

EMPIRICAL STUDIES OF THE DISTRIBUTION AND
FEEDBACK MECHANISMS OF MOBILE APP STORES

by

STUART MCILROY

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
September 2014

Copyright © Stuart Mcilroy, 2014

Abstract

Mobile app stores are online stores where users can purchase and download mobile apps. The marketplace for apps has exploded over recent years with hundreds of thousands of apps, millions of dollars in revenue and millions of users. Mobile app stores offer unique distribution and feedback mechanisms. The distribution mechanism of app stores acts as a central repository for all apps with an option to automatically update apps on user's devices. The feedback mechanism of app stores is a central rating and review system for all apps in a store. Consequently, there is a need for research to explore such unique mechanisms. In this thesis, we analyze over 10,000 of the top free apps across all the app categories in the Google Play Store. We first examine the distribution mechanisms of app stores and its impact on the release speed of apps. We then study the feedback mechanism of app stores to observe how it allows for a large influx of reviews and the benefits of responding to reviews. Finally, we propose an approach to automatically label reviews to help cope with the large influx of reviews by summarizing them.

In regards to the unique distribution mechanism, we observe that a subset of apps release very frequently - more than one release per week. Almost half of these apps do not provide users with a rationale for a new release. Developers are leveraging the distribution mechanism by choosing to rapidly release new versions of their app.

In regards to the unique feedback mechanism, we find that most apps do not receive

many reviews, however, approximately 1% of apps receive over 500 reviews per day. We find that responding to reviews can lead to a positive increase in the rating of the app. The feedback mechanism enables users to give rapid and rich feedback.

Finally, we propose an approach to label user reviews automatically using 14 different labels that capture the user's perceived quality of an app. We demonstrate the usefulness of our labelling approach through three different analytics use case scenarios: competitive analysis, app store overview and anomaly detection.

Acknowledgments

First of all, I'd like to thank my parents who provided constant love and support throughout my Master's degree. I wouldn't be the person I am today without them.

I'd like to thank Ahmed E. Hassan for providing me the opportunity to attend Queens and work in his lab. His wise words guided me throughout my Master's degree. He pushed me to exceed my own expectations and laid the foundations for my success as a researcher and academic writer.

I would like to thank my mentors Nasir Ali and Mei Nagappan. Nasir Ali was able to guide me out of research difficulties when my research progress stalled. His knowledge and constant encouragement were essential. Mei Nagappan helped find my footing and guide me early on in my Master's degree.

Finally, I'd like to thank my friends at Queen's and in the Kingston area that I met during my Master's degree for the fond memories and fun distractions from my work.

Statement of Originality

I, Stuart Mcilroy, hereby declare that I am the sole author of this thesis. All ideas and inventions attributed to others have been properly referenced. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made electronically available to the public.

Table of Contents

Abstract	i
Acknowledgments	iii
Statement of Originality	iv
List of Tables	vii
List of Figures	ix
1 Introduction	1
1.1 Research Statement	3
1.2 Thesis Overview	3
1.3 Thesis Contribution	5
1.4 Organization of the Thesis	5
Chapter 2:	
Background and Related Work	7
Chapter 3:	
Fresh Apps: An Empirical Study of Frequently-Releasing Mobile Apps	14
3.1 Introduction	15
3.2 Traditional Software Releases	17
3.3 Empirical Study Design	18
3.4 Empirical Study Results	21
3.5 Discussion	37
3.6 Threats to Validity	41
3.7 Conclusion	42

Chapter 4:		
	Review Overload: An Empirical study of the Influx of User Reviews for Mobile Apps	44
4.1	Introduction	45
4.2	Empirical Study Design	47
4.3	Empirical Study Results	49
4.4	Discussion	66
4.5	Threats to Validity	68
4.6	Conclusion	69
Chapter 5:		
	Analyzing and Automatically Labelling The Types of User Issues that are Raised in Mobile App Reviews	70
5.1	Introduction	71
5.2	Opinion Mining	74
5.3	Empirical Study	76
5.4	Applications of our Multi-labelling Approach	99
5.5	Discussion	115
5.6	Threats to Validity	123
5.7	Conclusion	125
Chapter 6:		
	Conclusions and Future Work	126
6.1	Summary	126
6.2	Limitations and Future Work	128

List of Tables

3.1	Number of versions per apps as of Feb 16, 2014	20
3.2	Types of tasks, the frequency of non-silent releases with a given task	33
3.3	The file type groupings, their descriptions and their extensions	35
4.1	Our observations on Google Play apps compared to Pagano and Maalej’s observations on Apple’s App Store	50
4.2	Types of topics for the most global and local topics.	60
4.3	Types of reviews written by users in our sample.	63
4.4	Types of replies written by developers in our sample.	64
4.5	Coupling of review types with response types.	66
5.1	The number of user reviews occurring in a five day window (Sept 4 to Sept 9 2013).	72
5.2	Datasets of prior work in mining mobile reviews	76
5.3	User reviews were collected from these 20 apps on the Apple App Store and user reviews from 4 of these apps were collected from the Google Play Store	80
5.4	Data statistics for data collected from the Apple App Store and the Google Play Store	81
5.5	Descriptions of each issue type.	82
5.6	The percentage of issue types in our labelled dataset as well as the probability that a specific issue type co-occurs with other issues i.e., a multi-labelled user review	84
5.7	Example of evaluation measures	95
5.8	Performance of a multi-labelled approach on 14 labels	96
5.9	Performance of a classifier on single labelled user reviews from the Apple App Store and Google Play Store	97
5.10	Performance of approaches on the user reviews from the Apple App Store and Google Play Store	99
5.11	Performance of our multi-label approach on a random sample of the Google Play Store user reviews	101

5.12	Store categories that have specific issue types above or below two standard deviations of the mean in our Google Play Store data set	108
5.13	Percentage of anomalous apps that are inaccessible in the store (1 year later) and anomalous apps flagged by our approach for 1, 2 and 3 standard deviations. N/A denotes there were no apps at that range	111
5.14	Most anomalous apps from the four categories that were flagged in our analysis of the Google Play Store dataset	113
5.15	Comparison of training on Google Play Store and testing on Apple App Store and vice versa	116
5.16	Examples of general topics about photo and videogames with LDA (14 topics) and specific topics about gps and requesting help with LDA (150 topics)	121
5.17	Results for keyword search	122
5.18	Top 20 most frequent anomalous (3 standard deviations) words across apps	123

List of Figures

2.1	The process that a developer follows to publish or update an app in the Google Play and Apple App Stores.	10
2.2	Example of the “What’s New” section for the app ‘Scribd’	10
2.3	The review and response process between a developer and user.	12
3.1	Three examples of calculating release period entropy. The release window is 7 days. For App A, 2 release periods are grouped together and 1 release period remains distinct. For App B, 3 release periods are grouped together. For App C, all release periods are distinct.	23
3.2	Analysis of release period consistency	24
3.3	Apps divided into three groups: releasing less than bi-weekly, releasing bi-weekly to weekly and releasing weekly or more frequently (based on crawling for over a six week period).	25
3.4	Percentage of studied apps in each category who release bi-weekly or more frequently	26
3.5	Mode day of the week for each app’s release date	27
3.6	Amount of coupling of tasks in the “What’s New” section. An edge between two tasks means that they co-occur. The number above the edge is the percentage of releases where both of these tasks co-occur. The number in a task circle indicates the percentage of releases where such a task occurred in the “What’s New” section.	29
3.7	The total APK file size growth of frequently-releasing silent and non-silent apps over the study period	36
3.8	The median change in file type groups of the frequently-releasing apps	37
3.9	Percentage of negative star ratings for infrequent and frequent releasing apps.	39
3.10	Percentage of negative star ratings for infrequent large, infrequent small, frequent large and frequent small apps	40
4.1	Beanplots showing amounts of reviews per day and in total.	52

4.2	Data plot of the total reviews (logged) on the y-axis and three separate graphs of the app categories, the number of downloads and the number of releases on the x-axis. The graph demonstrates the correlation between the three factors and the total number of reviews.	53
4.3	A nomograph of the impact of the new releases, app category and number of downloads on the total number of received reviews.	54
4.4	Standard deviation of new reviews every 24 hours before and after the first collected release for each app in our dataset. Each boxplot represents the standard deviation from the median for every app at that time.	55
4.5	Our topic modelling process.	57
4.6	Entropy of each topic across the the 100 most reviewed apps (100, 150 and 500 topics)	61
4.7	Percentage of replies to number of total reviews for each app separated by the number of downloads. Excluding apps with zero replies.	63
4.8	Percentage of negative app ratings for infrequently-responding vs. frequently-responding apps.	68
5.1	An example of a Google Play Store one-star user review of ‘theScore: Sports & Scores’ app. The review contains (raises) multiple issue types . . .	73
5.2	Amount of co-occurrence per issue types.	85
5.3	A bean plot of the length of user reviews in both single and multi-labelled data for both stores	86
5.4	Process to label multi-labelled user reviews.	87
5.5	Examples of Binary Relevance, Classifier Chains and Pruned Sets with Threshold Extension	90
5.6	Comparison of 3 similar rated popular apps with similar ratings in the Weather category	104
5.7	Distribution of issue types for 1 and 2 star user reviews in the Google Play Store dataset	108
5.8	Comparison of the Kindle app for the Google Play and Apple App Stores .	109
5.9	Box-plots with the anomalous apps as points in the graph above two standard deviations of the mean	114
5.10	Distribution of the frequency of occurrence of anomalous words across all apps (sorted by frequency).	122

Chapter 1

Introduction

Mobile app stores are online distribution channels for mobile applications (mobile apps). There are several hundred app stores in existence. Popular app stores include the Apple App Store (Apple, 2014c), the Blackberry World Store (BlackBerry, 2014), the Google Play Store (Google, 2014b), and Microsoft Phone Apps Store (Microsoft, 2014). Mobile app stores have seen massive growth in recent years. Apps stores see millions of dollars in revenue (Butler, 2011; Gartner, 2013) and Apple has paid over 9 billion to app developers (Apple, 2014a). There are over 1 million apps in both the Google Play Store and Apple App Store (Bostic, 2013; Ingraham). There are over 275,000 app developers on the Apple App Store. There have been over 50 billion downloads as of July 2013 (Statista, 2014). Researchers have estimated that there are almost 1.2 billion mobile app users worldwide (Whitfield, 2014).

Recent studies have shown that there exists numerous differences between mobile apps and traditional software (Minelli and Lanza, 2013; Syer, 2013). More specifically, that the different platforms have different definitions of software quality (Syer et al., 2011a) and that the app stores have different software economies than traditional software (Chen, 2009).

Additionally, mobile app stores offer mechanisms to developers and users that are unique from traditional software. Yet there is a lack of studies specifically on how the differences between app store distribution and feedback mechanisms effect developers, users and app store owners.

Distribution Mechanism Mobile app stores act as central repositories of apps where users are able to browse all available apps and update their current apps. Developers can author an app and upload the app to the app store. An author can set a price or release the app for free. When the app is approved by the app store owner, the app is released to the public. If a developer chooses to publish a new version of the app, the new version can be automatically pushed to the current users of the app. In this way, developers can rapidly release a new version of their app and see it pushed to users the same day (Google, 2014c). In traditional software development, software developers are shifting towards a more mobile-oriented development environment. Large open source companies are shifting towards faster release cycles (Khomh et al., 2012; Mntyl et al., 2013).

Feedback Mechanism Additionally, mobile app stores provide the ability for users to rate and leave reviews of apps that users have downloaded. The feedback is publicly available and offers developers (and even other users) insight into the problems and requests of users. The mechanism is convenient as users can be prompted within the app or by visiting the store page of the app. Reputation systems (i.e., ability for users to leave reviews) have been shown to reduce incoherent user behavior (Dini et al., 2012). The convenience of the review system allows for a potentially massive amount of reviews. For example, Facebook received over 4,275 reviews in one day (Pagano and Maalej, 2013). In traditional software, feedback usually comes in the form of emails and bug reporting (Crowston et al., 2012). Also developers in the Google Play Store are able to respond to individual

reviews allowing developers to interact with users and respond to complaints. In traditional software, developers have shown a willingness to engage with users for features and bug reporting (Crowston et al., 2008; Dabbish et al., 2012).

1.1 Research Statement

Mobile app stores have unique distribution and feedback mechanisms that do not exist in traditional software development. In depth empirical studies and new approaches are needed to better understand the impact of these mechanisms and leverage the rich information that such mechanisms provide.

1.2 Thesis Overview

We first conduct two empirical studies to better understand mobile app distribution and feedback mechanisms. We follow up these studies with a proposal for an approach to assist developers, users and app store owners in dealing with the challenges presented by the unique mechanisms. In all three studies, we analyze a set of over 10,000 apps. The 10,000 apps are taken from a list of the most popular apps in each of Google Play's thirty app categories. We give below a brief summary of our three empirical studies.

1. *Fresh Apps: An Empirical Study of Frequently-Releasing Mobile Apps* (Chap. 3)

In our first study, we focus on the distribution mechanism of apps. In the study, we observe the release patterns and characteristics of mobile apps. We focus our research on the most-frequently releasing apps in our studied dataset.

We find that most apps release in-frequently yet there is a very small subset of apps that release very frequently (as fast as once per week). These apps often do not mention what has changed in their release and when developers do communicate a change, the changes are often primarily due to fixing bugs.

2. ***Review Overload: An Empirical study of the Influx of User Reviews for Mobile Apps (Chap. 4)***

In our second study, we shift to the study of the feedback mechanism of app stores. We analyze the influx of user reviews in our dataset. We find that most apps receive 20 or less reviews but a small subset of apps receive over 500 reviews per day - highlighting the need for more research in user review summarization. We also find that developers benefit from responding to reviews with users changing their rating 38.7% of the time.

3. ***Analyzing and Automatically Labelling The Types of User Issues that are Raised in Mobile App Reviews (Chap. 5)***

In our third study, we propose an approach to assist developers, users and app store owners in dealing with the influx of user reviews. We apply machine learning tools to automatically label reviews with pre-defined labels that capture users' impressions of the quality of an app (i.e., user perceived quality).

We demonstrate three analytics use case scenarios of our automated labelling approach for three separate stakeholders (e.g., app store owners, developers and users).

1.3 Thesis Contribution

In this thesis, we empirically study over 10,000 apps and the impact of mobile store's distribution and feedback mechanisms. In particular, we make three contributions:

1. We show that the distribution of app release frequencies is highly skewed and small subset of apps release very frequently. Such frequent-releasing apps have a higher average rating than infrequent-releasing apps.
2. We show that some apps receive many more times the reviews than the average app receives. This finding motivates researchers to study approaches to summarize reviews. The benefits of these approaches are limited to the subset of apps that receive many reviews per day.
3. We propose an approach to automatically label reviews and showcase three analytics use case scenarios to aide three stakeholders in the mobile marketplace (e.g., app owners, developers and users).

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 provides an overview of the mobile marketplace and app stores as well as current research regarding mobile app releases, reviews and summarization of reviews.

Chapter 3 presents our empirical study on the release patterns and characteristics of frequently-releasing apps.

Chapter 4 presents our empirical study of the influx of reviews, their varied nature and the benefits of responding to reviews.

Chapter 5 presents an approach to automatically label reviews with pre-defined labels for three different analytics use case scenarios.

Chapter 6 concludes the thesis with a discussion of our results, the limitations and threats to validity, and possible avenues for future work.

Chapter 2

Background and Related Work

This section presents an overview of the mobile app marketplace, related work on empirical studies of mobile apps and mobile app analytics.

2.0.1 App Stores

The mobile app market continues to grow at a very rapid pace with thousands of developers, thousands of apps, and millions of dollars in revenue as of September 2013 (Gartner, 2013). Mobile apps are available through app stores, the largest app stores are the Apple App Store, the Blackberry World Store, the Google Play Store and the Microsoft Phone Apps Store. However, there are many more specialized or regional app stores with thousands of apps. Such stores provide convenient and efficient mechanisms for developers to distribute their apps. Such stores also provide a simple, public and intuitive mechanism for users to review an app. These rapidly growing mobile app stores with thousands of apps have not only attracted the attention of both researchers and software engineers but have also posed new challenges for them as well (Harman et al., 2012; Kim et al., 2011; Linares-Vásquez et al.,

2013; Mudambi and Schuff, 2010).

Ruiz *et al.* studied software reuse in the Google Play Store. They found a great deal of reuse which might explain some of the rapid growth of the market (Ruiz *et al.*, 2012). Linares-Vásquez *et al.* empirically analysed how the stability and fault-proneness of app store APIs used by some free Android apps could impact the success of apps (Linares-Vásquez *et al.*, 2013). Lim and Bentley analyzed different ranking algorithms for mobile stores and found that the frequency of new releases impacted the ranking of an app (Lim and Bentley, 2013). Chia *et al.* studied the use of permissions and their misuse (Chia *et al.*, 2012). Pandita studied automatic risk assessment of mobile app permissions (Pandita *et al.*, 2013).

Each app in the app stores has its own homepage. The page contains meta-data about the app such as the title, developer, average rating, description, similar apps, the upload date, recent user reviews, screenshots, video previews and a link to download the app. Some of the recent studies have shown that traditional software repositories and software are different from mobile apps (Harman *et al.*, 2012; Syer *et al.*, 2013). Thus, there is a need for specific studies related to mobile apps (Syer *et al.*, 2013). Some of the studies have been conducted by researchers to better understand the problems facing mobile app developers. Minelli and Lanza showed that mobile apps source code and API use are different than traditional software (Minelli and Lanza, 2013). Syer *et al.* explored the development of Android and Blackberry apps and found that Android apps relied heavily on the Android platform and had fewer files than the Blackberry apps (Syer *et al.*, 2011b). Linares-Vásquez *et al.* explored the issues that mobile developers discussed on StackOverflow (Linares-Vásquez *et al.*, 2013).

Apps can be downloaded and updated from the stores. Once downloaded a user can

review the app. The number of reviews associated to an app varies depending on the popularity of the app. Some popular apps have millions of user reviews.

Reviews in both stores contain a title, a date, a numerical rating between 1 and 5 (where 1 represents a poor app) and a comment section where the user is free to write whatever they wish.

2.0.2 Google Play Store and Apple App Store

In this thesis, we chose two of the most popular app stores to study, the Google Play Store and the Apple App Store. Our criteria for selection is based on popularity (the Google Play Store and the Apple App Store are the top two most popular app stores) and the availability of tools to automatically collect reviews from both app stores (the tool that we used for the Apple App Store has since been made incompatible with an updated version of the store).

The *Google Play Store* is a digital distribution outlet run by Google. Apart from apps, the Play Store sells other digital media e.g. e-books, movies and music. It has over 1,000,000 apps available as of July 2013 (Bostic, 2013). There are paid and free apps available in the stores. Apps can be updated from the store.

The *Apple App Store* is the digital distribution outlet for Apple where users can download third-party apps. The apps range from games, productivity apps, social networking and business apps. There are over 1 million apps in the App Store as of October 2013 (Ingraham).

2.0.3 Publishing an App

In order to publish an app on the Google Play Store or Apple App Store the developer must follow the guidelines and instructions of the app store as Figure 2.1 demonstrates. The

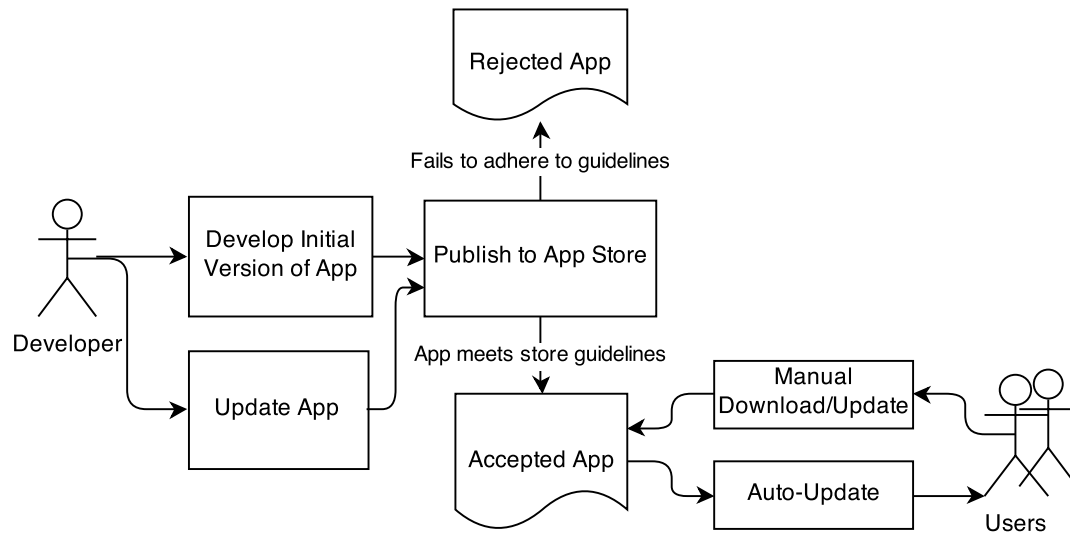


Figure 2.1: The process that a developer follows to publish or update an app in the Google Play and Apple App Stores.

