of Android apps. Moreover, none of the prior works has studied the relationship between prioritizing issue reports of mobile apps and star-ratings.

Section 2.5 discussed other aspects of app development, such as the best time for releasing a newer version of an app. None of the recent works has studied the factors that share a statistically significant relationship with app ranks. Many factors, such as the number of releases and release latency, share a significant relationship with app ranks that is missing in the prior literature.

Chapter 3

App and Device Metrics with a Significant Relationship with Star-Ratings

In this chapter, we study the relation of both device metrics and app metrics with star-ratings of Android apps. The goal of this chapter is twofold. First, we show that to succeed in the competitive market of mobile apps, developers should consider various factors, such as users' devices, very carefully. Second, we highlight the device and app metrics that are statistically significantly related to star-ratings. Therefore, developers would be able to proactively estimate the success of their apps prior to release with respect to the important metrics.

A brief introduction to this chapter is presented in Section 3.1. In Section 3.2, the study setup is explained. Section 3.3 describes the approach and the findings. Section 3.4 discusses the threats to the validity. Finally, Section 3.5 concludes this chapter.

3.1 Introduction

Due to the importance of star-ratings in the success of an app [17, 43, 106, 134, 135], it is vital to identify the metrics that have significant relationships with star-ratings. The knowledge of such metrics helps developers to quantify the expected star-ratings prior to release proactively. Developers can also limit the availability of their apps to specific devices to reduce the risk of receiving low ratings from specific devices. Moreover, developers could prioritize the app testing efforts on certain devices [104].

Recent studies mainly focus on the relationship between app metrics and star-ratings [106, 123, 135, 190]. For example, Taba *et al.* [190] found that the User-Interface (UI) complexity has a significant relationship with star-ratings. However, the observed star-ratings is the result of users' experience running the apps on a particular mobile device [205]. In this chapter, the following research questions are addressed:

RQ3.1) *Do users of different mobile devices give similar star-ratings to mobile apps?* The level of satisfaction with an app varies based on the device on which an app is running. By studying 30 Android mobile devices and 280 Android apps, we find that users of various devices give different star-ratings to the same set of apps.

RQ3.2) *Which device metrics share the most significant relationships with star-ratings?* We measure 20 device metrics, such as memory capacity, display size, and battery size. We find that the star-ratings of an app have a significant relationship with device metrics, such as Central Processing Unit (CPU), display, and Operating System (OS). For example, users of devices with more powerful CPUs give higher star-ratings to apps. Conversely, users having a newer version of the OS give lower star-ratings to the apps.

RQ3.3) *Do device metrics have a stronger relationship with star-ratings than app metrics?* We measure 13 app metrics, such as UI complexity and code size, in addition to the device metrics. We find that app metrics, such as code size, share significant relationships with star-ratings too. However, some device metrics, such as the CPU, have statistically significantly stronger relationships with star-ratings than the majority of app metrics.

3.2 Study Setup

Figure 3.1 depicts an overview of the study setup. First, user-reviews, app details, installation files, and device specifications are retrieved from Google Play Store and the technical device-specific Web-sites, including GSM Arena[1] and Phone Arena[2]. Technical device-specific Web-sites provide the required information about the device specifications, such as the capacity of RAM. Both Web-sites contain technical articles and details about mobile devices. When the details of a particular device were not available on GSM Arena, we checked Phone Arena to get the required device details.

3.2.1 Data Source

Android platform suffers from device fragmentation with many different devices running varying versions of the Android OS [76]. The data used in this chapter is an

[1]http://www.gsmarena.com/
[2]http://www.phonearena.com/

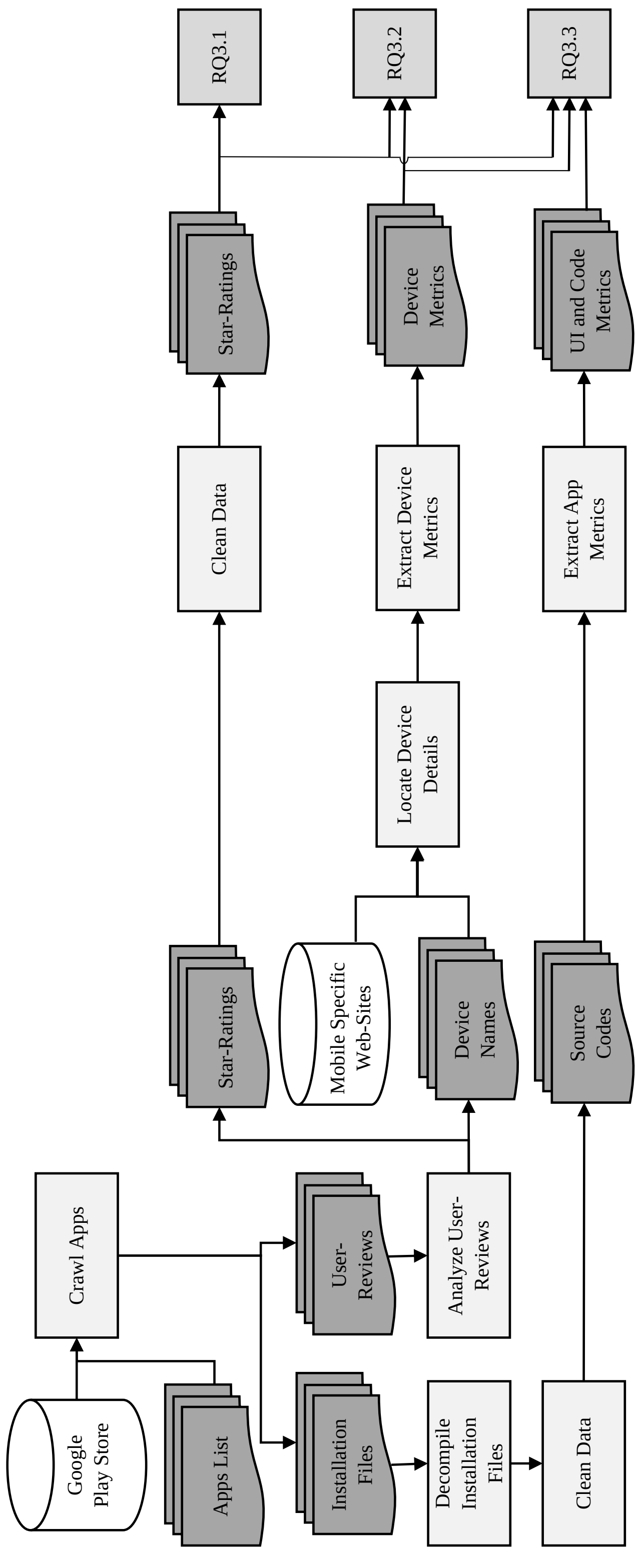


Figure 3.1: Study setup for identifying metrics of mobile apps and devices that share a significant relationship with star-ratings.

extension of the data of Khalid *et al.* [104]. The data includes app details, such as app names, device names, and star-ratings. Khalid *et al.* selected 99 apps for their study [104]. However, our dataset extends their dataset to 593 apps, $595,951$ reviews, and 210 devices. This dataset contains the star-ratings that are posted from 2009 to 2013. Table 3.1 provides a statistical overview of the star-ratings in the dataset.

3.2.2 Preprocessing Data

In this section, we discuss our data processing steps, such as data cleaning. Android apps on Google Play Store are accessible as APK files (i.e., installation files). We decompile the installation files first (see Part 1.4.2). We use the decompiled Java files to measure the app metrics, such as the number of Lines of Code (LOC) and the number of methods.

Cleaning Dataset

Our initial dataset contains $595,951$ reviews, 210 devices, and 596 apps. Google Play Store only allows us to download the latest version of an app [69]. There are 194 apps for which we did not have access to their older APK files. Therefore, we remove these 194 apps from our study. Moreover, we remove 22 apps that do not declare their UI using XML files from our study. We believe that UI is an important factor in star-ratings because users interact with the apps via their UIs. Furthermore, we remove 100 paid apps from our dataset because we would need to pay to get access to their installation files. We focus on free apps to reduce the influence of other confounding factors, such as the variance of user expectations based on the cost of an app [77].

We remove the devices whose metrics are not publicly available on the Internet.

Table 3.1: Overview of data that is used for identifying app and device metrics that share a significant relationship with star-ratings.

Device	**Total**	**Mean**	**Median**	**Deviation**	**Skew**
Galaxy S3	22,022	3.4	4.0	1.5	-0.4
Galaxy S2	18,898	3.3	4.0	1.5	-0.4
Galaxy S	13,417	3.2	3.0	1.5	-0.3
Nexus 7	6,322	3.3	4.0	1.5	-0.3
Galaxy Nexus	5,764	3.0	3.0	1.5	-0.1
Droid RAZR	6,892	3.2	3.0	1.5	-0.2
Evo 4G	7,121	3.1	3.0	1.5	-0.1
Optimus One	6,503	3.3	4.0	1.5	-0.3
Galaxy Note	4,776	3.4	4.0	1.5	-0.5
Galaxy Y	5,224	3.6	4.0	1.4	-0.7
Droid X	4,670	3.0	3.0	1.5	0.0
Desire HD	4,211	3.2	3.0	1.5	-0.2
Galaxy Ace	4,390	3.4	4.0	1.5	-0.5
One X	2,889	3.3	4.0	1.5	-0.3
Sensation 4G	2,824	3.2	3.0	1.5	-0.2
EVO 3D	2,582	3.2	3.0	1.5	-0.2
Droid Bionic	2,946	2.8	3.0	1.6	0.2
Nexus S	2,175	3.1	3.0	1.5	-0.1
Desire	2,715	3.1	3.0	1.5	-0.1
Thunderbolt	2,312	3.1	3.0	1.5	-0.1
Galaxy Tab 10.1	2,067	3.3	4.0	1.5	-0.4
Droid	2,699	3.0	3.0	1.4	0.1
Droid Incredible	2,459	3.0	3.0	1.5	0.0
Galaxy Tab 10.1	2,338	3.4	4.0	1.5	-0.5
Droid X2	2,048	3.0	3.0	1.5	0.0
myTouch 4G	1,913	3.2	3.0	1.5	-0.3
Wildfire S	2,169	3.3	4.0	1.5	-0.3
Galaxy Mini	2,171	3.5	4.0	1.4	-0.6
Droid II	1,991	3.0	3.0	1.5	0.0
Incredible 2	1,865	3.1	3.0	1.5	-0.1

We choose the devices that have the most number of apps in common. To mitigate our findings being skewed by the apps with few star-ratings [103], we consider the largest subset of the apps that have at least five star-ratings for each device in our dataset. Finally, we ended up with 30 mobile devices, 280 Android apps, and $150,373$ star-ratings in our dataset that met all the above criteria.

Extracting App Metrics

We describe the app metrics along three aspects: (i) source-code, (ii) user-interface, and (iii) miscellaneous.

Source Code. For each app, we count the number of methods and LOC in the decompiled Java files. LOC is a software metric that is used to measure the size of a program. Larger programs in terms of lines of code are potentially harder to maintain [117]. We measure the number of methods as another metric that captures the code size and the code complexity [97]. We calculate the complexity of each app using two measures: (i) McCabe's cyclomatic complexity [136] and (ii) Weighted Method per Class (WMC) [34]. After calculating the McCabe's cyclomatic complexity for each method, we count the number of methods with a cyclomatic complexity value of more than ten. Coppick and Cheatham [40] report that the methods with cyclomatic complexity values of less than ten are considered simple while the methods with cyclomatic complexity values of more than ten indicate a more complex code [40]. We measure the weighted average of the WMCs of each class.

User-Interface. App components are the fundamental building blocks of an Android app. There are four types of app UI components: (i) activities, (ii) services, (iii) content providers, and (iv) broadcast receivers [64]. An activity represents

Table 3.2: User-interface tags for identifying input and output elements.

Type	**Element Names**
Input	Button, EditText, AutoCompleteTextView, RadioGroup, RadioButton, ToggleButton, DatePicker, TimePicker, ImageButton, CheckBox, Spinner
Output	TextView, ListView, GridView, View, ImageView, ProgressBar, GroupView

a single-screen UI. A service is for performing functions for remote processes or for performing long-running operations. A content provider manages a shared set of app data. A broadcast receiver is a component that responds to broadcast announcements [64]. Users interact with activities only. There are two ways to define a UI layout for an activity: (i) declaring UI design elements in XML files and (ii) instantiating UI layout elements in the source-code. Similar to earlier studies (e.g., [190]), we study the apps that follow the best practice of UI implementation as suggested by Google [65]. With the best practice UI implementation, developers declare app layouts in XML files. We parse the XML files to calculate the set of UI metrics, as proposed by Taba *et al.* [190], including the number of inputs, outputs, and activities. We use the input and the output tags listed in Table 3.2 to identify inputs and output elements in a UI page [190]. The XML files are represented as trees [87, 204]. Therefore, we are interested in examining the relationship between the main characteristics of the UI trees and star-ratings. Hence, for each app, we extract the total number of leaves, nodes, and the layout depth of the UI trees [41].

Miscellaneous. We consider the size of the installation files as they contain various media and other resources that are associated with the apps. The size of an installation file can capture the overall complexity and richness of an app.

We summarize the app metrics in Table 3.3. Overall, we measure 13 metrics along

Table 3.3: App metrics for explaining star-ratings.

Category	Metric	Description
User-Interface	Number of Leaves	Number of leaves in XML layout files.
	Number of Nodes	Number of nodes in XML layout files.
	UI Depth	Deepest tree in XML layout files.
	Number of Activities	Number of activities in UI.
	Number of Inputs	Number of inputs in in UI.
	Number of Outputs	Number of outputs in UI.
	LOCUI	Number of lines of UI code.
Source Code	LOC	Number of lines of source code.
	Number of Methods	Number of methods in source code.
	WMC	Average of weighted method per class.
	Low McCabe	Number of methods with a cyclomatic complexity equal or less than 10.
	High McCabe	Number of methods with a cyclomatic complexity more than 10.
Miscellaneous	APK Size	Size of the downloadable APK installation file.

the three aforementioned categories. Table 3.4 shows a brief overview of the statistics of the computed app metrics.

Extracting Device Metrics

For each device, we extract 38 *direct metrics* of the hardware specifications. A direct metric is referred to as a metric without any processing on its value. All the 38 direct metrics are listed in Table 3.5. We remove the device metrics that have the same value for all of the devices because this information cannot differentiate among devices.

We did not include the prices of the devices as a device that is expensive today would be cheaper in the future. A user's experience with an app will remain the same independent of how much the user has paid for the device. We end up with 20 device metrics as summarized in Table 3.6.

Table 3.4: Overview of app metrics for explaining star-ratings.

Category	**Metric**	**Maximum**	**Minimum**	**Mean**	**Median**
User-Interface	LOCUI	4,222	4	605	371
	UI Leaves	4,222	2	605	160
	UI Nodes	1,846	1	264	183
	UI Depth	17	1	6	6
	Activity	495	1	49	34
	Input	769	1	52	22
	Output	1,375	1	152	79
Source code	LOC	607,458	315	140,541	137,598
	Code Method	54,444	40	12,162	11,692
	WMC	37	3	16	16
	Low McCabe	47,573	34	12,528	9,915
	High McCabe	1,661	0	380	315
Miscellaneous	APK Size	52,139	73	9,478	7,790

Table 3.5: List of direct device metrics.

Direct Metrics		
2G network support	3G network support	SIM support
Announcement date	Status of availability	Dimensions
Weight	Size of display	Multitouch display
Display colors	Display protection	Loudspeaker
3.5mm jack	Vibration	Internal RAM
GPRS support	Hot-spot support	EDGE support
WLAN support	Bluetooth support	NFC
USB support	Primary camera details	Video recording support
Secondary camera support	OS version	Upgradable OS
CPU	GPU	Accelerometer support
Compass support	Messaging features	Browser features
GPS	Java support	Battery
Standby	Talk time	

We classify the metrics as (i) nominal metrics that include two or more categories without considering the ordering among the categories, (ii) ordinal metrics considering the ordering among the metrics, and (iii) numeric metrics that are expressed with real numbers. We also split the device metrics that explain multiple aspects of the

Table 3.6: Device metrics for explaining star-ratings.

Category		**Metric**	**Description**
Body	**Dimension**	DeviceHeight	Height of a device in centimeters.
		DeviceWidth	Width of a device in centimeters.
		DeviceThickness	Thickness of a device.
Display	**Colors**	DisplayColors	Range of colors that are supported by the screen.
	Size	DisplayInch	Display size in inches.
	Resolution	ScreenWidth	Display width in pixels.
		ScreenHeight	Display height in pixels.
		DisplayPPI	Pixels per inch is a measurement of the pixel density (resolution).
Platform	**OS**	OSVersion	Initial Android version.
		OSUpgradeable	Indicates whether any OS upgrade is announced by manufacturers after the initial release.
	CPU	CPU	Powerfulness of the CPU.
	GPU	GPU	Powerfulness of the GPU.
Memory	**RAM**	InternalRAM	Capacity of the internal RAM.
	ROM	InternalROM	Capacity of the internal ROM.
Camera	**Camera**	CameraMegaPixel	Resolution of the primary camera in megapixels.
		Video	Resolution of recording video.
Battery	**Battery**	Battery	Battery size in ampere-hour.
	Talk time	Talktime	The longest time that a single battery charge lasts when a user is constantly talking on a device.
	Standby	Standby	The officially-quoted longest time that a single battery charge lasts when a device is constantly connected to the GSM network, but is not in active use.
Date	**Release Date**	Announced	Announcement date of a device.

devices. For example, we split the device size into the device height and the device width. For remaining metrics, such as GPU and CPU, we rank them in the order of their computation power.

3.3 Approach and Results

This section presents the approach and finding of the three research questions. For each research question, the motivation, the approach, and a discussion of the findings are presented.

RQ3.1) Do users of different mobile devices give similar star-ratings to mobile apps?

Motivation. Users run mobile apps on different devices with varying specifications. In this research question, we explore if users of different mobile devices give similar star-ratings to the same apps.

Approach. To address this research question, we investigate the star-ratings that are recorded by the users of each device. Figure 3.2 shows the distribution of the number of star-ratings per app for each device. As a null hypothesis, we suppose that the distributions of the star-ratings for the same set of apps running on different devices are the same. We use the Kruskal-Wallis test [111] to check the null hypothesis. The Kruskal-Wallis is a non-parametric method for testing whether different samples originate from the same distribution [111]. The test does not assume a normal distribution of the residuals as it is a non-parametric method. The Kruskal-Wallis extends the Mann-Whitney U test [131] to test more than two groups [111]. Hence, it shows

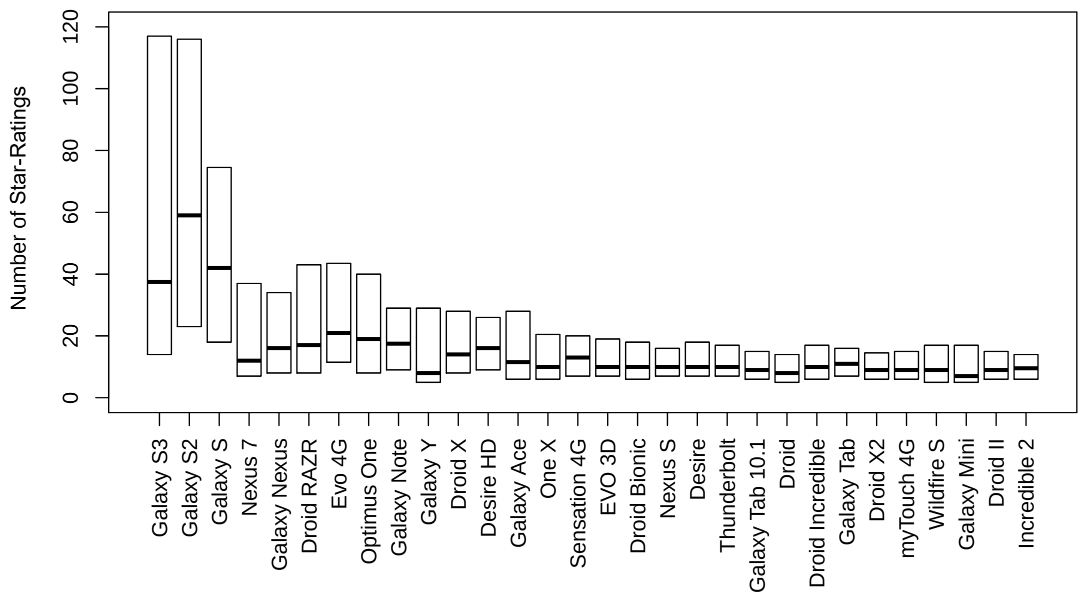


Figure 3.2: Number of star-ratings (per app) for each device.

whether different devices have the same distribution of star-ratings. We can reject the null hypothesis if the observed $p - value$ is less than 0.05. Hence, we can conclude that the users of at least one device give different star-ratings for the same set of apps. In addition, we use Dunn's test [54] to compare star-ratings of each pair of devices to determine how many pairs of devices are statistically significantly different in terms of the distributions of star-ratings. Finally, we depict the distribution of the star-ratings for each device using boxplots.

Findings. The Kruskal-Wallis test shows that the distribution of the star-ratings for the same set of apps of at least one device is significantly different with a $p - value$ of less than $2.2e - 16$ across the 30 devices. By using Dunn's test on the distribution of the star-ratings across each pair of devices, we find that 81% of the device pairs

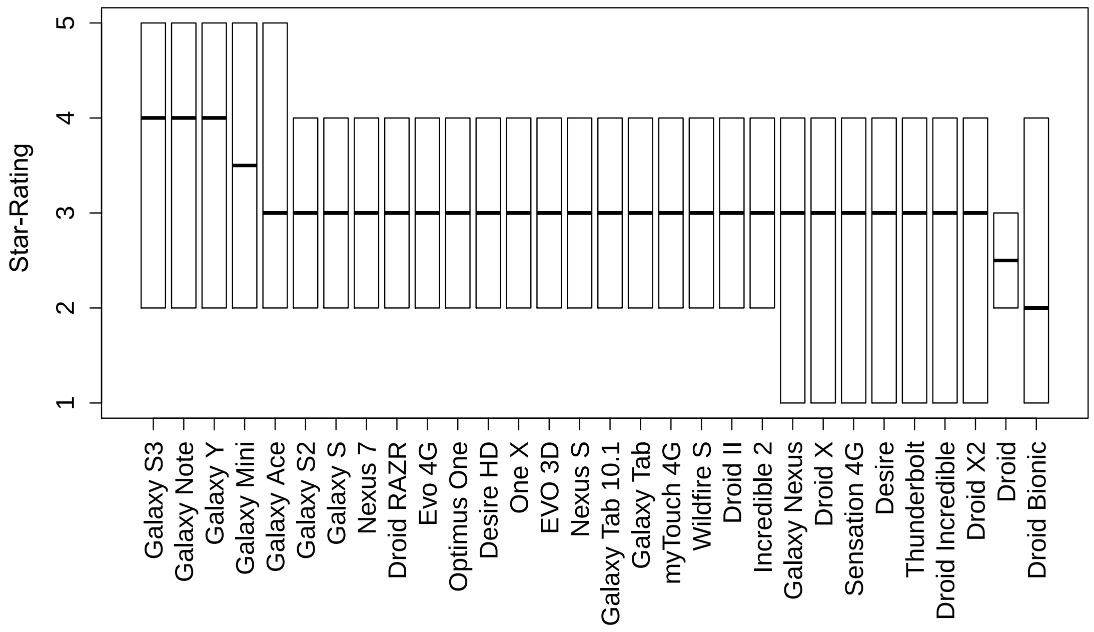


Figure 3.3: Given star-ratings by users of each device.

have a statistically significantly different distribution of star-ratings. In addition, by calculating the Cliff's delta effect size [38], we find that 42% of the device pairs have at least a small (effect size) difference. Figure 3.3 shows that the distributions of star-ratings vary across devices. Hence, we conclude that the devices on which the apps are running play a significant role in the received star-ratings.

Star-ratings of the same set of apps varies across devices. Users of some devices give higher star-ratings to mobile apps in comparison with users of other devices.

RQ3.2) Which device metrics share the most significant relationships with star-ratings?

Motivation. In the first research question, we found that users of some devices give higher star-ratings in comparison with users of other devices. However, a mobile device can be explained by various metrics, such as CPU, internal RAM, and screen size. In this research question, we explore the device metrics that share a significant relationship with star-ratings. The outcome of this research question helps both device manufacturers and mobile app developers. Mobile device manufacturers can figure out the device metrics that have the most significant relationships with star-ratings as estimated by their users. For example, manufacturers can decide whether it is better to allocate more resources to advance the screen resolution or the battery capacity for their next generation devices. Mobile app developers can understand the device metrics that have the most significant relationships with star-ratings. Therefore, app developers can determine the devices for which they would like to make their app available because Google Play Store permits developers to limit an app to specific devices.

Approach. We build a linear mixed effects model to study the device metrics that have the most significant relationships with star-ratings. The independent metrics of the model are the device metrics (as listed in Table 3.6) and the dependent metric is the star-ratings. We normalize the independent metrics so we can compare the coefficients.

Model Construction. We identify the correlated metrics because having correlated metrics negatively affects the stability of our model and prevents us from understanding the full impact of each metric [79]. We use the metric clustering analysis to build a hierarchical overview of the correlations between the explanatory metrics [84]. We choose the metrics that are more intuitive for inclusion in our model from each sub-hierarchy of metrics with Spearman's $|\rho| > 0.7$ [151]. Figure 3.4 shows the hierarchical clustering of the metrics according to Spearman's $|\rho|$. The Spearman correlation evaluates the monotonic relationship between continuous or ordinal metrics. In Figure 3.4, the horizontal line shows the threshold from which the metrics are highly correlated with each other. From each sub-hierarchy of correlated metrics, we select a representative independent metric to build our final model. We select the following metrics after the correlation analysis: OSVersion, CPU, DisplayInch, InternalROM, GPU, CameraMegaPixel, DisplayColors, DisplayPPI, OSUpgradable, and Standby.