What's New

Version 3.2.1

- Immersive mode is now supported on KitKat devices, so you can use your whole screen to read
- New, nicer-looking thumbnails
- Removed un-needed permissions
- Other bug fixes and improvements

Version 3.2.0

- Added more font size options so you can get just the right size for you
- You can now like a book on Scribd - just tap the share icon while reading and pick "Like on Scribd"
- Applied another coat of polish to the app

Figure 2.2: Example of the “What’s New” section for the app ‘Scribd’

developer must ensure that their app meets the usability and reliability standards of the app store. Once the developer is confident that the app meets the quality standards and respects the content guidelines of the app store, the developer may submit it to the store. Apple requires a 99\$ developer fee per year, whereas Google requires a one time 25\$ developer fee (Viswanathan, 2011).

Once an app is submitted, the app is reviewed by both stores. The Apple review process involves a manual review of the app, whereas the Google review process is automated. The Apple process can take several days or even weeks and it is not uncommon for apps to be rejected. In the case of the Google Play Store, the app is verified with automated tests for malware and within a few hours the app is available on the store.

In the Google Play Store, an app is stored as an APK file. In the case of the Apple App Store, an app is stored as an IPA file. The binary file, e.g. APK/IPA, contain the app's entire content including code, art assets, and configuration files.

Once an app is published, a developer can update it as much as he or she wishes. In Google's case, a new release is available immediately and users are notified that there is a new release to download. For Apple, the update undergoes a review by Apple (Apple, 2014b). Users also have the option to enable automated updates of their apps if a new release becomes available. There is an optional *release note*-like section on an app's store page called "What's New" where the developer can provide a brief message about what has changed in the new release. Figure 2.2 shows an example of a typical text for the "What's New" section.

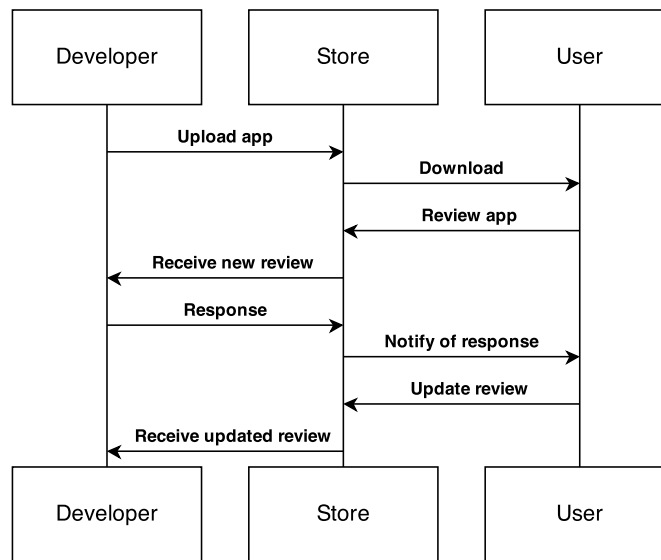


Figure 2.3: The review and response process between a developer and user.

2.0.4 User Feedback of Apps

Once an app is released to the store users have the option of downloading the app. Once a user downloads the app the user is then allowed to leave a rating, a review or both. The rating and review can both be updated. The previous rating or review is overwritten.

Developers are able to respond to a review. The developer's response is public for anyone to see (not just the particular user). The user is then notified that the developer has left a response. If the developer wishes, she may overwrite the response with a new response. Figure 2.3 shows the process of rating an app.

Harman *et al.* mined the Blackberry app store to analyze downloads and ratings of apps and found correlation between customer rating and number of downloads. They did not find any correlation between price and rating (Harman et al., 2012). Kim *et al.* (Kim et al., 2011) conducted interviews of app buyers and discovered that reviews are one of the key determinants in the user's purchase of an app.

2.0.5 Mobile App Analytics

Vision Mobile performed a survey of 7,000 developers and found that 40% of developers make use of user analytics tools (mobile, 2014) and 18% use crash reporting and bug tracking tools. Previous studies also highlight that developers need app analytics tools. For example, Pagano and Bruegge conducted a study on how feedback occurs after the initial release of a product (Pagano and Bruegge, 2013). The authors concluded that there is a need to structure and analyze feedback, particularly when it occurs in large quantities.

Nowadays, there are many app analytics companies that specialize in giving developers tools to understand how users interact with the developers' apps, how developers generate revenue (in-app purchases, e-commerce, direct buy) and the demographics of app users (Adobe, 2014; Annie, 2014; Distimo, 2013; Flurry, 2014). These app analytics companies also provide developers with overviews of user feedback and logged crash reports. Google has their own extensive analytics tools for Android developers. The tools measure how users are using an app (e.g., identify the locations of users and how they reached the app). The tools also track sales data (e.g., tracking how the developer makes money through in-app purchases and calculating the impact of promotions on the sales of the app (Google, 2014a)).

Chapter 3

Fresh Apps: An Empirical Study of Frequently-Releasing Mobile Apps

Mobile app stores provide a unique mechanism for developers to rapidly release and deploy new versions of their apps. In this chapter, we study the release patterns of 10,713 mobile apps (the top free 400 apps at the start of 2013 in each of the 30 categories in the Google Play store). We find that a small subset of these apps (98 apps representing ~1% of the studied apps) are released at a very frequent rate – more than one release per week. We examine 852 releases of the top 100 frequently-releasing apps and about half of the releases do not provide the users with any information about the rationale for the new release. When the rationale for the new release is communicated to users, it is often due to bug-fixing (63% of the time) – followed by new features (35%) and improvements (30%). The median growth rate of the binary of these frequent releasing apps is about 6% over 47 days, even after one year of development. Our study is the first to investigate the release patterns of frequently-releasing mobile apps.

3.1 Introduction

Mobile app stores provide a unique distribution mechanism that enables developers to rapidly release and deploy new versions of their apps. Once a developer uploads a new version of an app (i.e., a new release), the updated release is available to the app's user base immediately with no cost to the developer. Such a rapid and low cost deployment mechanism is unique compared to traditional software applications. The unique distribution mechanism is beneficial to the three stakeholders of the mobile marketplace (i.e., developers, users, and app store owners).

Moreover, app stores indirectly reward frequently-releasing apps by factoring in the freshness of an app for ranking apps across the store (Lim and Bentley, 2013; Lynch, 2012). Maintaining a high ranking is essential for developers of mobile apps given the large number of apps in mobile markets.

In the first part of our empirical study, we seek to shed some light on the mobile apps release patterns and the frequency at which developers make use of such a convenient distribution mechanism for updates, i.e., how often do developers update their apps. We examined the release patterns of 10,713 mobile apps (the top 400 free¹ apps at the start of 2013 in each of the 30 categories in the Google Play store). We crawled the Google Play store over a 47 day time period at the beginning of 2014. We chose apps that had been popular for at least one year to ensure that the observed release patterns are more representative of a stable app not one that is in its early lifetime (where frequent releases are more likely to be expected).

We structure our study around the following research questions:

RQ1: How frequent and consistent are apps updated?

¹We define a free app as an app that is free-to-download.

~14% of the studied apps release new versions on a bi-weekly basis (or more frequently). With ~1% of the apps releasing at least one update per week. Apps in the “Social” category in the Play market are the most frequently-updated. We also note that there is no consistent release pattern – no consistency on which day of the week or release period, i.e., length of time between releases.

RQ2: What is the rationale for such frequent releases?

Almost half (45%) of the releases in the top 100 frequently releasing apps do not contain any information about what has changed (i.e., silent releases). When the rationale for the release is communicated to users, the release is often due to bug-fixing (63% of the time) – followed by new features (35%) and improvements (30%)

RQ3: What is actually changing for such frequent releases?

The binaries (APK files) of silent releases exhibit a similar amount of churn as that of non-silent releases. The source code is the most modified part of the binaries. Overall, the binary files exhibit moderate growth – a median growth of 6% across the frequently-releasing apps.

This chapter is the first study of its kind to explore the release patterns of frequently-releasing mobile apps. Such a study sheds light onto how developers of frequently-releasing mobile apps make use of the convenient and unique release deployment frameworks offered by mobile stores. Much like how developers of mobile apps make use of analytical tools and surveys to understand how the market works (Adobe, 2014; Annie, 2014; Distimo, 2013; Flurry, 2014; mobile, 2014), our findings are of value to developers who wish to compare their own release patterns to that of their competitors and peers in the mobile

marketplaces. Moreover, developers do not often mention the rationale for a new release. Such expectations are in contrast to traditional software systems, where detailed release notes are expected and releases are done in a more systematic way and at a much slower pace (Michlmayr et al., 2007).

This chapter is organized as follows: Section 3.2 presents the related work on traditional software releases. Section 3.3 presents the design of our study. Section 3.4 provides the results of our empirical study. Section 3.5 discusses our findings. Section 3.6 discusses the threats to the validity of our study. Section 3.7 concludes our work.

3.2 Traditional Software Releases

To the best of our knowledge, our study is the first attempt to empirically analyze the frequently-releasing mobile apps. This section looks at the literature related to release cycles of traditional software development.

Due to the migration of open source projects to agile development, many projects moved to rapid release cycles (Khomh et al., 2012). Otte et al. performed a survey to analyze the frequency of software releases (Otte et al., 2008). Their survey shows that 49.3% of software is released at least once per a quarter, 32.7% releases every 6 month and roughly 18% have even longer intervals. Baskerville and Pries-Heje observed the reasons for short release cycles and noted that software projects continued to grow (Baskerville and Pries-Heje, 2004). Developers and researchers have developed new tools and techniques to enable continuous delivery (Humble and Farley, 2010; Porter et al., 2006). Dolstra et al. developed framework to automatically build and deploy code changes (Dolstra et al., 2004). Amazon, for example, deploys on average every 11.6 seconds (Jenkins, 2011), achieving more than 1,079 deployments per hour.

Some researchers have explored the impact of software releases on code quality. Kuppuswami et al. showed that small incremental releases reduce the required development by 2.67% (Kuppuswami et al., 2003). Marschal observed that short cycle of releases involve a steady flow of releases in order to control the number of reported bugs (Marschall, 2007). Escrow.com reduced 70% of the defects by reducing its release cycle to bi-weekly (Hodgetts and Phillips, 2002). Khomh *et al.* showed that shorter release cycles in open source software did not lead to more bugs and bugs were fixed faster (Khomh et al., 2012). Khomh *et al.* also found that the systems that they studied had bi-weekly release patterns. Our own work shows that 14% of apps released a new version on a bi-weekly basis or more frequently.

3.3 Empirical Study Design

In this section, we present the design of our study and the data collection and processing methods used in our study.

3.3.1 Study Goals and Quality Focus

The *goal* of our empirical study is to analyze the release patterns and characteristics of frequently-releasing mobile apps.

The *quality focus* of our empirical study is the frequency of releases of mobile apps, maintenance tasks and APK change. Our *perspective* is that of practitioners (i.e., stakeholders) and researchers interested in apps that are being released frequently. The *object* of our case study is the different releases of the top 12,000 apps in the Google Play Store. In this empirical study, we address three RQs as defined in Section 3.1.

Data Selection

We select the Google Play Store as our app store of interest. Our criteria for an app store is based on the popularity (the Google Play Store is one of the most popular app stores) and that tools exist to automatically collect information from the Google Play Store. We select a total of 12,000 free-to-download apps from the Google Play Store. We select the top 400 most popular apps in the USA for each of the thirty different categories e.g., Photography, Sports and Education. The popularity ranking is based on Distimo's ranking of apps. Distimo is an app analytic company (Distimo, 2013). The list was selected from the most popular apps according to Distimo in the Spring of 2013. As we previously stated, we chose apps that were popular one year ago to avoid the expected frequent releases of a brand new app.

Data Collection

We use an open source Google Play Store crawler (Akdeniz, 2013) to extract the apps' information and APK files. The crawler simulates a mobile device and interfaces with the Google Play API as a regular mobile device. We select the Samsung Galaxy S3 phone's as our simulated device since it is one of the most popular Android devices (Chen, 2012). We modified the crawler for our purposes. We only extract the relevant information for our study, we instituted a timer to pause the crawler to avoid sending too many requests to Google's servers and we scaled the crawler over multiple machines to distribute the load. The machines pool the results into a central database.

We ran the crawler on a daily basis over a period of 47 days beginning on January 1th 2014 to February 16th 2014. The crawler cycles over every one of the 12,000 apps in the span of 24 hours. In the first crawler run, the crawler downloads the static information,

Table 3.1: Number of versions per apps as of Feb 16, 2014

Total apps crawled	12,000
Inaccessible	1,287
Total accessible apps	10,713
Not updated during the study period	7,044 (66%)
Updated at least once	3,669 (34%)
Updated bi-weekly or more frequently	1448 (14%)
Updated weekly or more frequently	98 (0.9%)

such as the app title, author, and number of downloads. All the downloaded data is stored locally in a database. For each app, the crawler downloads the latest user reviews and app permissions i.e., list of contacts, ability to send notifications. Then, the crawler checks if there is a new release, if not, it moves to the next app. If there is a new release (i.e., the current release date is greater than the release date stored in the database) then the APK file of the new release is downloaded and the relevant information of the new release is downloaded and stored in the database. The information for a new release includes the size of the app, version number, release date, description of the app, installation size, description of recent changes to the app, and the current star ratings (one through five) given to the app.

11% (1,287) of the 12,000 apps could not be accessed by the crawler and so the crawler failed to extract the information on these apps. These apps no longer exist in the store and return a “Not Found” page result if one attempts to view the app’s page. These apps ceased to exist between the time we selected the app list and the period when we began to crawl the app store. We tried to access the store page of these apps but they were either not available, suggesting that the app was removed from the app store. In total, 10,713 apps were crawled regularly as Table 3.1 shows.

Data Processing

After collecting the data, we further process the data to calculate changes between successive releases.

We observe in our data that the release date will sometimes change but the version number and APK file remains the same. We therefore, use the version number to track unique releases for each app in our analysis.

The reason for this irregularity is due to a feature in the Google Play Store which allows developers to associate multiple APK files for different models of mobile devices. Developers are able to manage their apps' configuration and distribution to target specific users and phones. A developer can specify the features, e.g., OpenGL support, screen sizes that must be present on a mobile device etc., for a mobile device to be compatible. The Google Play Store will then intelligently select an APK file that best suits the user's mobile device. Therefore, the release date is somewhat misleading as it coincides with any new release by the developer and not the users mobile device's version.

In the following sub-sections, we present the results of our empirical study. We begin by answering RQ1 in order to ascertain the prevalence of frequently-updated apps.

3.4 Empirical Study Results

This section presents and discusses the results of our three research questions. For each research question, we present our motivation to study the question, the approach that we used to answer the question, and the results of our analysis.

3.4.1 RQ1: How frequent and consistent are apps updated?

Motivation

We investigate the release pattern for apps along two dimensions: frequency and consistency. Specifically, we ask what is the frequency of new releases and are the release patterns

of apps consistent? The frequency of the releases informs us how fresh the apps are and the consistency informs us how the developers organize their release pattern i.e., whether their patterns are sporadic ad-hoc, or planned. Developers are then able to match the practises of the app market and in doing so, meet the expectations of users.

Approach

We calculate the frequency of new releases by summing the number of crawled new versions of each app. We then separate the app into three groups based on their release frequency: less than bi-weekly, bi-weekly to weekly and weekly or more frequently. To further investigate the release frequency, we examine the categories that the apps belong to in the Google Play Store (e.g., Arcade, or Social), to study the frequency of releases across the various categories of the market.

To investigate the consistency of releases, we use the normalized Shannon’s entropy measure (Shannon, 2001) on the release periods between versions of an app. To calculate the release period of version i , we subtract the release date of version i from version $i - 1$. The normalized Shannon’s entropy equation is:

$$normH(X) = - \sum p(x_i) \log_b p(x_i) / \log_b(N)$$

where X is a list of release periods for an app, x_i is the probability of a given release period, \log_b denotes logarithm of base 2 and N is the number of unique release periods. The higher the Shannon entropy the more information gained, i.e., the more irregular the release periods are. Shannon entropy of 0 means the median release periods are all the same i.e., very regular release periods (Hassan, 2009).

In order to account for slight differences in release periods we use a release window of

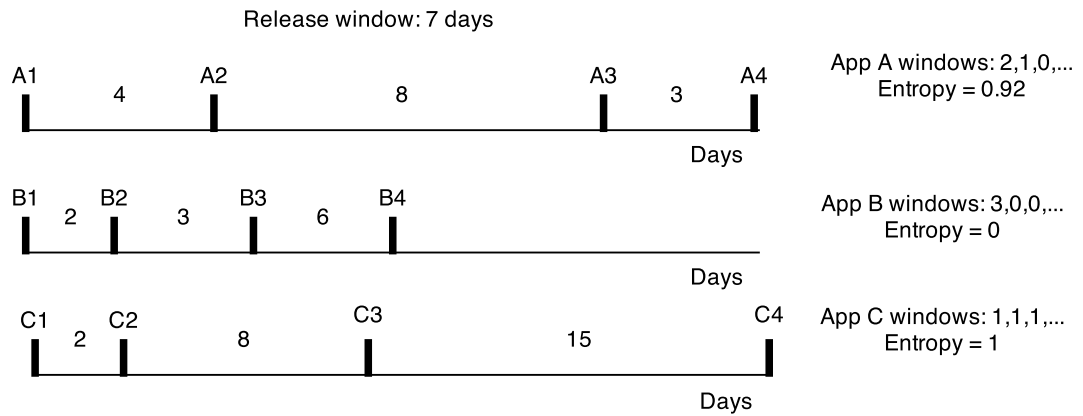
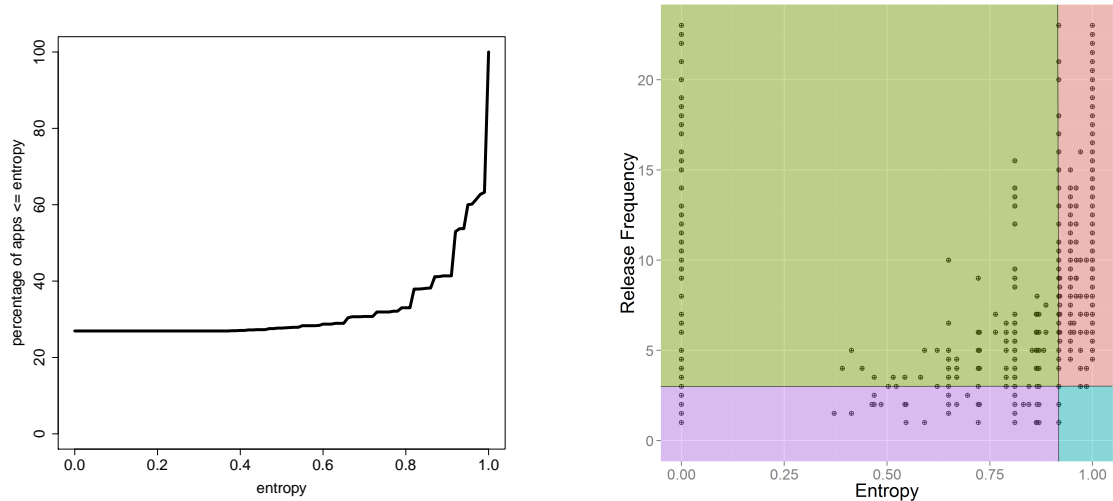


Figure 3.1: Three examples of calculating release period entropy. The release window is 7 days. For App A, 2 release periods are grouped together and 1 release period remains distinct. For App B, 3 release periods are grouped together. For App C, all release periods are distinct.

one week e.g., release periods are considered the same if they are within one week of each other.