Model Building. A mixed effects model is a model that contains both fixed effects metrics and random effects metrics [56, 216]. A fixed effects metric is treated with a constant coefficient and intercept for all the observations, while the coefficient and the intercept of a random effects metric can vary across individual observations [216]. By applying a mixed effects model, we can explain the relationships between the dependent metric and the independent metrics that are grouped according to one or more grouping factor(s). The mixed effects model can assume different intercepts or coefficients for each group [216].

Equation (3.1) shows the mixed effects model formula [155]. In Equation (3.1), g shows the grouping factor, n is the total number of metrics, and m is the number of metrics with fixed coefficients. Y_g holds the dependent metric. β_0 is the constant intercept and θ_{0g} is the intercept that varies across each grouping factor [155]. Let

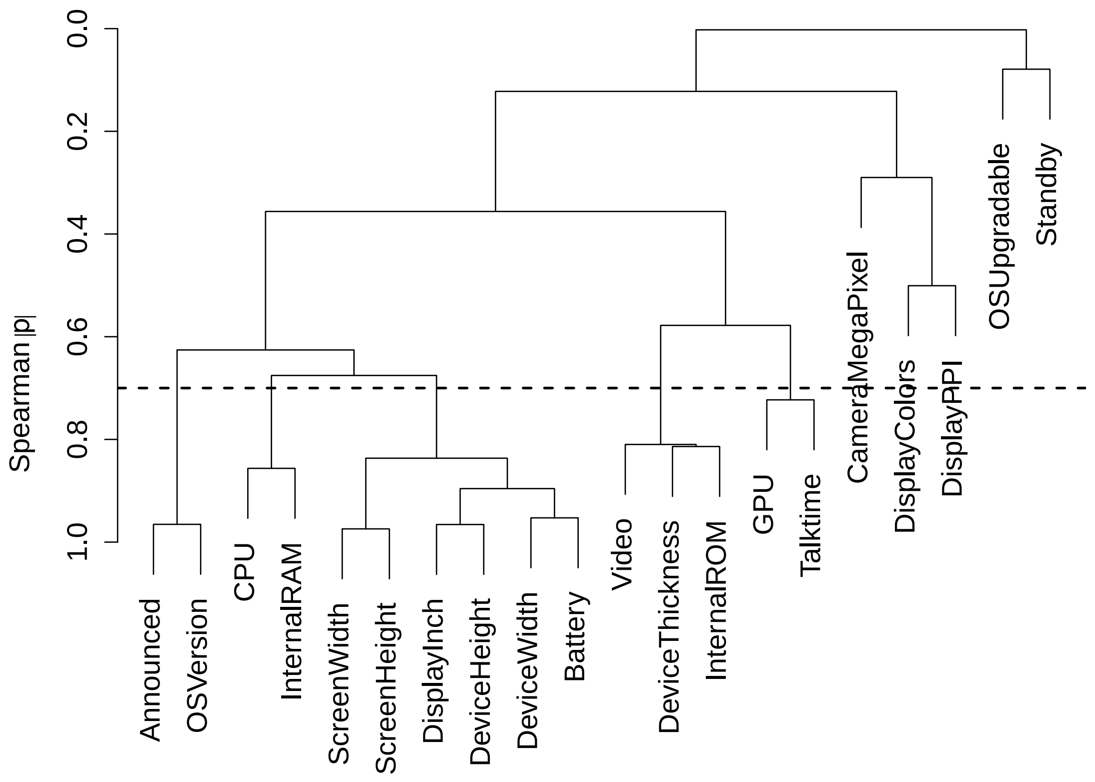


Figure 3.4: Hierarchical clustering of device metrics according to Spearman's $|\rho|$.

X_i denote the i_{th} independent metric; $\beta_i + \theta_{1g}$ and β_i indicate the coefficients of the X_i where θ_{1g} can vary across groups. ϵ_g is the indicator of errors. By having $\theta_{1g} = 0$, only the intercepts can vary. By setting $\theta_{0g} = 0$, only the coefficients can vary. By having both $\theta_{0g} = 0$ and $\theta_{1g} = 0$, the model would become a fixed effects model.

$$Y_g = \beta_0 + \theta_{0g} + \sum_{i=1}^{m} \beta_i X_i + \sum_{i=m}^{n} (\beta_i + \theta_{1g}) X_i + \epsilon_g \tag{3.1}$$

We let the independent metrics have variable intercepts (i.e., the intercept is a random effect) but with constant coefficients. We group our independent metrics according to each device and app. We build a mixed effects model with star-ratings as the dependent metric, the 20 device metrics as the independent metrics, and we let random effects for every app and device. Equation (3.2) shows the equation of our model. In Equation (3.2), Y_g denotes the star-ratings. X_i represents the independent metrics. β_i show the coefficients of each X_i and θ_g shows the intercepts that vary across each app and device.

$$Y_g = \beta_0 + \theta_g + \sum_{i=1}^{n} \beta_i X_i + \epsilon_g \tag{3.2}$$

Mobile apps may behave differently on different devices that are manufactured by different companies. Hence, we model the individual differences by assuming different random intercepts for each app and device. We assign a different intercept for each app and device and the mixed model estimates the intercepts [207].

To estimate the goodness of fit in a mixed effects model, two measures are used [147]: (i) marginal R^2 and (ii) conditional R^2. The marginal R^2 describes the proportion of the variance explained by the fixed effects metrics, while the conditional R^2 describes the proportion of the variance explained by both fixed and random effects metrics. The marginal R^2 of the model is 0.010 in this research question, but the conditional R^2 is 0.190. The higher value of the conditional R^2 indicates that the proportion of the variance that is explained by the fixed effects metrics is much less than the proportion of variance that is explained by both the fixed and the random metrics.

The significant metrics are marked with asterisks in the output of the model using the ANOVA test [168] (see Table 3.7). The significant metrics have $Pr(>|F|)$ —the $p-value$ that is associated with the F-statistic— less than 0.05. The upward arrows

Table 3.7: Linear mixed effects model using only device metrics for explaining star-ratings.

Metric	**Pr(>F)**		$\tilde{\chi}^2$	**Estimate**	
OSVersion	0.0007	***	27.07	1.410e-01 ± 0.153	↘
DisplayPPI	0.0035	**	8.55	7.476e-02 ± 0.026	↘
CPU	0.0162	*	17.21	7.661e-02 ± 0.126	↗
Standby	0.0263	*	4.94	3.011e-04 ± 0.001	↗
CameraMegaPixel	0.0907	+	2.86	1.385e-02 ± 0.008	↘
InternalROM	0.0915	+	2.85	1.482e-02 ± 0.009	↘
GPU	0.2865		1.14	1.945e-02 ± 0.018	↘
DisplayInch	0.4197		0.65	3.020e-02 ± 0.037	↗
DisplayColors	0.7548		0.56	4.225e-02 ± 0.096	↗
OSUpgradable	0.9335		0.01	4.075e-03 ± 0.049	↘

Codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '+' 0.1 ' ' 1

indicate the metrics with a positive relationship with star-ratings and the downward arrows indicate the metrics with a negative relationship. The values of the $\tilde{\chi}^2$ show if a model is statistically different from the same model in the absence of a given independent metric in the model according to the degrees of freedom.

Findings. Table 3.7 displays the results of the model. We identify that the OS version, the CPU, the display resolution, and the standby time as the metrics that have significant relationships with star-ratings. The CPU and the standby time have positive relationships with star-ratings, while the display resolution and the OS version have an inverse relationship with star-ratings. First, we discuss the metrics with a positive relationship with star-ratings. Then, we discuss the metrics with a negative relationship with star-ratings.

Device Metrics with a Positive Relationship with Star-ratings. We observe that the user of more powerful CPUs give higher star-ratings as the CPU is responsible for computations on a device. Standby is an upfront-derived specification that may not directly impact star-ratings. Standby time quantifies the longest time that a battery charge will last when the phone is constantly connected to the network but is not in an active use. One reason for standby having a positive relationship with star-ratings could be the energy consumption of mobile apps. One of the main energy consuming features of mobile apps is network usage [13, 115, 116]. Li *et al.* [116] reported that the network is the most energy consuming component among all the device components. Moreover, Kaup and Hausheer [101] observed that the user-perceived quality of an app can be affected by the data rates in networks and the energy consumption. Hence, if an app over-consumes energy, it would create an inconvenient user experience for a user that owns a device with low standby time. The users who can use their phones for a longer time may run the apps with a higher quality than the users who need to recharge their devices more often. For example, the following user-reviews denote the users' concerns in terms of battery consumption issues.

> “
> *Why is this app using 7% of my battery when not in use???*

> “
> *It is ok, but is takes up 18mb, not 4.5mb. Also, it massively drains the battery even when not in use and the phone is switched off. Uninstalled*

Device metrics with a Negative Relationship with Star-ratings. Table 3.7 shows that having a higher version of the OS has a significant inverse relationship with star-ratings. Moreover, the ability of a device to upgrade to a higher version of the OS has an inverse relationship with star-ratings too. One of the reasons for this observation can be the API updates. Updating the Android OS requires the apps to update their API usages [189]. Hence, new bugs might emerge. Such bugs are likely to impact star-ratings negatively as McDonnell *et al.* [138] show that many developers are not fast enough in adapting to new APIs and they often fail to catch up with the pace of Android API evolution. The following user complaints support this observation.

> “
> *...since upgrading to Android 4.2.1, the messages will not update correctly displaying the...”*

> “
> *After upgrade to ice cream sandwich, won't sync with Android Motorola bionic...*

The display resolution has an inverse relationship with star-ratings, i.e., the higher the display resolution, the lower the star-ratings. Android has the APIs that allow developers to precisely control the layout resources of the apps for different screen sizes [66]. To further investigate the relationship between the display resolution and star-ratings, we investigate the apps to see whether they have controlled their layout resources. 75%, 81%, and 93% of the apps in our dataset have a specific UI declaration for low-density, medium-density, and high-density screens, respectively. However, 63% and 99% of apps do not have a specific UI declaration for extra-high-density and extra-extra-high-density screens. Therefore, all the apps are not perfectly compatible with all the devices with different screen resolutions. The following example user-reviews support this observation.

> “ *Scrolling and loading are much improved, but please update display size for larger phones.*

> “ *...says my resolution isn’t high enough, then won’t let me tap a single button. Bad app.*

Some device metrics, such as the CPU, have significant relationships with star-ratings. However, having a better device characteristic, such as a higher display resolution, does not necessarily share a positive relationship with star-ratings.

RQ3.3) Do device metrics have a stronger relationship with star-ratings than app metrics?

Motivation. In this research question, we study and compare the relation of both device and app metrics with star-ratings. We aim to investigate whether device metrics have stronger relationships with star-ratings than app metrics. Therefore, app developers can target the devices on which they would like to make their app available. By limiting an app to specific devices, app developers can reduce the risk of receiving negative star-ratings from possibly underpowered devices. Moreover, developers would be able to prioritize the app testing efforts on certain devices [104].

Approach. We add the app metrics (as listed in Table 3.3) to the mixed effects model in addition to the device metrics (see Part 3.3).

Model Construction. Figure 3.5 shows the result of the hierarchical clustering of the metrics according to Spearman's $|\rho|$. Based on Figure 3.5, all the UI metrics, such as the number of inputs and outputs, the number of activities, and the Lines of UI Code (LOCUI), are correlated with each other. We select the total number of inputs as the representative metric for the group that contains the UI complexity metrics. From the other groups, we select WMC, OSUpgradable, Standby, APK-Size, LOC, OSVersion, CPU, DisplayInch, InternalROM, GPU, CameraMegaPixel, DisplayColors, and DisplayPPI.

Model Building. To determine the device and the app metrics that have statistically significant relationships with star-ratings, we build a linear mixed effects model [56]. The independent metrics are the combination of both the device and the app metrics. The marginal R^2 of the model is 0.013 and the conditional R^2 is 0.191. The higher value of the conditional R^2 indicates that the proportional of the variance that is explained by the fixed effects metrics is much less than the proportional of the variance that is explained by both the fixed effects and the random effects metrics.

Findings. Table 3.8 shows the output of the mixed effects model. We observe both the app and the device attributes have significant relationships with star-ratings.

The LOC is an app metric that has a significant relationship with star-ratings. This observation could be the result of the relationship between the app maintainability and the code size (the larger the code, the harder to maintain) [3, 117]. We also note that the WMC has a significant relationship with star-ratings. One possible explanation for this observation could be that more development effort is required to

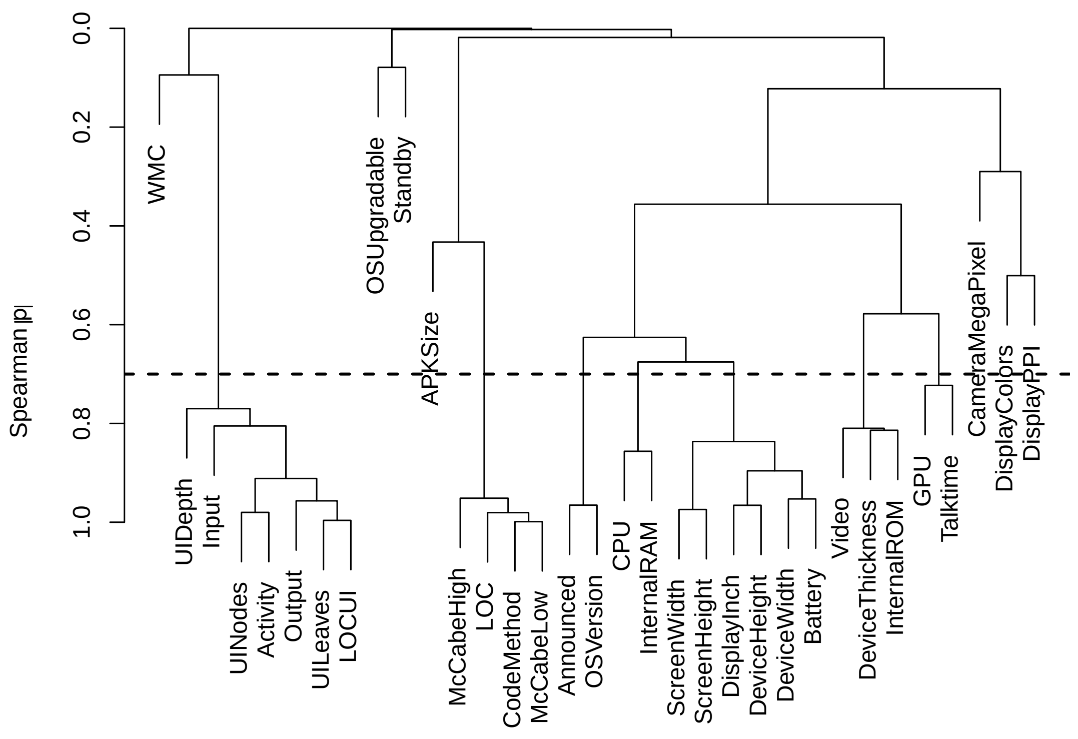


Figure 3.5: Hierarchical clustering of app and device metrics according to Spearman's $|\rho|$.

maintain more complex apps [14, 15].

The device metrics, such as the CPU and the screen resolution, play a significant role in star-ratings too. In other words, the characteristics of both the apps and the devices are statistically significant related to star-ratings. The LOC (an app metric) and the OS version (a device metric) both share significant relationships with star-ratings.

Table 3.8: Linear mixed effects model using app and device metrics for explaining star-ratings.

Category	**Metric**	**Pr(>F)**		$\tilde{\chi}^2$	**Estimate**	
App	**LOC**	1.326e-09	***	36.90	3.343e-02 ± 0.0050	↘
	WMC	4.892e-05	***	16.51	1.129e-02 ± 0.0003	↘
	Input	0.0205	*	5.37	2.150e-02 ± 0.0093	↗
	APKSize	0.2710		1.21	5.619e-03 ± 0.0051	↗
Device	**OSVersion**	0.0005	***	27.78	1.406e-01 ± 0.1253	↘
	DisplayPPI	0.0025	**	9.13	7.688e-02 ± 0.0254	↘
	CPU	0.0136	*	17.73	7.638e-02 ± 0.1258	↗
	Standby	0.0276	*	4.86	2.971e-04 ± 0.0001	↗
	CameraMegaPixel	0.0778	+	3.11	1.438e-02 ± 0.0008	↘
	InternalROM	0.0829	+	3.00	1.516e-02 ± 0.0087	↘
	GPU	0.2876		1.13	1.932e-02 ± 0.0182	↘
	DisplayInch	0.4301		0.62	2.939e-02 ± 0.0372	↗
	DisplayColors	0.7300		0.63	4.632e-02 ± 0.0952	↗
	OSUpgradable	0.9071		0.01	5.666e-03 ± 0.0486	↘

Codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '+' 0.1 ' ' 1

Some app and device metrics share significant relationships with star-ratings. However, some device metrics, such as the CPU and the display resolution, share stronger relationships with star-ratings than some app metrics, such as the number of inputs in the UI.

3.4 Threats to Validity

In this section, we discuss the potential threats to the validity of this chapter.

3.4.1 Construct Validity

Threats to construct validity concern the degree to which the study represents the constructs in the real world [184]. Some device metrics, such as 2G or 3G connectivity support, may not be available in practice if such networks are not accessible from a specific location. In this study, we consider the metrics that are available in a non-controlled empirical study.

3.4.2 Internal Validity

Threats to internal validity concern our selection of subject systems and analysis methods [184]. We select a set of apps with at least five star-ratings for each of the 30 devices. The threshold of five star-ratings mitigates the risk of our results being skewed by the set of apps and devices that have a small number of star-ratings [103]. Finally, for each research question, we dug into the data to provide some context to our findings (e.g., the maintainability of a large code). We are not claiming an exhaustive list of relationships between the suggested causes and star-ratings.

3.4.3 External Validity

Threats to external validity concern the possibility of generalizing our results. Our study is performed on free apps. Hence, our results may not generalize to paid apps. We focus on free to download apps as it reduces the impact of other confounding factors, such as the variance of user expectations based on the cost of an app [77]. Intuitively, certain types of apps, in particular, games which are traditionally hardware taking, may be more dependent on the device metrics. We add an interaction term to allow the coefficients in our model to vary depending on whether we are modeling star-ratings of a game app or a non-game app. We found that the interaction terms for the GPU and the OS Version are statistically significant. Hence, games might be more dependent on the GPU and the OS Version of devices than other apps. However, we could not improve our model fit. Therefore, we further built two separate models: (i) one model for game apps and (ii) another model for non-game apps. Again, we could not improve the model fit indicating that a *general* model of star-ratings performs better than more specific models, such as the model that is built only based on game apps. Our inability to improve on the model fit may be due to (i) some games might be more dependent on device metrics than other games (e.g., a complex graphics-intensive game compared to a simpler game) or (ii) star-ratings does not vary by app type.

3.5 Summary

In this chapter, we identify the app and the device metrics that share a statistically significant relationship with star-ratings. Our analysis is based on 30 Android mobile devices, 280 Android apps, and $150,373$ star-ratings. App metrics, such as LOC, share significant relationships with star-ratings. However, to succeed in the competitive market of mobile apps, app developers should carefully consider the available devices on the market too. We observe that the star-ratings of an app vary across different devices. Device metrics, such as the CPU and the screen resolution, have significant relationships with star-ratings. However, having better device specifications, such as a higher display resolution, does not always share a positive relationship with star-ratings.

Chapter 4

Key Topics of User-Reviews

Each user-review that is posted on Google Play Store is associated with a star-rating. In this chapter, we look into the user-reviews to find the topics of user-reviews that share a statistically significant relationship with star-ratings (i.e., key topics). Given the importance of high star-ratings to the success of an app, it is crucial to help developers find the key topics of user-reviews in each category. By identifying the key topics of user-reviews, app developers could narrow down their development efforts to the user-reviews that matter for receiving higher star-ratings.

An introduction to this chapter is provided in Section 4.1. Section 4.2 describes the study setup of this chapter. Section 4.3 discusses the key topics identification method and the identified key topics. In Section 4.4, we evaluate the relationship between the key topics and star-ratings. Section 4.5 lists the potential threats to the validity. Finally, we conclude this chapter in Section 4.6.

4.1 Introduction

A star-rating can be associated with a user-review in Google Play Store [69]. A user-review is a piece of informal text without a predefined structure [160]. User-reviews could potentially contain useful information, such as bug reports, feature requests, and user experiences [59, 160]. Developers can use the information that is reported in user-reviews to improve their apps in terms of fixing bugs, adding new features, and catching up with the latest demands of users. Recent studies (e.g., [60, 59, 159, 160, 163]) have shown the unavoidable importance of star-ratings and user-reviews in success and profitability of apps.

Some recent studies aimed to summarize user-reviews and classify them into meaningful groups [135]. For example, Villarroel *et al.* [203] tried to help app developers in release planning by classifying user-reviews into clusters of bug reports and feature requests. Panichella *et al.* [163] classified user-reviews to help app developers maintain their apps. Palomba *et al.* [160] showed the relation between user-reviews and code changes towards app success. Iacob *et al.* [90] summarized the topics of user-reviews which cover users' complaints. Recent work is subject to the context of a single or a few apps. None of the earlier work specified a set of topics with a broader view, i.e., category, that are statistically significantly related to star-ratings, while every category of mobile apps shares a specific set of characteristics and requirements [128]. In this chapter, we show that addressing all the issues that are reported in user-reviews do not necessarily increase the star-ratings of mobile apps. Fixing a frequently reported issue can have a minor effect on star-ratings.

We introduce the *key topics*, which are the topics of user-reviews that share a significant relationship with star-ratings. For example, we identify *messages* as a key topic of the category of *social*. The following example user-reviews show the importance of *messages* in the category of *social*:

“ *Love this app. Great to be able to check up on the site for my messages when I'm on the go!*

“ *Wouldn't let me read my messages so I uninstalled it [...]*

“ *It wont let me recieve messages or send them fix it and i'll rate 5 star again*

The issues that are related to the key topics are statistically significantly related to star-ratings. Developers can use our approach on top of previous work, such as Villarroel *et al.* [203] and Palomba *et al.* [160], to better manage the user-reviews. It may be time-consuming to address all the issues that are reported in user-reviews. Therefore, it is beneficial to understand and consider the key topics for better managing time and efforts to receive higher star-ratings and succeed in the competitive market of mobile apps.

For each category, we identify a set of key topics that have a significant relationship with star-ratings through analyzing the contribution of each topic to the performance of the model that is built with the star-ratings as the dependent metric. For example, we find that *speed*, *battery consumption*, and *searching* are the key topics of the category of *travel and local*. Moreover, we observe that the most frequent topics of each category are not necessarily the key topics of the same category, i.e., most frequent topics do not always share a significant relationship with star-ratings.

We investigate a large number of user-reviews, i.e., $4,193,549$ user-reviews of 623 apps in ten different categories. With such a number of user-reviews, we conduct a fine-grained analysis per topic on the user-reviews and associated star-ratings. Considering a large number of user-reviews mitigates bias in the data [134].

We evaluate the identified key topics with respect to the the reported changes in the apps for 19 months. Release notes could be a great indicator of the changes that are made to each app. Johann *et al.* [96] reported a precision of up to 88% for app descriptions in identifying app features. For each app, we calculate the similarity score of the release notes with key topics versus non-key topics. We observe that, for 77% of the subject apps, higher star-ratings are received when release notes are more similar to the key topics.

4.2 Study Setup

Figure 4.1 shows an overview of the study setup process. As shown in Figure 4.1, we preprocess the user-reviews and release notes of each app. Then, we identify the topics of each user-review. After that, we compute the metrics of the topics and identify the key topics of each category. Finally, we evaluate the changes in star-ratings with respect to the release notes.

4.2.1 Data Source

We retrieve the user-reviews, app details, and release notes from Google Play Store. Due to the processing time for handling all the user-reviews in over thirty categories of Google Play Store, we study ten categories. We randomly selected ten categories to avoid introducing a bias towards a specific set of categories, e.g., popular categories. The selected categories are listed in Table 4.1. The distribution of the number of user-reviews varies across the categories. Some categories, such as *communication*

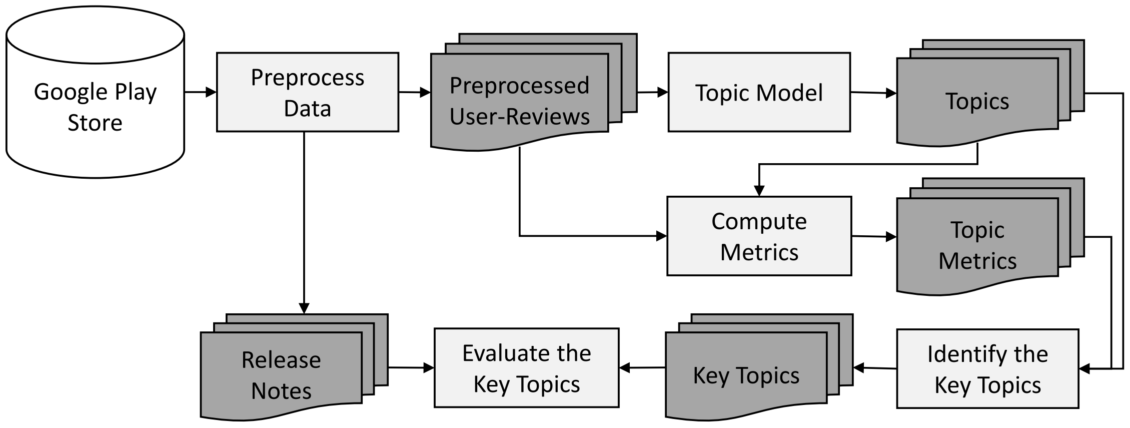


Figure 4.1: Overview of study setup for identifying the key topics of user-reviews.

and *social*, have a variety of popular apps and users. Some other categories, such as *travel and local*, have a fewer number of user-reviews.

We run our implemented crawler (see Part 1.4.1) to extract app details and the associated user-reviews on a daily basis from *April* 2014 to $November$ 2015. Initially, we retrieved $14,241,915$ user-reviews for $2,723$ apps.

The number of Events Per Variable (EPV) is a metric that calculates the ratio of data points to the number of variables [192]. To avoid the risk of over-fitting and having unstable results, having an EPV ≥ 10 is recommended [192]. To ensure that we have enough data to study the evolution of star-ratings over time, we exclude $2,100$ apps based on two criteria:

(i) The apps updated less than 10 times during our study [192] as we study the relation of addressing the issues raised by the key topics with star-ratings over time.

(ii) The apps that receive less than 10 user-reviews between each update to avoid the apps with few user-reviews skew the findings [103, 192].

Table 4.1: Distribution of categories that are studied to identify the key topics of user-reviews.

Category	Apps	User-Reviews	Consistent User-Reviews	Informative User-Reviews
Business	42	277,564	187,343	102,375
Communication	70	3,055,079	1,972,061	950,007
Health and Fitness	57	342,003	249,534	171,220
Media and Video	44	1,020,481	651,444	342,610
Photography	56	827,870	631,629	274,611
Productivity	67	1,271,049	925,077	438,505
Shopping	63	660,784	484,433	293,943
Social	99	2,739,505	1,712,657	953,311
Tools	76	1,476,290	1,019,811	477,804
Travel and Local	49	435,628	293,355	189,163

The numbers of selected apps in each category are listed in Table 4.1. By removing $2,100$ apps, the number of user-reviews decreased from $14,241,915$ to $12,106,253$ (i.e., 15% of the initial user-reviews excluded from our study). The number of releases for all the apps is illustrated in Figure 4.2.

4.2.2 Preprocessing Data

First, we preprocess the data (e.g., applying natural language processing techniques) (see Part 1.4.2). We remove $468,473$ user-reviews (i.e., 4%) that were written in a non-English language. We identify $8,127,344$ consistent user-reviews out of $11,137,780$ user-reviews (i.e., 73% of all the retrieved user-reviews). The total number of the user-reviews and the number of consistent user-reviews are listed in Table 4.1. Then, we identify $4,193,549$ informative user-reviews (i.e., 52% of the consistent user-reviews). The last column in Table 4.1 shows the number of informative user-reviews.

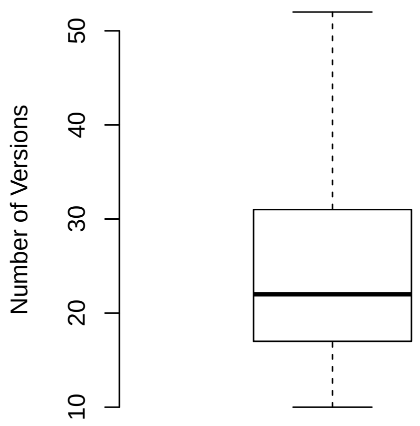


Figure 4.2: Number of versions for all apps that are studied for identifying the key topics of user-reviews.

Building Corpus

A text corpus is a set of texts for statistical analysis, such as topic modeling [133]. To build the corpus of user-reviews, we normalize the user-reviews before applying topic modeling approaches, following the steps below:

(i) *Removing Stop Words.* Stop words are referred to as the most common words in a language, such as *"been"* and *"the"*. By removing the stop words, we can focus on the important words and obtain topics more accurately. Stop words are usually filtered out before applying the natural language processing techniques [172].

(ii) *Removing Punctuations.* We put aside the punctuations in the user-reviews, such as commas and semicolons. Hence, we can focus on the words of the user-reviews to identify the topics of the user-reviews.

(iii) *Stemming Words.* Stemming [126] is the process of reducing inflected words to their word stem. Having the words stemmed reduces the inflectional forms of the word to a common base form. Hence, all forms of a word can contribute equally to the same word (i.e., the stem). For instance, *"run"*, *"runner"*, and *"running"* have the same word stem that is *"run"*.

We also normalize the release notes to better compare them with the identified topics and evaluate the key topics.

4.2.3 Topic Modeling

Topic modeling techniques let us assign topics to each user-review, so we can understand the topics of each user-review. We use the Latent Dirichlet Allocation (LDA) technique [26] to capture the topics of user-reviews. LDA allows sets of observations to be described by unobserved groups and can determine the similar parts of data. LDA can process a large number of user-reviews to identify the topics of user-reviews. Latent parameters, such as the number of topics, need to be set up first. The most important parameters of the LDA are:

- α is set up according to the distribution of topics.
- β is the parameter of the word distribution for each topic.
- θ_d is the distribution of the topics for a document d.
- ϕ_t is the distribution of the words that describe the topics t.
- t_{wd} is the topic for the w_{th} word in document d.
- $|N|$ is the number of topics.
- $|W|$ is the number of words in the vocabulary.

- $|w|$ is the number of words in each document.

- $|W_d|$ is the total number of words in all documents.

More details on the LDA parameters can be found in the work of Blei *et al.* [26]. Panichella *et al.* [162] proposed an approach to determine the optimal configuration of LDA for software engineering tasks. Most of the parameters are calculated automatically with respect to the distribution of data (i.e., words, documents, and topics) and the input parameters using JGibbLDA [167] (i.e., the tool that we use to identify the topics). JGibbLDA [167] does the parameter estimations using related work [7, 25, 26, 71, 81]. We employ two approaches (i.e., Griffiths *et al.* [71] and Cao *et al.* [30]) to identify the optimum number of topics. The method of Griffiths *et al.* [71] compares the likelihood of the topic models for each number of topics. The best number of topics has the highest likelihood value. Cao *et al.* [30] investigate the optimum number of topics in LDA based on the topic density. The approach of Griffiths *et al.* suggests 15 as the optimum number of topics. However, the approach of Cao *et al.* suggests 35 as the optimum number of topics. Hence, the best guess would be between 15 and 35. To avoid losing any potential topics, we pick a safe approach by choosing the maximum of the suggested numbers of topics (i.e., 35).

We apply LDA to our corpus to find the topics of the user-reviews with $2,000$ Gibbs sampling iterations [171]. Although more iteration is better, with $2,000$ iterations, we can achieve a high accuracy with topic modeling [71, 171]. The Gibbs sampling is a Markov chain Monte Carlo algorithm [62] to acquire a sequence of observations. The observations are approximated by a specified multivariate probability distribution [72]. We manually analyze the identified topics with a help of an external evaluator who was a graduate student in software engineering. We discussed each topic with the external evaluator until we reach an agreement. We merge the topics that have the same or very close semantic meaning based on the words given by the LDA and the associated user-reviews. We merged the topics having considered the following criteria:

- Each topic is defined by a set of words. We compare the words of each topic (especially the words appearing more frequently for each topic). If the words have the same semantic meaning, we merge them into one group.
- We also look into the user-reviews that are associated with each topic, especially if the words of a topic were not clear enough to understand the meaning of the topic. If the user-reviews of each topic explain the same concept, we merge them together.

We list the final 23 topics in Table 4.2 followed by the five most frequently appearing words from the output of LDA and a brief description of each topic. Table 4.3 shows the frequency of topics in each category.

4.2.4 Computing Metrics

We follow the *Goal / Question / Metric* (GQM) paradigm [16, 199] to capture the metrics. The GQM 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. We bring up nine questions to quantify the user-reviews and the corresponding topics. Then, we identify the metrics that can be measured to

Table 4.2: Identified topics of user-reviews.

#	Topic Name	Top Words	Brief Description
T_1	**Advertisements**	ad, screen, remove, annoy, button	Advertisement-related issues are along with this topic.
T_2	**Authentication Issues**	account, log, wrong, check, password	More related to authentication process of an app.
T_3	**Battery Consumption**	battery, save, life, power, charge	More about the issues of energy and battery consumption.
T_4	**Bug Reports**	star, problem, only, poing, issue	Users explain their problem, specially the bugs.
T_5	**Comparing Versions**	update, version, latest, free, upgrade	Discussing different versions, such as free vs. full version.
T_6	**Connection**	call, voice, connect, contact, number	The words are about calls and internet connections. quality.
T_7	**Device Compatibility**	phone, android, mobile, device, tablet	Comparisons based on different devices are along with this topic.
T_8	**Feature Requests**	better, improve, perform, add, feature	The verbs, such as “add”, are in this topic to request a feature.
T_9	**Language Support**	language, translate, English, speak	Users discuss language issues, such as adding a specific language.
T_{10}	**Messages**	post, message, view, text, receive	The words are more related to sending and receiving messages.
T_{11}	**Pictures**	photo, picture, edit, share, view	Related to viewing, sharing, and processing pictures.

Table 4.2: Identified topics of user-reviews (continued).

#	Topic Name	Top Words	Brief Description
T_{12}	**Playing Audio & Video**	video, play, watch, player, quality	The words are related to watching and playing audio & video.
T_{13}	**Praising Features**	well, perfect, proper, interesting, pretty	User praise various features.
T_{14}	**Purchases**	deal, money, price, store, shop	The words of this topic deal with purchasing and monetary issues.
T_{15}	**Repeating Issues**	time, every, day, try, long	Adverbs of time can be seen in this topic.
T_{16}	**Searching**	google, search, map, find, locate	Searching is the main concern in this topic.
T_{17}	**Social Networking**	people, meet, talk, chat, friend	About connecting users to other people, such as friends.
T_{18}	**Speed**	slow, load, quick, fast, speed	More related to speed of an app, a process, or a functionality.
T_{19}	**Storage**	file, data, manage, space, card	The words are more about storage management.
T_{20}	**Task Tracking & Notifications**	help, track, activity, list, sync	The words are related to tracking reminders, and notifications.
T_{21}	**Technical Support**	guy, support, hope, team, job	User discuss and share issues regarding technical supports.
T_{22}	**User-Interface**	screen, button, bar, interface, theme	Issues and experiences about user-interface.
T_{23}	**Web Browsing**	browser, open, speed, load, slow	The words in this topic are more related to Web browsing issues.

Table 4.3: Distribution of topics in different categories.

#	**Business**	**Communication**	**Health and Fitness**	**Media and Video**	**Photography**	**Productivity**	**Shopping**	**Social**	**Tools**	**Travel and Local**
$\boldsymbol{T_1}$	4%	4%	2%	4%	3%	3%	2%	4%	4%	2%
***T*2**	2%	4%	1%	1%	1%	2%	2%	3%	1%	2%
***T*3**	2%	1%	1%	1%	1%	18%	3%	1%	9%	2%
***T*4**	2%	2%	1%	2%	3%	2%	3%	4%	3%	4%
***T*5**	6%	4%	2%	4%	2%	3%	3%	5%	6%	4%
***T*6**	5%	14%	2%	6%	5%	4%	3%	7%	7%	3%
***T*7**	8%	3%	2%	4%	3%	8%	2%	3%	7%	2%
***T*8**	10%	4%	2%	3%	5%	5%	2%	2%	4%	3%
***T*9**	1%	2%	1%	2%	3%	2%	1%	2%	3%	1%
***T*10**	4%	12%	1%	2%	2%	2%	2%	10%	2%	2%
***T*11**	2%	1%	1%	1%	26%	1%	1%	4%	1%	1%
***T*12**	1%	2%	1%	31%	4%	1%	1%	4%	2%	1%
***T*13**	7%	7%	5%	6%	12%	8%	9%	11%	9%	5%
***T*14**	1%	1%	1%	1%	1%	1%	40%	1%	1%	6%
***T*15**	1%	1%	3%	1%	2%	2%	3%	2%	2%	1%
***T*16**	1%	1%	2%	1%	1%	1%	3%	1%	2%	33%
***T*17**	2%	8%	1%	2%	3%	2%	3%	14%	2%	2%
***T*18**	6%	5%	5%	6%	5%	5%	8%	8%	6%	9%
***T*19**	8%	1%	1%	2%	2%	7%	1%	1%	5%	1%
***T*20**	12%	4%	60%	3%	6%	12%	6%	4%	7%	9%
***T*21**	5%	3%	2%	3%	3%	4%	2%	2%	5%	3%
***T*22**	6%	6%	4%	5%	5%	5%	2%	6%	7%	4%
***T*23**	3%	11%	1%	13%	3%	3%	2%	2%	5%	2%

Table 4.4: GQM model to quantify metrics of user-reviews for identifying the key topics of user-reviews.

Goal: Quantifying the user-reviews and the associated topics.		
Questions	**Metric**	**Count**
How often each app gets released?	Number of releases	1
What is the estimated amount of information in each user-review based on the identified topics?	Entropy of topics	1
How well is the users' experience with each topic?	Sentiment scores	6
How many users discuss the same topic?	Number of user-reviews	2
What is the proportion of positive and negative user-reviews?	Proportion of user-reviews	2
How much information can be extracted from the user-reviews?	Number of words and sentences	12
What is the proportion of each topic in the user-reviews?	Proportion of topics	1
How often does each topic appear in the user-reviews?	Topic recurrence length	1
For how long does each topic appear in the user-reviews?	Duration of topics	1
	Total:	27

address the questions. We measure 27 metrics of user-reviews. Table 4.4 shows the GQM model and the list of the metrics.

Number of Releases

Khalid *et al.* [105] reported that many users complain about app updates. For each app, we count the number of releases as a control variable during the period of our study (i.e., *April* 2014 to *November* 2015).

Entropy of the Topics

We use Shannon entropy [122, 183] to measure the probability of topics appearing in user-reviews of an app using Equation (4.1).

$$H(a) = -\sum_{i=1}^{n} Pr_a(T_i) \cdot log(Pr_a(T_i)) \tag{4.1}$$

In Equation (4.1), $H(a)$ represents the entropy of an app a. n shows the number of topics. T_i demonstrates the i^{th} topic. $Pr_a(T_i)$ shows the probability of the occurring topic T_i in the user-reviews of the app a. We compute $Pr_a(T_i)$ using Equation (4.2).

$$Pr_a(T_i) = \frac{\eta_a(T_i)}{\eta_a} \tag{4.2}$$

In Equation (4.2), η_a shows the number of user-reviews for an app a, and $\eta_a(T_i)$ represents the number of user-reviews with the i^{th} topic for the same app.

Sentiment Score of the Topics

We use SentiStrength [193, 194] to measure the sentiment scores of the topics for each user-review (see Part 1.4.2). SentiStrength scores the user-reviews by a quantitative value between -5 and $+5$ from the most negative to the most positive. For each app, we measure the statistical characteristics of the sentiments scores, including the mean, median, minimum, maximum, 1^{st} quartile, and 3^{rd} quartile of the sentiment scores of the user-reviews per topic. Moreover, we calculate the number of negative and the number of positive user-reviews. We also count the proportion of negative and positive user-reviews per topic for each app.

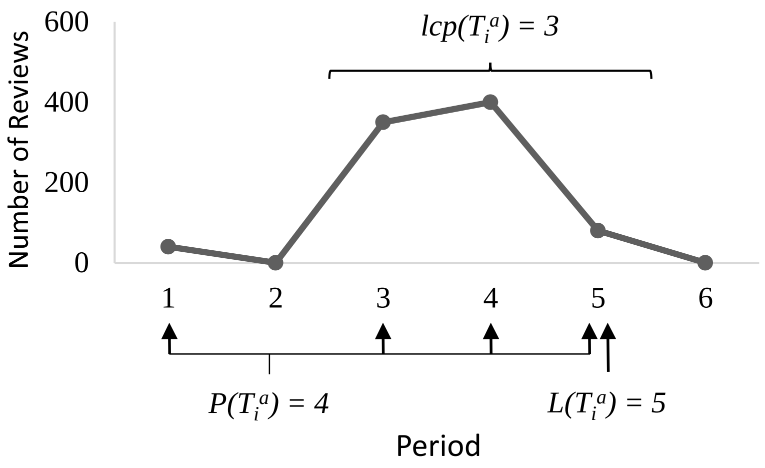


Figure 4.3: An example for computing *topic recurrence length*.

Number of Words and Sentences

The sizes of the user-reviews can implicitly show the helpfulness and the mood of user-reviews [108]. As the number of words and sentences increases, users might give more information about a specific topic. To calculate the size of user-reviews, we count the number of words and the number of sentences of each user-review. We use Stanford Parser [48] and Stanford Tokenizer [132] to count the number of words and sentences. For each topic, we measure the mean, median, minimum, maximum, 1^{st} quartile, and 3^{rd} quartile of the number of words and sentences of the user-reviews.

Proportion of the Topics

Some topics frequently appear in a specific category1 and other topics appear infrequently. For example, in the category of *shopping*, users talk more about *buying and purchasing* than *watching videos*. To capture the varying popularity of a topic, we measure the proportion of each topic for each app.

Topic Recurrence Length

A topic can be hot in the past but off subject now. Users may write reviews for some topics frequently but rarely for other topics. Inspired by prior studies [82, 191], we introduce the *topic recurrence length* metric to capture the consecutive occurrences of each topic. We use the following equation to measure the topic recurrence length of each topic for each app:

$$TRL(T_i^a) = lcp(T_i^a) \cdot e^{\frac{1}{n}((P(T_i^a)+L(T_i^a))} \tag{4.3}$$

For each app, we break the 19 months duration of our study into the periods between each release. In Equation (4.3), T_i^a shows the i^{th} topic from the user-reviews of an app a. n is the total number of periods between the releases of an app a during the period of our study. lcp is the longest consecutive period of T_i^a with at least one user-review between each release. $P(T_i^a)$ is the number of periods with at least one user-review for a topic T_i^a. $L(T_i^a)$ is the last index of the longest period.

We give an exponential effect to $P(T_i^a)$ as the emphasize of TRL is on the recurrence of the topics. As the recurrence of a topic increase, TRL grows exponentially. Similarly, we gave $L(T_i^a)$ the same effect. Figure 4.3 shows an example for computing the topic recurrence length. Suppose that we want to calculate the topic recurrence length of the i^{th} topic for an app a. Presume this app has updated and released 5 times during our study. Then, the number of periods is 6. Let us assume the distribution of the number of user-reviews for each period is $\{40, 0, 350, 400, 80, 0\}$. For this distribution, the longest consecutive period $lcp(T_i^a) = 3$, the number of periods with at least one user-review $P(T_i^a) = 4$, and the last index of the longest period is 5. Consequently, $TRL(T_i^a)$ (i.e., the topic recurrence length) equals to 13.45 using equation (4.3). A greater value of TRL shows that the topic had appeared more often and was on the matter of user-reviews for a longer time. For example, a topic with a $TRL = 13.45$ (as in the example above) has recurred more often than a topic having a $TRL = 10$.

Duration of the Topics

During the lifetime of an app, some topics might appear for a specific time and then disappear from the user-reviews. For example, if an app suffers from battery issues, the users might comment intensively on battery consumption problems. After the resolving the problem by developers, users may not complain about the battery issues anymore. Therefore, this topic would disappear from the user-reviews. To measure the duration of each topic, we calculate the difference between the earliest and the latest time of a continuous period with at least one user-review regarding the same topic.

4.3 Key Topics Identification

In this section, we describe the key topics identification method and the identified key topics.

4.3.1 Method

Google Play Store reports the average of star-ratings as an indicator of the overall star-rating of an app [69] (see Figure 1.1). Therefore, for each app, we investigate the relationship between the average of star-ratings and the identified topics, such as *user-interface* and *battery consumption*. The independent metrics are listed in

Table 4.2.

Having correlated metrics affects the stability of our models negatively and makes it difficult to distinguish the impact of the metrics [79]. We use the variable clustering analysis technique according to Spearman's $|\rho| > 0.7$ [151] to investigate the correlation between the explanatory metrics [84]. The Spearman correlation evaluates the monotonic relationship between the continuous or ordinal metrics.

Proportional Marginal Variance Decomposition (PMVD) is an approach that is based on sequential R^2s. It averages over orderings by using weighted averages with data-dependent weights. Therefore, PMVD mitigates the risk of dependence on the orderings of the metrics of topics. We apply PMVD [58, 73] to detect the topics that hold the highest contributions to star-ratings.

For each topic, we measure a set of metrics as explained in Part 4.2.4. Using PMVD, we group the metrics of each topic together. Therefore, PMVD would calculate the contribution of each topic (i.e., each group of metrics) on star-ratings. For simplicity, we explain the calculation of PMVD for $m_i^{t_j}$ (i.e., i^{th} metric of j^{th} topic) in Equation (4.4). In this Equation, s is the set of understudy metrics and n denotes the number of metrics in set s. r is a permutation of metrics of set s. $\omega(m_i^{t_j}|s)$ is the additional R^2 when adding the $m_i^{t_j}$ the model as shows in Equation (4.5). $\sigma(s)$ is the set of all $n!$ permutations of metrics of set s. $\rho(s)$ denotes the data-dependent PMVD weights that is calculated using Equation (4.6) [73].

$$PMVD(m_i^{t_j}) = \frac{1}{n!} \sum_{\sigma(s)} \rho(r)\omega(m_i^{t_j}|s) \tag{4.4}$$

$$\omega(m_i^{t_j} s) = R^2(m_i^{t_j} s) - R^2(s) \tag{4.5}$$

Equation (4.6) shows an overview of computing the PMVD weights. The weights have been derived from a set of axioms (see Feldman [58]), such as the axiom of proper exclusion. The axiom of proper exclusion makes an independent metric with a coefficient 0 to have an R^2 share of 0 [73]. In Equation (4.6), $\sigma(s)$ is the set of all $n!$ permutations of metrics of set s. r is a permutation of metrics in set s. $\rho(r)$ shows the weight for a list of metrics r. $\Gamma(s)$ can be computed using Equation (4.7) for a permutation r of the metrics of set s. In Equation (4.7), m_{ri} is the i^{th} metric of a permutation r of set s.

$$\rho(r) = \frac{\Gamma(r)}{\sum_{\sigma(s)} \Gamma(s)} \tag{4.6}$$

$$\Gamma(r) = \prod_{i=1}^{n-1} \omega(m_{ri+1}, ..., m_{rn} | m_{r1}, ..., m_{ri})^{-1} \tag{4.7}$$

Using PMVD allows us to identify the likelihood of independent metrics that are correctly ordered with respect to their relative importance. We compute the expected contribution of all the metrics of each topic to the model performance. We group the metrics of each topic together. PMVD gives the expected contribution of each topic based on its group of metrics. We mark a topic as a key topic when the expected contribution of the metrics that describe the topic (i.e., the PMVD score) is more than 0.05 [73].

4.3.2 Findings

The identified key topics are listed in Table 4.5. As shown in Table 4.5, each category has its own set of key topics that share a significant relationship with star-ratings. For example, the key topics of the category of *business* are *battery consumption*, *speed*,

Table 4.5: List of the key topics of each category.

Category	Key Topics	Count
Business	Battery Consumption (0.33), Speed (0.31), Device Compatibility (0.25)	3
Communication	Searching (0.58), Battery (0.11), Speed (0.08)	3
Health and Fitness	Task Tracking and Notifications (0.43), Messages (0.21), Speed (0.08), Battery Consumption (0.05)	4
Media and Video	Playing Audio & Video (0.71), Speed (0.19), Repeating Issues (0.06)	3
Photography	Pictures (0.43), Advertisement (0.24), Searching (0.06), User-Interface (0.06), Speed (0.05), Battery Consumption (0.05)	6
Productivity	Pictures (0.20), Advertisements (0.12), Authentication Issues (0.12), Repeating Issues (0.07), Task Tracking and Notifications (0.07)	5
Shopping	Searching (0.59), Bug Reports (0.10), Purchases (0.07)	3
Social	Messages (0.15), User-Interface (0.08), Comparing Versions (0.08), Social Networking (0.06), Pictures (0.06), Searching (0.05)	6
Tools	Battery Consumption (0.24), Speed (0.23), Bug Reports (0.17), Language Support (0.12)	4
Travel and Local	Searching (0.21), Battery Consumption (0.19), Advertisements (0.13), Speed (0.08)	4

and *device compatibility*. Some categories, such as *business*, have fewer key topics in comparison with other categories. The categories of *business*, *communication*, and *shopping* have only three key topics each, while the categories of *photography* and *social* have the most number of key topics with six key topics.

App developers should take care of the issues that are related to the key topics of the category in which they publish their apps to achieve higher star-ratings. App developers can also refer to related work, such as Chen *et al.* [33], Villarroel *et al.* [203], and Di Sorbo *et al.* [52], to identify the bug reports and feature requests that are reported in the user-reviews of their own apps. Following our findings, they can focus on the user-reviews that are related to the key topics to better manage time, budget, and efforts.

We made some additional interesting observations, as listed below, that can further help app developers:

(i) For each category, some key topics have a higher relationship with star-ratings in comparison with the other key topics. PMVD scores are listed in parentheses in Table 4.5. PMVD scores are between 0 and 1 [73]. A higher value of a PMVD score shows that the metrics of the topic have a higher contribution to the performance of the model. For example, in the category of *business*, *battery consumption* and *speed* are relatively more important than the issues related to *device compatibility*. Hence, developers would be able to prioritize the key topics.

(ii) The key topics are not *necessarily* the most *frequent* topics of the user-reviews (see Table 4.3) by comparing the most frequent topics of each category and the key topics of the same category. For example, in the category of *business*, *task tracking and notifications* is more frequent than other topics while it is not a key topic of the category of *business*. This is an interesting observation as developers are likely to be distracted by the frequent topics that do not actually share a significant relationship with star-ratings. The list of our extracted key topics helps developers prioritize the development efforts to address the issues that are related to the key topics.

(iii) Although we study the relationship between a group of metrics and star-ratings, highlighting the most significant metrics can provide developers more insights

Table 4.6: Significant metrics of each category (# sign means number of).

Category	**Significant Metrics**	$\boldsymbol{R^2}$
Business	T_2(#Words), T_3(Sentiment), T_9(#Words)	0.93
Communication	T_3(Sentiment, #Sentences), T_8(Sentiment), T_9(Sentiment), T_{12}(Sentiment), T_{14}(#Sentences), T_{16}(Sentiment), T_{12}(Sentiment), T_{20}(#Sentences)	0.87
Health and Fitness	T_3(Sentiment), T_{10}(Sentiment), T_{20}(#Sentences),	0.73
Media and Video	T_1(Sentiment), T_7(#Sentences), T_8(#Sentences), T_{11}(#Words), T_{12}(Sentiment), T_{18}(Sentiment)	0.94
Photography	T_1(Sentiment), T_3(Negative Reviews%), T_{11}(Sentiment), T_{18}(Sentiment, #Words)	0.86
Productivity	T_2(#Negative Reviews), T_{11}(Sentiment)	0.75
Shopping	Entropy, T_4(Sentiment), T_{19}(#Sentences), T_{10}(Sentiment), T_{16}(Sentiment), T_{21}(Sentiment), T_{23}(#Sentences)	0.90
Social	T_4(Sentiment), T_9(#Sentences), T_{10}(Sentiment), T_{16}(Sentiment)	0.74
Tools	T_4(Sentiment) , T_9(Sentiment, TRL), T_{16}(Proportion), T_{18}(Sentiment)	0.73
Travel and Local	T_1(Sentiment), T_{16}(#Words)	0.82

into the important aspects. Table 4.6 shows the significant metrics of each model. For example, the *sentiment scores* of *battery consumption* (T_3) and *messages* (T_{10}) share significant relationships with star-ratings in the category of *health and fitness.*

(iv) Some key topics are shared by the majority of the categories, while other key topics only appear in one category, such as *play audio & video* in the category of *media and video.* The most popular topic is *speed* which appears in seven categories (i.e., *business*, *communication*, *health and fitness*, *media and video*, *photography*, *tools*, and *travel and local*). The common key topics should be taken care of by the majority of app developers and app development companies.