Figure 3.1 demonstrates three examples. App A's release periods are 4, 8 and 3. The first release window contains 2 release periods, the second release window contains 1 release period and the subsequent release windows are empty. There exists a mix of different release periods and so the entropy is 0.92. App B's release periods are 2, 3 and 6. All the releases fall within the same release window, hence the entropy is 0. Finally, App C's release periods are 2, 8 and 15. Each release period falls inside a different release window, therefore the entropy is 1.

We continue our investigation of the release consistency by examining the consistency of days of the week. We also calculate the mode day of the week for each app.



(a) Cumulative distribution of the percentage of apps less than or equal to the release period entropy

(b) App entropy and release frequency divided by the median entropy on the x-axis and the median release frequency on the y-axis

Figure 3.2: Analysis of release period consistency

Results

~1% of the apps are released at least once a week. Approximately 1% of the studied apps release weekly or more frequently as Table 3.1 shows. The frequency of releases are quite high considering that these apps are at least one year old. A sizable group of ~14% release bi-weekly or more frequently as Figure 3.3 shows. 34% of apps released at least one version during our study version, while the remaining apps (66%) did not release a new version. The aforementioned percentages are calculated based on the successfully crawled apps.

We further investigate the release frequency across the different categories. We calculate the number of times each category occurs in the bi-weekly or more frequently-releasing apps. We divide the amount by the total number of crawled apps in that category. The most frequent Google play categories are ‘social’ followed by ‘productivity’ as Figure 3.4 shows.

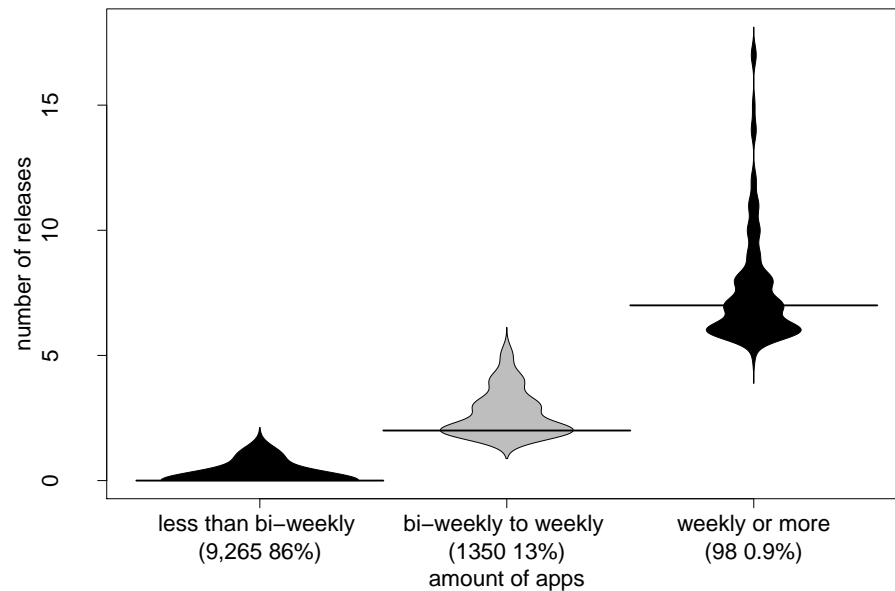


Figure 3.3: Apps divided into three groups: releasing less than bi-weekly, releasing bi-weekly to weekly and releasing weekly or more frequently (based on crawling for over a six week period).

Social apps have large user bases and complex social interactions that probably necessitate constant maintenance. Some categories, e.g., medical apps, have a much slower release frequency.

The release pattern for the majority of apps is not consistent. We find that most apps do not share the same release periods instead they have several different release periods as Figure 3.2a demonstrates. The cumulative distribution shows that the data is skewed towards a high release entropy. In other words, most apps have entropy which translates to an inconsistent release pattern. However, there is a portion with lower release entropy, i.e., higher consistency. 20% of the apps had a release entropy of 0.

We investigate this subset of apps further by examining if the release frequency plays a role in the consistency. We plot the release period entropy against the release frequency count. To avoid skewed results, we select only apps which released at least one app on a

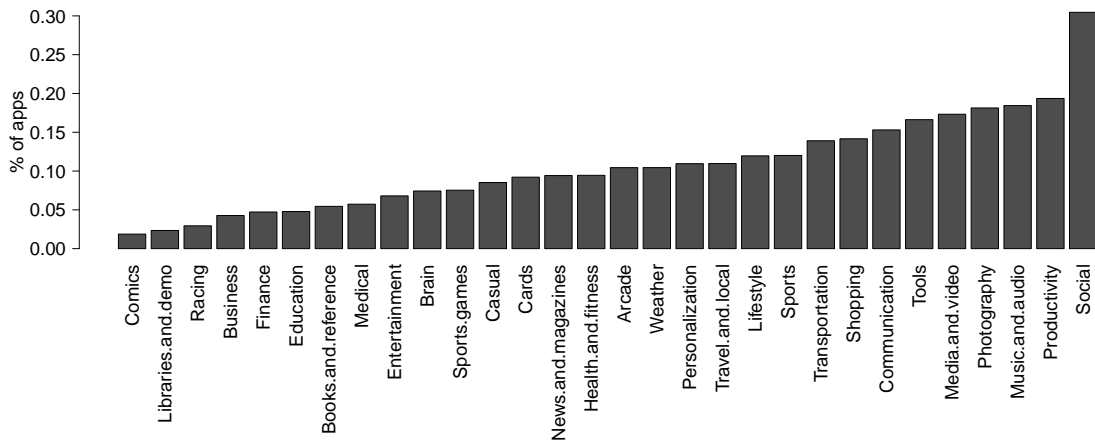


Figure 3.4: Percentage of studied apps in each category who release bi-weekly or more frequently

bi-weekly basis or more frequently. We divide the graph into four regions based on the median of the entropy and release frequency counts. (i) low entropy, low frequency, (ii) low entropy, high frequency, (iii) high entropy, low frequency and (iv) high entropy, high frequency.

There are 345 low entropy, low frequency apps; 422 low entropy, high frequency apps; 397 high entropy, low frequency apps; and 284 high entropy and high frequency apps as Figure 3.2b shows. In the low entropy, high frequency region there are apps such as “3D Cruz Azul Fondo Animado” that had three consecutive days of new releases. None of the releases of “3D Cruz Azul Fondo Animado” described what occurred in the “What’s New” section. An app from the high entropy, high frequency “Pou” would alternate between short releases and long releases. In the beginning of January, the app released several days apart but in the next half of January the app took 10 days and 11 days to release two new versions respectively. The release pattern of “Pou” is an example of an inconsistent and bursty release periods. We further show inconsistent release patterns with our release day analysis.

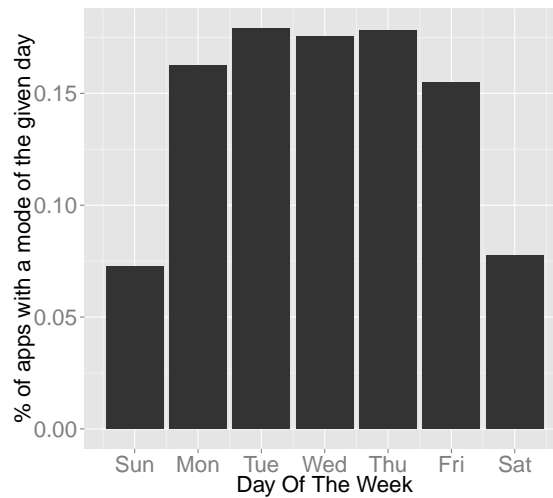


Figure 3.5: Mode day of the week for each app’s release date

Releases do not occur on consistent days of the week. By observing the mode day of the week for release dates per app, we find that most apps do not often release on weekends as Figure 3.5 shows. The rest of the week is spread evenly. This demonstrates that there is no favored day of the week for app developers to release a new version.

~1% of apps release at least one update per week. We also note that there is no consistent release pattern – no consistency on which day of the week or release period.

3.4.2 RQ2: What is the rationale for such frequent releases?

Motivation

Now that we have established that there is a small subset of apps that are released at a very frequent rate, we wish to investigate the rationale behind such frequent releases. However, the frequent releases do not reveal the rationale of such releases. Our goal is to understand what types of activities (referred to as tasks that developers perform), e.g., new features,

improvements, bug fixes, new content, or new permissions. With this knowledge, developers can better understand the tasks performed on new releases in the marketplace, just as developers access knowledge on the revenue and popularity of apps in the marketplace through app analytics tools (Annie, 2014; Distimo, 2013; Flurry, 2014).

Approach

We analyze the text in the publicly available “What’s New” section (see Figure 2.2 for an example). This optional section is authored by the developer and denotes what has been changed in the new release. We choose to analyze the “What’s New” sections for each new release of the top 100 most frequently-releasing apps. Our choice is motivated by our focus on frequently-releasing apps.

We manually label each of the “What’s New” sections. In total, we manually labelled 852 different releases and their corresponding “What’s New” sections for the top 100 most frequently-releasing apps. We discover four common tasks that are performed in the new releases. The four tasks are (i) adding new content, (ii) fixing bugs, (iii) new features, (iv) implementing improvements and (v) adjusting permissions. We record the amount of the same task that occurs in the text, wherever possible. For example if the “What’s New” section says “added a new like button in the menu and added the ability to scroll” then we record two new features for that release. We do not count how many improvements or bug fixes occur in the same “What’s New” section because developers often state an unknown amount of bug-fixes or improvements i.e., “bug fixes and some improvements”.

From our manual labelling, we calculate the co-occurrence of the tasks with one another. This information gives us an idea about the amount of coupling between the different tasks.

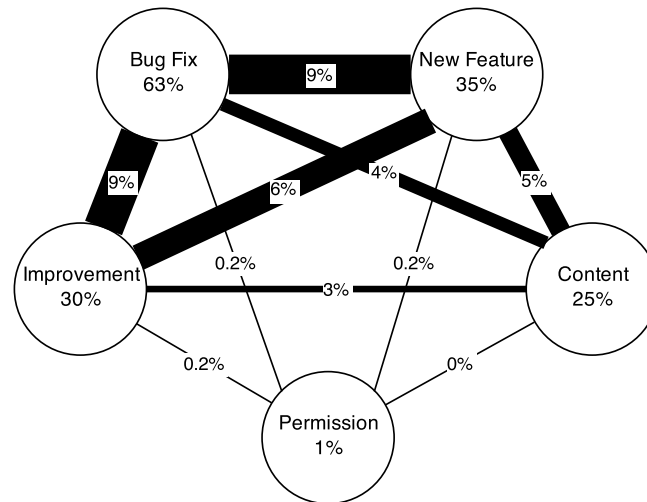


Figure 3.6: Amount of coupling of tasks in the “What’s New” section. An edge between two tasks means that they co-occur. The number above the edge is the percentage of releases where both of these tasks co-occur. The number in a task circle indicates the percentage of releases where such a task occurred in the “What’s New” section.

To complement our manual analysis, we perform an automated analysis of the “What’s New” section. We calculate the size of the text and the change in size between successive releases. The automated analysis is done to study the size of the developer communication, and whether developers simply append to or update the “What’s New” text or replace it in its entirety.

Results

Almost half (45%) of the new releases are silent, i.e., do not contain any information as to what has changed. Silent releases are releases where the “What’s New” section is either left empty or is identical to the “What’s New” section from the previous release. We designate the apps that only release silent releases as “silent apps”. In fact, the most frequently-releasing app, “3D Cruz Azul Fondo Animado” is completely silent. Further,

there are semi-silent apps that contain some silent releases and some non-silent releases.

We find that non-silent releases are often followed by silent releases. For example, an app may release a big new feature then subsequently release silent bug fixes. An example of a semi-silent app is “InstaWeather” which communicated that the developer has completely re-wrote their app in native Java in version 300300002 then there was no text in the “What’s New” section in the following version i.e., 300300003. Presumably, the changes were minor fixes to their previous major revision. We observe a mixture of complete, semi, and non-silent apps. Approximately 20% of the apps have no silent releases. On the other end of the spectrum there are approximately 20% of the apps who are completely silent.

Silent releases leave users completely in the dark as to what has changed and users may not download the new release especially if they are bandwidth-limited. Further, two developers actually apologized in the “What’s New” section for frequent releases, evidently the frequent updates upset their users because they were asked to update their app so frequently. A weather app “InstaWeather” in version 64 had added several small features in each release along with bug fixes and apologized for the many releases that week. The bandwidth cost should be a consideration in the decision to release a new version, or to wait and bundle several completed tasks together. We next discuss the tasks mentioned in the “What’s New” section of the non-silent app releases.

The most frequently communicated task is bug fixing which occurs in 63% of the non-silent releases. As Table 3.2 shows, the rest of the non-silent releases contain 35% new features, 30% improvements, 25% new content and very few permission changes (1%).

The style that developers use to communicate tasks varies. For some bugs and improvements, developers do not explicitly state the amount of bugs or improvements. Instead they

simply say things such as “fixed bugs” or “fixed crashes”. However, in other cases, developers might communicate specific bug fixes or specific improvements. For example, version 44 of the fitness app “Sports Tracker”, the developer states explicitly “fix losing edited workout details”.

However, new features, new content and new permissions are always stated explicitly if mentioned in the text (this of course does not include permissions or features added silently). New features and content are tasks that developers want to advertise to the users so it is understandable that they would feature them prominently in the “What’s New” section.

New permissions were mentioned in only three apps. The first app is a tool to manage invoices and manage your expenses called “Zoho Invoice and Time Tracking”. In version 33, “Zoho Invoice and Time Tracking” required the ability to read the user’s phone state to know when a customer of the user calls. The second app was a communication app called “Open Garden” that wanted permission in version 259 for sending text messages and reading users’ contact list. In each case “Open Garden” mentioned explicitly the reason for adding a new permission. The third app “Ultimate custom widget” had accidentally removed a permission and added the permission back in the subsequent release.

We further investigate if the frequently-releasing apps are adding permissions silently. We collected the permission data of each new version as part of our crawling. For each of the frequent releasing apps we determine if a new version has added or removed permissions.

Permissions are changed silently. We observe that 31% of the apps are silently changing permissions at least once. This contrasts with only 3 apps which reported permission

changes in their “What’s New” section. The changes to permissions are split evenly between additional permissions and removal of permissions with 46% added and 54% removed in total. An example of an app which added permissions silently is the app “Customizable Countdown Widget” which added the ability to read and write to external storage. Their “What’s New” section said: “3.1.2 Fixed a few issues with uploading images. It should be fixed. Email me if it is not”. The absence of notice about a change in permissions shows a clear lack of transparency in frequently-releasing apps. Users still must agree to permission changes when they’ve downloaded a new update for their app. Hence, permission changes are not completely hidden from the users. Yet, the developers choose not to present a reason for the permission changes which may trouble some users.

We now examine the co-occurrence of tasks in a release.

Tasks do not occur in isolation. Tasks often co-occur together and new features and new content occur more than once in the same “What’s New” section. 47% of the studied “What’s New” sections contained more than one task. Figure 3.6 shows that bug fixes, new features and improvements co-occur in about 10% of tasks. The mix of tasks suggests that developers are in a continuous flux of development and maintenance. If we consider how often a task occurs in the “What’s New” section, we find that new features have the highest occurrence in the same “What’s New” section. The highest occurrence of new features was 7 in the “Solo Launcher for Your Theme” social app — in that particular release, the developer redesigned the app. Finally, we end the discussion of the results of RQ2 by presenting the results of our automated analysis.

Developers follow two styles in the “What’s New” section. We observe that developers tend to organize their “What’s New” section along two styles. The first style is to replace the text in the “What’s New” section completely across releases. The replaced text can

Table 3.2: Types of tasks, the frequency of non-silent releases with a given task

Task	Description	Frequency
New content	New content is the addition of information separate from the core features of an app. For example, a new video game character, or chat icons for users, or a new map for a GPS system	114 (25%)
New feature	New features encompass any new addition to the core functionality of an app.	158 (35%)
Improvement	Improvements are tweaks, performance, memory improvements or small improvement to existing features.	136 (30%)
Bug fix	Reported fixes of bugs, defects, crashing or unexpected events.	284 (63%)
Permission	A change in permissions reported by the developer	5 (1%)

either be larger or shorter depending on the content of the next release. The second style maintains the content from previous “What’s New” sections then appends new information relevant to the current release. The median change is quite small because in both cases the text is not increasing or decreasing sizeably. In total, the text size is relatively short (300-400 characters long).

45% of frequently-releasing apps do not inform the users as to what has changed (including 28% of apps that modified their permissions silently). When they do inform the user, the release is a mix of development and maintenance: 63% bug-fixes, 35% new features, 30% improvements and 25% new content.

In the following section we focus on what actually changes in the code as opposed to what is reported by developers.

3.4.3 RQ3: What is actually changing for such frequent releases?

Motivation

RQ2 shows that non-silent frequently-releasing apps inform the users that they are continually developing and maintaining their apps. However, we wish to complement our analysis of the developer's communications by analyzing the changes in the APK files. We perform this analysis for two reasons. The first reason is we do not know what is actually changing inside the APK files or by how much regardless of what the developer communicates. Secondly, as we have seen, there are apps that do not communicate any information regarding the rationale of a new release. We investigate the amount of change of the top 100 frequently-releasing silent and non-silent apps.

Approach

To investigate how apps are changing over multiple releases, what is occurring inside the apps and the differences between silent and non-silent apps, we analyze the binaries of the apps (i.e., the APK files). We select the contents of the APK files for each of the top 100 most frequently-releasing apps whose "What's New" sections we had manually labelled in RQ2. For each app and its releases, we extract the associated APK files. In total, we analyze 852 releases. The total number of silent apps are 17 and the total number of non-silent apps is 83.

We use a tool called APK Tool (Brut.alll and Tumbleson, 2014) to extract the content of the APK files. The output of the tool are the resources, assets and files used by the app. The java code is not available as the tool does not decompile the source code. The java code remains in files ending with the extension .smali which is the Dalvik bytecode language (Google). Dalvik is a virtual machine used to run the apps.

Table 3.3: The file type groupings, their descriptions and their extensions

File Type Group	Extensions
Code	smali,js,css,lua php
Markup languages	xml, html, json, yml, grxml
Native code	.so
other	none, ccz, en, machinex, 30287,clientx 35536, 35536, cafe, g2g, svntmp, pkm, bks, LICENSE, pkcs12, atlas, acv, coords, armeabi, bc, 0, 3, 7, 8
images	png, jpg, gif, svg, tbn, xcf, raw, jpeg, ico, psd, PNG
font	ttf, otf, fnt, ttc, TTF
config	properties, plist, prop, config, ini
archive	zip, jar, apk
audio	wav, aac, smf, mp3, m4a, mp4
binary	ccbi, bin
multimedia	ogg, swf
graphics	vsh, fsh
data	dat, assets, csv
db	db, sql

Once the files of the app are extracted, we perform a time series analysis on the contents of the APK files. We are interested in observing the changes between APK size and the frequency of churn of the various file types e.g., .smali, .xml. To simplify the presentation of our analysis, we group similar file names together e.g., .smali, .xml, .js are grouped as “code” and .png, .jpeg .gif are grouped as “images”. Table 3.3 presents the file types and the groups we selected.

For each file type group in every release of every studied app, we calculate the size of the release, the percentage of code churn and the total change in size of the app’s APK file over our data collection period. Once we calculate the amount of change for each file type in each release of an app, we calculate the median change across every release of that app to determine how each file type changes over time.

Results

Silent apps grow as much as non-silent apps. During the study period, the median growth rate of non-silent versus silent apps is very close at 6.6% versus 4% as Figure 3.7 shows.

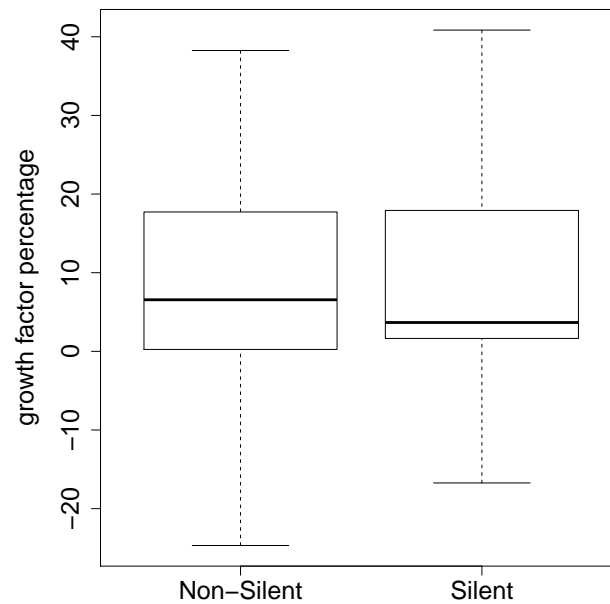


Figure 3.7: The total APK file size growth of frequently-releasing silent and non-silent apps over the study period

This similar growth rate demonstrates that similar changes are occurring in frequently-releasing silent apps as they are in frequently-releasing non-silent apps.