For example, testing companies should pay more attention to the issues that are related to the common key topics.

The key topics of each category are the topics of user-reviews that share a significant relationship with star-ratings. The key topics are not the most frequent topics of user-reviews.

4.4 Evaluation

In this section, we evaluate the relationship between the identified key topics and star-ratings. We check if the changes in star-ratings after each release are related to the key topics.

4.4.1 Method

We follow the steps below to evaluate the identified key topics for each category. For simplicity, let us describe the approach for the release$_s$ of an app, where release$_s$ is a sample release of the app during the period of our study.

1. Google Play Store shows the average of star-ratings as an indicator of the overall star-rating of the apps [69]. Therefore, we calculate (i) the average of star-ratings that are received after release^{s-1} and before release^{s} and (ii) the average of star-ratings that are received after release^{s} and before release^{s+1}. Then, we measure the difference between the above averages (Δ).

2. We identify the positive changes in star-ratings where $\Delta > 0$. Then, we calculate the weighted cosine similarity [208] between each release note and key topics versus non-key topics. We calculate the similarity using the vectors of words that are extracted by applying the topic modeling (see Part 4.2.3). We get the similarity scores for the key topics (λ_s^k) and the similarity scores for the non-key topics ($\lambda_s^{\neg k}$). If $\lambda_s^k > \lambda_s^{\neg k}$, then positive changes are associated with higher similarity scores of the key topics.

3. For each app, we build two vectors: (i) Λ^k for the key topics (Equation (4.8)) and (ii) $\Lambda^{\neg k}$ for the non-key topics (Equation (4.9)).

$$\Lambda^k = \{\lambda_m^k, \lambda_{m+1}^k, ..., \lambda_n^k\} \tag{4.8}$$

$$\Lambda^{\neg k} = \{\lambda_m^{\neg k}, \lambda_{m+1}^{\neg k}, ..., \lambda_n^{\neg k}\} \tag{4.9}$$

   λ_i^k and $\lambda_i^{\neg k}$ are the similarity scores for the i^{th} version of the app. $m \leq i \leq n$ where m is the oldest release and n is the latest release of the app during our study.

4. Mann-Whitney U test [131] is a non-parametric test to determine whether two datasets have the same distribution without an assumption of the normality of datasets. We compare Λ^k and $\Lambda^{\neg k}$ using paired Mann-Whitney U test [169]. As a null hypothesis, we assume that two λs are similar. The Mann-Whitney U test rejects the hypothesis with a $p - value < 0.05$ [131].

5. We group the apps into two groups: (i) one group where there is a significant difference in the similarity scores between key topics and non-key topics ($p -$

value < 0.05) and (ii) one group where there is no significant difference in the similarity scores between key topics and non-key topics. If we observe a greater proportion of apps in the first group, we can conclude that the release notes that are related to the key topics are more associated with positive changes in star-ratings.

In addition, we repeat the above steps considering the negative changes in star-ratings (i.e., $\Delta < 0$). The goal is to find out if the negative changes in star-ratings are also related to the key topics.

4.4.2 Findings

First, we describe our findings regarding the positive changes in star-ratings. Then, we explain our findings regarding the negative changes.

Positive Changes

For 77% of the apps on average, having a similar release note to the key topics led to increases in star-ratings. For each category, the second column of Table 4.7 shows the proportion of apps with $p - value$s less than 0.05 (see step 5 of the evaluation approach). The last column of Table 4.7 shows the differences between the average of the similarity scores for key topics and non-key topics. As shown in Table 4.7, for the majority of the apps in all the categories, having a more similar release note to the key topics continues with increases in star-ratings.

Table 4.7: Proportion of apps with significant differences when evaluating the key topics of user-reviews.

Category	**p-value<0.05**	**p-value ≥ 0.05**	**Diff**
Business	76%	24%	+0.08
Communication	77%	23%	+0.08
Health and Fitness	82%	18%	+0.06
Media and Video	77%	23%	+0.05
Photography	80%	20%	+0.04
Productivity	67%	33%	+0.02
Shopping	81%	19%	+0.07
Social	67%	33%	+0.04
Tools	87%	13%	+0.05
Travel and Local	71%	29%	+0.02

Negative Changes

By repeating the experiment considering the negative changes in star-ratings, we observed that, for 26% of the apps on average, there is a significant difference between the key topics and non-key topics. To investigate the reasons of the above observation, we randomly selected 384 user-reviews of the cases where there is a decline in star-ratings (with a confidence level of 95% and a confidence interval of 5). We find two main reasons:

1. The changes related to the issue of the key topics (reported in the release notes) did not satisfy users. Consequently, the same issues are repeated in the user-reviews of the next versions.

2. Developers make a change related to the key topics that makes users unhappy. For example, the *user-interface* is a key topics for the category of *social*. We observed that after a release note concerning the *user-interface*: *"We've added a brand new notification center, so now you can choose which items you get."*,

star-ratings decreased. The users were complaining about the new changes in the user-interface.

Developers should consider the key topics very carefully for the next releases to reduce the risk of receiving low star-ratings.

For 77% of the apps on average, having a similar release note to the key topics share a statistically significant relationship with positive changes in star-ratings.

4.5 Threats to Validity

In this section, we discuss the potential threats to the validity of this chapter.

4.5.1 Internal Validity

Regarding removing inconsistent user-reviews, we manually analyzed the inconsistent user-reviews on a statistically representative sample with confidence level of 95% and confidence interval of 5 (i.e., 384 user-reviews). We found that 78% are inconsistent user-reviews. In the evaluation section, we consider all the release notes of the subject apps regardless of the user-reviews. Additionally, we repeat the evaluation approach with an alternation where we only consider the release notes that come after the user-reviews that discuss a key topic. Even with the new alternation in the experiment, we still observe increases in star-ratings.

4.5.2 External Validity

Threats to external validity concern the possibility to generalize the findings [184]. Due to the processing time for handling all the user-reviews, we gather the user-reviews from ten randomly selected categories. Therefore, our extracted key topics are limited to the ten categories. However, future research can follow our approach to mine the key topics for other categories.

4.6 Summary

In this chapter, we identify the key topics on which the developers should focus for the next releases. The identified key topics provide app developers a smaller subset of user-reviews for investigation, rather than all the user-reviews. We study the topics of $4,193,549$ user-reviews from Google Play Store that are collected in a 19 month period. Our analysis is based on 623 Android apps in ten randomly selected categories. We employ PMVD to find the key topics of each category. We find that each category has a specific set of key topics that are not necessarily the most frequent topics of the user-reviews. Considering the key topics is recommended for developers as there is a statistically significant relationship between the key topics and star-ratings. Finally, we evaluated our findings using the release notes. We observed that having similar release notes to the key topics continued with increases in star-ratings.

Chapter 5

Prioritizing User-Related Issue Reports

Having considered the continuous app development paradigm (see Part 1.1.3), developers should prioritize and address the issues that are reported in issue tracking systems (i.e., issue reports) in the next releases. In this chapter, we propose an approach to integrate user-reviews into the process of issue reports prioritization. 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 show that prioritizing the issue reports that are related to user-reviews shares a significant positive relationship with star-ratings. Therefore, we suggest developers a solution for prioritizing user-related issue reports. The results show that the mobile apps with a similar prioritization approach to our solution receive higher star-ratings than other apps.

Section 5.1 gives an introduction to this chapter. Section 5.2 explains the study setup process. Section 5.3 describes the details of the research questions and the findings. Section 5.4 discusses the potential threats to the validity. Finally, we provide a summary of this chapter in Section 5.5.

5.1 Introduction

Traditionally, issues are managed and prioritized through issue tracking systems. Many mobile apps use GitHub [63] as a source code repository and an issue tracking system to manage the identified issues. Addressing the issues that are reported in user-reviews can increase the star-ratings [160]. However, there is no precise link between issue reports in issue tracking systems and 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 would be able to focus on the issue reports that can potentially increase the star-ratings. However, it is a hard task for developers to decide which user-related issue report should be addressed first. For instance, an issue that is reported by an expert developer could receive high priority [209], as well as an issue appearing in many user-reviews. A resolution on such various aspects is beneficial for prioritizing issue reports. Second, developers would be able to identify the issues from the user-reviews that have already been reported in the issue tracking system. Hence, developers can avoid issue report duplications [32] if they plan to add the issues that are reported in the user-reviews to the issue tracking system.

In this chapter, we consider all the Android apps (i.e., $1,310$ apps) that are listed on F-Droid [57]. 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 [103, 192]. We address the following research questions:

RQ5.1) *How precisely can user-reviews be mapped to issue reports?* A user-review is a piece of unstructured text [160] that is not longer than two lines on average. We cluster the related 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 clusters of user-reviews and issue reports. The results show that our approach achieves a precision of 79% in mapping clusters of user-reviews to issue reports.

RQ5.2) *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 values to model the issue reports prioritization of each app. Our models fits well (i.e., adjusted $R^2 \geq 0.5$ [149]) for 37% of the apps but fails to fit for 63% of the apps. The apps for which the model fits well receive higher star-ratings than the rest of the apps (i.e., the apps for which the model failed to fit well). In other words, apps for which there is a significant relationship between star-ratings and our metrics tend to have higher star-ratings. The results imply that prioritizing the issue reports with respect to our suggested metrics is beneficial for receiving higher star-ratings.

RQ5.3) *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 random forest [119]. 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

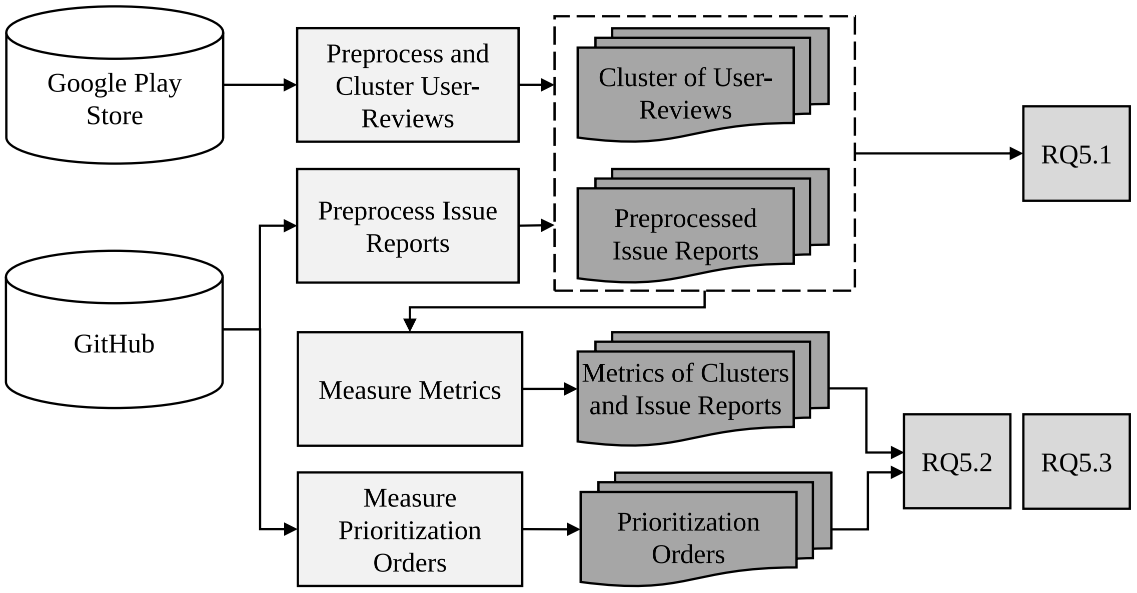

Google Play Store
GitHub
Preprocess and Cluster User-Reviews
Preprocess Issue Reports
Measure Metrics
Measure Prioritization Orders
Cluster of User-Reviews
Preprocessed Issue Reports
Metrics of Clusters and Issue Reports
Prioritization Orders
RQ5.1
RQ5.2
RQ5.3

Figure 5.1: Overview of study setup for prioritizing user-related issue reports.

the first group of apps receive higher star-ratings than the second group. Hence, our suggested method can provide developers a helpful approach for prioritizing issue reports in order to receive higher star-ratings.

5.2 Study Setup

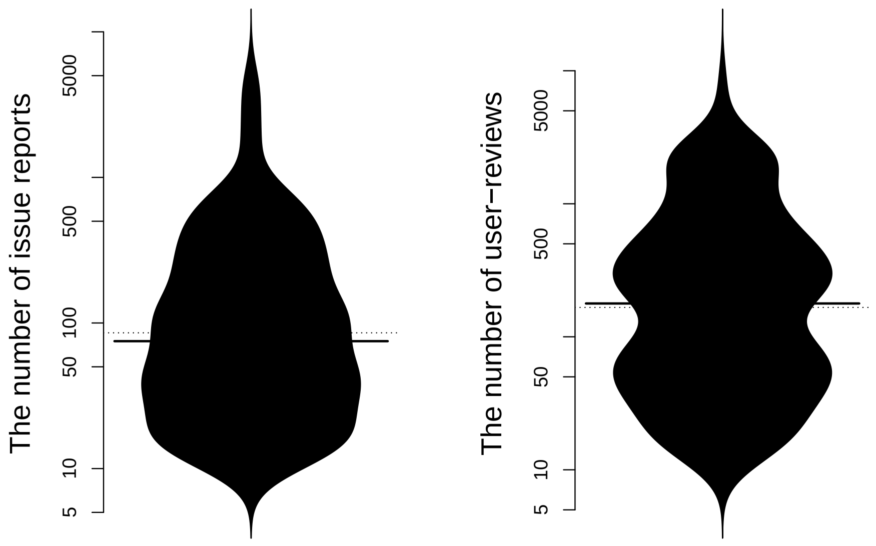
An overview of our study setup is depicted in Figure 5.1. As shown in Figure 5.1, the study setup process mainly consists of four steps: (i) preprocessing user-reviews and issue reports, (ii) clustering user-reviews, (iii) computing metrics from both user-reviews and issue reports, and (iv) measuring prioritization orders of issue reports.

5.2.1 Data Sources

We retrieved a set of open-source apps associated with their GitHub repositories from F-Droid app market [57]. F-Droid is an app store for open-source Android apps that provides access to the source code and the binary files of apps [57]. $1,310$ open-source Android apps were hosted on F-Droid app market [57] 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 5.3). 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 Part 1.4.2) and less than 10 issue reports [103]. The number of Events Per Variable (EPV) is a metric that calculates the ratio of data points to the number of variables [192]. To avoid the risk of over-fitting and having unstable results, having an EPV ≥ 10 is recommended [192]. 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 met the aforementioned criteria.

We gradually retrieve all the user-reviews of our subject apps (see Part 1.4.1). We retrieved all the available issues of our subject apps using GitHub API [50] (see Part 1.4.1). As shown in Figure 5.2, the number of issue reports varies for each app.



(a) Issue Reports (b) User-Reviews

Figure 5.2: Number of issue reports (from GitHub) and user-reviews (from Google Play Store) for all the apps that are studied for prioritizing user-related issue reports.

5.2.2 Preprocessing Data

To reprocess the data, we follow the preprocessing steps as discussed in Part 1.4.2. In addition, we remove stop words and stem the words as explained in Part 4.2.2. However, in this chapter, unlike the approach that is discussed in Part 1.4.2, we did not use AR-Miner [33] to identify the uninformative user-reviews. Instead, we use linguistic rules [89] to identify the uninformative user-reviews (see Part 5.2.2). Moreover, for each user-review and issue report, we extract n-grams of words with n varying from 2 to 4 (see Part 5.2.2).

We asked three non-authors to evaluate the mappings between the user-reviews and issue reports. The evaluators are graduate students in computer and software engineering. We randomly select 384 user-reviews with the associated issue reports with a 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. Correcting typos allows us to increase the mapping precision by 4%. Resolving synonyms let us increase the mapping precision by 3%. Resolving negations increases the precision of our mapping by 3%.

Removing Uninformative User-Reviews

In this chapter, as we cluster the related user-reviews together, a group of uninformative user-reviews can potentially become informative when they are grouped together. Also, having groups of related user-reviews (informative or uninformative) allows us to calculate the required metrics more accurately (see Part 5.2.4). Therefore, we only filter out the user-reviews that solely praise or condemn an app without giving additional details. We use linguistic rules [89] to identify the uninformative user-reviews in this chapter. The main author defined the linguistic rules by manually investigating 5, 000 randomly selected user-reviews. Among 5, 000 user-reviews, we identified 3, 789 user-reviews as informative and 1, 211 user-reviews as uninformative ones according to the identified rules. The rules are listed in Table 5.1. As examples 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!!!"*. By putting aside the uninformative user-reivews and the non-English user-reviews, we ended up with 130, 712 user-reviews. Figure 5.2 shows the number of user-reviews for each of the 326 subject apps.

Table 5.1: Linguistic rules for identifying 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*

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 the real meaning of such a phrase. A n-gram is a contiguous sequence of n words from a given sentence or a given sequence of words [27]. For each user-review and issue report, we extract n-grams of words 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.* [203], we extract the n-grams from the text 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%.

5.2.3 Clustering User-Reviews

We cluster the related user-reviews by customizing the Villarroel *et al.* approach [203], such as adding a step for correcting typos. By clustering the user-reviews together, even short and uninformative user-reviews can become helpful when they are considered together. Furthermore, clustering the user-reviews is required 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.

We apply DBSCAN [55] on the 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 [179] cosine similarity between (i) the associated star-ratings [203], (ii) the post-processed user-reviews, and (iii) the lists of n-grams. We adopt the term frequency-inverse document frequency (TF-IDF) [178] on the vector of each user-review. TF-IDF allows us to measure the frequency of each term in the user-reviews 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 on our dataset. We set the minimum number of points of DBSCAN to 1, as even one user-review may be useful in identifying a potential issue in the user-reviews.

Table 5.2: 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

An example of an issue report that is matched with a cluster of user-reviews is shown in Table 5.2. As shown in Table 5.2, the reported issue is about an issue in the auto-correcting system where there exists some user-reviews reporting the same issue.

5.2.4 Computing Metrics of User-Reviews

We follow the *Goal / Question / Metric* (GQM) paradigm [16, 199] to capture the metrics of user-reviews (see Part 4.2.4). We set our goal to quantify the user-reviews. Table 5.3 shows our GQM model for capturing the user-reviews. As shown in Table 5.3, we measure 31 metrics from the user-reviews, such as the number of similar user-reviews and the proportion of negative and positive user-reviews.

Number of Similar User-Reviews

An issue may be reported by multiple users, while some other issues may be reported by only an individual user. 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

Table 5.3: GQM model to capture metrics of user-reviews for prioritizing user-related issue reports.

Goal: Quantifying the user-reviews for a given issue		
Question	**Metric(s)**	**Count**
How many users reported the same issue?	Number of similar user-reviews	1
How did the users reporting the same issue rate an app?	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	3
How much effort do users put to describe an issue and how much information is provided?	Sizes of user-reviews	12
How was the users' experience with a given issue?	Sentiment scores	6
What is the proportion of user-reviews with positive, negative, and neutral sentiment scores?	Proportion of negative, neutral, and positive sentiment scores	3
		Total: 31

majority of users report the same issue. To count the number of similar user-reviews, we measure the number of user-reviews that appear in the same clusters.

Star-Ratings

To maintain the level of star-ratings, developers are more likely to prioritize the issues that are reported with low star-ratings before the issues that are reported with high star-ratings. For each cluster of the user-reviews, we compute the mean, median, minimum, maximum, 1^{st} quartile, and 3^{rd} quartile of the star-ratings in each cluster.

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 1st quartile and the 3rd quartile in addition to the median. For example, if the 1st quartile is far away from the median but the 3rd quartile is close to the median, we can understand 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 [102].

Proportion of Negative and Positive Star-ratings

Each cluster of user-reviews have positive and negative star-ratings. To capture the diversity of ratings, we measure the proportion of negative, positive, and neutral user-reviews within each cluster. The users usually do not download the apps with the star-rating of less than 3 [155]. Therefore, 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 [155].

Sizes of User-reviews

The size of a user-review can reflect the helpfulness and the importance of the user-review [108]. For example, a longer user-review in terms of the number of words and sentences can show that the user is more concerned about the associated issue. We use the Stanford parser [48] to count the number of words and sentences as the measurements of the size of the user-reviews. For each cluster of the user-reviews, we measure the mean, median, minimum, maximum, 1st quartile, and 3rd quartile of the number of words and sentences in the user-reviews.

Sentiment Scores of User-Reviews

Although a star-rating is the score of an app from user perspectives, the star-rating does not always reflect the real mood and feeling of users. For example, a user may not be satisfied with a certain feature but may still give a high star-rating due to the overall satisfaction of the app. For example, *"Unfortunately I cannot see get the notifications on time"* is associated with a five star-rating. Nevertheless, it does not provide a positive content about the notification feature.

The star-ratings do not always reflect the real sentiments of user-reviews. To capture the sentiment scores of user-reviews, we apply sentiment analysis on the user-reviews using the SentiStrength-SE tool [92]. For each cluster of the user-reviews, we measure the mean, median, minimum, maximum, 1^{st} quartile, and 3^{rd} quartile of the sentiment scores of the user-reviews.

Proportion of Negative and Positive Sentiment Scores

There is a combination of user-reviews with positive and negative sentiment scores in each cluster of user-reviews. We measure the proportion of negative, positive, and neutral user-reviews with respect to their sentiment score. We consider the sentiment scores $\{-1, 0, +1\}$ as neutral, $\{+2, +3, +4, +5\}$ as positive, and $\{-5, -4, -3, -2\}$ as negative [92].

5.2.5 Computing Metrics of Issue Reports

Table 5.4 shows the GQM model to quantify the issue reports.

Table 5.4: GQM model to capture metrics of issue reports for prioritizing user-related issue reports.

Goal: Quantifying the issue reports		
Question	**Metric(s)**	**#**
How many users contributed to resolving an issue?	Number of users	1
How many interactions have been happened for resolving an issue?	Number of comments	1
How well an issue is described?	Size of issue reports	3
How well the comments of an issue are described?	Sizes of comments	12
What is the contribution of an issue reporter?	Reporter contribution	1
What is the contribution of the users who have involved in resolving an issue?	Contribution of users who have involved in issues	6
For how long a reporter is a member of GitHub?	Time since reporter has joined GitHub	1
For how long the users who have involved in an issue are members of GitHub?	Time since contributors have joined GitHub	6
How popular is the reporter?	Number of following and followers of reporters	2
How popular are the users who have involved in an issue?	Number of following and followers of contributors	12
How many code snippet a reporter has shared?	Number of gists of reporter	1
How many code snippet the contributors have shared?	Number of gists of contributors	6
What is the number of repositories of the reporter?	Number of public repositories of reporters	1
What is the number of repositories of contributors?	Number of public repositories of contributors	6
	Total:	59

Number of Users

There are often some discussions among app developers to resolve an issue report. The number of GitHub users who have involved in the discussions can implicitly show the importance of an issue. For each issue report, we count the number of users who contribute (e.g., post a comment) to the issue report.

Number of Comments

Developers may discuss the reported issues. The discussions are usually recorded in the issue tracking system as comments. The number of comments can reflect the complexity of an issue. For each issue report, we count the number of comments.

Sizes of Issue Reports and Comments

The size of an issue report or the associated comments reflects the amount of information contained in the issue report [108]. Moreover, Yu *et al.* [212] indicate that the size of a given document can be associated with the quality and the complexity of a document. For example, consider the example issue reports that are listed in Table 5.5. 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. The second issue with a smaller size report does not provide enough context to understand and resolve the issue. To capture the size of an issue report, we count (i) the number of words of the title, (ii) the number of words and sentences of the body, and (iii) the mean, median, minimum, maximum, 1^{st} quartile, and 3^{rd} quartile of the number of words and sentences of the comments.

Table 5.5: 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*

Contribution of Issue Reporter and Contributors

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 [63] computes a contribution score based on the activities of the user, such as committing to a repository and opening an issue report [63]. The contribution score reflects the experience of a user. We also quantify the mean, median, minimum, maximum, 1^{st} quartile, and 3^{rd} quartile of the contribution of the distinct users who contribute (e.g., post a comment) to each issue report.

Time Since Issue Reporter and Contributors Joined the GitHub

A more experienced user may be active on the GitHub for a long time. For each user, we compute the time since the user joined GitHub till *September* 1, 2016. We measure such time for the reporter of an issue. We calculate the mean, median, minimum, maximum, 1^{st} quartile, and 3^{rd} quartile of such time for every distinct user who contributed to each issue report.

Number of Followers and Followings of Issue Reporter and Contributors

The users of GitHub can follow each other. The number of followers and followings of a user show the popularity of the user [176, 21]. A user with many followers could be a very popular user. This can result in addressing the issues that are reported by such user earlier than other issues. Thus, we measure the number of followers and the number of followings of the reporter, and the mean, median, minimum, maximum, 1^{st} quartile, and 3^{rd} quartile of the number of followers and followings of the users who contribute to each issue report.

Number of Gists of Issue Reporter and Contributors

Gist is a GitHub service that allows users to share code snippets with others [63]. The number of gists can show how much a developer intends to help the development community that may be associated with the developers' activity. We measure the number of gists of the reporter, and the mean, median, minimum, maximum, 1^{st} quartile, the 3^{rd} quartile of the number gists of the users who involve in an issue report.

Number of Public Repositories of Issue Reporter and Contributors

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. We count the number of public repositories of the reporter, and the mean, median, minimum, maximum, 1st quartile, and 3rd quartile of the number public repositories of the users who involve in an issue report.

5.2.6 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.

(iii) Add 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.

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%, a confidence interval of

5, and a 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.

5.3 Approach and Results

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

RQ5.1) How precisely can user-reviews be mapped to issue reports?

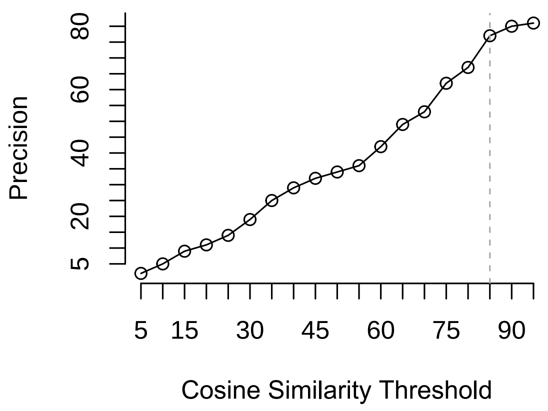
Motivation. Many apps have hundreds or even thousands of user-reviews. It is not a trivial task for app developers to manually analyze all of the user-reviews. By automatically mapping the user-reviews to the issue reports, app developers would be able to use the user-reviews to prioritize the issue reports. Moreover, having the knowledge of the issues that are also mentioned in the user-reviews can help app developers to manage the issues and prevent issue report duplications better [32].

Approach. As described in Part 5.2.3, 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 [179]. First, we compute TF-IDF [178] to obtain the vector of each document, i.e., either a cluster of user-reviews or an issue report. Second, we calculate the cosine similarity between the 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.

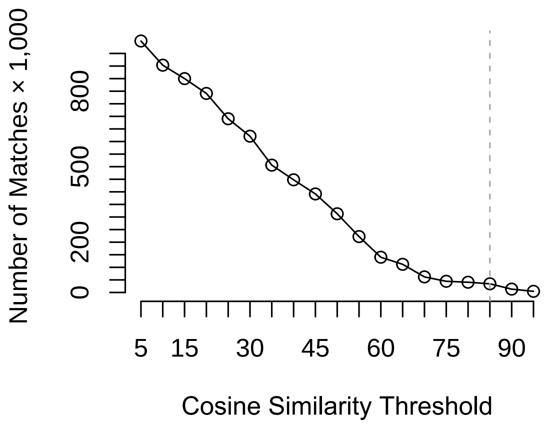
To measure the performance of our mapping approach, we compute precision: the proportion of correctly matched pairs of clusters of user-reviews and issue reports among all the matched pairs. We asked three non-authors to manually examine the correctness of each matched pair on a statistically representative 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 among the evaluators.

Findings. Figure 5.3 shows the precision achieved by our approach with various thresholds and the number of matches between the user-reviews and the 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 $\tau = 0.85$ (i.e., the similarity between the issue reports and the clusters of the user-reviews).

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 [93], while user-reviews are from external users. The matches between the issue reports and the clusters of the 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,



(a) Precisions



(b) Number of Matches

Figure 5.3: Precisions of the mapping approach and the number of matches between user-reviews and issue reports obtained using thresholds from 0.05 to 0.95.

developers can create new issue reports concerning the user-reviews that are left out. Therefore, with 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%.

RQ5.2) Does prioritizing the user-related issue reports have a relationship with star-ratings?

Motivation. Developers may take different priority orders when addressing user-related issue reports. Although the lack of issue reports prioritization can negatively impact the star-ratings, there is no empirical evidence to show the relation between prioritizing the user-related issue reports and star-ratings. Therefore, we study the relationship between star-ratings and the metrics of user-reviews and issue reports.

Approach. We model the issue reports prioritization using linear regression models [56]. 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 [149], of the linear regression models shows whether issue reports prioritization has a relationship with the metrics of user-reviews and issue reports. Figure 5.4 shows an overall overview of our approach.

Before building the regression models, we identify the correlated variables. We apply variable clustering analysis [84] to build a hierarchical overview of the correlation between the independent metrics [155]. The metrics within each sub-hierarchy of metrics with Spearman's $|\rho| > 0.7$ are considered as correlated variables [151]. We choose the most intuitive metrics for inclusion in our model from each sub-hierarchy of metrics. We build two types of regression models:

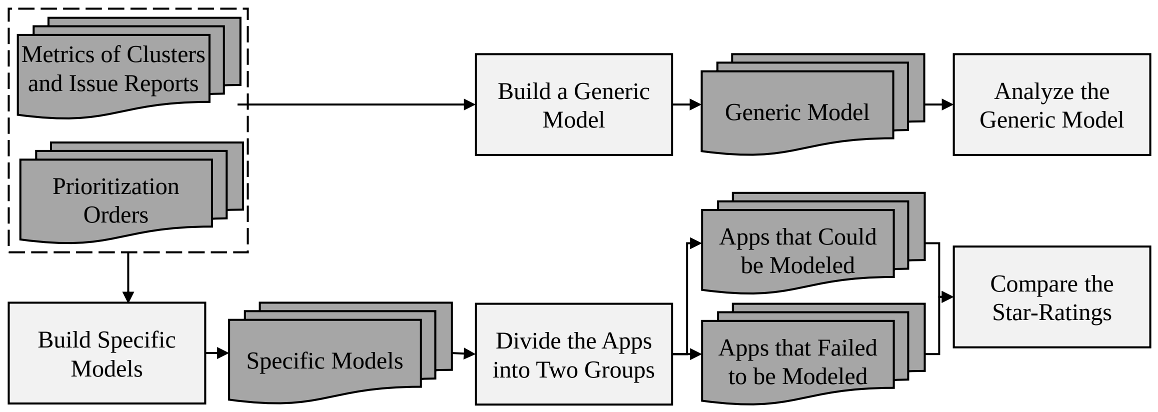

Metrics of Clusters and Issue Reports
Prioritization Orders
Build a Generic Model
Generic Model
Analyze the Generic Model
Apps that Could be Modeled
Compare the Star-Ratings
Build Specific Models
Specific Models
Divide the Apps into Two Groups
Apps that Failed to be Modeled

Figure 5.4: Overview of the approach for addressing RQ5.2.

(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 the 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 reports.

The EPV measures the ratio of data points to the number of variables [192]. An EPV of greater than 10 is recommended to have a very low risk of over-fitting and getting unstable results [192]. 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 $R^2 < 0.5$ [149]. We compare the star-ratings of the two groups of apps using Mann-Whitney U test [131]. 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.

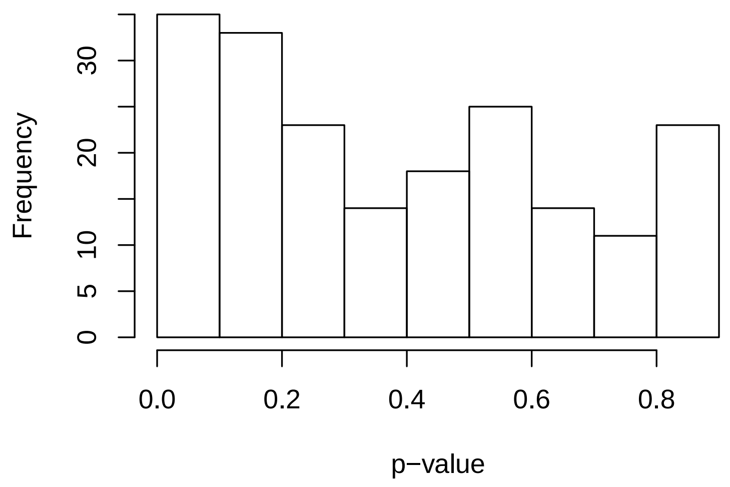


Figure 5.5: Adjusted R^2s that are obtained from regression models that are built for each app explaining issue reports prioritizations.

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 [38]. Cliff's δ is a non-parametric measure without assumptions about the distribution of data [38]. 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 [38]. 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 [39] to interpret the effect size [214]. 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 [214].

Table 5.6: Model that is built for explaining issue reports prioritization of *Indic Keyboard* app.

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		↘

Codes: '***'< 0, '**'< 0.001, '*'< 0.01, '.'< 0.05

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 $R^2s \geq 0.5$, i.e., the issue reports prioritizations share

Table 5.7: Model that is built for explaining issue reports prioritization of *Vanilla Music* app.

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		↘

Codes: '***'< 0, '**'< 0.001, '*'< 0.01, '.'< 0.05

significant relationships with the metrics of user-reviews and issue reports. Figure 5.5 shows the obtained R^2s. Tables 5.6 and 5.7 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.

[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 5.8: Ranks and percentages of occurrence of 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%

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 5.8 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 5.8, 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 issue report content plays 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 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 [155].

Addressing the issues in the user-reviews has a statistically significant relationship with star-ratings. The results in Figure 5.6 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 5.6 and the apps that we could not match their issues reports with the user-reviews (the third boxplot in Figure 5.6 are statistically significant with a $p-value$ of $5.37e-05$ and a medium effect size with $Cohen's\ d$ of 0.37.

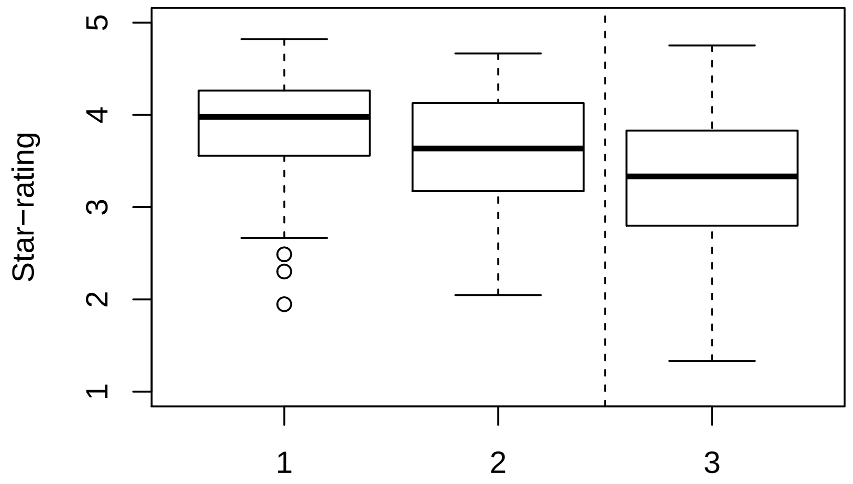


Figure 5.6: Average of star-ratings of the apps that:

(i) We could match their user-reviews with the issue reports and the issue reports prioritization is statistically significantly related to the metrics.

(ii) 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.

(iii) We could not match their user-reviews with the issue reports.

Prioritizing the issue reports with respect to our metrics shares a statistically significant relationship with star-ratings. The first two boxplots in Figure 5.6 show the apps that we could 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 5.6 and (ii) the issue reports prioritization is not statistically significantly related to our metrics (the second boxplot in Figure 5.6 are statistically significant with a $p-value$ of $8.82e-06$ and a medium effects size with a $Cohen's\ d = 0.55$. Figure 5.6 shows that the star-ratings of the first group of apps (i.e., the apps that their issue reports prioritization share a statistically significant relationship with our metrics) 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.

RQ5.3) How can app developers prioritize the user-related issue reports to achieve higher star-ratings?

Motivation. To utilize the findings of the previous research question, we suggest a prioritization method for ranking issue reports in order to achieve higher star-ratings.

Approach. Figure 5.7 shows an overview of our approach. To prioritize the issue reports better, we define four levels of prioritization. Given a list of issue reports that are ranked based on the prioritization order of issue reports (see Part 5.2.6), 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.

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

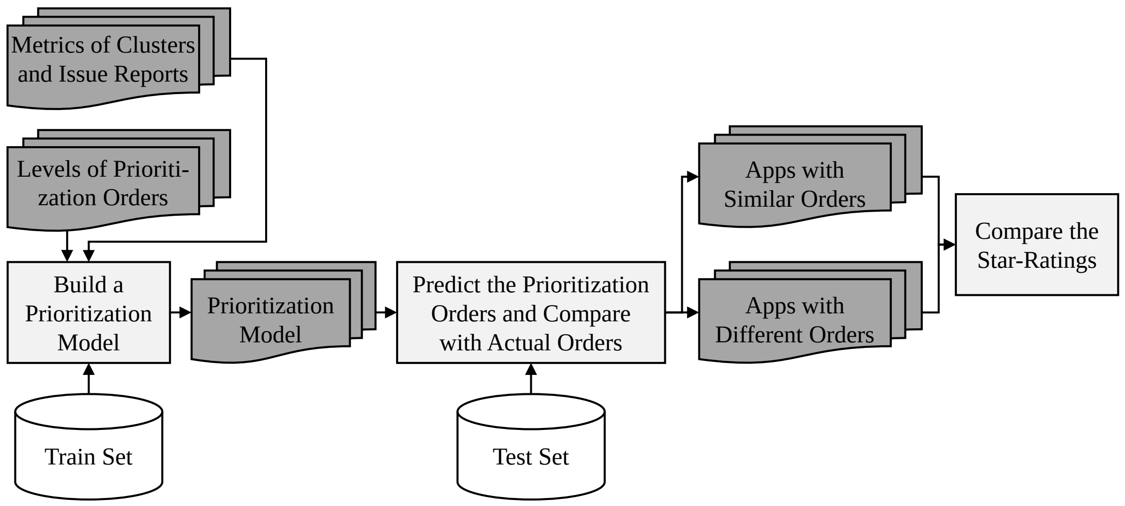

Metrics of Clusters and Issue Reports
Levels of Prioritization Orders
Build a Prioritization Model
Prioritization Model
Predict the Prioritization Orders and Compare with Actual Orders
Apps with Similar Orders
Apps with Different Orders
Compare the Star-Ratings
Train Set
Test Set

Figure 5.7: Overview of the approach for addressing and evaluating RQ5.3.

(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 [209]. For example, Bugzilla [29] 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. We observe that 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, and the the trivial priority issue reports are addressed after five days.

We build a random forest model [119] to predict the prioritization levels of issue reports. Random forest [85] 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 [85]. First, we train a model with the issue reports (that are labeled with the four levels of prioritization as mentioned above) 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 (5.1). In Equation (5.1), 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.

$$Accuracy(i) = \frac{I_c(i)}{I(i)} \tag{5.1}$$

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 [131] 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 [38].

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 solely with the app that has received 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:

(i) The apps that have similar prioritization orders to our predicted levels.

(ii) The apps that 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.

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 5.8 shows the distribution of star-ratings for the apps that: (i) the prioritizations of the issue reports are similar to our predicted prioritizations, and (ii) the prioritizations of the issue reports are different from our predicted prioritizations. As shown in Figure 5.8, 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

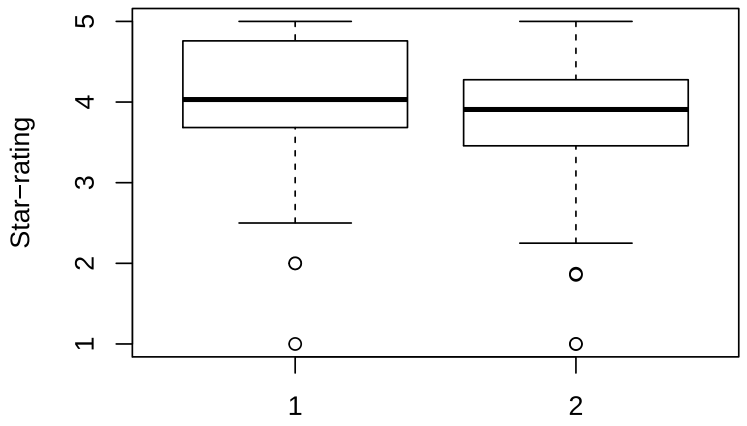


Figure 5.8: Average of star-ratings of the apps that: (i) have similar prioritizations as our predicted prioritizations, and (ii) have different prioritizations from our predicted prioritizations.

medium with a *Cohen's* $d = 0.30$, indicating that the difference is observable and cannot be neglected.

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 5.3), we match could 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.

5.4 Threats to Validity

In this section, we discuss the threats to the validity of this chapter [210].

5.4.1 Conclusion Validity

Threats to conclusion validity concern the relationship between the treatment and the outcome. 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 5.6 is obtained).

5.4.2 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., $N = 10$) [103], 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 are mapped to the user-reviews. Our conclusion remains the same with or without the two outlier apps. 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 chapter, 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 of our approach.

5.4.3 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.

5.5 Summary

The issue tracking systems of most mobile apps are not publicly available. Hence, there is a notable lack of studies investigating how to prioritize user-related issue reports recorded in issue tracking systems. Issue reports are mainly used by developers, as opposed to user-reviews that are open to all users. Therefore, the issues described in the user-reviews do not necessarily overlap with the issue reports.

In this chapter, first, we introduce an approach for mapping user-reviews and 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 63% of our subject apps may not consider the metrics of user-reviews and issue reports when prioritizing issue reports. However, prioritizing issue reports positively associates with the increases in star-ratings. Therefore, it is necessary to assist app developers to prioritize issue reports by considering both user-reviews and issue reports. Finally, we propose a prioritization prediction method using the top apps. The prioritization model can be applied to each app to get the prioritization orders of issue reports. Our results show that the apps that their real prioritizations are similar to our recommended prioritizations achieve higher star-ratings. Therefore, our suggested approach can be used to assist app developers in prioritizing issue reports.

Chapter 6

Ranks

When a user looks for an Android app in Google Play Store, a number of apps appear in a specific rank. Mobile apps with higher ranks are more likely to be noticed and downloaded by users. The goal of this chapter is to understand the evolution of ranks and identify the metrics that share a statistically significant relationship with ranks.

An introduction to this chapter is provided in Section 6.1. Section 6.2 explains the study setup process. Section 6.3 describes the research questions, including the motivations, the approaches, and the findings. Section 6.4 highlights the important implications for new app developers. Section 6.5 discusses the potential threats to the validity. Finally, we summarize this chapter in Section 6.6.

6.1 Introduction

When a user looks for a particular sort of app (i.e., area), Google Play Store lists the most related apps according to the app ranks [140]. Improving the rank of an app increases the chance of being noticed and downloaded by users. Having a higher number of users is desired as it can increase revenues [17, 107].

We define *rank trend* as an evolution of ranks (i.e., declines and inclines over time) that happens similarly among a set of apps. We study the rank trends and highlight the metrics that are statistically significantly related to the rank trends. In addition, we conduct fine-grained analyses per app and per version of each app over time. The goal is to find the most important metrics that share a significant relationship with the changes in the ranks. We investigate 900 apps in 30 different areas that are associated with $4,878,011$ user-reviews in two years.

Our findings reveal practical factors, such as the appearance of new topics, which share a significant relationship with ranks. Such findings can help both developers of currently published apps and developers of start-up apps by providing actionable guidelines to achieve higher ranks. We address the following research questions:

RQ6.1) *What are the rank trends of mobile apps?* We identify rank trends (i.e., evolutions of ranks) by applying a fuzzy clustering [22, 213] on the ranks. We observe that, in some areas, falling in the ranks is more likely to happen than rising in the ranks. Moreover, competing in the areas with dominant high ranked apps (i.e., the apps that maintain their ranks constantly on top), such as *budget*, might be harder than competing in areas with fewer dominant apps.

RQ6.2) *Which metrics can improve the rank trends?* We apply a regression model to explain the identified rank trends [36, 206]. We identify actionable ways, such as introducing new features, to achieve better ranks over time.

RQ6.3) *What metrics have a significant relationship with the ranks over time?* We build a mixed effects model [216] to quantify the ranks across apps and

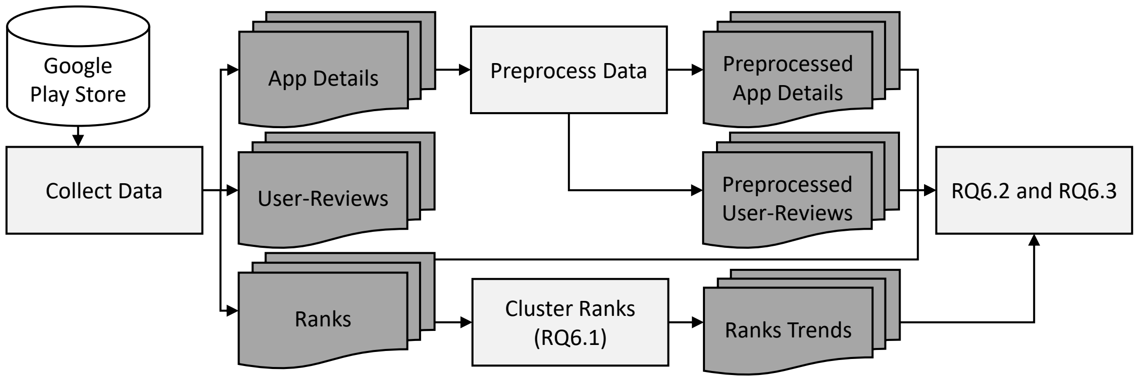


Figure 6.1: Overview of study setup process for identifying and explaining ranks.

different versions of each app. A mixed effects model allows us to explain the ranks across various trends, areas, and app versions. We observe that developer-related activities, such as release latency, play an important role. We also find that not all the metrics that common-wisdom would deem important share a significant relationship with the ranks.

RQ6.4) *How do app developers evaluate the findings?* We asked 51 app developers to evaluate our findings. 60.8%, 84.3%, and 80.4% of the developers strongly agree that the findings of each research question can be useful in practice, respectively. The feedback indicates that higher ranks can be achieved by following our guidelines when developing an app.

6.2 Study Setup

Figure 6.1 shows the major steps of the study setup process. As shown in Figure 6.1, we collect the required data, such as release notes and user-reviews, from Google Play Store [69]. Then, we preprocess the collected data. We explain the data collection and the preprocessing steps in the following parts.

6.2.1 Data Collection

Google Play Store has the largest number of apps and users [155]. When it comes to success in the app markets, the reachability of apps speaks first. Therefore, the most effective data collection scenario comes from an ordinary user side. We impersonate an ordinary user by using Google Play Store search engine to find apps, although every user is subject to different attitudes [18, 46].

An area, such as *calculator*, is a subset of apps that share similar purposes and functionalities. We collect the most searched areas using the *Semrush* service [182] which tracks the popular search engines, such as the *Google* search engine. We look for the top keywords that are mostly searched via the mobile devices for finding apps. Table 6.1 shows the list of 30 top searched keywords (i.e., top wanted areas). The list is determined by manually removing brand names, such as *"Amazon"* and pornographic keywords. By removing brand names, (i) we can focus on the areas that share similar applications, purposes, and functionalities and (ii) we mitigate the skewness of the collected apps towards certain apps.

Like an ordinary user, who uses the search bar to find the desired app, we search each area in Google Play Store. Since the majority of queries are generated by users, we searched the exact identified area names in Google Play Store. We scroll down each search result to the end. For each area, we find a median of 250 app. We randomly select 30 apps from each area to perform our analysis. Considering a higher number of apps (i.e., more than 30 for each area) was not feasible in this study because (i) not all the areas have over 30 different apps and (ii) not all the apps can be studied for two years since some apps are periodically cleaned up from the store [177]. Having 30 apps for each of our 30 subject areas allows us to investigate 900 apps. Moreover,

Table 6.1: List of areas with the number of versions and user-reviews.

Area	**Versions**	**User-Reviews**	**Area**	**Versions**	**User-Reviews**
Airline	422	51,408	**Health**	263	88,306
Bible	339	92,075	**Mailbox**	339	56,391
Budget	206	18,356	**Messaging**	525	230,248
Calculator	294	7,316	**Movie**	279	164,195
Calling	658	210,483	**News**	690	186,827
Camera	466	55,383	**Paint**	122	6,674
Chess	186	22,811	**Piano**	202	27,909
Cloud	546	162,591	**Radio**	504	238,336
Coupon	348	61,160	**Reminder**	292	108,461
Dating	916	165,869	**Sleep**	159	42,713
Dictionary	292	74,073	**Spy Phone**	162	27,328
Emoji	101	4,105	**Talking Pet**	235	31,662
Fitness	272	84,436	**Translator**	206	36,375
GPS	630	151,844	**Weather**	845	178,968
Grocery List	264	50,944	**Weight Loss**	278	58,590

our random selection provides a better combination of apps than picking only the top apps.

We retrieve the app details (e.g., releases notes and descriptions), the user-reviews, and the ranks of the selected apps from Google Play Store from $March$ 1, 2015 to $March$ 1 2017. We collect $4,878,011$ user-reviews for $10,508$ distinct versions of our 900 apps. The number of app versions and informative user-reviews (see Part 1.4.2) are listed in Table 6.1 for each area.

6.2.2 Preprocessing Data

To reprocess the data, we follow the preprocessing steps discussed in Part 1.4.2. Table 6.1 shows the number of informative user-reviews in each area. Unlike user-reviews, app descriptions and release notes benefit from a semi-structured form of writing. Google Play Store demonstrates app descriptions and release notes using the standard HTML format [69, 173]. We extract each item from each release note. We also break app descriptions into separate paragraphs and items of the existing lists in the descriptions. Organizing app descriptions and release notes allows us to measure the metrics more accurately (see Part 6.2.3).

We apply topic modeling [2] on app description, release notes, and user-reviews for two reasons. First, we would be able to unify all the user-reviews, app descriptions, and release notes within a certain number of topics and assign mathematically comparable vectors to each user-review, app description, and releases note. Therefore, we could measure the introduction of new topics and the similarity of release notes and app descriptions with user-reviews (see Part 6.2.3). Second, using topic modeling helps us to mitigate the lack of up-to-date dictionary of words for the terms that are used by users [155]. Similarly, Noei and Heydarnoori [155] employ topic modeling for finding the software engineering terms that represent the same concepts.

We apply the Latent Dirichlet Allocation (LDA) technique [26, 154] on the user-reviews, app descriptions, and release notes. Each document in the corpus is considered as a combination of a number of topics [26]. We employ three latest approaches (i.e., Griffiths *et al.* [71], Deveaud *et al.* [49], and Cao *et al.* [30]) to estimate the optimum number of topics. The summary of the outputs of the three approaches (i.e., Griffiths, Deveaud, and Cao) suggests that the optimum number of topics is between 25 and 165. To avoid losing any potential topic, we pick 165 (i.e., maximum) as the optimum number of topics. We run the LDA with $2,000$ Gibbs sampling iterations [72, 171]. With $2,000$ iterations, we can achieve a high accuracy of LDA [71, 171].