Negative changes do occur. The APK file changes for non-silent apps is above zero, however 25% of changes were negative as Figure 3.7 demonstrates. We see the same percentage of negative changes among the non-silent apps as well. However, if we only consider the APK file changes we do not gain insight into which types of files are changing. Therefore we explore below the file types that are changing the most.

Source code changes the most. Figure 3.8 presents the percentage of change of the different file types. It is clear that the source code is where most of the churn occurs. The only other file group with a median percentage churn greater than 0 is markup languages such as XML. One use of XML in Android apps is to represent the UI layout of apps.

Frequently-releasing apps are growing moderately even though they are relatively

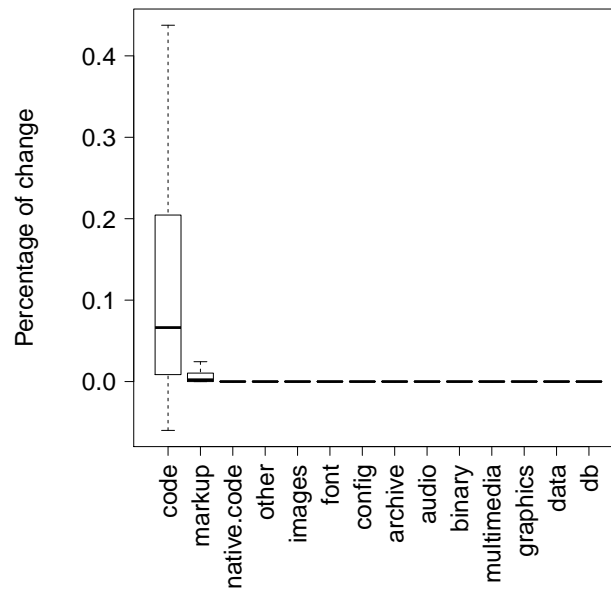


Figure 3.8: The median change in file type groups of the frequently-releasing apps

mature apps. We find that the apps undergo a moderately large median increase of 6% in size for our entire study period. Figure 3.7 shows the growth factor of apps over the period of our study. The growth factor is the relative amount of growth from the start of our study till the end. The apps continue to grow even after at least one year of existence on the store.

Frequently-releasing apps continuously undergo code churn and have moderate growth for their APK file. These changes occur long after the initial release date. Silent apps undergo similar changes.

3.5 Discussion

We discuss the results of our research questions in this section.

3.5.1 The impact of release frequency on “app ratings”

We divide our entire dataset of apps into two categories using the median release frequency: frequently-releasing and infrequently-releasing. We compare the quality of infrequently vs. frequently-releasing apps. Our measure of quality is the percentage of “negative ratings”. We chose the percentage of negative ratings because most apps in our dataset have an overall four star rating. The percentage of negative ratings allows us to see the amount of negative sentiment associated with the apps (Maas et al., 2011; Pang and Lee, 2004). We calculate the percentage of “negative ratings” (1 or 2 star ratings) for each app. The percentage of negative ratings is simply the ratio of one-star and two-star ratings over the total amount of ratings.

We only use the negative ratings given to apps during the time of our study. The Google Play Store only keeps a single global rating of the app over its entire lifespan (this includes ratings given many years ago). Therefore, we only calculate the negative ratings by subtracting time snapshots of the global rating from previous versions to arrive at a rating delta. This rating delta is the amount of new ratings. We calculate the delta for every release and the median delta per app in which we are interested.

Frequently-releasing apps have a lower percentage of negative ratings. As Figure 3.9 demonstrates, there are more negative ratings for infrequently-releasing apps. The results are statistically significant with a Mann-Whitney test p-value < 0.05 and an effect size of 22 (small effect) Kampenes et al. (2007). Additionally, we control for the size of apps since larger apps may take longer to release. We divide each frequent and infrequent app groups into two by the median size of the app’s APK file. Figure 3.10 demonstrates that our observation still holds – frequently-releasing large/small apps have lower percentages

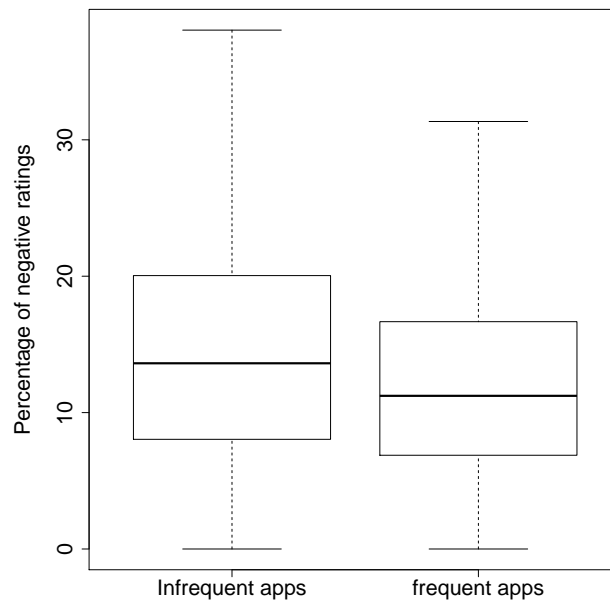


Figure 3.9: Percentage of negative star ratings for infrequent and frequent releasing apps.

of negative ratings than their respective infrequently-releasing large/small apps. The difference in ratings between frequent small and infrequent small apps as well as the difference in ratings between frequent large and infrequent large apps is statistically significant with a Mann-Whitney test p -value < 0.05 . The comparison is conducted on popular apps drawn from Distimo's most popular apps. Therefore, our results may not generalize over less popular apps. Additionally, even a small difference in rating can affect the visibility of an app on the marketplace. As previous research has shown, the ranking of apps in app stores are influenced by the rating of apps (Lim and Bentley, 2013).

3.5.2 The impact of release frequency on app popularity

We further investigate how popular the 100 most frequently-releasing apps are after one year has passed since the apps were in the list of most popular apps in Distimo. The

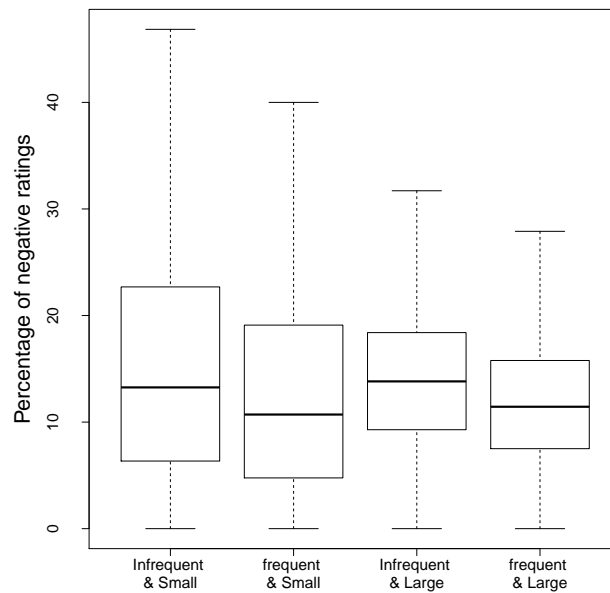


Figure 3.10: Percentage of negative star ratings for infrequent large, infrequent small, frequent large and frequent small apps

popularity is a judgement of success that includes overall downloads and ratings according to Distimo. We compare the 100 most frequently-releasing apps against all other apps in our dataset. We verify if an app is still in the most popular apps of its category in Distimo for January 2014 (a total of approximately 10,000 apps). In the case of the 100 most frequently-releasing apps, we manually verify that each app still appears in Distimo's popularity list. For the rest of the apps, we perform automated verification. We crawl Distimo's rankings and compare the list of apps that we collected in the Spring of 2013 with all the popularity lists of Distimo. As we crawl, there may be some false negatives because apps may have changed their names since our original crawl in Spring 2013. However, manual verification of approximately 10,000 apps would be prohibitively time-consuming, therefore we use the infrequent apps popularity as an estimate only.

Frequently-releasing apps remain popular. We find that 81% of apps in the frequently-releasing apps are still found in the list of most popular apps of their category, whereas only 45% of the rest of the apps are in the list of the most popular apps of their category.

3.6 Threats to Validity

Some threats could potentially limit the validity of our results. We now discuss such threats and how we control or mitigate them.

3.6.1 Threats to Construct Validity

I manually labelled the maintenance tasks mentioned in “What’s New” section. The labelling may have produced some false labels. In order to mitigate this threat, a post-doctoral fellow verified the manually-labelled tasks. If there were disagreements between labels, both raters came to a consensus. The raters could not come to a consensus 2% of the time – in such cases a professor was used to break ties.

Our study investigates various characteristics and release patterns of frequently-releasing apps. However, to truly understand the rationale behind the release pattern, we need to conduct developer interviews. Interviews will be part of future studies.

3.6.2 Threats to Internal Validity

The selection of the top 100 most frequently-releasing apps may bias any conclusion that we may draw on apps in general. We therefore only draw conclusions on frequently-releasing apps.

The APK tool that we used may not be perfect in disassembling all the APK files. We

mitigate this threat by only focusing on the file extensions and not the reverse engineered code in the files.

The relationship between release frequency and the ratings and popularity of an app may be influenced by other factors. We mitigate this threat by not implying causation but merely that there is a correlation in their relationships.

3.6.3 Threats to Conclusion Validity

The selection of statistical test could bias our results. We mitigate this threat by using non-parametric tests because our data has a non-normal distribution.

3.7 Conclusion

The frequent release patterns of some popular apps demonstrates a need to adapt to frequent releases on mobile marketplaces. In fact, we already see this adaptation occurring in the Google Play Store. App stores mechanisms, for example the Google Play Store's distribution mechanism, can update apps automatically without requiring user confirmation. Users benefit from no longer dealing with the hassle of manually downloading a new version. Additionally, Google Play Store has an option to only update a user's apps when the user is connected to Wi-fi. Such an option prevents users from depleting their potentially limited bandwidth on their cellphone plans. Both options allow users to more effectively handle continual new releases of their apps. Developers are also adapting as they sometimes do not even bother to notify the user of what has changed between releases and a release may only be a bug fix or several improvements. These factors contribute to a mentality in developers, app stores and users that a new release is not nearly as significant as they traditionally were

considered.

One downside is the decreased time for quality assurance. As we have shown, the “What’s New” section of new releases contains 63% of some form of bug fix. Releasing a working version then fixing bugs in subsequent releases is more important for developers than taking longer to release a more stable version. Many of the new releases have a dual role as a patch and as an improved version of their prior release.

Finally, what is interesting to note is the amount of upkeep that is being placed in maintaining apps for free whereas traditional software may have charged a fee. The economies of mobile app stores have improved the viability of free apps as app revenue can be generated from ads.

The various points demonstrate that the stakeholders in the mobile marketplace are already adjusting to the frequent-releasing reality.

The following chapter forms the second part of our empirical study. This chapter demonstrated that due to the distribution mechanism, apps are able to release more frequently and generate more reviews after a release. The following chapter focuses on the unique feedback mechanism of app stores and the consequences of the feedback mechanism for the various stakeholders of app stores (developers, users and app store owners).

Chapter 4

Review Overload: An Empirical study of the Influx of User Reviews for Mobile Apps

The feedback mechanism of mobile app stores is a useful source of feedback. The feedback mechanism allows users can write reviews of apps in the app store. However, highly popular apps receive thousands of reviews per day which makes it difficult for developers to read them all. Previous work has proposed methods to help developers cope with the overwhelming amount of reviews (e.g., by summarizing user reviews). However, to the best of our knowledge, there are no studies that demonstrate what percentage of the apps in the market is overwhelmed with reviews. By analyzing reviews from 10,713 top apps from the Google Play Store, this chapter highlights that a very small number (0.19%) of the studied apps receive more than 500 reviews per day while most (88%) of the studied apps receive less than 20 reviews per day. We also show that similar issues emerge across reviews. We also observe that developers of frequently-reviewed apps never response to reviews, yet

we show that there are positive effects to responding to reviews (users change their rating 38.7% of the time following a developer response) and the median rating sees an increase of 20%.

4.1 Introduction

As we have shown in the previous chapter the distribution mechanism available to developers allows new versions of apps to be released quickly prompting a burst of reviews. Next, we observe the feedback mechanism and its consequences for developers.

App stores provide feedback mechanisms for developers by allowing a user to rate an app using a 5 star rating system and to write a review. Addressing feedback is an important part of developing and maintaining popularity in app stores. However, a developer may not have time to read all the received reviews due to the overwhelming number of received reviews. Several papers have proposed solutions designed to help developers cope with the large number of overwhelming reviews (Chen et al., 2014; Fu et al., 2013; Galvis Carreño and Winbladh, 2013). The papers develop methods to visualize, summarize and rank reviews. The daily review counts for several large apps such as Facebook have often been used to motivate this line of research (Chen et al., 2014; Galvis Carreño and Winbladh, 2013). However, to the best of our knowledge, the percentage of apps in an app store that are overwhelmed with reviews is unknown. This chapter answers the question of how pervasive are the frequently-reviewed apps in the market.

Developers of infrequently-reviewed apps could easily read all the received reviews per day, yet, frequently-reviewed apps are overwhelmed with reviews. We investigate how varied the issues are in overwhelmed apps. Developers may find that certain issues dominate in occurrence over others. We investigate how varied the review issues are for the

overwhelmed apps to see how the influx of reviews varies over time.

Developers can address user feedback by publishing a new release or responding to a review. By responding to a review, a developer can address the concerns of the user at a more personal level. However, a response can be time consuming and knowing when a response is likely to have a positive impact on the user's rating of an app is important for managing the limited resources of a developer.

The goal of this chapter is to empirically investigate app reviews from the perspective of developers who must read and address such reviews on a daily basis. Through an analysis of 10,713 apps from the Google Play Store over a period of two months, we explore the following research questions:

RQ1: How many reviews does an app receive on a daily basis?

0.19% of the studied apps receive over 500 reviews daily. The majority of the studied apps (88%) receive a small number (20 or less reviews) daily. The number of downloads and releases impact the number of received reviews (the category of the app does not play a major role).

RQ2: Do similar issues emerge in the reviews of frequently-reviewed apps?

Users share similar issues across the market. However, the issues are not distributed equally among apps and issues vary for each app as new reviews are received.

RQ3: Is there value in responding to reviews?

13.8% of the studied apps replied to at least one review during the studied time period. The most-reviewed apps never replied to a review during our study period. Users change their rating 38.7% of the time following a developer response. The median rating change is a one-star increase out of five.

To the best of our knowledge, this chapter is the first study to investigate the amount of reviews that occur per day over a large segment of the Google Play Store and to show how similar issues emerge across apps. Finally, this study empirically shows the usefulness of responding to reviews.

This chapter is organized as follows: Section 4.2 presents the design of our study. Section 4.3 provides the results of our empirical study. Section 4.4 discusses our findings. Section 4.5 discusses the threats to the validity of our study. Section 4.6 concludes our work.

4.2 Empirical Study Design

In this section, we present the design of our study and the data collection and processing methods used in our study.

Study Goals

The *goals* of our empirical study are to analyze the frequency of reviews, the variation of the issues raised in reviews and the benefits of responding to reviews.

The *focus* of our empirical study is the amount of reviews, the textual information in reviews and the ratings of reviews. Our *perspective* is that of app developers (i.e., stakeholders) and researchers interested in mobile app reviews. The *objects* of our case study are the different reviews and their responses in the Google Play Store. In this empirical study, we address the three RQs from Section 4.1.

4.2.1 Data Selection

We select the same data as described in Chapter 3 in Section 3.3.1. We chose apps that were popular one year ago because we are interested in studying stable mature apps that had not been released recently to avoid the expected frequent burst of reviews following the early releases of an app (Pagano and Maalej, 2013).

4.2.2 Data Collection

Our data collection process follows the same process described in Chapter 3 in Section 3.3.1 with a few differences specific to how we collect reviews and length of time during which the app store was crawled. We detail such differences next.

We run the crawler on a daily basis over a period of approximately two months beginning on January 1st 2014 to March 2nd 2014. 11,047 different releases of apps were collected in the studied time period. An app can have multiple releases. In our dataset, on average, an app has 0.86 releases, i.e., most apps did not publish a new release during our studied release period. 4,073 apps published at least one release. Apps that published at least one new release had an average of 2.28 releases.

A limitation of the Google Play Store is that only the 500 latest reviews are accessible. The crawler is unable to access any earlier reviews. That means if more than 500 reviews occur within the 24 hour period between runs of our crawler, then the crawler will not collect those reviews. The limitation means we have a conservative estimate of the number of reviews for apps that receive more than 500 reviews per 24 hours.

In the following subsections, we present the results of our empirical study. We begin by answering RQ1 to ascertain the prevalence of frequently-reviewed apps.

4.3 Empirical Study Results

This section presents and discusses the results of our three research questions. For each research question, we present our motivation to study the question, the approach used to answer the question, and the results of our analysis.

4.3.1 RQ1: How many reviews does an app receive on a daily basis?

Motivation

App stores provide a unique platform that differs from traditional software. One of the unique aspects of app stores is the convenience of providing feedback. Users are able to quickly leave a review and a rating on the store page of an app (app stores also allow users to rate and review an app within the app). The speed and convenience of rating and reviewing an app allows rapid feedback to reach the developers. In turn, the rapid feedback assists developers in creating new iterations of the app. This lightweight feedback mechanism contrasts with traditional software feedback mechanisms such as bug reporting systems e.g., Bugzilla. Bug reporting systems are negative in nature i.e., only bugs, unlike reviews which can be positive. Additionally, reviews can even serve as a means to gather additional requirements (Galvis Carreño and Winbladh, 2013).

When the number of reviews being written exceeds the ability of the developer to read the reviews, the developer loses out on feedback. To date, there have been a limited number of studies which investigated the number of received reviews by developers. The most relevant study is recent work by Pagano and Maalej on the Apple App Store, which shows that on average a free app receives 37 reviews per day (with paid apps receiving around 7 reviews per day) (Pagano and Maalej, 2013). We compare our results with the results

Table 4.1: Our observations on Google Play apps compared to Pagano and Maalej’s observations on Apple’s App Store

Item	Pagano & Maalej (Apple App Store)	Our contribution (Google Play Store)	Note
Amount of received reviews on a daily basis	Average of 22 reviews per day; 36.87 for free apps, 7.18 for paid apps	Average of 7 reviews per day for free apps	Compared to Pagano and Maalej, we found a much lower average number of received user reviews. Further analysis shows that user reviews are very skewed. The median number of received is zero. 88% of free apps receive 20 reviews or less.
Highly reviewed apps	Facebook received 4,275 reviews in one day	0.19% of apps receive over 500 reviews, top 100 most reviewed apps have 6,000 to 43,000 reviews in studied period.	Pagano and Maalej were the first to observe that some apps (in their case the Facebook app) might receive a large number of reviews per day. Our work is the first to explore this observation in depth. Our key finding is that while some apps might receive a large amount of reviews, most apps (99%) receive very few reviews). Hence most apps might not benefit much from approaches for automated analysis of user reviews.
Impact of categories	Individual categories have different amounts of daily reviews	No relation	We did not find a relation between an app’s category and of received user reviews once we control for the # of downloads of an app and its number of releases.
Spike in reviews after a release	Reviews decrease over time after a release	The standard deviation of received reviews deviates from the median directly after a release and returns back to normal afterwards	Both stores show evidence of spike in reviews after a release.
Average length of a review	61 characters	Average of 64 characters; median of 36	Both stores have similar average lengths of reviews. However, the median length of reviews demonstrates that the distribution of review lengths has a right skew.

of Pagano and Maalej in Table 4.1. Yet little is known about the characteristics of the Google Play Store and the characteristics of the received reviews (e.g., is the data normally distributed or is there a small number of apps with a substantial number of reviews on a daily basis). We investigate how prevalent frequently-reviewed apps are in the studied apps.

Approach

Calculate the amount of reviews per app: We first investigate the distribution of total reviews per app. We calculate the total amount of received reviews for each studied app throughout our study period and the total amount of reviews received per day. We focus

our study on free apps, since recent work shows that free apps receives five times as much reviews as paid apps (Pagano and Maalej, 2013).

Results

Most apps (88% of Play Store apps) receive very few reviews over our studied time period. We plot the reviews per day and the total received reviews with a beanplot. The beanplot combines a boxplot with a kernel density estimation function. Figure 4.1a shows that the median number of reviews per day is 0. We find that 20 or 0.19% of the studied apps received 500 or more reviews (as previously mentioned in Section 4.2.2 this is a conservative estimate). Whereas, 88% of the apps in our dataset have less than 20 reviews per day. Additionally, the median total number of reviews is 0 for our studied period of time. We find less average reviews per day than Pagano and Maalej. This could be due to several factors. The first factor is that we collect reviews from apps that have been released for at least one year whereas Pagano and Maalej collected apps that may have been released recently. The second factor is that our estimates are conservative and do not count reviews that occurred after the first 500 of the day. We separate the apps into the 100 most-reviewed apps and all other apps. Figure 4.1b shows, there is a large gap in total number of reviews for the 100 most-reviewed apps. The total number of reviews of the 100 most-reviewed apps range from 43,000 reviews to 6,000 in the studied period. The review themselves are short, only 64 characters on average. This result is similar to Apple App Store's 61 characters found by Pagano and Maalej.

Next, we investigate the impact that number of downloads of the app, number of releases and the app store category of the app have on the number of received reviews. We wish to determine if there is a correlation between these three factors and the number of

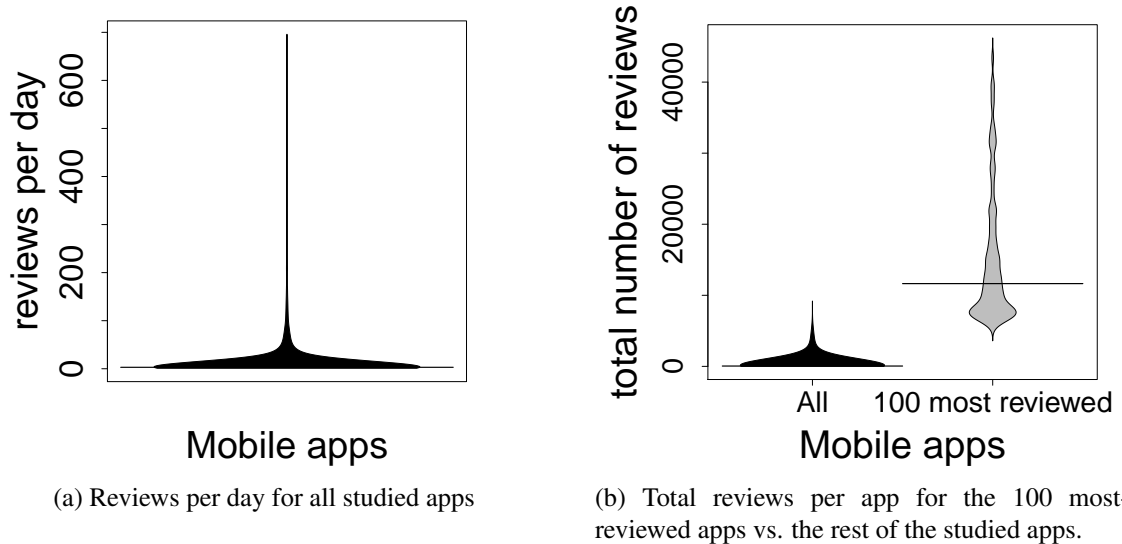


Figure 4.1: Beanplots showing amounts of reviews per day and in total.

reviews. In order to perform this step, we build a regression model with the number of received reviews of an app as the dependent variable.

Figure 4.2 plots the total number of reviews using the built regression model. We have three plots, for each plot we keep the median values of the other factors the same to see how each factor effects the total number of reviews. The grey bands around the plotted lines are the confidence intervals.

We generate a nomogram to visualize the results of our model (Harrell, 2001) where each factor is an independent variable. The nomogram permits us to examine the impact of each factor while controlling for the other factors. We find that as the number of downloads and the number of releases increase, the total number of reviews increases. There exists no relationship between individual categories e.g., communications, social, tools and review count (once we control for the number of downloads and releases). Pagano and Maalej found a relation between categories and the number of received reviews on the Apple App

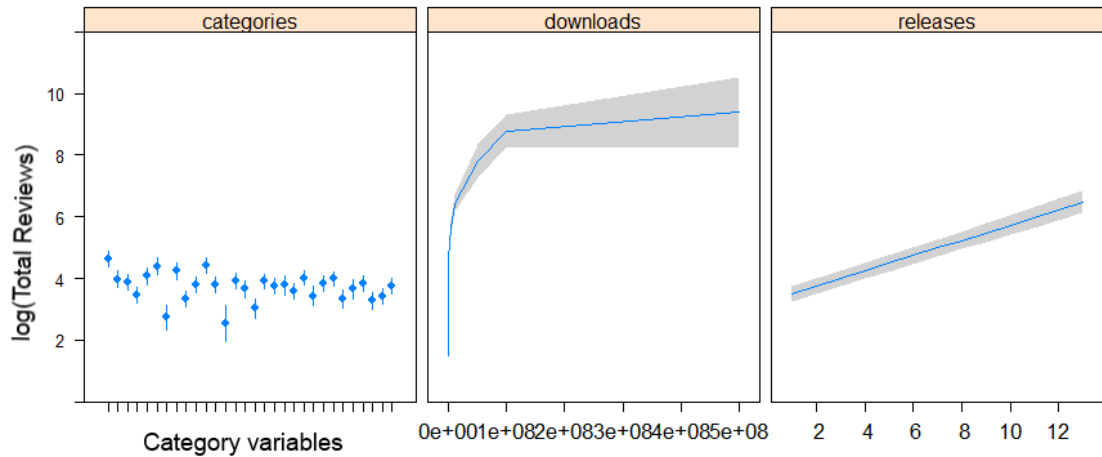


Figure 4.2: Data plot of the total reviews (logged) on the y-axis and three separate graphs of the app categories, the number of downloads and the number of releases on the x-axis. The graph demonstrates the correlation between the three factors and the total number of reviews.

Store.

We construct a nomogram in Figure 4.3 to further study the impact of the factors. A nomogram is used to graphically calculate the function of multiple factors, in this case it is fitted to our regression model. The nomogram is a series of scales. The Linear Predictor scale is the total number of reviews in log scale. To calculate the total number of reviews, one draws a straight line from the value of the “total points” scale to the Linear Predictor scale. The total points are calculated by summing the points that each of the three scale factors (releases, downloads and categories) have. To calculate the points value of each factor one would draw a line from the value in the factor scale to the points scale. The value in the points scale becomes the points for that factor. For example, releases = 2, downloads = 100,000 and categories = “tools”. We find that 2-releases corresponds to approximately 7 points, 100,000 downloads corresponds to approximately 20 points and the “tools” category corresponds approximately to 5 points. The sum is 32 total points

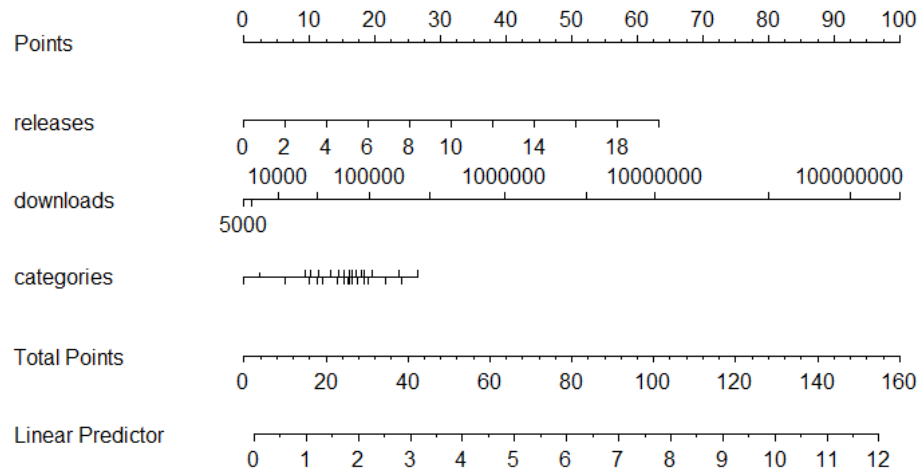


Figure 4.3: A nomograph of the impact of the new releases, app category and number of downloads on the total number of received reviews.

which corresponds to approximately 2.5 log scale i.e., (316) total user reviews.

We find that a high number of downloads corresponds to a high number of points. We also find that the impact of many releases (greater than 20) is as impactful as an app with 10,000,000 downloads. It is intuitive to find that the amount of downloads corresponds to a higher number of reviews as the more users that download an app, the more likely the app will receive additional reviews. We investigate why more releases correspond to a higher number of reviews.

We first note that Pagano and Maalej studied the number of reviews in paid and free apps occurring after a release and found the amount of reviews decreased over time after a release suggesting that releases contribute to new reviews (Pagano and Maalej, 2013). We find the same effects over the release periods of our studied apps by studying before and after an app release. For each app, we calculate the median number of reviews per day.

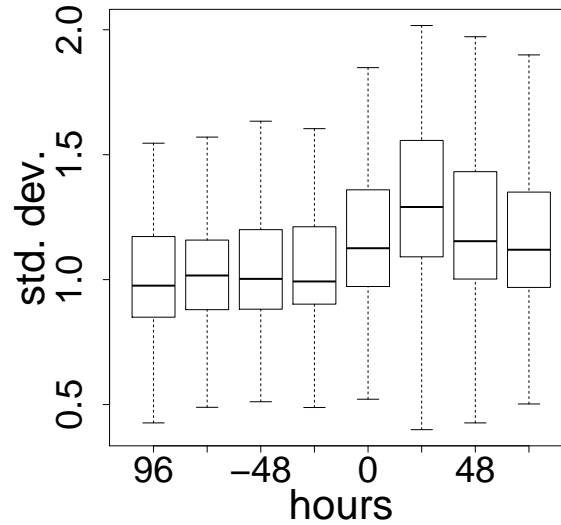


Figure 4.4: Standard deviation of new reviews every 24 hours before and after the first collected release for each app in our dataset. Each boxplot represents the standard deviation from the median for every app at that time.

Then we calculate the standard deviation from the median in eight different 24 hour periods (before and after a release). We calculate the median standard deviation at each time period if an app has multiple releases. Finally we divide the standard deviation by the total number of reviews in each app.

The previous steps produce Figure 4.4 which is a boxplot of all the apps. The figure shows a spike in reviews directly on and after the release day of an app.

0.19% of apps have more than 500 reviews per day, yet most studied apps receive few reviews per day. The number of downloads and releases impact the number of reviews that an app receives.

4.3.2 RQ2: Do similar issues emerge in the reviews of frequently-reviewed apps?

Motivation

As we have previously seen, a few apps receive many reviews per day but we do not know how varied the issues raised in these reviews are. Furthermore, we investigate if raised issues are universal across apps or if they occur in only some apps.

Approach

To study the variation in issues, we turn our attention to the frequently-reviewed apps. These apps are the ones that must cope with a large number of reviews. The more varied the issues in reviews, the more difficult it would be to group (e.g., summarize) the large number of received reviews. We examine the top 100 frequently-reviewed apps.

Uniqueness of topics: To measure the variation in issues within reviews we use Latent Dirichlet Analysis (LDA) to determine the issues raised in reviews. We use LDA to uncover topics which refer to issues.

Topic modelling: LDA is a popular statistical topic model (Blei et al., 2003). LDA generates topics from the corpus of documents and represents each document as a set of topics with associated probabilities. A topic is a mixture of terms drawn from the documents with associated probabilities for that topic. To generate the topic and term distributions, a generative process is applied to learn the correct distributions such as Gibbs Sampling or Collapsed Variation Bayes. For example, LDA will generate a list of words that are most associated with each topic and infer which documents are most associated with each topic. The output of LDA is a probability distribution for each topic and for each document. If

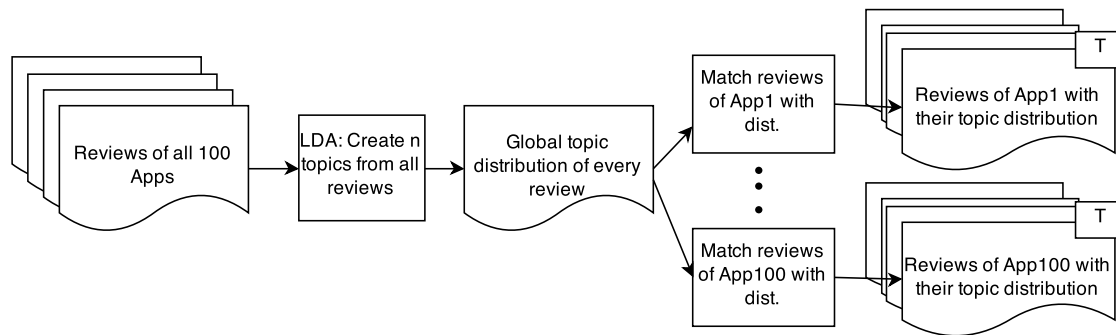


Figure 4.5: Our topic modelling process.

a topic's probability in a review is greater than 0.3 then we consider the topic (i.e., issue) to have occurred in the review. Figure 4.5 outlines the process of global and local topic modelling.

Parameter settings: We build an LDA model with the Stanford Topic Modelling Tool (STMT) (sta, 2012). We use the same settings as (Ramage and Rosen, 2011). The topic and term smoothing parameters are set at 0.1 and 0.1 respectively. The number of sampling iterations is 1,000. Document similarity is computed using the cosine similarity measure. To infer the distributions of words and topics, we use the collapsed variational bayes (CVB) inference.

STMT produces a probability distribution of the issues in each review. In order to determine if an issue is contained in a review we select a threshold where if the probability distribution value is higher than the threshold then the issue raised is in the review. The value that we selected is 0.3 after internal experimentation. Similar works have chosen values between 0.01 and 0.02 (Chen et al., 2014; Tian et al., 2009). We selected a higher value since we wish to be more conservative (i.e., certain) that the issue is contained in the review.

Selection of topic count: Selecting the number of topics for LDA remains an open

research question in the natural language processing field. We select a range of topic counts similar to (Titov and McDonald, 2008) and (Zhao et al., 2011), in order to determine a reasonable number of topics. The topic counts we select are 100, 150 and 500.

Preprocessing: We preprocess the review text before calculating the topics. Punctuation and stop words are removed. The words are stemmed. Each review is treated as a document and the reviews for each app are sorted by their review date. We filter out words that are in less than four reviews, and we filter out reviews with two or less words. The total number of reviews after preprocessing is 1.7 million.

Global and local: LDA will find two different types of topics (i.e. issues) that arise across the reviews of all apps: global and local. Global topics appear across many apps whereas local topics are specific to a particular app.

Understanding issues in reviews: We assume that if an issue occurs in a review according to LDA then the review is a representative occurrence of that issue.

Entropy: After we perform LDA modelling, we wish to find out the distribution of LDA topics across the studied apps. We investigate if the topics are local in a few apps or in global across all 100 apps. In order to investigate this research question, we use normalized Shannon's entropy Shannon (2001). The formula is the same as previously stated in Section 3.4 of Chapter 3. A high entropy denotes a global topic. A low entropy denotes a local topic.

Results

Global issues emerge across apps. As Figure 4.6 shows, the majority of topics are featured across most of the apps. We refer to these topics as global topics. We also find that there are more localized topics at the bottom of the plots. These topics are more concentrated on

just several apps or less. We refer to these topics as local topics.

The topic distributions remain relatively stable as we increase the number of topics.

We manually investigate the ten topics with the most entropy and the ten topics with the least entropy for the 100, 150 and 500 topic LDA model. I identified the topics based on the word summaries of each topic that LDA generates. For example the top 10 words for an topic on Google Maps where: map, location, navigation, Google, place, direction, find, offline, route and city.

We find that the global topics are general comments that could be about any app. The topics include aspects of the look and feel of apps, praise, thankfulness, recommending the app and discussing the quality of the developer's customer service. The local topics are all app specific. They include topics on maps and navigation for the app 'Google Maps', discussing listening to music for the music app 'Pandora' or a topic about messaging for a social network app called 'Kik'. Our results for 100 topics are displayed in Table 4.2 (we observed nearly identical topics in the 150, and 500 runs). In the table we include the top 10 keywords of each topic.

4.3.3 RQ3: Is there value in responding to reviews?

Motivation

By responding to a review a developer can engage the user personally. Developers are able to respond to a complaint or thank the user for kind remarks about the app. The response may motivate the user to change the rating of their app review or write a more positive review. However, developers have limited time. Responding to reviews takes away time that the developer could use to enhance their app. It is not clear how often users change their rating after a response, if at all. Additionally, it would be beneficial to know which

Table 4.2: Types of topics for the most global and local topics.

Topic #	Description	Entropy	Keywords	Topic #	Description	Entropy	Keywords
100							
Global				Local			
43	look and feel (size)	0.92	much, pretty, way, look, far, different, seem, actual, try, other	90	(App specific) Talking Angela	0.23	ask, man, see, dont, angela, talking, look, person, question, kid
9	praise and thanks	0.92	well, far, done, job, hope, enjoy, get, seem, thanks, pretty	21	(App specific) Kik	0.30	kik, girl, talk, gui, look, hey, chat, bore, people, old
91	recommend the app	0.91	recommend, highlight, without, everyone, anyone, amazing, definite, worth, live, have	27	(App specific) Google Maps	0.33	map, locate, navigation, google, place, direct, find, offline, route, city
15	quality of customer service	0.91	still, over, improve, service, year, customer, user, limit, long, experience	3	(App specific) Pandora	0.33	music, song, listen, pandora, playlist, skip, station, spotify, radio, album
7	Thankful	0.91	thankful, happy, never, found, amazing, day, feel, bore, first, final	31	(App specific) Zedge	0.33	ringtone, wallpaper, tone, lot, ring, select, variety, free, choice, zedge
51	Request change	0.91	thing, always, something, want, same, many, give, differ, think, try	39	(App specific) Groupon	0.43	deal, order, item, buy, price, groupon, purchase, save, pizza, stuff
65	Change or else won't receive 5 stars	0.90	give, rating, five, chang, ill, gave, reason, higher, problem, once	95	(App specific) Netflix	0.44	show, movie, watch, netflix, favorite, wish, put, want, chromecast, stream
80	Time	0.89	day, few, last, hour, month, year, ago, week, two, second	33	(App specific) Youtube	0.46	video, watch, youtube, upload, buffer, audio, record, post, problem, instagram
87	General issue, problem	0.89	problem, issue, never, have, always, soon, same, hope, trouble, seem	56	(App specific) Skype	0.47	call, video, viber, skype, chat, free, social, network, voice, media
40	Warning	0.89	never, try, out, before, get, give, think, once, anything, again	68	(App specific) Note Everything	0.49	perfect, note, list, everything, simple, look, handy, thing, quick, come

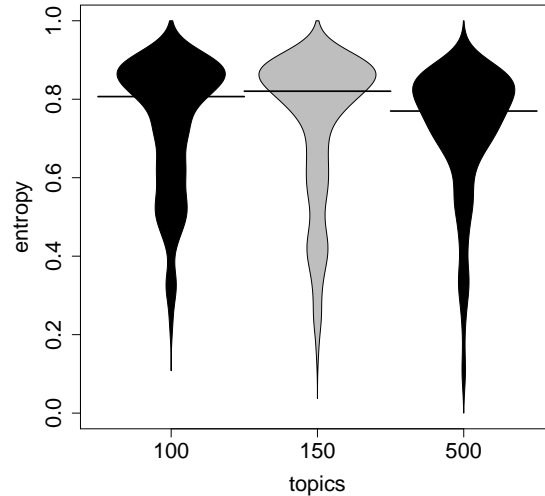


Figure 4.6: Entropy of each topic across the the 100 most reviewed apps (100, 150 and 500 topics)

types of reviews i.e., in terms of contents, are most likely to have their rating updated if the developer replied to the review. A strategic choice should be made to response to reviews which are most likely to positively update their ratings.