6.2.3 Measuring Metrics

We use the *Goal / Question / Metric* (GQM) paradigm [16, 199] to capture the required metrics (see Part 4.2.4). Our goal is to understand ranks and rank trends using the available data. Google considers star-ratings and the number of downloads when calculating the ranks [67, 68]. Therefore, we consider star-ratings and the number of downloads as *control variables* in our study. We briefly describe our metrics in the following paragraphs.

Star-Ratings

We consider star-rating in our study as a control variable because Google uses star-ratings as one of the factors to calculate the ranks [67, 68]. We collect the star-ratings that are associated with each version of the subject apps.

Number of Downloads

We measure the number of downloads as a control variable as Google uses the number of downloads to calculate the ranks [67, 68]. For each version of an app, we retrieve the latest number of downloads from Google Play Store.

Number of User-Reviews

The number of user-reviews can represent the interest of users in an app [155]. For each version of an app, we count the number of user-reviews since the last version to the current version.

Sentiment Scores

As the interpretation of different users is not the same regarding the number of stars [155], measuring the sentiment scores of the user-reviews is an alternative to measure the user-perceived quality (see Part 1.4.2). We measure the average of sentiment scores of the user-reviews for each version using the SentiStrength-SE tool [92]. Unlike star-ratings that are explicitly given by end users, sentiment scores cannot be measured directly from Google Play Store. We manually analyzed the output of the SentiStrength-SE on a sample of 384 user-reviews with the confidence level of 95% and the confidence interval of 5. SentiStrength-SE gives 71% correct sentiment scores.

Proportion of User-Reviews

To capture the diversity of user-reviews, we measure the proportion of negative, positive, and neutral user-reviews from two perspectives: (i) star-ratings, and (ii) sentiment scores. Users usually do not install the apps with a star-rating of less than 3 [155]. We consider user-reviews with star-ratings equal to 3, greater than 3, and less than 3 as neutral, positive, and negative user-reviews, respectively [155].

App Price

The price of an app can encourage or dissuade a user to download an app [61]. We catch the price of each app from Google Play Store.

Number of Releases

The process of changing the code, integrating with the older versions, and releasing as a new version is called release engineering [1]. The higher number of releases can indicate an active release cycle. We collect the number of releases for each app.

Release Latency

Different apps might have differing release cycles [1] that may affect the ranks [83]. We measure the time between two releases as an indicator of the release cycle latency for each version. We also measure the average of latencies for each app.

Text Size

Larger app descriptions can give users more information about an app, while users may be interested in brief descriptions and release notes. We measure the following metrics to capture the size of app details: (i) the number of words of app names, (ii) the number of words of app descriptions, (iii) the number of sentences of app descriptions, (iv) the number of words of release notes, and (v) the number of sentences of release notes. Moreover, the size of a user-review can indicate its helpfulness [108]. For each version of an app, we calculate the average of (vi) the number of words and (vii) the number sentences of the user-reviews that are posted for the corresponding version. We use the Stanford Parser and Tokenizer [48, 132] to count the number of words and sentences.

App Details Similarity

The first impression of an app might lie upon its name and description. A relevant name and description can motivate more users to download an app. We measure the similarity of app names and app descriptions with our 30 subject areas using the WordNet similarity package [165].

Number of Pictures

Every app on Google Play Store can be associated with some pictures [69]. The pictures may influence the users' decision. We count the number of pictures on the page of each app.

Installation File Size

Users may refrain to download a big application due to network and storage limitations [155]. For each version of an app, we collect the size of the installation file.

Launch Date

An app that has been released prior to other competitors may have a better chance to get the attention of users. We measure the first release date of each app.

Addressing User-Reviews

Recent studies investigate the importance of user-reviews in the deployment of mobile apps [59, 60, 89, 159, 160]. Some researchers focus on the user-reviews [33, 163, 203] to help developers in the process of app deployment. Therefore, we quantify the degrees to which app developers address the user-reviews. For each version of apps, we measure the cosine similarity of the app description with the user-reviews that are posted from the last version to the current version. We also measure the similarity of release notes with the corresponding user-reviews [96].

Introduction of New Topics

Different apps may introduce new features and functionalities in their new versions. We estimate the addition of new functionalities by counting the number of topics that are added to a newer version of an app in comparison with the preceding versions. For each version, we count (i) the number of added topics to the description [96], (ii) the number of removed topics from the description, (iii) the number of added topics to the release note, and (iv) the number of removed topics from the release note.

Area

An area with plenty of dominant apps might be hard to compete in. This can influence other apps within the same area (see Table 6.1). For each app, we consider its area of development.

Category

A category in which an app is released might affect its rank. Some categories may be very competitive with a large number of popular apps. For each app, we retrieve its category from Google Play Store. In addition, for each category, we measure the number of published apps on Google Play Store, star-ratings of the apps, the number of paid apps, the average price of the paid apps, and the ratio of paid apps per total apps.

Company

A reputation of a company may impact all the apps that belong to the same company. For example, having a successful app can motivate users to try other products. For each company, we measure the number of published apps on Google Play Store, star-ratings of the apps, the number of downloads of the apps, the number of paid apps, the average price of the paid apps, and the ratio of paid app per total apps.

6.3 Approach and Results

In this section, we present the approach and the findings of the research questions.

RQ6.1) What are the rank trends of mobile apps?

Motivation. When searching for new apps, Google Play Store shows a ranked list of apps. Apps with higher ranks are more likely to be downloaded and installed [42, 121, 215]. Observing the evolution of ranks helps developers to see the odds of rising and falling in ranks over time. Moreover, we can discover the areas where apps tend to change or tend to maintain their ranks. Thus, developers can take a wiser decision when implementing a new app or trying to improve the ranks of the currently published apps.

Approach. We apply time series clustering [118] to identify the rank trends. Time series clustering requires (i) a clustering algorithm, (ii) a distance measurement method, and (iii) the number of clusters. We apply a fuzzy clustering algorithm [213] with dynamic time warping (DTW) [20] as the method of measuring the distance between the time series of ranks.

Fuzzy Clustering. Partition clusterings, such as k-means [208], build n partitions where each cluster contains at least one object and each object belongs to one cluster. Fuzzy clustering inherits from the fuzzy logic and fuzzy sets where the truth values of metrics can be real numbers between 0 and 1 instead of being exactly 0 or

1 [213]. Therefore, with fuzzy clustering, a rank trend can belong to more than one cluster [22]. For example, if a rank is fluctuating while declining over time, the fuzzy algorithm can put such a rank into two clusters. Therefore, the metrics associated with such a trend can contribute to both clusters of ranks. The fuzzy algorithms can be set with a level of fuzziness m where $m \geq 1$. A larger m makes the clustering fuzzier, while when $m = 1$, the fuzzy algorithm works like a partition algorithm [118].

Dynamic Time Warping (DTW). DTW aligns two time series (rank trends) in a way that the differences between the two trends are minimized [20]. The Euclidean distance that is commonly used in clustering algorithms, such as k-means [129, 208], assumes that the i^{th} point in one trend should be aligned with the i^{th} points of other trends. DTW alignment allows the offsets in the rank trends to be varied [180]. For example, suppose two rank trends $R_1 = \{r_{11}, r_{12}, ..., r_{1n}\}$ and $R_2 = \{r_{21}, r_{22}, ..., r_{2m}\}$, where n is the number of time points in R_1 and m is the number of time points in R_2. If the trends are very similar but with an offset of d days, Euclidean distance fails to converge. However, DTW can fit such trends in one cluster. Equation (6.1) shows the distance of warping path between two trends.

$$dist_{DTW} = \frac{\sum_{i=1}^{\kappa} \omega_i}{\kappa} \tag{6.1}$$

DTW build a warping path $W = \{\omega_1, \omega_2, ..., \omega_\kappa\}$ where κ denotes the number of points in W and $max(m, n) \leq \kappa \leq m + n - 1$ [118].

Number of Clusters. Prior to applying a clustering algorithm, it is required to determine the number of clusters to obtain the highest clustering quality [144]. To calculate the number of clusters, we follow two approaches. First, we run the clustering algorithm with different numbers of clusters from 2 to 50 and visually inspect the results until we achieve distinguishable clusters [201]. We find 13 as the best number of clusters. Second, we employ the gap statistic approach [197] to estimate the optimum number of clusters. The gap statistic uses the output of the clustering algorithm and compares it with the change in a within-cluster dispersion. The procedure tries different numbers of clusters to maximize the gap statistic value. We apply the gap statistics algorithm with the number of clusters ranging from 2 to 50. The best result achieved with 13 clusters. Both approaches (i.e., visual inspection and statistical analysis) suggest 13 as the optimum number of clusters.

Findings. In this part, we explain our findings and observations regarding clustering the rank trends.

There are 13 rank trends and three general trends. Figure 6.2 shows the centroids of the 13 identified rank trends. As it is shown in Figure 6.2, three general trends among the rank trends can be observed: (i) falling, (ii) rising, and (iii) maintaining. In our experiments, we tried increasing the fuzziness of the clustering. However, rank trends tend to stay categorized in only one cluster. This can confirm that the number of clusters is set optimally.

61% of the studied apps fell in the rankings over time. By comparing the number of apps that lay on each of the three general trends, we observe that the majority of the apps (i.e., 61%) had a falling trend, which may implicitly denote the competitive nature of app markets. Only 6% of the subject apps could have improved the ranks during the studied period. Finally, 33% of our subject apps have maintained their ranks with very slight fluctuations.

Some areas, such as *news* and *budget*, are harder to compete than other areas. Figure 6.3 shows the distribution of the rank trends for each area. *News* and

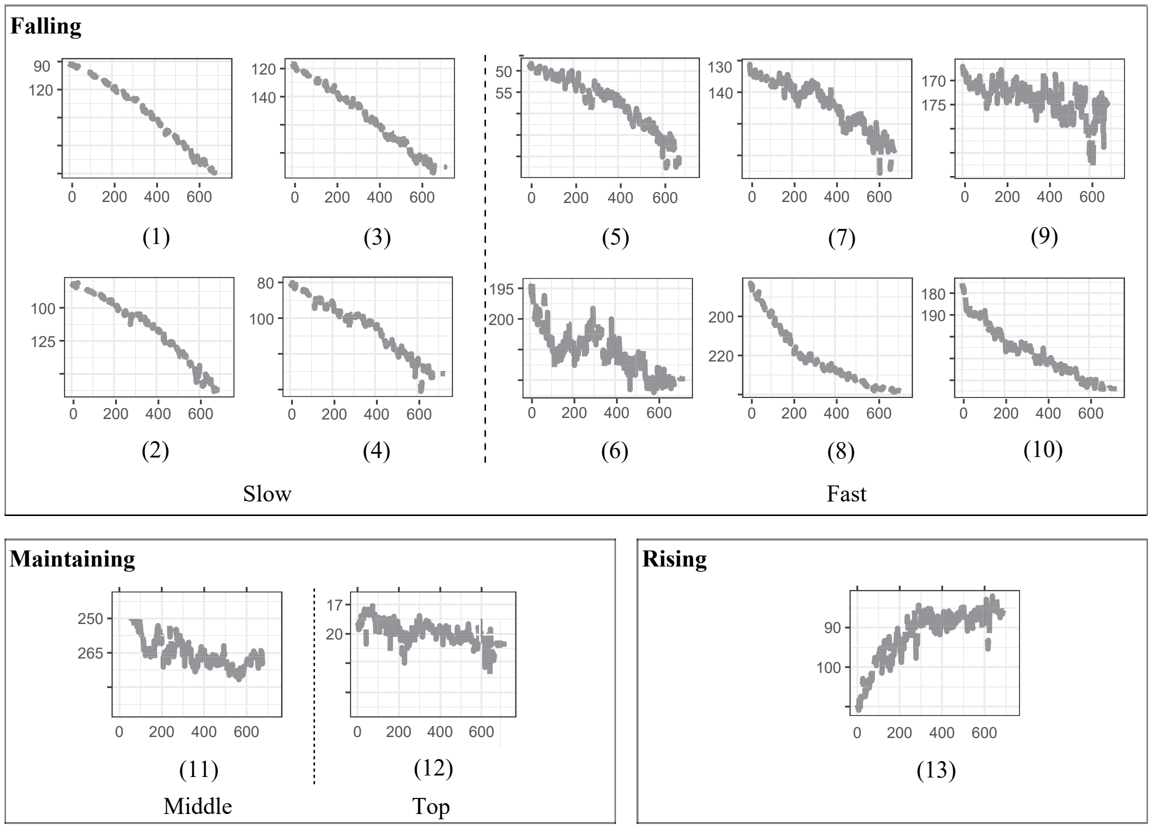


Figure 6.2: Centroids of each cluster of rank trends. X-axis denotes time (day) and Y-axis shows ranks.

budget are two areas with the highest number of apps that could have maintained their ranks among the top apps (i.e., cluster 12), while only 3 apps improved their ranks in the *news* and *budget* areas. This observation can suggest that it is harder for newcomers to compete in such areas since there are dominant apps that maintained their ranks during the studied period.

Cluster 11 contains the most number of apps (i.e., 19%) in comparison with other clusters. The apps that are in cluster 11 maintained the ranks in the middle. The *radio* apps failed more in maintaining the ranks in comparison with other areas. The *chess* and *cloud* areas have the most number of apps with a rising trend. Such a result

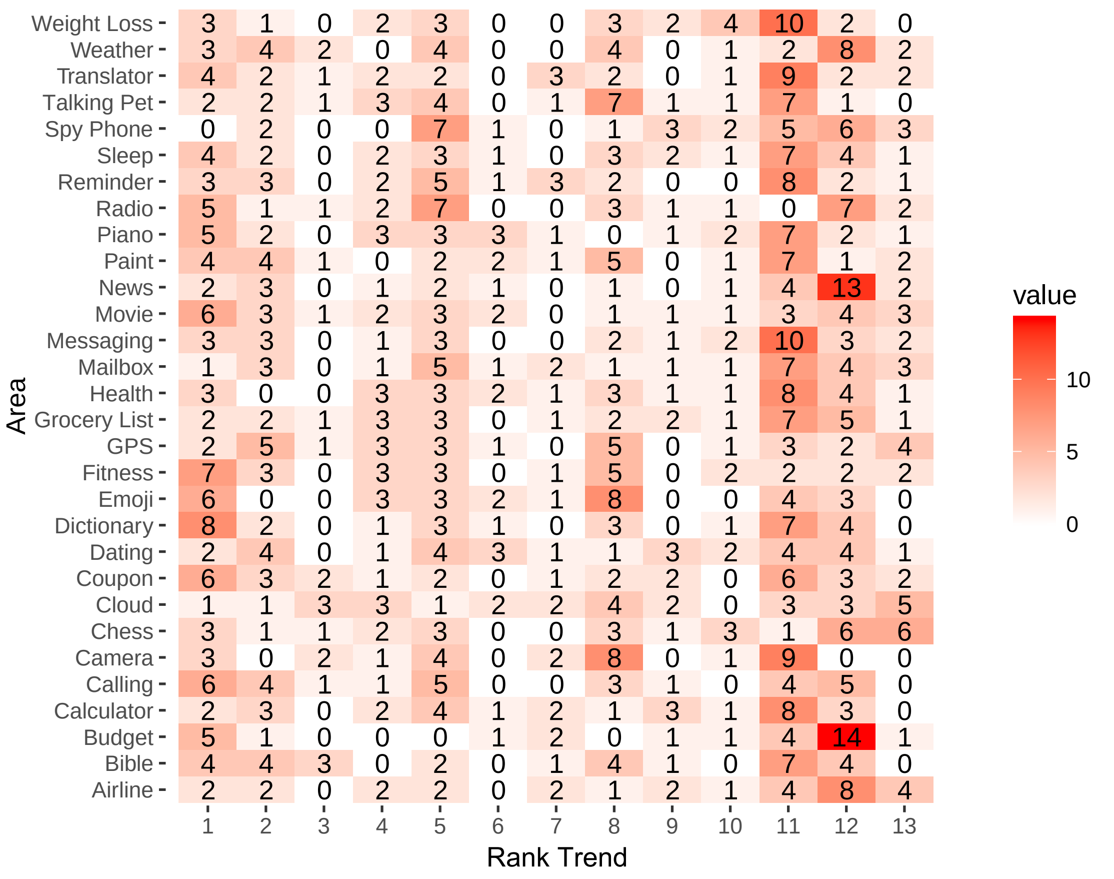


Figure 6.3: Distribution of apps in each cluster of rank trends (X-axis) versus areas (Y-Axis).

suggests that entering to *chess* and *cloud* areas might be a wiser choice to succeed in the competitive market of mobile apps.

Most of the subject apps tend to fall in ranks within the duration of our study. In some areas, such as chess, it is easier to compete and improve ranks. However, having a large number of dominant apps in some areas, such as news and budget, makes it harder for developers of new apps to enter the areas.

RQ6.2) Which metrics can improve the rank trends?

Motivation. In the previous research question, we identified the rank trends. Although one could speculate that the area is an important metric to explain the rank trends (see Figure 6.3), it is unclear what metrics share a strong relationship with the rank trends. Discovering such metrics is important to help developers and app development companies to improve the ranks of their apps proactively.

Approach. We treat metrics as (i) nominal metrics that include two or more categories, such as the category in which an app is published, (ii) ordinal metrics that are nominal metrics for which the order of values matters, and (iii) numeric metrics, such as the number of releases. We also normalize the independent metrics to bring the values into the same scale. Before building a model, we identify the correlated metrics. Having correlated metrics negatively affects the results of our model [79]. We apply metric clustering analysis [84] to build a hierarchical overview of the correlation between the independent metrics [155]. We choose the metrics that are more intuitive for inclusion in our model from each sub-hierarchy of metrics with Spearman's $|\rho| > 0.7$ [151]. The correlated metrics are listed in Table 6.2.

Studying rank trends to get practical results requires ordering them from the worst to the best. However, there is no standard definition of the goodness of a rank trend to treat the trends as an ordinal metric. Therefore, we mitigate the bias of judging the superiority of each rank trend over another by breaking the trends into two groups of optimistic rank trends and pessimistic rank trends. We define optimistic rank trends as (i) the ranks that are maintained among top apps during our study (i.e., trend 12), and (ii) the ranks with a rising trend (i.e., trend 13). Otherwise, a rank trend is

Table 6.2: List of correlated variables for explaining rank trends.

Selected Variable	**Correlated Variables**
Description Similarity	Number of Words in Description
Number of User-Reviews	Average Number of Downloads (Company), Number of Negative, Positive, and Neutral User-Reviews, Number of User-Reviews with Negative, Positive, and Neutral Sentiment Scores
Average Rating (Company)	Number of Apps (Company)
Address User-Reviews (Release Note)	Address User-Reviews (Description)
Average Price (Company)	Number of Paid Apps (Company), Price
Number of Releases	Average Time Between Releases
Number of Apps (Category)	Number of Paid Apps (Category)

considered as a pessimistic trend.

Having two groups of trends allows us to build an ordered regression model to determine the odds ratios of the metrics that play a significant role in explaining the two groups of trends (i.e., optimistic and pessimistic). *Logistic* regression model and *probit* regression model are both appropriate approaches to explain an ordered dichotomous dependent metric (i.e., optimistic vs. pessimistic) [36, 88, 206]. The main differences between *logit* and *probit* regression models are the link function and the assumption of each model about the distribution of the errors [4]. As we are interested to observe the odds ratios, we report the output of the ordered *logistic* regression model in this chapter. Nonetheless, both approaches (i.e., *logit* and *probit*) produce the same results with our data. We use metrics that are introduced in Part 6.2.3 as the independent metrics. Our *logistic* regression model obtains a great fit with the McFadden's $\rho^2 = 0.24$ [139]. The McFadden's ρ^2 compares the log-likelihood of the model with the *null* model to measure the level of improvement of the model [53]. Moreover, the number of Events Per Variables (EPV) of our model is 11.5, which indicates that our model has a low risk of over-fitting [192].

We determine the significant metrics using the ANOVA χ^2 test [168]. The significant metrics have $Pr(>\chi^2)$ less than 0.05. $Pr(>\chi^2)$ is the $p-value$ that is associated with the $\chi-statistic$ [137].

Findings. Table 6.3 shows the results of our ordered *logistic* regression model. The Likelihood-ratio χ^2 [200], $p-value$, and effect of each independent metrics are shown in Table 6.3. The significant metrics are marked with asterisks.

The *launch date* shares a significant relationship with the rank trends. Interestingly, we observe that the apps that are launched later tend to be more successful in the market. This is good news for newcomers and start-up companies not to be hopeless when launching an app for the first time. Although the initial idea and innovation are essential to succeed [175], competitors might overcome the seminal apps over time if the necessary effort is invested. This observation led us to check how the launch date works when comparing top apps and rising apps only. We observe that the launch date no longer appears as a significant metric when rising apps and top apps are compared.

Developers of published apps should constantly work on improving their apps to strive. The appearance of new topics in the release notes and the descriptions are two significant metrics of the model. A release note usually contains a report of the fixed issues and the addition of new features [69]. A description usually describes the functionality and the purpose of an app [96]. The addition of new topics has a positive relationship with the success of an app.

More releases are encouraged. Some users may feel frustrated when apps get

Table 6.3: Logistic regression model of rank trends.

Metric	**LR χ^2**	**Pr**$(> \chi^2)$		**Effect**
Launch Date	11.94	5.49E-04	***	↗
Appearance of New Topics (Release Note)	10.63	1.12E-03	**	↗
Number of Releases	7.07	7.85E-02	**	↗
Category	47.69	2.13E-02	*	-
Star-Rating	4.71	3.00E-02	*	↗
Area	44.66	3.18E-02	*	-
Appearance of New Topics (Description)	4.39	3.61E-02	*	↗
Average Rating (Company)	4.32	3.76E-02	*	↗
Number of User-Reviews	4.04	4.44E-02	*	↗
Description Similarity	3.66	5.57E-02	.	↗
Name Size	3.08	7.92E-02	.	↘
Number of Pictures	2.91	8.80E-02	.	↗
Address User-Reviews (Release Note)	2.40	1.22E-01		↗
Sentiment Score	2.28	1.31E-01		↗
Number of Sentences (Description)	1.87	1.71E-01		↘
Average Price (Company)	1.75	1.86E-01		↘
Installation File Size	1.63	2.81E-01		↘
Name Similarity	0.05	8.17E-01		↗
Average Star-Rating (Category)	1.00	9.95E-01		↗
Average Price (Category)	0.00	1.00E+00		↘
Number of Apps (Category)	0.00	1.00E+00		↘

Codes: '***'< 0, '**'< 0.001, '*'< 0.01, '.'< 0.05

frequently updated [170]. When it comes to statistical observations, we note that the number of releases has a positive relationship with the success of apps. Therefore, developers should not be afraid of keeping their apps up-to-date. The average of days between releases for the top and rising apps is 14 days. However, developers should be cautious about the quality of the app version they are about to release as Ruiz *et al.* [177] find that releasing a low-quality app endangers the survival of the app.

Developers, especially from start-up companies, should carefully consider the area and the category on which they intend to work. Both the *area* and the *category* of apps have a significant relationship with the success of an app. A *chess*, *cloud*, *radio*, *spy phone*, *weather*, *news*, or *budget* app has a higher chance to lie upon optimistic trends (by a positive effect). In addition, when we compare only the rising and top maintaining apps, we note that among the aforementioned areas, the ranks of *chess*, *cloud*, *radio*, and *spy phone* apps are more likely to rise.

App developers should not be afraid of a late entry into the market as newer apps achieved higher ranks during the period of our study. There is a chance of winning the market by constantly improving an app, such as adding new features and fixing bugs. However, developers should be careful in choosing the area in which they intend to succeed.

RQ6.3)What metrics have a significant relationship with the ranks over time?

Motivation. In the previous research question, we identify the metrics that have a significant relationship with the rank trends. However, an app can face various rises and falls over time. We break the timeline of our subject apps with respect to the release dates of each app. Therefore, we can identify the metrics that have a significant relationship with the ups and downs of the ranks over time. In this research question, we study the ranks with a finer grain (version) while, in the previous research question, we study the ranks with a broader view (i.e., rank trends).

Table 6.4: List of correlated variables for explaining changes in ranks.

Selected Variable	**Correlated Variables**
Number of Apps (Category)	Number of Paid Apps (Category)
Number of User-Reviews	Number of Negative, Positive, and Neutral User-Reviews, Number of User-Reviews with Negative, Positive, and Neutral Sentiment Scores
Address User-Reviews (Release Note)	Address User-Reviews (Description)
Number of Releases	Average Time Between Releases
Number of Downloads	Average Number of Downloads (Company)
Description Similarity	Number of Words in Description
Average Price (Company)	Price, Number of Paid Apps (Company)
Average Rating (Company)	Number of Apps (Company)

Approach. First, we remove correlated metrics with Spearman's $|\rho| > 0.7$ [151]. Table 6.4 shows the list of the correlated metrics. Then, we build a mixed effects model [216, 155] to determine the metrics that have a significant relationship with the ranks (see Section 3.3).

We let the dependent metric to be the rank of each version of the apps. We set the independent metrics to have random intercepts while keeping the coefficient of the independent metrics fixed. Thus, we give a chance to the mixed effect model to tune metric intercepts according to the grouping factors but keep the coefficients union across different observations. Therefore, independent metrics can contribute equally to the observations with a minor tune with respect to the grouping factors.

We group the independent metrics according to (i) 13 rank trends and (ii) 30 areas nested within various versions of each individual app. Setting a grouping factor for rank trends gives a fair comparison between the apps. Instead of comparing all the apps together, we compare an app with its own competitors. A good analogy is the different weight classes in wrestling competitions. By having different weight classes, competitors can be judged on their techniques rather than their weights. With a similar reasoning as above, we set another grouping factor to the area of the app. We let different versions have varying intercepts as there might be other factors for each version but cannot be considered in a non-controlled empirical study [155], such as contextual requirements of the time in which a version is released.