Approach

Manual analysis: We manually label a statistically representative sample of 384 reviews and their replies from 111,099 reviews with replies that occurred during the studied time period. The amount is the number required for a statistical sample with a confidence level of 95% and confidence interval of 5%. In total, I spent approximately 8 hours to manually analyze and label each review and response. A post-doctoral fellow reviewed the labels for consistency. If they disagreed, they came to a consensus, which occurred for very few reviews.

Automated analysis: To complement our manual analysis of replies we perform additional automated analysis to calculate the average rating change for reviews with replies, the probability of a rating change and the magnitude of the rating change. We then separate the reviews into 25 LDA topics and find which topics are most likely to lead to a positive change in rating. Our choice of 25 topics is motivated by a desire for general topics that are broad and that most developers would face.

For 20 days from April 7th to April 27th we monitored if reviews changed either the rating, comment or response. We denote a review and all subsequent changes as a review chain. The median review-chain length is 2 (meaning one review and one response). The maximum review chain is 8. We automatically analyzed 15,208 review chains in total. We analyze all the studied apps, as all developers, regardless of the number of downloads, benefit from the analysis.

Results

Most apps do not respond to reviews. Only 13.8% of 10,713 apps responded to reviews during the studied time period. As Figure 4.7 shows, apps with greater number of downloads never respond, but some apps in the mid-range of number of downloads response often. The apps with high response percentages have a low number of reviews.

Replies often lead to a positive change in review rating. Looking at all review chains, we find that 38.7% of users changed their rating after a response. We also find that the median change in rating for the 38.7% was a positive increase of one star (20%). This finding demonstrates that developers can benefit from responding to reviews. Some users even updated their review to notify the developer that the response had solved their problem or that the user was thankful that the developer had directly responded to them. We also

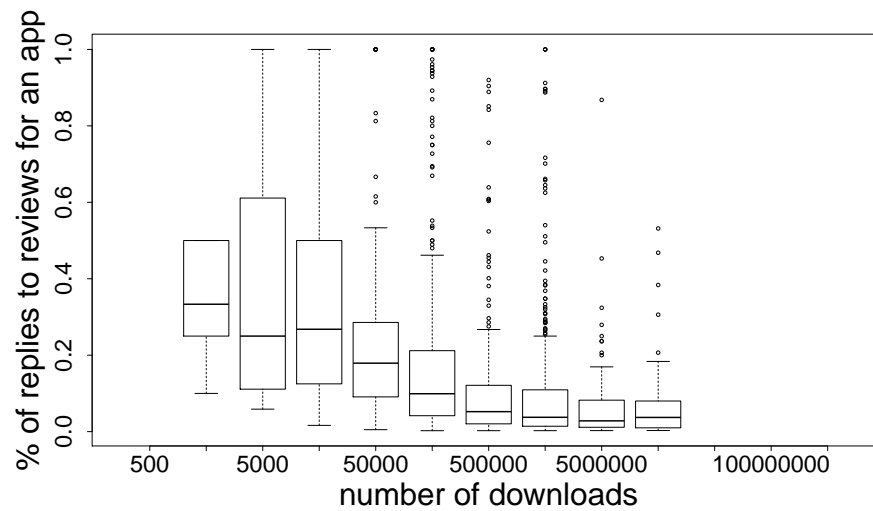


Figure 4.7: Percentage of replies to number of total reviews for each app separated by the number of downloads. Excluding apps with zero replies.

Table 4.3: Types of reviews written by users in our sample.

Review Type	Description	Amount
Praise	A user states they are pleased with the app.	129
Functional problem	A user states there is an error or unexpected behavior occurring.	126
Complaint	A user complains about the content or features of the app.	120
Request	A user makes a request for a new feature or addition to the app.	89
Update issue	A user preferred the previous release of the app.	36
Competitor	A user mentions a competitor to the app.	10
Other	Another language.	10

find that most reviews with replies are low rated with an average of 2.2 stars. This finding supports prior research which targeted negative reviews (1 and 2 star reviews (Maas et al., 2011) (Pang and Lee, 2004)) as being reviews of great interest and concern to developers of mobile apps (over higher star-rating reviews).

To ascertain which reviews were contributing most to rating changes, we perform LDA with 25 topics on both the reviews and the replies separately. Our selection of 25 topics is based on internal testing of an adequate number of representative topics that were not too specific but maintained some broad characteristics. Previous studies have divided reviews

Table 4.4: Types of replies written by developers in our sample.

Response Type	Description	Amount
Instructions	A developer provides assistance on how to use the feature or steps to fix the problem the user is having	120
Request Contact	A developer asks the user to contact the developer through a given email.	119
Thanks	A developer thanks the user for a positive review.	49
Next Release Fix	A developer states they will fix the problem the user has in the next release.	31
Current Release Fix	A developer states that the problem is already fixed in the current release. The user is asked to update their software.	26
Next Release Feature	A developer states they will add the feature the user requests in the next release.	12
Question	A developer asks for clarification.	9
Other	A response is in another language or does not address the user.	6
Rating-review mismatch	A developer asks why the positive review does not match the negative rating or vice-versa.	6
Current Release Feature	A developer states the current release contains the feature. The user is asked to update their software.	3

into a similar number of groupings (Khalid et al., 2014c; Pagano and Maalej, 2013).

The most common review topic that received replies was about crashing at 8%. The issue of crashing is a serious one and has a great impact on the experience of the user. It's understandable that developers focus on these reviews. We find that the chance of a rating change for each review topic was distributed between 15% to 42%. The two topics with highest chance of a rating change were about notifications and not being able to connect to the app. These are specific issues that can be addressed by developers. Users that wrote reviews about a specific Samsung phone had the single highest rating change on average following a response. The replies gave specific advice about the phone.

The most common response topic, at 7%, was concerned about developers notifying the user that a requested feature is either in development or is planned. Once again, the most common response topic is not the one associated with the greatest rating change. The response topic that had the highest chance of changing a rating (with 39.1% chance) was a topic on notifying the user that the issue the user had originally complained about had been resolved. The users may not have known that the issue was fixed and being told

personally resulted in more rating changes than other topics. We next look at the review and response types that occur in our manually labelled data.

In our manual analysis, more than one type can occur in the same review or response. Any review or response that does not conform to one of the types is considered as ‘other’. The ‘other’ reviews and replies were usually written in a language other than English or were not written in coherent English.

We find six common review types as Table 4.3 shows. The six types are praise, functional problem, complaint, request, update issue and mentioning competitors. Reviews contained a mixture of types between praise and complaints. The most common issues were praise for the app, followed by complaints of functional errors and complaints about the apps content.

We also find ten common response types in replies from developers as Table 4.4 shows. The nine types are instructions, request contact, thanks, next release fix, current release fix, next release feature, question, rating review mismatch and current release feature. The most common response type was instructions on how to solve the users problem, the second most common being a canned request to email the developers. The other types included reassurances that a problem was already fixed or that the problem would be fixed in an upcoming release. The same reassurances were provided to users who complained about a lack of a feature. The last three types were either thanking the user for leaving a kind review, asking why the negative rating of the review didn’t match the positive review or inquiring for further information about the user’s complaint.

Finally we match the reviews and replies in a table to show which response types occur most often with review types. As Table 4.5 shows, the most replied to reviews are praise, request for feature, and functional problem types. The majority of the replies to these

Table 4.5: Coupling of review types with response types.

	Instructions	Request contact	Thanks	New Re-release fix	New Re-release Feature	Current Re-release Fix	Other	Rating and Review Mismatch	Question	Total Replies
Praise	33	24	45	8	6	4	1	6	3	134
Request	35	31	3	7	10	4	2	0	1	130
Functional	39	60	3	10	4	12	0	0	6	124
Complaint	52	39	2	16	2	10	1	0	2	93
Was good	11	15	1	3	1	7	0	0	0	38
Other	3	4	1	0	0	0	2	0	0	10
Competitor	3	5	1	0	0	0	0	0	1	10

reviews were either direct instructions from the developer on how to solve the problem or the developer asked the user to contact them by email. We found that often a user would leave praise for an app but then either have a request or a minor problem to which the developer would respond.

Most apps do not respond to reviews, however responding can lead to a positive change in rating. Addressing specific issues and notifying the users that requested features are available are most likely to lead to a change in the review rating.

4.4 Discussion

We discuss the results of our research questions in this section.

4.4.1 The impact of replies on “app ratings”

We have shown that ratings of reviews change with replies. Now we investigate if the app rating is affected by developers who are active in responding to their users. We divide our entire dataset of apps into two categories using the median amount of responses over

the total amount of reviews: frequently-responding and infrequently-responding apps. We compare the quality of infrequently vs. frequently-responding apps. Our measure of quality is the percentage of “negative ratings” of reviews. We chose the percentage of negative ratings because most apps in our dataset have an overall four star rating. The percentage of negative ratings allows us to measure the amount of negative sentiment associated with the apps (Maas et al., 2011; Pang and Lee, 2004). We calculate the percentage of “negative ratings” (1 or 2 star ratings) for each app. The percentage of negative ratings is simply the ratio of one-star and two-star ratings over the total amount of ratings. Each review has an accompanying rating which we use as our metric. We only use the negative ratings given to apps during the time period of our study.

Frequently-responding apps have a lower percentage of negative ratings. As Figure 4.8 demonstrates, there are more negative ratings for infrequently-responding apps. The results are statistically significant with a Mann-Whitney test p -value < 0.05 .

4.4.2 The impact of popularity on app reviews

The apps that we chose in our dataset were popular one year ago. Over this time some apps are no longer popular and may not receive as many reviews as before. The percentage of apps that are highly-reviewed would be higher if the apps were all popular today. We recognize this issue and address it by determining the popularity of apps in our dataset now. We find that of the 10,713 apps that were popular according to Distimo, 4,354 of apps or 41% remain in Distimo’s top 400 categories. We find that all the 20 most frequently-reviewed apps that received over 500 reviews per day remain popular.

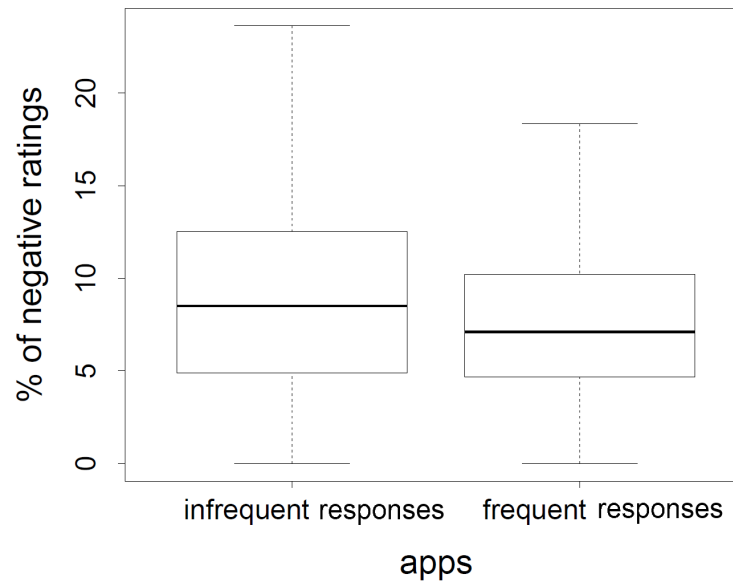


Figure 4.8: Percentage of negative app ratings for infrequently-responding vs. frequently-responding apps.

4.5 Threats to Validity

Some threats could potentially limit the validity of our results. We now discuss such threats and how we control or mitigate them.

4.5.1 Threats to Construct Validity

Since, we manually labelled our dataset of reviews with the different issue types, some reviews may have been incorrectly labelled. To mitigate this threat, I performed this labelling in an iterative manner and went over each review multiple times to ensure correct labelling of the reviews. To avoid any bias in the labelling process a post-doctoral fellow, verified the labelling. If they disagreed on a label they came to a consensus.

4.6 Conclusion

Future studies of review mining should consider the implications of our findings to motivate their own work. The usefulness of approach to assist developers in coping with large number of received reviews should be put into context. Apps benefit to varying degrees depending on the number of reviews that they receive and the amount of resources that they have devoted to responding and reacting to user feedback.

Responding to reviews improves the rating of reviews and occurs for both apps with many reviews and those with few reviews. Developers that take the time to respond to reviews will see the same benefits in the form of increased review ratings. Again, developers can make the decision based on their time constraints.

As we have shown in our empirical studies in Chapter 3 and Chapter 4, some developers face a torrent of feedback based in part on the distribution and feedback mechanisms. The distribution mechanism allows for an increase in frequency of releases which prompts bursts of new reviews and the feedback mechanism allows for a constant stream of reviews for popular apps. Clearly, developers need some way to cope with the overwhelming feedback due to the distribution and feedback mechanisms of app stores. In the next chapter, we propose an approach to help developers summarize the feedback that they receive.

Chapter 5

Analyzing and Automatically Labelling The Types of User Issues that are Raised in Mobile App Reviews

Mobile app reviews by users contain a wealth of information on the issues that users are experiencing. For example, a review might contain a feature request, a bug report, and/or a privacy complaint. Developers, users and app store owners (e.g., Apple, Blackberry, Google, Microsoft) can benefit from a better understanding of these issues – developers can better understand users' concerns, app store owners can spot anomalous apps, and users can compare similar apps to decide which ones to download or purchase.

However, user reviews are not labelled, i.e., we do not know which types of issues are raised in a review. Hence, one must sift through potentially thousands of reviews with slang and abbreviations to understand the various types of issues. Moreover, the unstructured and informal nature of reviews complicates the automated labelling of such reviews.

In this chapter, we study the multi-labelled nature of reviews from 20 mobile apps in

the Google Play Store and Apple App Store. We find that up to 30% of the reviews raise various types of issues in a single review (e.g., a review might contain a feature request and a bug report). We then propose an approach that can automatically assign multiple labels to reviews based on the raised issues with a precision of 66% and recall of 65%. Finally, we apply our approach to address three analytics use case scenarios: (i) we compare competing apps to assist developers and users, (ii) we provide an overview of 601,221 reviews from 12,000 apps in the Google Play Store to assist app store owners and developers and (iii) we detect anomalous apps in the Google Play Store to assist app store owners and users.

5.1 Introduction

As we have shown in the two empirical studies from previous chapters, app stores distribution and feedback mechanisms provide a convenient, simple and public way for users to download apps and to write reviews on their user-experience which leads to a large influx of reviews.

Such mobile app user reviews (from this point forward, we refer to the description section of a mobile app user review as simply a user review) contain valuable information e.g., feature requests, functional complaints, and privacy issues. This information is valuable to mainly three stakeholders (from this point forward, we will use the term “stakeholders” to refer to developers, users, and app store owners): (i) developers receive timely feedback about issues related to their apps e.g., bugs, feature requests, and any other issue, (ii) a user can read user reviews to make an informed decision whether or not to download/purchase an app, and (iii) app store owners, e.g., Apple, Blackberry, Google and Microsoft, can analyze user reviews to uncover anomalous apps, e.g., an app with an unexpected number or

Table 5.1: The number of user reviews occurring in a five day window (Sept 4 to Sept 9 2013).

App Name	User Reviews in a 5 day period
AccuWeather	878
The Weather Channel	665
WeatherBug	298

type of issues relative to other apps.

From Chapters 3 and 4, we have seen that the distribution and feedback mechanisms allow for an influx of reviews for certain popular, frequently-releasing apps. Due to the large number of user reviews and their free form, it is infeasible for stakeholders to fully benefit from the valuable information in user reviews through manual inspection. As seen in Table 5.1, there are hundreds of reviews that may occur per day for popular apps. The valuable information in such reviews has led to the emergence of a whole set of companies - mobile app analytics companies. Such companies specialize in providing detailed statistics and comparative analysis of user reviews and app revenues to their clients i.e., app developers (Annie, 2014; Flurry, 2014). However, much of the provided analytics are not software engineering oriented yet. For example, the occurrence of words in reviews across competing apps are presented, however, the provided analysis would not link such word occurrences to software related concepts (e.g., software quality).

Automated approaches are needed to automatically label reviews based on the types of raised issues e.g., feature requests, functional complaints, and privacy issues. However, users might raise several issues within a particular review. Figure 5.1 shows an excerpt of a user review where the user raises three issues about an update issue, the response time, and a functional complaint. We cannot label such user reviews with only a single issue. Moreover, such labelling is a difficult task due to the unstructured nature of reviews with

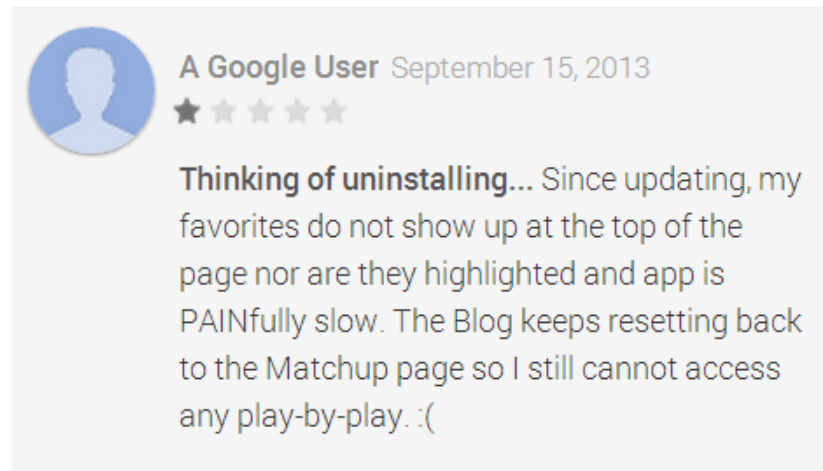


Figure 5.1: An example of a Google Play Store one-star user review of ‘theScore: Sports & Scores’ app. The review contains (raises) multiple issue types

many of them containing slang, lacking punctuation, and containing improper grammar.

In this chapter, we present an approach that can automatically assign multi-labels (i.e., multi-labelling) to user reviews. The approach is a direct response to the results we found in chapters 3 and 4. The approach helps the various stakeholders gain an overview of the users’ feedback that is readily available in the reviews, provides an overview of an app store and allows for the detection of anomalous apps. We perform a large scale empirical study to answer the following three research questions:

RQ1: How many user reviews contain multiple issue types?

Up to 30% of the user reviews raise more than one issue type, i.e., 30% of the data is multi-labelled. We identify 14 types of issues, e.g., feature requests, functional complaints, and privacy issues. We find that the issue types in our manually-labelled data are skewed towards certain issue types and that some issue types are correlated (i.e., they often co-occur often in the same review).

RQ2: How well can we automatically multi-label user reviews?

Our multi-labelling approach can correctly label issue types in user reviews with a precision of 66% and a recall of 65%. We also find that our approach can label some types more accurately (i.e., higher precision and recall) than other types.

RQ3: Are multi-label approaches useful for stakeholders?

We demonstrate several useful analytics use case scenarios of our multi-labelling approach for stakeholders using three different scenarios. The first scenario is useful to developers and users when comparing issues across competing apps. The second scenario is useful to app store owners and developers as we analyze the issue distribution of apps in the Google Play Store by app category (e.g., social, finance) and compare the issues from competing app stores for the same app titles. The third scenario is useful to app store owners when detecting anomalous apps based on user reviews and finding apps that violate or disregard the policies and guidelines of an app store (i.e., Google Play Store).

This chapter is organized as follows: Section 5.2 presents the related work. Section 5.3 provides the details of our empirical study and presents our approach. Section 5.4 presents the application of our approach on three analytics use case scenarios. Section 5.5 discusses the results of applying our approach. Section 5.6 discusses the threats to the validity of our study. Section 5.7 concludes our work.

5.2 Opinion Mining

Opinion mining is the study of user's sentiment. Opinion mining focuses on discovering the positive, negative or neutral opinion expressed by a user in a review (Pang and Lee,

2008). There are different levels of analysis: document level, sentence level and aspect (word) level (Hu and Liu, 2004). Document level analysis assumes one sentiment (Pang et al., 2002). Aspect level opinion mining concerns the pairing of an opinion word such as “like” with an aspect such as “smartphone”.

Ganesan *et al.* (Ganesan et al., 2012) proposed an approach to automatically summarize users opinions. Their approach generates a summary of users’ opinions; independent of different types of opinions in a cluster.

Pak and Paroubek used Twitter as a corpus of data (Pak and Paroubek, 2010) which is similar in many respects to the style of user reviews.

5.2.1 Opinion Mining of User Reviews

The importance of user reviews has motivated many recent studies on analyzing and summarizing user reviews for mobile apps as Table 5.2 shows. Khalid *et al.* manually analyzed and categorized one-star and two-star mobile app reviews (Khalid, 2013). Jacob and Harrison (Jacob and Harrison, 2013) built a rule-based automated tool to extract feature requests from user reviews of mobile apps – their approach identifies whether a user review contains a feature request or not. Chandy and Gu identified spam messages in reviews on the Apple iOS App Store (Chandy and Gu, 2012). Carreño *et al.* (Galvis Carreño and Winbladh, 2013) used opinion mining techniques and topic modelling to extract requirements from user reviews. Fu *et al.* present an approach that discovered inconsistencies in apps and analyzed the negative reviews of apps using topic analysis (Fu et al., 2013). Khalid *et al.* manually analyzed and categorized one and two star mobile app reviews (Khalid et al., 2014c). They manually identified the different issues users complained about in mobile apps. Pagano and Maalej analyzed the content of users’ feedback of both free and paid

Table 5.2: Datasets of prior work in mining mobile reviews

Paper	App Store	Apps	Reviews
Iacob and Harrison (Iacob and Harrison, 2013)	Google Play Store	161	3,279
Galvis and Carreno (Galvis Carreño and Winbladh, 2013)	Google Play Store	2	710
Fu et al. (Fu et al., 2013)	Google Play Store	171,493	13,286,706
Chen et al. (Chen et al., 2014)	Google Play Store	4	169,097
Pagano and Maalej (Pagano and Maalej, 2013)	Apple App Store	1,100	1,126,453

apps and its impact on the user community (Pagano and Maalej, 2013). We observed that most of the reviews are provided after a release, with a rapidly decreasing frequency over time. Chen *et al.* is the most extensive summarization approach to date (Chen et al., 2014). They remove un-informative reviews, and prioritize the most informative reviews before presenting a visualization of the content of reviews.

5.3 Empirical Study

In the empirical study section, we present our study design, the results of our preliminary study and our multi-labelled approach.

5.3.1 Background

In this section we introduce the goals of our study, we provide a brief background about app stores and mobile app analytics. We also motivate our choice to focus our study on negative reviews.

Study Design

The *goals* of our empirical study are to analyze the extent of multi-labelled user reviews, evaluate the effectiveness of automatically labelling multi-labelled user reviews, and present several analytics use case scenarios of automated labelling of user reviews for various stakeholders.

The *quality focus* of our empirical study are the evaluation measures that we use to evaluate the performance of our multi-labelling approach. Our *perspective* is that of practitioners (i.e., stakeholders) and researchers interested in user reviews for mobile apps. The *object* of our case study is user reviews from the Google Play Store and the Apple App Store.

Automated labelling of data in Software Engineering

Automated labelling approaches are widely used for sentiment analysis, spam detection and language detection. Melville *et al.* used Logistical Regression (LR) on text within blogs to identify the sentiment of writers (Melville et al., 2009). Jindal *et al.* used a LR model to find spam within reviews of products (Jindal and Liu, 2007).

Several researchers have explored the effectiveness of multi-labelling in software engineering (Ahsan et al., 2009; Han et al., 2012). Han *et al.* (Han et al., 2012) used LDA and LLDA to understand Android fragmentation identifying vendor-specific bugs of mobile device manufacturers. Ahsan *et al.* (Ahsan et al., 2009) demonstrated SVM to be a very effective classifier for labelling multi-labelled software change requests. Ramage *et al.* (Ramage et al., 2009) showed that SVM and LLDA performed similarly on multi-labelled data. The contents of bug reports and change requests are assumed to contain a bug report or change request whereas the contents of user reviews is open-ended and no

assumptions can be made as to what a user review will contain.

Some researchers have reported that even developers incorrectly label bug reports and feature requests (Antoniol et al., 2008; Herzig et al., 2013). To label bug reports automatically, Antoniol et al. proposed an approach to label change requests as a bug or a feature request (Antoniol et al., 2008). However, their approach can only handle binary labelled data. In addition, they analyze change requests only. User reviews are different from bug reports and change requests because bug reports and change requests have more structured style and are often much longer. There is an ability to exchange messages with the reporter, the reporter may assign priority and type i.e., whether it is an enhancement or defect report. Also the report can be filed into categories such as ‘accepted’, ‘closed’ and ‘duplicate’. There are no such mechanisms in user reviews on app stores.

Motivation to study one-star and two-star reviews

We select only one-star and two-star user reviews for analysis as we and previous literature make the assumption that one-star and two-star reviews are indicative of negative issues (Maas et al., 2011) (Pang and Lee, 2004). We decided to focus on negative issues since users and developers are most interested in addressing such issue types. We confirm our groupings of bad and good reviews by running a sentiment analysis tool (Thelwall et al., 2012) over all of the bad (one-star and two-star), neutral (three-star) and good (four-star and five-star) reviews of the studied apps. This tool estimates the positive and negative sentiment expressed in the text. As expected, one-star and two-star reviews were given a negative score while four-star and five-star reviews were given a positive score.

We present the results of our empirical study in the following sub-sections. We first

present a preliminary study to motivate our work then we present our approach to automatically multi-label user reviews. We follow the results of our empirical study with a discussion of three scenarios that demonstrate the application of our approach.

5.3.2 Preliminary Study

We first explore how often do multiple issues occur in user reviews and if so, how many issue types co-occur within a user review. The amount of multi-labelled data informs our decision to use single-labelled or multi-labelled approaches. Single-labelled approaches map user reviews to one label whereas multi-labelled approaches map user reviews to more than one label. One can use multi-labelled approaches on single labelled data, however; multi-label approaches are more complex as the hypothesis space is larger and so multi-labelled approaches would decrease prediction performance unnecessarily. For example, determining whether a user review contains a functional complaint issue or not is an inherently simpler task than determining whether a user review contains a functional complaint, and/or a network problem, and/or a crashing issue. If we use single-labelled approaches on multi-labelled data we will be guaranteed to mis-label the multi-labelled user reviews. Therefore, we carefully consider our choice of approach based on the amount of multi-labelled data that we observe in our preliminary study.

RQ1: How many user reviews contain multiple issue types?

Our process to answer RQ1 is divided into several sections: data collection, our approach to analyze the data, and our results.

Table 5.3: User reviews were collected from these 20 apps on the Apple App Store and user reviews from 4 of these apps were collected from the Google Play Store

Name of Apps
Weight Watchers Mobile, Evernote, Hulu Plus, Yelp, Netflix, CNN App for iphone, Farmville by Zynga, Find my iphone, Word Lens, Foursquare, Facebook, Wikipedia Mobile, Adobe Photoshop Express, Last fm, Kindle, Metalstorm Wingman, ESPN Scorecenter, Mint, Epicurious Recipes shopping list, Gmail

Data Collection

We analyze the user reviews of 20 Apple App Store apps and 4 Google Play Store apps. The list of apps can be found in Table 5.3. We select apps which cover a broad range of categories and have a significant number of user reviews. Half of the apps that we choose have an above average rating, while the other half have a below average rating. We select 4 matching Google Play Store apps from the Apple App Store to examine how issue types from Google Play Store apps differ from those of Apple App Store apps given the exact same apps. After identifying the apps, we built a simple web crawler to automatically collect the user reviews of these apps from the website of each app store.

In total, we downloaded 226,797 one-star and two-star user reviews for the Apple App Store and 3,480 one-star and two-star user reviews for the Google Play store (see Table 5.4). The discrepancy between the number of user reviews across the app stores is due to the different APIs that Apple and Google exposed for data collection at the time of our study. Apple App Store allows us a user to subscribe to an RSS feed for user reviews, however the Google Play Store only lists the top 500 reviews for each star rating. Hence, the collection of user reviews of the Google Play Store was restricted as no tool existed at the time, to collect the reviews other than crawling the Google Play Store Website.

Table 5.4: Data statistics for data collected from the Apple App Store and the Google Play Store

Collected		Manually Labelled		Auto-Labelled
Apple App Store	Google Play Store	Apple App Store	Google Play Store	Google Play Store
226,797	3,480	6,390	1,066	601,221

Approach

In our previous study (Khalid, 2013), we manually labelled a statistically representative sample of 6,390 and 1,066 user reviews for the Apple App Store and the Google Play store respectively. Both amounts are above the number required for a statistical sample with a confidence level of 95% and confidence interval of 5%. We labelled only user reviews with a one-star or two-star rating. In total, a graduate student spent approximately 125 hours to manually analyze and label each user review. A faculty member and a post-doctoral fellow reviewed the labels for consistency. If they disagreed on a label, they took the majority opinion, i.e., in total there were three votes, one from the individual who labelled the data and two from the individuals who verified the data. If the vote was a three-way tie they came to a consensus which occurred for very few reviews.

Our manual labelling process identified issue types from app reviews via manual inspection. Instead of beginning with a set of issues, we analyzed the data to identify common concepts which can be grouped together. We refer to these concepts as issue types. Our selection of issues was motivated by what we believed would be useful for developers and independent of particular apps. There is no doubt that there exist multiple different issues that may or may not be relevant for developers that we do not consider. However, we believe that the selected issues is a baseline from which future work can be done to

Table 5.5: Descriptions of each issue type.

Issue Type	Description
Additional Cost	Complain about the hidden costs to enjoy the full experience of the app
Functional Complaint	Unexpected behavior or failure
Compatibility Issue	App has problems on a specific device or an OS version
Crashing	The app is often crashing
Feature Removal	One or more specific feature is ruining the app
Feature Request	App needs additional feature(s) to get a better rating
Network Problem	The app has trouble with the network e.g., network lag
Other	A review-comment that is not useful or doesn't point out the problem
Privacy and Ethical Issue	The app invades privacy or is unethical
Resource Heavy	The app consumes too much battery or memory
Response Time	The app is slow to respond to input, or is laggy overall
Uninteresting Content	The specific content is unappealing
Update Issue	The user blames an update for introducing new problems
User Interface	Complain about the design, controls or visuals

explore the remaining issues (Khalid, 2013). In total, we identify 13 issue types from the user reviews that we randomly sampled as shown in Table 5.5. We include a 14th issue type (“other”) that covers any user review that does not conform to the 13 issue types.

In our current study, to observe the relationship between issue types, we calculate the percentage of multi-labelled data per issue type e.g., 39% of functional complaints are multi-labelled and the amount of co-occurrence between the various issue types. For example, a functional complaint occurred in 44% of the user reviews that contained a network problem issue. We calculate the word length of the single versus the multi-labelled data to see how they differ. Finally, to study the type of multi-labelled data, we calculate the label cardinality and label density of each review. Label cardinality and label density are

commonly used statistics to describe multi-labelled data (Tsoumakas and Katakis, 2007). These measures are used to quantify the amount of multi-labelling in the studied data set. Let Y be the number of relevant labels, L be the number of all the different label types and N be the size of the dataset.

$$\text{label cardinality} = \frac{1}{N} \sum_{j=1}^N Y_j \quad (5.1)$$

Label density is the average number of labels in the example divided by the total number of different label types L .

$$\text{label density} = \frac{1}{N} \sum_{j=1}^N \frac{Y_j}{L} \quad (5.2)$$

Results

We find that the data set is skewed towards certain issue types as seen in Table 5.6. Functional complaint, feature request and crashing dominate over the other issue types, while issue types such as uninteresting content and resource heavy occur at a much lower frequency.

For the Google Play Store user reviews, 30% (317) of the 1,066 user reviews are multi-labelled. The maximum number of labels for a user review is 4; the average number of labels per review is 1.4.

For the Apple App Store user reviews, 22% (1431) of the 6,390 user reviews are multi-labelled. The maximum number of labels for a user review is 5; the average number of labels per review is 1.2.

We observe that the reviews associated with certain issue types are more multi-labelled than other reviews. The multi-labelled column in Table 5.6 shows how many times an issue

Table 5.6: The percentage of issue types in our labelled dataset as well as the probability that a specific issue type co-occurs with other issues i.e., a multi-labelled user review

Issue Type	Apple App Store		Google Play Store	
	Freq. %	Multi-Labelled %	Freq. %	Multi-Labelled %
Additional Cost	10.8	27.8	1.7	66.7
Functional Complaint	26.7	39	34.1	50.3
Compatibility Issue	2.4	27.7	14.3	11.2
Crashing	16.2	40.3	11.3	56.7
Feature Removal	6	33.4	3	62.5
Feature Request	19.7	28.8	23.1	43.1
Network Problem	10.3	58.1	19.3	65
Other	11.7	0	6.9	0
Privacy and Ethical Issue	2.4	17	0.7	28.6
Resource Heavy	0.4	41.7	6	56.3
Response Time	1.6	64.1	2.3	80.8
Uninteresting Content	0.7	32.6	0.2	50
Update Issue	11.3	74.7	9	78.1
User Interface	4.6	57.4	3.7	56.4

type co-occurred with others. As Figure 5.6 demonstrates ‘functional complaint’ occurs with many other issue types. ‘Update issue’ is in multi-labelled data 75% of the time. On the other side of the spectrum, certain issue types co-occur less frequently with others, such as ‘privacy and ethical issue’ with only 17% co-occurrences. The issue types with the most co-occurring labels are crashing, feature request, functional complaint, network problem, and update issue.

We find that certain issue types co-occur together with increased frequency than others as Figure 5.2 shows. For example, we observe that the issue type ‘crashing’ and ‘functional complaint’ occur frequently with the ‘update issue’. Such co-occurrences of issue types demonstrate that issue types are not independent.

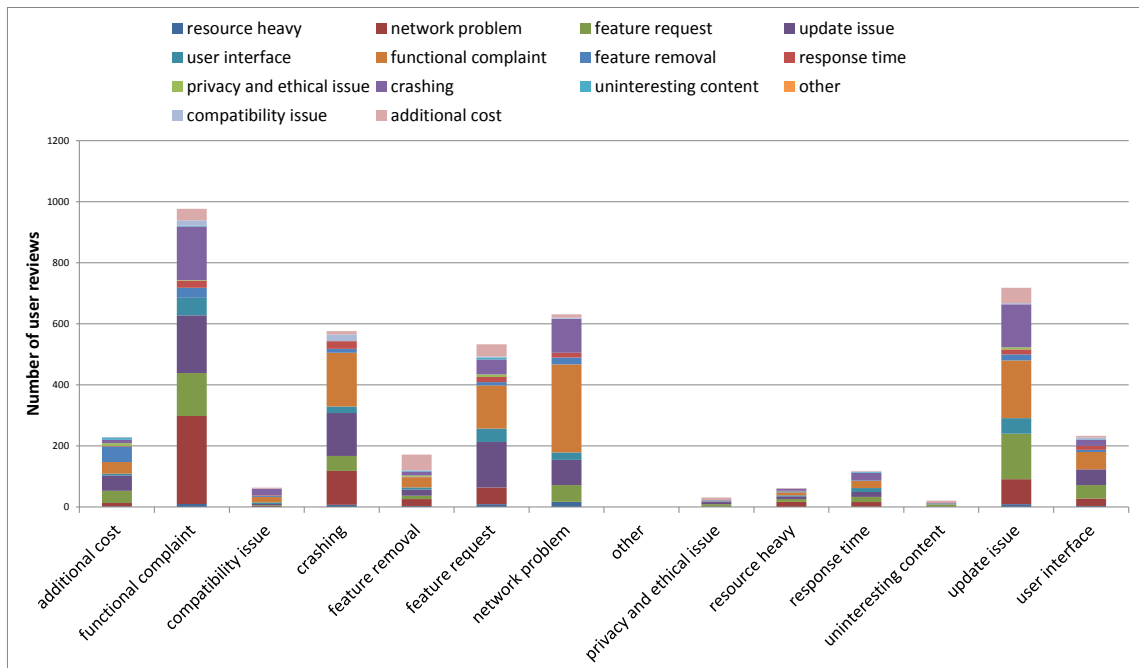


Figure 5.2: Amount of co-occurrence per issue types.

As Figure 5.3 shows, multi-labelled user reviews are longer than single-labelled user reviews. The intuitive reason is that the reviewer has more to say when raising several types of issues in the same review. On average, a user review contains 32 and 33 words for Google Play Store and Apple App Store user reviews respectively. Multi-labelled user reviews are on average 41 words for both Google Play and Apple App Stores.

Both stores show similar patterns. The user reviews of both stores have low label cardinality and label density. The observed values for label cardinality and density in our dataset are similar to prior research on multi-label text datasets (Read, 2010). The label cardinality is 1.36, 1.25 and 1.26 for the Google Play Store, Apple App Store and combined respectively. The label density is 0.10, 0.09 and 0.09 for the Google Play Store, Apple App Store and combined respectively. In the following section we make use of prior research

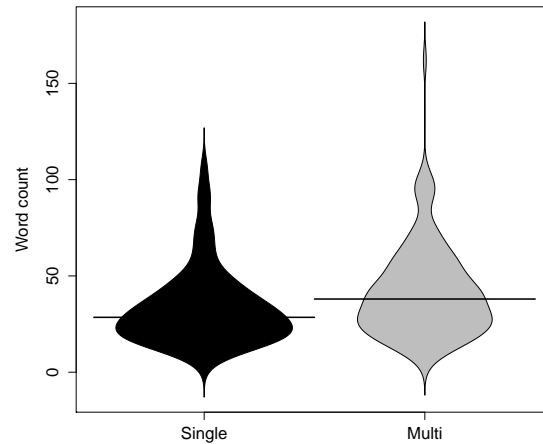


Figure 5.3: A bean plot of the length of user reviews in both single and multi-labelled data for both stores

techniques for automated labelling of multi-labelled datasets.

30% of Apple App Store and 22% of Google Play Store user reviews are multi-labelled. Users raise multiple issue types in a single review and the distribution of issue types is skewed with some issue types occurring more frequently than others. Moreover, some issue types tend to co-occur at a higher frequency than other issue types.

5.3.3 Automated Labelling of User Reviews

From the results of RQ1, we conclude that the user reviews of the Google Play Store and Apple App Store contain a substantial amount of multi-labelled data. Hence, we wish to explore how well we can automatically label user reviews using machine learning models that are tailored specifically to multi-labelled data.

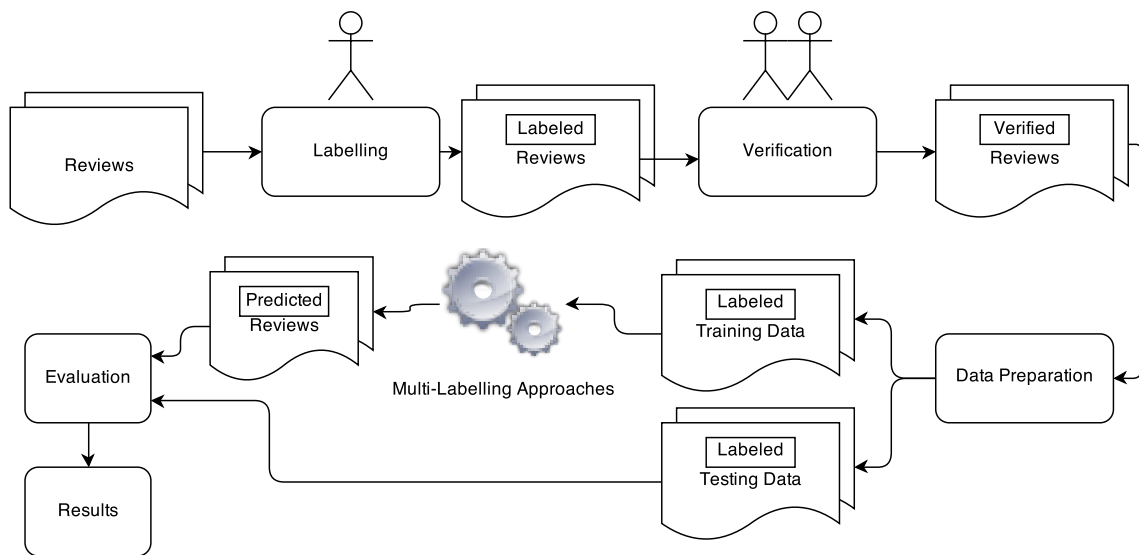


Figure 5.4: Process to label multi-labelled user reviews.

RQ2: How well can we automatically multi-label user reviews?

Our process is divided into several steps: data preparation, multi-label approaches and evaluation and results (as shown in Figure 5.4). In the following subsections, we explain each step in detail.

Data Preparation

The input to the data preparation step is the manually-labelled data from RQ1. The output is a list of processed user reviews.