The marginal R^2 [147] of our model is 0.02, but the conditional R^2 is 0.68. The lower value of the marginal R^2 compared to the conditional R^2 denotes that the proportion of the variance explained by both fixed and random metrics is considerably higher than the proportion of the variance explained by the fixed metrics. Thus, modeling the random effects greatly improves the explanatory power of our model [155].

Findings. Table 6.5 shows the output of the mixed effects model. Significant metrics are highlighted with asterisks along with their effects on the ranks.

User-reviews are important artifacts to explain the ranks. The sentiment scores have a significant relationship with the ranks. Developers should take care of user-reviews and keep the users happy to achieve better ranks. For example, developers should resolve the bugs that are reported in the user-reviews. For extracting bug reports, developers can refer to related work on this matter, such as Chen *et al.* [33], Villarroel *et al.* [203], and Sorbo *et al.* [52].

Apps should be assigned with a proper name and description. Name and description of an app should be related to the area of the app. Although there exist some apps, such as *Facebook*, whose own commercial name may not reflect their true area, less famous apps should either find a way to advertise their apps or keep the

Table 6.5: Mixed effects model of ranks.

Metric	χ^2	Pr(> |F|)		Estimate	Effect
Sentiment Score	62.09	3.55E-15	***	1.14E-01±1.45E-02	↗
Number of Sentences (Description)	46.10	1.19E-11	***	-5.08E-02±7.48E-03	↗
Name Similarity	41.16	1.47E-10	***	5.18E-02±8.08E-03	↗
Sentiment Score (Total)	23.19	1.49E-06	***	2.76E-02±5.73E-03	↗
Release Latency	19.71	9.13E-06	***	3.01E-02±6.78E-03	↘
Installation File Size	13.49	2.41E-04	***	1.79E-02±4.88E-03	↘
Star-Rating	12.07	5.15E-04	***	-4.20E-02±1.21E-02	↗
Appearance of New Topics (Release Note)	11.20	8.20E-04	***	2.75E-02±8.20E-03	↗
Number of Downloads	10.64	1.11E-03	**	-2.88E-02±8.82E-03	↗
Average Star-Rating (Category)	9.08	2.60E-03	**	-1.93E-02±6.42E-03	↘
Address User-Reviews (Release Note)	3.94	4.71E-02	*	-1.89E-02±9.51E-03	↗
Description Similarity	2.75	9.76E-02	.	2.98E-02±1.80E-02	↗
Average Price (Category)	2.50	1.14E-01		-1.31E-02±8.28E-03	↗
Appearance of New Topics (Description)	2.17	1.41E-01		1.23E-02±8.37E-03	↗
Number of Releases	1.67	1.97E-01		7.65E-03±5.93E-03	↗
Launch Date	1.39	2.38E-01		-1.91E-02±1.62E-02	↗
Number of Pictures	1.29	2.56E-01		-5.77E-03±5.08E-03	↗
Name Size	1.22	2.70E-01		-5.34E-03±4.84E-03	↘
Ratio of Paid Apps per Total (Category)	0.70	4.02E-01		-5.35E-03±6.38E-03	↗
Number of Apps (Category)	0.40	5.28E-01		-3.54E-03±5.60E-03	↘
Average Star-Rating (Company)	0.34	5.62E-01		3.19E-03±5.49E-03	↗
Deletion of Previous Topics	0.09	7.70E-01		2.28E-03±7.80E-03	↘
Average Price (Company)	0.08	7.75E-01		-6.41E-03±2.25E-02	↘
Number of User-Reviews	0.04	8.49E-01		-1.18E-02±6.22E-02	↗

Codes: '***'< 0, '**'< 0.001, '*'< 0.01, '.'< 0.05

similarity of names in mind.

Developers should not wait too long to publish an update. As the latency of an update increases for an app, the app tends to lose ranks. Such behavior might be caused by a probable Google's ranking strategy as Henze *et al.* [83] observed that releasing new updates is an effective strategy for increasing apps in App Store. Another possible explanation can be the users' preferences on seeing an updated version of an app earlier. Future studies should look into this matter more in-depth. Also, it is better to keep the installation files as small as possible.

Introducing new topics, such as new features, helps developers to improve the ranks. We observe that the introduction of new topics in the release notes has a significant positive relationship with the ranks. Developers can refer to related work, such as Villarroel *et al.* [203], and Sorbo *et al.* [52], to mine the user-reviews and get some idea on the topics that they may want to add to the next versions.

The common-wisdom does not always hold when it comes to improving the ranks. Various metrics, such as the number of pictures and the number of apps in a given category, may sound critical to have a better rank. For example, Tian *et al.* [196] observed that the number of pictures is an influential metric for receiving higher star-ratings. Conversely, as shown in Table 6.5, the number of pictures does not have a significant relationship with the ranks. In fact, only 11 metrics share a significant relationship with the ranks.

Developers should take care of user-reviews and constantly improve their apps (e.g., introducing new features) to achieve higher ranks. They should be careful not to spend too much budget and time on the metrics that sound important but actually do not share a significant relationship with changes in the ranks.

RQ6.4) How do app developers evaluate the findings?

Motivation. To better understand the practical value of our findings, we conducted a survey and in-depth interviews with app developers discussing our findings.

Approach. In this part, we describe our participants, the design of the survey, and the procedure of the survey.

Participants. The participants of our survey are mobile app developers. To find the participants, we contacted four app development companies that agreed to cooperate with us in conducting the survey. Finally, 51 app developers participated in our survey. We also had the opportunity to discuss our observations in details with two developers who were willing to discuss more.

Design. We gathered the developers' opinions using questionnaires. We asked five questions regarding the usefulness of our findings. The questions are listed in Table 6.6. Also, for each question, we asked the developers to provide further comments in case they did not agree with our findings or in case they would like to provide additional feedback (follow-up comments). We asked the developers to answer the questions on a five-point Likert scale: (i) strongly agree, (ii) agree, (iii) neutral, (iv) disagree, and (v) strongly disagree [120].

Procedure. For each company, we presented our research and our results to the developers. It took around 30 minutes to do each presentation using slides. There was no time limit for developers to finish the survey. However, it took less than 15 minutes to have the questionnaires filled out by all the developers. Regarding the in-depth discussions, the first author conducted the interviews. Each interview took around 60 minutes.

Table 6.6: List of survey questions for evaluating the usefulness of the findings of Chapter 6.

#	Question
1	Do you think that the studied areas are the top areas of mobile apps?
2	Are the considered variables reasonable and sufficient for explaining ranks and rank trends?
3	Do you think the findings of RQ6.1 are useful and practical for the industry?
4	Do you think the findings of RQ6.2 are useful and practical for the industry?
5	Do you think the findings of RQ6.3 are useful and practical for the industry?

Findings. The majority of the developers agree with our findings. For each question, first, we present the result of the survey. Then, we summarize our in-depth discussion with two developers.

(i) ***Do you think that the studied areas are the top areas of mobile apps?***

Survey. As shown in Figure 6.4, developers agree that the investigated areas are the hot areas of mobiles apps. On the other hand, 11.8% and 2.0% of the developers disagree and strongly disagree with the question, respectively. According to the follow-up comments, one developer thinks that the area of *photography* might be important too. Two developers think that *social networking* is missing. However, as we look for the top keywords that are searched by real users, the exact name of the dominant social networking apps, such as *Facebook*, are more likely to be searched rather than searching for other alternative apps.

In-depth Discussions. Both developers approved the areas subject of our study with *agree* and *strongly agree.* First developer said *"you have covered all the top areas, such as dating, coupons, weather apps, health, and messaging"*.

(ii) ***Are the considered metrics reasonable and sufficient for explaining ranks and rank trends?***

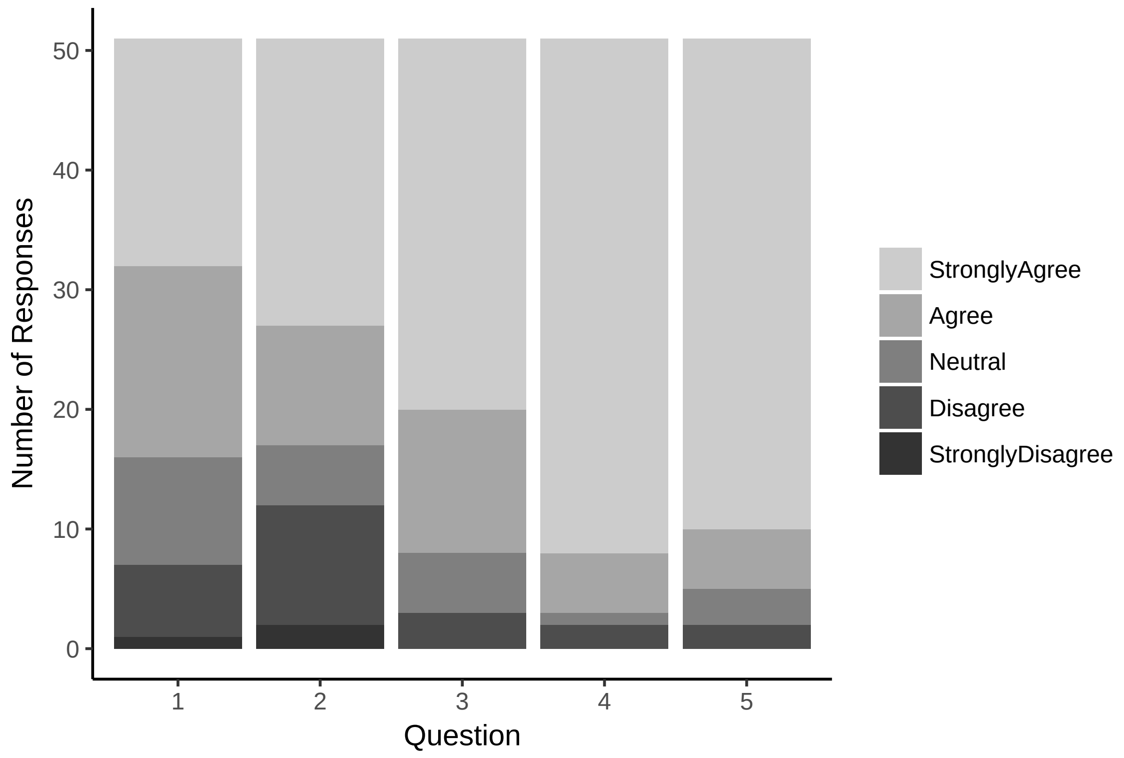


Figure 6.4: Distribution of developers responses to each question regarding the usefulness of the findings of Chapter 6.

Survey. As shown in Figure 6.4, 47.1% and 31.4% of the developers strongly agree and agree with the considered metrics, respectively. However, 11.8% and 3.9% of the developers disagree with the metrics. Summarizing the follow-up comments, developers that do not agree with our metrics suggested the following metrics: (i) code metrics, (ii) team size, (iii) communication between the development team, (iv) expertness of developers, (v) marketing and advertisement, (vi) project plan, and (vii) the novelty of an app. Although we agree with the developers' suggestions, it is challenging to add more details about the code metrics and team characteristics as such data is not publicly available. Future studies should look into the suggested metrics in more details.

In-depth Discussions. Both developers confirmed our metrics with *strongly agree*. They indicated that we have considered almost all the important metrics.

(iii) ***Do you think the findings of RQ6.1 are useful and practical for the industry?***

Survey. As shown in Figure 6.4, 60.8% of the developer strongly agree with the findings of the first research question. According to the follow-up comments, one developer found the drops from the ranks very challenging for the developers of existing apps. This is due to our observation that most of the apps tend to fall in the ranks.

In-depth Discussions. Both developers confirmed the question with *strongly agree*. First developer mentioned: *"I think the reason that most apps have fallen in the ranking is the introduction of new apps to the market"*. Regarding competitiveness of different areas, he said: *"As an example, in the area of chess, new apps do not have serious barrier entry and can easily take the position of older apps by introducing new features"*. Second developer mentioned that seeing different rank trends in different areas is very interesting. It shows that choosing an area is an important decision.

(iv) ***Do you think the findings of RQ6.2 are useful and practical for the industry?***

Survey. 84.3% of the developers strongly agree that the findings can be useful in practice.

In-depth Discussions. Both developers *strongly agreed* with the question. First developer said: *"We do not launch an app unless we make sure that our app can receive high star-ratings. Releasing newer versions normally include adding new features, fixing bugs, and improvement in the app, that all improve users' trust in* *our app. I think this is why the number of releases is an important metric. I also think there are a variety of apps that are better than the older ones, so the launch date makes much sense to me"*. Second developer said: *"I agree that the developers' tasks, such as constantly working on an app, introducing new topics, and releasing an updated version, lead to better ranks"*.

(v) ***Do you think the findings of RQ6.3 are useful and practical for the industry?***

Survey. As shown in Figure 6.4, 80.4% of the developers strongly agree with the findings. However, one developer thinks some suggestions, such as addressing user-reviews, may need a lot of effort.

In-depth Discussions. Both developers *strongly agreed* with the question. First developer said: *"Even not all the famous app names are irrelevant. For example, the names of famous apps, such as Facebook and Whatsapp, are quite relevant to their area."*. Second developer mentioned: *"The highlighted factors, including the sentiment scores and installation file size, correctly show that the users need to be satisfied. Moreover, the release latency is an important issue. Developers need to address the issues quickly and release a newer version"*.

The majority of the developers agree that our findings can be very useful in practice.

6.4 Implications for New App Developers

Newcomers to the app production rally, including start-up founders, entrepreneurs [9], and even a small team of developers, may be worried about the competitive nature of app markets [69, 187]. An important message of this work for newcomers is that *"app developers should not stop improving their apps as they have a great chance to win the rally!"*. Unfortunately, even if developers start with a novel idea, they are in danger of losing their ranks to other companies as soon as they reveal their idea. In other words, newcomers must have a strong personal drive to success [31]. It is not surprising that the lack of discipline and work could make it infeasible to succeed [10].

The key points for newcomers that are discussed in this chapter can show them the right path:

(i) Developers should carefully select the area in which they intend to work. Developers should do their homework and see if there is a huge number of dominant apps in an area or not. The most wanted areas of apps can be a decent start.

(ii) *Never is late* as in the past two years, many apps (other than our subject apps) have achieved higher ranks in Google Play Store.

(iii) Investing time and money on the user-reviews, adding new features, and releasing improved versions can give developers a higher chance to win the app production rally.

(iv) Developer should work on the metrics that can potentially improve the rank of an app. Developers should not be distracted by the metrics that sound critical but are not important.

Developers of new apps with no (or a limited number of) user-reviews can study the user-reviews, features, and descriptions of other apps to get some hints. We investigate the descriptions and release notes of each app in each area and compare them with the user-reviews of other apps in the same area. We notice that, on average, there is a similarity of 63% between the descriptions and release notes of each app and the user-reviews of other apps. Newcomers can identify popular features by studying other apps within the same area.

6.5 Threats to Validity

In this section, we discuss the potential threats to the validity of our experiments [184].

6.5.1 Construct Validity

Martin *et al.* [134] reported that using an incomplete set of user-reviews in Blackberry World app store can introduce bias to the findings. To mitigate such a threat, we gradually retrieved all the user-reviews of our subject apps for two years. Google Play Store does not reveal the information about all the available apps. We mitigate this limitation by clicking on *"Show More"* button as many times as possible to retrieve all the accessible apps.

One way to capture app features is decompiling installation files to byte code. Unfortunately, Google Play Store only reveals the latest version of the apps. We captured the concepts of features by breaking down the release notes and descriptions as Johann *et al.* [96] reported a precision of up to 88% for app descriptions in identifying app features.

6.5.2 External Validity

We rely on Google to get the app rankings. Our findings and experiments will be still useful even if Google completely changes the ranking algorithm in the future for three main reasons. First, the majority of our metrics are related to users and developers activities, such as addressing user-reviews and introducing new topics. Second, the same approach can be used to find the most important metrics in the future with the most current data. Third, Google improves the ranking algorithm over time [166]. However, the metrics that are identified in this chapter have constantly appeared as the metrics that share a significant relationship with the ranks for two years.

The top areas may be subject to change in different geographical locations and time. However, the emphasis of our work is on the differences in areas.

If a company is not interested in the ranks, the result of our study may not be generalized to the usage of such a company. However, it would be still a great opportunity for such a company to improve its rank and attract more customers.

6.5.3 Internal Validity

We study 45 metrics to explain the ranks. However, we do not claim an exhaustive list of explanatory metrics. Future work can shed more light on explaining the ranks by taking more metrics into consideration. Some metrics may not directly impact the ranks. For example, adding a new feature to an app may make its users more satisfied. As a result, the rank of the app can be improved.

6.6 Summary

In this chapter, we study the evolution of ranks in Google Play Store. We observe that there are 13 trends in the ranks, including falling, maintaining, and rising trends. In some areas, such as *budget*, existing apps tend to maintain their ranks. Therefore, it is harder for newer apps to compete with the existing apps. However, by drawing a regression model, we observe that app developers should not be afraid of a late entry into the market because the apps that appeared later achieved higher ranks during our study. There is a chance of improving the ranks by continuously improving the app, such as adding new features and fixing bugs. We also model the ranks across apps and versions of each app. We identified 11 metrics that share a significant relationship with the changes in the ranks, such as the name and the appearance of new topics. Furthermore, developers should not be distracted by some metrics that sound important but do not share a significant relationship with the ranks, such as the number of pictures. Finally, the feedback obtained from 51 app developers confirms that our findings can help app developers to achieve higher ranks in Google Play Store.

Chapter 7

Summary and Conclusions

In this chapter, a summary of the thesis is provided in Section 7.1. Section 7.2 outlines future work. Finally, Section 7.3 concudes the thesis.

7.1 Summary

Having considered the immensely competitive markets of mobile apps, such as Google Play Store [69], developers follow continuous mobile app development paradigm in order to succeed [1]. Receiving high star-ratings and achieving higher ranks are two important elements of success [135]. Therefore, it is vital for app developers to understand the factors that play a major role in receiving high star-ratings and achieving higher ranks.

Chapter 3 investigates app and device metrics that share a significant relationship with star-ratings. We show that star-ratings given by users of different devices are statistically significantly different from each other. Thus, developers should not only consider their own apps but also the devices available in the market. We observe that device metrics, such as display size and CPU, share a significant relationship with star-ratings.

Chapter 4 identifies the key topics of user-reviews, i.e., the topics of user-reviews that are statistically significantly related to star-ratings. We explore 4, 193, 549 user-reviews of 623 apps in ten different categories of apps, such as *Business* and *Tools*. We show that the key topics are not necessarily the most frequent topics of user-reviews. A topic of user-reviews can be frequently discussed, however, such a frequent topic may have a minor relationship with the associated star-ratings. Therefore, developers should focus on the key topics instead of getting distracted by topics that may not actually share a relationship with star-ratings.

Chapter 5 proposes a solution for mapping user-reviews (from Google Play Store) to issue reports (from GitHub). The proposed approach achieves a precision of 79%. In addition, Chapter 5 investigate the metrics of issue reports prioritization that share a significant relationship with star-ratings, such as title size and minimum star-rating.

Chapter 6 identifies the rank trends in the market. We observe that the majority of our subject apps (i.e., 61%) tend to lose their ranks within the two years of our study. App developers should not be afraid of a late entry into the market as newer apps can achieve higher ranks. We show that app developers should take care of user-reviews and constantly improve their apps. Also, developers should be careful not to spend too much budget and time on the factors that sound important but actually do not share a significant relationship with changes in the ranks. Instead, developers should focus on important metrics, such as release latency and introduction of new features.

7.2 Future Work

Although we believe that this thesis has made positive contributions to help developers succeed in markets of mobile apps, there is plenty of room for future research and further contributions. We outline some future opportunities below:

(i) Future studies should explore other aspects and other metrics, such as contextual circumstances (e.g., users' location), that could potentially share a significant relationship with star-ratings. We believe that such aspects may also exhibit relationships with star-ratings and the user-perceived quality of apps.

(ii) Future studies should integrate earlier works that summarizes user-reviews (e.g., [33, 160, 203]) with the key topics of user-reviews. Thus, developers could prioritize and summarize the future changes by focusing on the identified key topics.

(iii) Identifying the key topics of user-reviews can be utilized with code localization techniques (e.g., Palomba *et al.* [161]) to make a direct link to the parts of the source code that must be modified with respect to the key topics.

(iv) Future work should study the generalizability of mapping user-reviews with issue reports more deeply. For example, our approach should be tested with user-reviews from other app markets, such as Apple App Store [8], and issue reports from other issue tracking systems and source code repositories.

(v) Future work should study the ranks in other app markets, such as Apple App Store. The differences and the similarities of various app markets can provide new insights into cross-platform app development.

7.3 Conclusion

Succeeding in the competitive market of mobile apps is not an easy task. App developers should keep improving their apps with respect to the important factors that are identified in this thesis, such as available devices and user-reviews. When addressing the issues that are reported in user-reviews, developers should pay a special attention to the key topics of user-reviews. In addition, with our proposed approach, developers can map user-reviews to issue reports to better manage the issues that are (or are not) reported in the user-reviews. For achieving higher ranks, developers should work on the factors that can potentially help them achieve higher ranks. It is never late for newcomers to the market as we observed that many apps could achieve higher ranks in Google Play Store in the last two years. Our findings and our proposed approaches help developers to enhance their app development process to be more successful in the market.

Bibliography

[1] Bram Adams and Shane McIntosh. Modern release engineering in a nutshell–why researchers should care. In *Proceedings of 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 78–90, 2016.

[2] Charu C Aggarwal and ChengXiang Zhai. *Mining text data*. Springer Science & Business Media, 2012.

[3] Allan J Albrecht and John E Gaffney Jr. Software function, source lines of code, and development effort prediction: a software science validation. *IEEE Transactions on Software Engineering*, SE-9(6):639–648, 1983.

[4] John H Aldrich and Forrest D Nelson. *Linear probability, logit, and probit models*, volume 45. Sage, 1984.

[5] Mamdouh Alenezi and Shadi Banitaan. Bug reports prioritization: Which features and classifier to use? In *Proceedings of the 12th International Conference on Machine Learning and Applications (ICMLA)*, volume 2, pages 112–116. IEEE, 2013.

[6] Allacronyms. Acronyms and abbreviations. [Online]. Available: https://www.allacronyms.com/, 2017.

[7] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1):5–43, 2003.

[8] Apple. App store. [Online] Available: https://www.apple.com/ca/ios/app-store/, 2018.

[9] David B Audretsch and Max Keilbach. Entrepreneurship and regional growth: an evolutionary interpretation. *Journal of Evolutionary Economics*, 14(5):605–616, 2004.

[10] Bill Aulet. *Disciplined entrepreneurship: 24 steps to a successful startup*. John Wiley & Sons, 2013.

[11] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[12] Normi Sham Awang Abu Bakar and Iqram Mahmud. Empirical analysis of android apps permissions. In *Proceedings of the 2013 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pages 406–411. IEEE, 2013.

[13] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pages 280–293. ACM, 2009.

[14] Rajendra K Bandi, Vijay K Vaishnavi, and Daniel E Turk. Predicting maintenance performance using object-oriented design complexity metrics. *IEEE Transactions on Software Engineering*, 29(1):77–87, 2003.

[15] Rajiv D Banker, Srikant M Datar, Chris F Kemerer, and Dani Zweig. Software complexity and maintenance costs. *Communications of the ACM*, 36(11):81–94, 1993.

[16] Victor R Basili. Software modeling and measurement: the goal/question/metric paradigm. Technical report, University of Maryland at College Park, 1992.

[17] Gabriele Bavota, Mario Linares-Vasquez, Carlos Eduardo Bernal-Cardenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. The impact of api change-and fault-proneness on the user ratings of android apps. *IEEE Transactions on Software Engineering*, 41(4):384–407, 2015.

[18] Steven Bellman, Robert F Potter, Shiree Treleaven-Hassard, Jennifer A Robinson, and Duane Varan. The effectiveness of branded mobile phone apps. *Journal of interactive Marketing*, 25(4):191–200, 2011.

[19] Stephen P Berczuk and Brad Appleton. *Software configuration management patterns: effective teamwork, practical integration*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[20] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[21] Dane Bertram, Amy Voida, Saul Greenberg, and Robert Walker. Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 29–300. ACM, 2010.

[22] James C Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2-3):191–203, 1984.

[23] Pamela Bhattacharya, Liudmila Ulanova, Iulian Neamtiu, and Sai Charan Koduru. An empirical analysis of bug reports and bug fixing in open source android apps. In *Proceedings of the 17th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 133–143. IEEE, 2013.

[24] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[25] David M Blei and John D Lafferty. A correlated topic model of science. *The Annals of Applied Statistics*, pages 17–35, 2007.

[26] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[27] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166, 1997.

[28] Andreas Bruns, Andreas Kornstadt, and Dennis Wichmann. Web application tests with selenium. *IEEE software*, 26(5), 2009.

[29] Bugzilla. Bugzilla. [Online]. Available: https://www.bugzilla.org/, 2018.

[30] Juan Cao, Tian Xia, Jintao Li, Yongdong Zhang, and Sheng Tang. A density-based method for adaptive lda model selection. *Neurocomputing*, 72(7):1775–1781, 2009.

[31] Gary J Castrogiovanni. Pre-startup planning and the survival of new small businesses: Theoretical linkages. *Journal of management*, 22(6):801–822, 1996.

[32] Yguaratã Cerqueira Cavalcanti, Paulo Anselmo da Mota Silveira Neto, Daniel Lucrédio, Tassio Vale, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. The bug report duplication problem: an exploratory study. *Software Quality Journal*, 21(1):39–66, 2013.

[33] Ning Chen, Jialiu Lin, Steven CH Hoi, Xiaokui Xiao, and Boshen Zhang. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering*, pages 767–778. ACM, 2014.

[34] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[35] Rosanne Chiem, Jacob Arriola, Derek Browers, Jerome Gross, Enrico Limman, Paul V Nguyen, Dwi Sembodo, Young Song, and Kala Chand Seal. The critical success factors for marketing with downloadable applications: Lessons learned from selected european countries. *International Journal of Mobile Marketing*, 5(2), 2010.

[36] Ronald Christensen. *Log-linear models and logistic regression*. Springer Science & Business Media, 2006.

[37] Adelina Ciurumelea, Andreas Schaufelbhl, Sebastiano Panichella, and Harald Gall. Analyzing reviews and code of mobile apps for better release planning. In *Proceedings of the 24th International Conference on Software Analysis Evolution and Reengineering*. IEEE, 2017.

[38] Norman Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, 114(3):494, 1993.

[39] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Academic press, 2013.

[40] J Chris Coppick and Thomas J Cheatham. Software metrics for object-oriented systems. In *Proceedings of the 1992 ACM annual conference on Communications*, pages 317–322. ACM, 1992.

[41] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[42] Enrique Costa-Montenegro, Ana Belen Barragans-Martinez, and Marta Rey-Lopez. Which app? a recommender system of applications in markets: Implementation of the service for monitoring users' interaction. *Expert systems with applications*, 39(10):9367–9375, 2012.

[43] Enrique Costa-Montenegro, Ana Beln Barragns-Martnez, and Marta Rey-Lpez. Which app? a recommender system of applications in markets: Implementation

of the service for monitoring users interaction. *Expert Systems with Applications*, 39(10):9367 – 9375, 2012.

[44] David Crystal. *Dictionary of linguistics and phonetics*, volume 30. John Wiley & Sons, 2011.

[45] Teerath Das, Massimiliano Di Penta, and Ivano Malavolta. A quantitative and qualitative investigation of performance-related commits in android apps. In *Proceedings of the 2016 International Conference on Software Maintenance and Evolution (ICSME)*, pages 443–447. IEEE, 2016.

[46] Diya Datta and Sangaralingam Kajanan. Do app launch times impact their subsequent commercial success? an analytical approach. In *Proceedings of the 2013 International Conference on Cloud Computing and Big Data (CloudCom-Asia)*, pages 205–210. IEEE, 2013.

[47] Soumya Kanti Datta. Android stack integration in embedded systems. In *Proceedings of the International Conference on Emerging Trends in Computer & Information Technology, Coimbatore, India*, 2012.

[48] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, volume 6, pages 449–454, 2006.

[49] Romain Deveaud, Eric SanJuan, and Patrice Bellot. Accurate and effective latent concept modeling for ad hoc information retrieval. *Document numérique*, 17(1):61–84, 2014.

[50] GitHub Developer. Github developer. [Online]. Available: https://developer.github.com/v3/, 2018.

[51] Dex2jar. dex2jar. https://code.google.com/p/dex2jar/, 2016.

[52] Andrea Di Sorbo, Sebastiano Panichella, Carol V Alexandru, Junji Shimagaki, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. 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*, pages 499–510. ACM, 2016.

[53] Thomas A Domencich and Daniel McFadden. Urban travel demand-a behavioral analysis. Technical report, TRID, 1975.

[54] Olive Jean Dunn. Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252, Aug 1964.

[55] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, volume 96, pages 226–231, 1996.

[56] Julian J Faraway. *Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models.* CRC press, 2005.

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

[58] Barry E Feldman. Relative importance and value. *Available at SSRN 2255827*, 2005.

[59] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th International Conference on Knowledge Discovery and Data Mining*, pages 1276–1284. ACM, 2013.

[60] Laura V Galvis Carreño and Kristina Winbladh. Analysis of user comments: an approach for software requirements evolution. In *Proceedings of the 35th International Conference on Software Engineering*, pages 582–591. IEEE, 2013.

[61] Rajiv Garg and Rahul Telang. Inferring app demand from publicly available data. *SSRN*, 2012.

[62] Walter R Gilks. *Markov chain monte carlo*. Wiley Online Library, 2005.

[63] Github. Github. [Online]. Available: http://www.github.com/, 2017.

[64] Google. Application fundamentals. http://developer.android.com/ guide/components/ fundamentals.html/, 2015.

[65] Google. Layouts. http://developer.android.com/intl/ru/guide/topics /ui/declaring-layout.html/, 2015.

[66] Google. Supporting multiple screens. http://developer.android.com/guide/practices/, 2015.

[67] Google. Android developers blog. [Online] Available: https://android-developers.googleblog.com/2017/02/welcome-to-google-developer-day-at-game.html/, 2017.

[68] Google. Get discovered on google play search. [Online] Available: https://support.google.com/googleplay/android-developer/answer/4448378/, 2017.

[69] Google. Google play store. [Online] Available: http://play.google.com/, 2017.

[70] Michael Goul, Olivera Marjanovic, Susan Baxley, and Karen Vizecky. Managing the enterprise business intelligence app store: Sentiment analysis supported requirements engineering. In *Proceedings of the 45th Hawaii International Conference on System Science (HICSS)*, pages 4168–4177. IEEE, 2012.

[71] Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *the National Academy of Sciences*, 101(suppl 1):5228–5235, 2004.

[72] Tom Griffiths. Gibbs sampling in the generative model of latent dirichlet allocation. *Citeseer*, 2002.

[73] Ulrike Grömping et al. Relative importance for linear regression in r: the package relaimpo. *Journal of statistical software*, 17(1):1–27, 2006.

[74] Xiaodong Gu and Sunghun Kim. " what parts of your apps are loved by users?"(t). In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, pages 760–770. IEEE, 2015.

[75] Emitza Guzman and Wiem Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Proceedings of the 22nd International Conference on Requirements Engineering*, pages 153–162. IEEE, 2014.

[76] Dan Han, Chenlei Zhang, Xiaochao Fan, Abram Hindle, Kenny Wong, and Eleni Stroulia. Understanding android fragmentation with topic analysis of

vendor-specific bugs. In *Proceedings of the 19th Working Conference on Reverse Engineering*, pages 83–92. IEEE, 2012.

[77] Marian Harbach, Markus Hettig, Susanne Weber, and Matthew Smith. Using personal examples to improve risk communication for security & privacy decisions. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pages 2647–2656. ACM, 2014.

[78] Mark Harman, Yue Jia, and Yuanyuan Zhang. App store mining and analysis: Msr for app stores. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 108–111. IEEE, 2012.

[79] Frank E. Harrell. *Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis.* Springer, 2001.

[80] Marc Hassenzahl and Noam Tractinsky. User experience - a research agenda. *Behaviour & Information Technology*, 25(2):91–97, January 2006.

[81] Gregor Heinrich. Parameter estimation for text analysis. *University of Leipzig, Tech. Rep*, 2008.

[82] Gilbert Held and Thomas Marshall. *Data compression; techniques and applications: Hardware and software considerations.* John Wiley & Sons, Inc., 1991.

[83] Niels Henze and Susanne Boll. Release your app on sunday eve: Finding the best time to deploy apps. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, pages 581–586. ACM, 2011.

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

[85] Tin Kam Ho. Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282. IEEE, 1995.

[86] Matthew Hoffman, Francis R Bach, and David M Blei. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*, pages 856–864, 2010.

[87] Ellis Horowitz et al. *Fundamentals of data structures in C++*. Galgotia Publications, 2006.

[88] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[89] Claudia Iacob and Rachel Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 41–44. IEEE, 2013.

[90] Claudia Iacob, Varsha Veerappa, and Rachel Harrison. What are you complaining about?: a study of online reviews of mobile applications. In *Proceedings of the 27th International BCS Human Computer Interaction Conference*, page 29. British Computer Society, 2013.

[91] Selim Ickin, Katarzyna Wac, Markus Fiedler, Lucjan Janowski, Jin-Hyuk Hong, and Anind K. Dey. Factors influencing quality of experience of commonly used mobile applications. *IEEE Communications Magazine*, 50(4):48–56, 2012.

[92] Md Rakibul Islam and Minhaz F Zibran. Leveraging automated sentiment analysis in software engineering. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 203–214. IEEE Press, 2017.

[93] Jiří Janák. Issue tracking systems. *Brno, spring*, 2009.

[94] Jazzy. Jazzy spell checker. [Online] Available: http://jazzy.sourceforge.net/, 2017.

[95] JD. Java decompiler. http://jd.benow.com/, 2016.

[96] Timo Johann, Christoph Stanik, Walid Maalej, et al. Safe: A simple approach for feature extraction from app descriptions and app reviews. In *Proceedings of the 25th International Conference on Requirements Engineering*, pages 21–30. IEEE, 2017.

[97] Ralph E Johnson and Brian Foote. Designing reusable classes. *Object-oriented Programming*, 1(2):22–35, 1988.

[98] Robbert Jongeling, Subhajit Datta, and Alexander Serebrenik. Choosing your weapons: On sentiment analysis tools for software engineering research. In *Proceedings of the 31st Conference on Software maintenance and evolution*, pages 531–535. IEEE, 2015.

[99] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. Real challenges in mobile app development. In *Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 15–24. IEEE, 2013.

[100] Jaweria Kanwal and Onaiza Maqbool. Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology*, 27(2):397–412, 2012.

[101] Fabian Kaup and David Hausheer. Optimizing energy consumption and qoe on mobile devices. In *Proceedings of the 21st IEEE International Conference on Network Protocols*, pages 1–3. IEEE, 2013.

[102] Truman Lee Kelley. *Fundamentals of statistics.* Harvard University Press, 1947.

[103] Hammad Khalid, Meiyappan Nagappan, and Ahmed E Hassan. Examining the relationship between findbugs warnings and app ratings. *IEEE Software*, 33(4):34–39, 2016.

[104] Hammad Khalid, Meiyappan Nagappan, Emad Shihab, and Ahmed E. Hassan. Prioritizing the devices to test your app on: A case study of android game apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pages 370–379, 2014.

[105] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed Hassan. What do mobile app users complain about? a study on free ios apps. *IEEE Software*, pages 1–1, 2014.

[106] Hee-Woong Kim, HL Lee, and JE Son. An exploratory study on the determinants of smartphone app purchase. In *Proceedings of the 11th International Decision Science Institute and the 16th Asia Pacific Decision Sciences Institute Joint Meeting*, 2011.

[107] Hee-Woong Kim, HL Lee, and JE Son. An exploratory study on the determinants of smartphone app purchase. In *Proceedings of the 11th International DSI and the 16th APDSI Joint Meeting*, 2011.

[108] Soo-Min Kim, Patrick Pantel, Tim Chklovski, and Marco Pennacchiotti. Automatically assessing review helpfulness. In *Proceedings of the 2006 Conference on empirical methods in natural language processing*, pages 423–430. Association for Computational Linguistics, 2006.

[109] Nozomi Kobayashi, Kentaro Inui, and Yuji Matsumoto. Extracting aspect-evaluation and aspect-of relations in opinion mining. In *EMNLP-CoNLL*, volume 7, pages 1065–1074. Citeseer, 2007.

[110] Hannu Korhonen, Juha Arrasvuori, and Kaisa Väänänen-Vainio-Mattila. Analysing user experience of personal mobile products through contextual factors. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*, MUM '10, pages 11:1–11:10, New York, NY, USA, 2010. ACM.

[111] William H. Kruskal and W. Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.

[112] Hyun Jung La and Soo Dong Kim. A service-based approach to developing android mobile internet device (mid) applications. In *Proceedings of the 2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–7. IEEE, 2009.

[113] Ahmed Lamkanfi, Serge Demeyer, Emanuel Giger, and Bart Goethals. Predicting the severity of a reported bug. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 1–10. IEEE, 2010.

[114] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the 50th Internrational Conference on Computational Natural Language Learning: Shared Task*, pages 28–34, 2011.

[115] Ding Li and William GJ Halfond. An investigation into energy-saving programming practices for android smartphone app development. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, pages 46–53. ACM, 2014.

[116] Ding Li, Shuai Hao, Jiaping Gui, and William GJ Halfond. An empirical study of the energy consumption of android applications. In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution*, pages 121–130. IEEE, 2014.

[117] Wei Li and Sallie Henry. Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, 23(2):111–122, 1993.

[118] T Warren Liao. Clustering of time series data survey. *Pattern recognition*, 38(11):1857–1874, 2005.

[119] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[120] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[121] Soo Ling Lim and Peter J Bentley. Investigating app store ranking algorithms using a simulation of mobile app ecosystems. In *Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC)*, pages 2672–2679. IEEE, 2013.

[122] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[123] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. Api change and fault proneness: A threat to the success of android apps. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, pages 477–487. ACM, 2013.

[124] Mario Linares-Vásquez, Sam Klock, Collin McMillan, Aminata Sabané, Denys Poshyvanyk, and Yann-Gaël Guéhéneuc. Domain matters: bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in java mobile apps. In *Proceedings of the 22nd International Conference on Program Comprehension*, pages 232–243. ACM, 2014.

[125] Mario Linares-Vásquez, Christopher Vendome, Qi Luo, and Denys Poshyvanyk. How developers detect and fix performance bottlenecks in android apps. In *Proceedings of the 31st Conference on Software Maintenance and Evolution*, pages 352–361. IEEE, 2015.

[126] Julie B Lovins. *Development of a stemming algorithm.* MIT Information Processing Group, Electronic Systems Laboratory, 1968.

[127] Stacy K Lukins, Nicholas A Kraft, and Letha H Etzkorn. Source code retrieval for bug localization using latent dirichlet allocation. In *Proceedings of the 15th Working Conference on Reverse Engineering*, pages 155–164. IEEE, 2008.

[128] Walid Maalej, Maleknaz Nayebi, Timo Johann, and Guenther Ruhe. Toward data-driven requirements engineering. *IEEE Software*, 33(1):48–54, 2016.

[129] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[130] Amiya Kumar Maji, Kangli Hao, Salmin Sultana, and Saurabh Bagchi. Characterizing failures in mobile oses: A case study with android and symbian. In *Proceedings of the 21st International Symposium on Software Reliability Engineering (ISSRE)*, pages 249–258. IEEE, 2010.

[131] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[132] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[133] W Martin, B Al, and P Van Sterkenburg. On the processing of a text corpus: From textual data to lexicographical information. *Academic Press*, 1983.

[134] William Martin, Mark Harman, Yue Jia, Federica Sarro, and Yuanyuan Zhang. The app sampling problem for app store mining. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 123–133. IEEE, 2015.

[135] William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. A survey of app store analysis for software engineering. *IEEE Transactions on Software Engineering*, PP(99), 2016.

[136] Thomas J McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, 1976.

[137] James T McClave, Terry Sincich, and Terry T Sincich. *A first course in statistics*. Pearson Education, 2003.

[138] Tyler McDonnell, Bonnie Ray, and Miryung Kim. An empirical study of api stability and adoption in the android ecosystem. In *Proceedings of the 29th IEEE International Conference on Software Maintenance*, pages 70–79. IEEE, 2013.

[139] Daniel McFadden et al. Conditional logit analysis of qualitative choice behavior. *Institute of Urban and Regional Development, University of California*, 1973.

[140] Stuart McIlroy, Nasir Ali, and Ahmed E Hassan. Fresh apps: an empirical study of frequently-updated mobile apps in the google play store. *Empirical Software Engineering*, 21(3):1346–1370, 2016.

[141] Tim Menzies and Andrian Marcus. Automated severity assessment of software defect reports. In *Proceedings of the International Conference on Software Maintenance*, pages 346–355. IEEE, 2008.

[142] Microsoft. Microsoft store. [Online] Available: http://www.microsoft.com/Store/, 2018.

[143] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[144] Glenn W Milligan and Martha C Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50(2):159–179, 1985.

[145] Kevin Moran, Mario Linares-Vásquez, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. Auto-completing bug reports for android applications. In *Proceedings of the 10th Joint Meeting on European Software Engineering and Foundations of Software Engineering*, ESEC/FSE 2015, pages 673–686, New York, NY, USA, 2015. ACM.

[146] Meiyappan Nagappan and Emad Shihab. Future trends in software engineering research for mobile apps. In *FOSE*, pages 21–32, 2016.

[147] Shinichi Nakagawa and Holger Schielzeth. A general and simple method for obtaining r2 from generalized linear mixed-effects models. *Methods in Ecology and Evolution*, 4(2):133–142, 2013.

[148] Maleknaz Nayebi, Bram Adams, and Guenther Ruhe. Release practices for mobile apps–what do users and developers think? In *Proceedings of the 23rd*

International Conference on Software Analysis, Evolution, and Reengineering (SANER), volume 1, pages 552–562. IEEE, 2016.

[149] John A Nelder and R Jacob Baker. Generalized linear models. *Encyclopedia of statistical sciences*, 1972.

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

[151] Thanh HD Nguyen, Bram Adams, and Ahmed E Hassan. Studying the impact of dependency network measures on software quality. In *Proceedings of the 26th International Conference on Software Maintenance*, pages 1–10. IEEE, 2010.

[152] Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using em. *Machine learning*, 39(2):103–134, 2000.

[153] Ehsan Noei, Daniel da Costa, and Ying Zou. Winning the app production rally. In *Proceedings of the the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, 2018.

[154] Ehsan Noei and Abbas Heydarnoori. Exaf: A search engine for sample applications of object-oriented framework-provided concepts. *Information and Software Technology*, 75:135–147, 2016.

[155] Ehsan Noei, Mark D Syer, Ying Zou, Ahmed E Hassan, and Iman Keivanloo. A study of the relation of mobile device attributes with the user-perceived quality of android apps. *Empirical Software Engineering*, 22(6):3088–3116, 2017.

[156] Ehsan Noei, Feng Zhang, Shaohua Wan, and Ying Zou. Towards prioritizing user-related issue reports of mobile applications. *Empirical Software Engineering*, 2018.

[157] Christiane Nord. *Text analysis in translation: Theory, methodology, and didactic application of a model for translation-oriented text analysis.* Rodopi, 2005.

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

[159] Dennis Pagano and Walid Maalej. User feedback in the appstore: An empirical study. In *Proceedings of the 21st International Conference on Requirements Engineering*, pages 125–134. IEEE, 2013.

[160] Fabio Palomba, Mario Linares-Vásquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In *Proceedings of the 31st International Conference on Software Maintenance and Evolution*, pages 291–300. IEEE, 2015.

[161] Fabio Palomba, Pasquale Salza, Adelina Ciurumelea, Sebastiano Panichella, Harald Gall, Filomena Ferrucci, and Andrea De Lucia. Recommending and localizing change requests for mobile apps based on user reviews. In *Proceedings of the 39th International Conference on Software Engineering*, pages 106–117, 2017.

[162] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimilano Di Penta, Denys

Poshynanyk, and Andrea De Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 522–531. IEEE, 2013.

[163] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, C Visaggio, Gerardo Canfora, and H Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *Proceedings of the 31st International Conference on Software Maintenance and Evolution*, 2015.

[164] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. Ardoc: app reviews development oriented classifier. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 1023–1027. ACM, 2016.

[165] Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. Wordnet:: Similarity: measuring the relatedness of concepts. In *Demonstration papers at HLT-NAACL 2004*, pages 38–41. Association for Computational Linguistics, 2004.

[166] Sarah Perez. Google play now considers user engagement, not just downloads, in ranking games. [Online] Available: https://beta.techcrunch.com/2017/02/28/google-play-now-considers-user-engagement-not-just-downloads-in-ranking-games/, 2017.

[167] Xuan-Hieu Phan and Cam-Tu Nguyen. Jgibblda. [Online]. Available: http://jgibblda.sourceforge.net/, 2008.

[168] Jose Pinheiro, Douglas Bates, Saikat DebRoy, Deepayan Sarkar, et al. Linear and nonlinear mixed effects models. *R package version*, 3:57, 2007.

[169] Susan L Prescott, Claudia Macaubas, Troy Smallacombe, Barbara J Holt, Peter D Sly, and Patrick G Holt. Development of allergen-specific t-cell memory in atopic and normal children. *The Lancet*, 353(9148):196–200, 1999.

[170] Quora. What are the downsides to releasing frequent app updates? [Online] Available: https://www.quora.com/What-are-the-downsides-to-releasing-frequent-app-updates/, 2017.

[171] Adrian E Raftery, Steven Lewis, et al. How many iterations in the gibbs sampler. *Bayesian statistics*, 4(2):763–773, 1992.

[172] Anand Rajaraman, Jeffrey D Ullman, Jeffrey David Ullman, and Jeffrey David Ullman. *Mining of massive datasets*, volume 77. Cambridge University Press Cambridge, 2012.

[173] Alan A Ramaley, Darrell L Aldrich, David M Buchthal, and Thomas W Olsen. Method for managing embedded files for a document saved in html format, July 1 2003. US Patent 6,585,777.

[174] Partha P Ray. Home health hub internet of things (h 3 iot): An architectural framework for monitoring health of elderly people. In *Proceedings of the 2014 International Conference on Science Engineering and Management Research (ICSEMR)*, pages 1–3. IEEE, 2014.

[175] Eric Ries. *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses.* Crown Books, 2011.

[176] Daniel M Romero, Wojciech Galuba, Sitaram Asur, and Bernardo A Huberman. Influence and passivity in social media. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 18–33. Springer, 2011.

[177] Israel J Mojica Ruiz, Meiyappan Nagappan, Bram Adams, Thorsten Berger, Steffen Dienst, and Ahmed E Hassan. Examining the rating system used in mobile-app stores. *IEEE Software*, 33(6):86–92, 2016.

[178] Gerard Salton and J Michael. Mcgill. *Introduction to modern information retrieval*, pages 24–51, 1983.

[179] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[180] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.

[181] Selenium. Selenium - web browser automation. http://seleniumhq.org/, June 2017.

[182] Semrush. Semrush. [Online] Available: https://www.semrush.com/, 2017.

[183] Claude E. Shannon, Warren Weaver, and Arthur W. Burks. The mathematical theory of communication (review). *Philosophical Review*, 60(3):398–400, 1951.

[184] Forrest Shull, Janice Singer, and Dag I.K. Sjøberg. *Guide to Advanced Empirical Software Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[185] Statista. 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, 2017.

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

[187] Statista. Number of apps available in leading app stores as of 2018. [Online] Available: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/, 2018.

[188] AppBrain Stats. Number of android applications. [Online]. Available: http://www.appbrain.com/stats/number-of-android-apps, 2018.

[189] Mark D Syer, Bram Adams, Ying Zou, and Ahmed E Hassan. Exploring the development of micro-apps: A case study on the blackberry and android platforms. In *Proceedings of the 11th IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 55–64. IEEE, 2011.

[190] Seyyed Ehsan Salamati Taba, Iman Keivanloo, Ying Zou, Joanna W. Ng, and Tinny Ng. An exploratory study on the relation between user interface complexity and the perceived quality. In *Proceedings of the 14th International Conference on Web Engineering*, pages 370–379, July 2014.

[191] Seyyed Ehsan Salamati Taba, Foutse Khomh, Ying Zou, Ahmed E Hassan, and Meiyappan Nagappan. Predicting bugs using antipatterns. In *Proceedings of the*

29th International Conference on Software Maintenance, pages 270–279. IEEE, 2013.

[192] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 43(1):1–18, 2017.

[193] Mike Thelwall, Kevan Buckley, and Georgios Paltoglou. Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology*, 63(1):163–173, 2012.

[194] Mike Thelwall, Kevan Buckley, Georgios Paltoglou, Di Cai, and Arvid Kappas. Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*, 61(12):2544–2558, 2010.

[195] Yuan Tian, David Lo, and Chengnian Sun. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In *Proceedings of the 19th Working Conference on Reverse Engineering (WCRE)*, pages 215–224. IEEE, 2012.

[196] Yuan Tian, Meiyappan Nagappan, David Lo, and Ahmed E Hassan. What are the characteristics of high-rated apps? a case study on free android applications. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 301–310. IEEE, 2015.

[197] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[198] Raoul Vallon, Lukas Wenzel, Martin E Brüggemann, and Thomas Grechenig. An agile and lean process model for mobile app development: Case study into austrian industry. *JSW*, 10(11):1245–1264, 2015.

[199] Rini Van Solingen, Vic Basili, Gianluigi Caldiera, and H Dieter Rombach. Goal question metric (gqm) approach. *Encyclopedia of software engineering*, 2002.

[200] Brandon K Vaughn. Data analysis using regression and multilevel/hierarchical models, by gelman, a., & hill, j. *Journal of Educational Measurement*, 45(1):94–97, 2008.

[201] Pradeep K Venkatesh, Shaohua Wang, Feng Zhang, Ying Zou, and Ahmed E Hassan. What do client developers concern when using web apis? an empirical study on developer forums and stack overflow. In *Proceedings of the 2016 IEEE International Conference on Web Services (ICWS)*, pages 131–138. IEEE, 2016.

[202] Nicolas Viennot, Edward Garcia, and Jason Nieh. A measurement study of google play. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 221–233. ACM, 2014.

[203] Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering*, pages 14–24. ACM, 2016.

[204] Haixun Wang, Sanghyun Park, Wei Fan, and Philip S Yu. Vist: a dynamic index method for querying xml data by tree structures. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 110–121. ACM, 2003.

[205] Anthony I. Wasserman. Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, pages 397–400, New York, NY, USA, 2010. ACM.

[206] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.

[207] Bodo Winter. A very basic tutorial for performing linear mixed effects analyses. *arXiv preprint arXiv:1308.5499*, 2013.

[208] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[209] Jifeng Xuan, He Jiang, Zhilei Ren, and Weiqin Zou. Developer prioritization in bug repositories. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 25–35. IEEE, 2012.

[210] Robert K Yin. *Case study research: Design and methods*. Sage publications, 2013.

[211] Lian Yu, Wei-Tek Tsai, Wei Zhao, and Fang Wu. Predicting defect priority based on neural networks. In *Proceedings of the International Conference on Advanced Data Mining and Applications*, pages 356–367. Springer, 2010.

[212] Yue Yu, Huaimin Wang, Vladimir Filkov, Premkumar Devanbu, and Bogdan Vasilescu. Wait for it: determinants of pull request evaluation latency on github. In *Proceedings of the 12th working conference on Mining software repositories (MSR)*, pages 367–371. IEEE, 2015.

[213] Lotfi Asker Zadeh. Fuzzy logic. *Computer*, 21(4):83–93, 1988.

[214] Feng Zhang, Audris Mockus, Iman Keivanloo, and Ying Zou. Towards building a universal defect prediction model with rank transformed predictors. *Empirical Software Engineering*, pages 1–39, 2015.

[215] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. Discovery of ranking fraud for mobile apps. *IEEE Transactions on knowledge and data engineering*, 27(1):74–87, 2015.

[216] Alain Zuur, Elena N Ieno, Neil Walker, Anatoly A Saveliev, and Graham M Smith. *Mixed effects models and extensions in ecology with R*. Springer Science & Business Media, 2009.