Preprocessing: For all user reviews, we merge the title and comment together. We choose not to remove stop words because certain issue types benefit from words such as

‘should’ and ‘could’ as shown by Iacob and Harrison (Iacob and Harrison, 2013). We remove all numbers and special characters except hyphens and apostrophes. We filter words that occur less than three times in our data set. Such filtering removes rare instances of misspellings and long sequences of meaningless characters. Filtering words that occur less than three times reduces the complexity of assigning labels to user reviews and the approaches are less likely to overfit to the data. We stem the words using the snowball stemmer (Porter) because it implements the popular Porter stemming algorithm (Porter, 1980). We expand abbreviated words such as ‘couldn’t’ to ‘could not’. We ensure that user reviews are meaningful with at least a one-word title and a three-word comment. Hence, we remove any user review that is three words or less e.g. *review: bad!, Not working*. At the end of the preprocessing step, we are left with 7,290 reviews out of the initial set of 7,458 manually labelled user reviews.

Frequency Building of Words: We transform the text of the user reviews into an array of word frequencies. The word frequency format is readable by machine learning algorithms. We use the `String To Word Vector` filter, available in MEKA (a multi-label classification tool (Read, 2013) which is an extension to WEKA (Hall et al., 2009a)), in order to build a dictionary of all words left after preprocessing. There are 6,525 unique words before preprocessing and 2,771 unique words after preprocessing. The attributes of each user review are the words. The MEKA filter forms a $M \times N$ matrix of M user reviews and N word attributes. $M_i N_j$ represents the frequency of a word. We employ the term frequency–inverse document frequency (TF-IDF) as a means to increase the weight of words that occur frequently in a single user review and decrease the weight of words that occur frequently in many user reviews (Rajaraman and Ullman, 2012). For example, the

word ‘app’ would be penalized because it occurs in many user reviews whereas an uncommon word like ‘omission’ would be given a higher weight. $f_{i,j}$ is the frequency of a word i in user review j :

$$TF-IDF(word_{i,j}) = f_{i,j} * \log\left(\frac{total_user_reviews}{total_user_reviews_with_word_i}\right) \quad (5.3)$$

Multi-labelling Approaches

In this chapter, we experiment with several different multi-labelling approaches e.g., Binary Relevance (BR), Classifier Chains (CC), and Pruned Sets with threshold extension (PSt) as well as several different classifiers e.g., support vector machines (SVM), decision tree (J48) and Naive Bayes (NB).

Selection of approaches and classifiers: We selected three approaches for multi-labelling the user reviews. The approaches transform the problem of classifying multi-labelled data into one or more problems for single labelling. For example, if a multi-labelled problem had two labels, the approach would predict if the user review contained the first label, the second label, both labels, or neither. Such prediction is accomplished by building two classifiers, a classifier for label one and a classifier for label two. The results from both classifiers are combined. Such a problem transformation allows standard discriminative machine learning models like SVM to be used to multi-label user reviews.

As Figure 5.5 demonstrates, BR transforms the problem into many single-labelled problems. BR will construct N binary models for N labels. The models can be any binary machine learning algorithm. The main weakness of BR is that it does not leverage the correlations between labels. BR’s loss of information is problematic because the issue types in our dataset are correlated (i.e., some of them have high co-occurrence probabilities) as

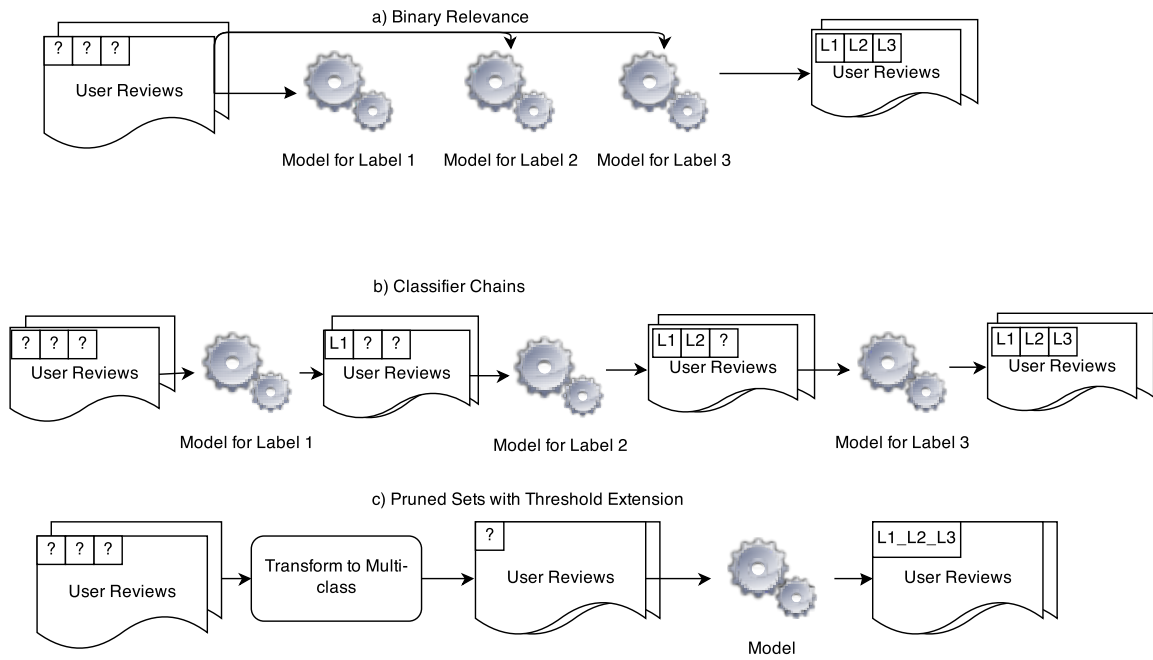


Figure 5.5: Examples of Binary Relevance, Classifier Chains and Pruned Sets with Threshold Extension

shown in the results of our preliminary study (see Section 5.3.2) i.e., a network problem is more likely to be in a user review that has a bug complaint.

CC extends BR by building models in a serial fashion and every k^{th} model takes into account the prediction of the $k - 1^{th}$ model. As such, CC does not assume independence between labels and correlates labels with one another (Read et al., 2009).

PSt differs from BR and CC by treating each possible multi-label combination as a value in a single label. PSt leverages the correlations between labels. PSt produces a large number of possible outcomes that contain each single label e.g., 2^n possible combinations exist for n labels. PSt removes infrequently-occurring label combinations and any user reviews that contained the infrequent label combination are duplicated into multiple

single labelled user reviews. For example, an infrequently-occurring combination label $\{\textit{functional complaint}, \textit{additional cost}\}$ would be removed and the user reviews i containing the combination would be duplicated into two user reviews $i_1 \{\textit{functional complaint}\}$ and $i_2 \{\textit{additional cost}\}$. Furthermore, any combinations that had never occurred in the training data and therefore would be excluded from the list of possibilities would be given a posterior probability. The posterior probability is the percentage of a label's frequency in the training data. For example $\{\textit{functional complaint}, \textit{response time}, \textit{network problem}\}$ which never occurred in the training data would be assigned a probability based on the probability of each label occurring together e.g., probability of $\{\textit{functional complaint}, \textit{response time}, \textit{network problem}\} = \text{probability of } \{\textit{functional complaint}\} \text{ and } \{\textit{response time}\} \text{ and } \{\textit{network problem}\}$ occurring in the same user review.

Our selection of the SVM, J48 and NB classifiers is motivated by prior studies which demonstrate the good performance of these classifiers with multi-labelled data (Read, 2010).

The BR, CC, PSt, SVM, J48 and Naive Bayes classifiers are implemented in MEKA. We use the default values of each classifier in MEKA.

Threshold Optimization: Multi-labelled classification requires a threshold. Surpassing the threshold with a prediction confidence indicates that a user review has such a particular label. There is a corresponding threshold value for each label. More formally, in multi-labelled classification the output of the model is a matrix of $d \times l$ confidence predictions; where d are the user reviews and l are the labels. For example, the model predicts that the label l_i is in user review d_j because the model's prediction confidence is 0.7 for l_i . 0.7 is higher than the threshold 0.6 for l_i . The confidence predictions are real numbers ranging from 0 to 1. The goal of threshold optimization is to determine at what value a confidence prediction denotes class membership of a label. More information on threshold

optimization can be found in (Fan and Lin, 2007).

We use the *Proportional Cut* threshold algorithm and determine a separate threshold for each of the 14 labels. The thresholds are set based on the occurrence of label values in the training data. This threshold optimization assumes that the test data will have a similar distribution. This algorithm exhibits similar performance as well as other methods in literature (Read, 2010).

Evaluation

Cross-validation: To test the accuracy of performing multi-labelled classification on our labelled dataset, we perform a 10-fold cross-validation. The data is split into 10 equal groups (i.e., folds). The first group is selected as the testing data and the other 9 groups are selected as the training data. The process is repeated 10 times with a different group chosen as the testing data for each repetition.

Justification of evaluation measures: We provide six evaluation measures for multi-labelled data. The rationale for providing six different measures is for the following reasons: (i) There are multiple different evaluation measures used in the literature for multi-labelled data (Tsoumakas et al., 2010). (ii) BR's, CC's and PSt's performance, relative to one another, are affected by the choice of evaluation measure (Read, 2010). (iii) Each measure has different levels of strictness and penalizes the errors in the classifier differently. (iv) There are two categories of evaluation measures for multi-label classification, i.e., label-set and label-based. Label-set measures (we refer to label-set measures as review-based measures from this point forward) independently evaluate each user review, whereas label-based measures independently evaluate each label. Accuracy and exact match and F-measure micro are review-based measures, f-measure macro (by label) is an example of a

label-based measure. An approach may have superior performance in one measure relative to another measure.

Evaluation measures: The first selected measure is called exact match (Read et al., 2009). Exact match is a very strict evaluation measure. Exact match is defined as follows: let \mathbf{S} be the predicted label values, let \mathbf{Y} be the actual label values. Let r_j be a user review. Let n be the total number of user reviews.

$$f(r_j) = \begin{cases} 1 & \text{if } \mathbf{S}_j = \mathbf{Y}_j \\ 0 & \text{if } \mathbf{S}_j \neq \mathbf{Y}_j \end{cases} \quad (5.4)$$

$$exact_match = \frac{1}{n} \sum_{j=1}^n f(r_j) \quad (5.5)$$

For example, if a model predicts {functional complaint, response time, network problem} and the actual labels are {functional complaint, response time, crashing, feature request} the exact match score is 0 because there is at least one label incorrectly labelled.

A less strict alternative is multi-labelled accuracy (Read et al., 2009). The intersection of the predicted labels with the actual labels is divided over the union of the predicted labels with the actual labels. Let \mathbf{Y}_j be the actual labels for user review j , let \mathbf{S}_j be the predicted labels for user review j and n be the number of user reviews.

$$accuracy = \frac{1}{|n|} \sum_{j=1}^n \frac{|\mathbf{S}_j \cap \mathbf{Y}_j|}{|\mathbf{S}_j \cup \mathbf{Y}_j|} \quad (5.6)$$

For example, if a model predicts {functional complaint, response time, network problem} for a review and the actual labels are {functional complaint, response time, crashing, feature request} then the intersection is {functional complaint, response time} and the

union is {functional complaint, response time, crashing, feature request, network problem}.

The accuracy is $\frac{2}{5}$ for the example review.

F-measure is another common evaluation measure in the multi-labelled literature. It is the harmonic mean of precision and recall. Let \mathbf{z} be a vector of 0/1 values of actual labels and $\hat{\mathbf{z}}$ be the predicted labels.

$$precision(\mathbf{z}, \hat{\mathbf{z}}) = (|\mathbf{z} \cap \hat{\mathbf{z}}|)/|\hat{\mathbf{z}}| \quad (5.7)$$

$$recall(\mathbf{z}, \hat{\mathbf{z}}) = (|\mathbf{z} \cap \hat{\mathbf{z}}|)/|\mathbf{z}| \quad (5.8)$$

$$F\ measure(\mathbf{z}, \hat{\mathbf{z}}) = \frac{2 * precision(\mathbf{z}, \hat{\mathbf{z}}) * recall(\mathbf{z}, \hat{\mathbf{z}})}{precision(\mathbf{z}, \hat{\mathbf{z}}) + recall(\mathbf{z}, \hat{\mathbf{z}})} \quad (5.9)$$

There are different ways to calculate the precision, recall and f-measure for the entire dataset. The method of calculation is dependent on how the \mathbf{z} vector is created. Let N be the number of user reviews and L be the number of labels. In the micro version \mathbf{z} is $L \times N$ data points:

$$\mathbf{z} = [y_j^i, \dots, y_L^N] \quad (5.10)$$

In the micro version there is one precision, one recall and one f-measure for the entire dataset. The f-measure micro is a global f-measure across all reviews. The \mathbf{z} and $\hat{\mathbf{z}}$ are calculated per review and then the value of both are summed up across all the reviews. This summation produces two global values that are used to calculate the precision and recall.

Table 5.7: Example of evaluation measures

Two Example Reviews	Labels	L1	L2	L3	L4	L5	L6	L7
Review 1	Actual	✓	✓	✓	✓			
	Predicted	✓	✓			✓		
Review 2	Actual	✓	✓			✓		✓
	Predicted					✓	✓	
Measure		Value						
Exact Match		$\frac{1}{2}(0 + 0) = 0$						
Accuracy		$\frac{1}{2}(\frac{2}{5} + \frac{1}{5}) = 0.3$						
Precision-Micro		$\frac{3}{5} = 0.6$						
Recall-Micro		$\frac{3}{8} = 0.375$						
Precision-Macro by Label		$\frac{1}{7}(\frac{1}{2} + \frac{1}{2} + \frac{0}{1} + \frac{0}{1} + \frac{1}{2} + \frac{0}{0} + \frac{0}{1}) = 0.214$						
Recall-Macro by Label		$\frac{1}{7}(\frac{1}{2} + \frac{1}{2} + \frac{0}{1} + \frac{0}{1} + \frac{1}{1} + \frac{0}{0} + \frac{0}{1}) = 0.286$						

$$F \text{ measure } micro(D) = F \text{ measure}(\mathbf{z}, \hat{\mathbf{z}}) \quad (5.11)$$

The \mathbf{z} vector for macro (by label) is of size N :

$$\mathbf{z} = [y_j^1, \dots, y_j^N] \quad (5.12)$$

Given that there are L labels, there are L precision, recall and f-measures. The f-measure macro (by label) is calculated as the average f-measure for each label. The \mathbf{z} and $\hat{\mathbf{z}}$ are summed per label.

$$F \text{ measure } macro \text{ by Label} = \frac{1}{|L|} \sum_{j=1}^L F \text{ measure}(\mathbf{z}_i, \hat{\mathbf{z}}_i) \quad (5.13)$$

We evaluate the models using the following measures: exact match, accuracy, micro precision, micro recall, micro f-measure and macro f-measure L (by label). Table 5.7 demonstrates examples of the different measures.

Table 5.8: Performance of a multi-labelled approach on 14 labels

Evaluation Measure	Pruned Sets with threshold extension - SVM
Precision	65%
Recall	64%
F-measure micro	64%
F-measure Macro (by label)	56%
Accuracy	59%
Exact Match	44%

Results

Results for all 14 labels: As Table 5.8 shows average performance results, we achieve 59% accuracy, 44% exact match, 65% precision, 64% recall, 64% F-measure micro and 56% F-measure macro (by label). The results are well above 1/15 (0.07%), which is the random chance of guessing that a user review contains one of the 14 labels or no label at all. We find the results to be satisfactory as they compare well with prior results of other multi-labelled classification efforts (Read, 2010). We leave the improvement in performance to future work as this is a first step.

Ambiguous issue types: As a precaution, we investigated the performance of each label individually. We remove the multi-labelled Google Play Store and Apple App Store user reviews (the single labelled user reviews allows us to observe individual f-measures and precision-recall curves (PRC) for each label). We follow the data preparation steps in RQ2 and run the SVM classifier that is available in WEKA. The PRC is the area under a graph with recall on the x-axis and precision on the y-axis. The PRC ranges from 0 to 1. A PRC of 1 denotes 100% precision and recall. As Table 5.9 demonstrates, certain labels perform poorly compared to others. In particular, the response time, uninteresting content, and user interface labels perform poorly.

Table 5.9: Performance of a classifier on single labelled user reviews from the Apple App Store and Google Play Store

Issue	F-Measure	Precision-Recall Curve Area
Additional Cost	78	84
Functional Complaint	64	69
Compatibility Issue	64	73
Crashing	89	94
Feature Removal	56	61
Feature Request	66	72
Network Problem	63	64
Other	52	54
Privacy and Ethical Issue	45	48
Resource Heavy	64	61
Response Time	23	27
Uninteresting Content	19	18
Update Issue	53	49
User Interface	38	34

We conclude that future research may be needed to accurately predict these ambiguous issue types at a higher f-measure. Hence, for now we merge these three ambiguous issue labels into the ‘other’ issue type. In doing so our average f-measure may improve as the model is no longer burdened with the hard to identify issue types.

The fact that we removed the poor performing labels does not invalidate the results of our other labels. We do not remove the data with poor performing labels. We replace the labels of the reviews with poor performing labels with the “other” label. The model must predict the “other” label correctly in the poor performing reviews. The better accuracy is due to the reduced number of labels. An $n - 1$ label problem is easier to predict than an n label problem just as a binary problem is easier for a model to predict than a 14 label problem.

Results for 11 labels: We now perform the multi-label classification on the 11 remaining issue types. The results show that the f-measure macro (by label) measure increases by 56% to up to 63%. The increase demonstrates that the merging of the ambiguous issue types with the ‘other’ issue improved the performance of individual labels.

We find that the LibSVM classifier performs the best amongst the three different classifiers and that PSt performs the best amongst the multi-label approaches. Table 5.10 shows the results of the various classifiers using the aforementioned cross-validation process on the Google Play Store. We find that CC does not perform as well as BR in the f-measure evaluation measures. The lower performance of CC is likely due to the low label cardinality of 1.26 and the limited number of cross-label correlations for CC to take advantage of. We perform the Kruskal-Wallis statistical test ($\alpha = 0.05$) on the accuracy evaluation measure to determine if the differences in the results are statistically significant 1) between the three models: Naive Bayes, J48 Decision trees, and LibSVM 2) between the three approaches using LibSVM: BR, CC, and PSt and 3) between all nine multi-labelling model combinations. The differences in performances are statistically significant. The findings hold even after applying a bonferroni correction.

The approaches that we use in this study i.e., BR, CC, and PSt have known biases (Read, 2010). The performance of the BR and CC approaches favors label-based evaluation measures e.g., f-measure macro (by label) whereas the performance of the PSt approach favors review-based evaluation measures (Read, 2010).

Pst and BR have similar precision, recall and f-measure micro results using LibSVM. PSt outperforms BR and CC according to the accuracy measure and exact match measure. However, BR outperforms PSt and CC according to f-measure macro (by label). We ultimately, choose PSt as it has superior accuracy and exact match results compared to BR which has only superior f-measure macro (by label).

<p>Our multi-labelled classification of mobile user reviews has a precision of up to 66% and a recall of up to 65%.</p>

Table 5.10: Performance of approaches on the user reviews from the Apple App Store and Google Play Store

Approaches	Binary Relevance			Classifier Chains			Pruned Sets with threshold extension		
	Naive Bayes	Decision Tree	SVM	Naive Bayes	Decision Tree	SVM	Naive Bayes	Decision Tree	SVM
Precision	17	38	66	17	53	75	32	47	66
Recall	76	55	65	82	51	54	38	49	65
F-Measure Micro	28	45	65	28	52	63	35	48	65
F-Measure Macro L	28	48	63	27	47	60	31	43	62
Accuracy	21	33	56	20	47	52	27	45	60
Exact Match	0.2	11	41	0.17	36	42	11	33	45

5.4 Applications of our Multi-labelling Approach

As we have shown, we can label user reviews with 66% precision and recall of up to 65%. Therefore, we wish to demonstrate the application of our approach. In other words, we ask how would each of the three main stakeholders of mobile apps benefit from the availability of automatically labelled reviews. To address the applicability of our approach, we pose the following research question:

RQ3: Are multi-label approaches useful for stakeholders?

To address RQ3, we define three scenarios: app comparison, app store overview, and anomaly detection. Each scenario requires the analysis of user reviews.

The datasets used in these scenarios are separate from the reviews downloaded in our

manual analysis because the previous reviews were selected from 20 apps and our scenarios require a much larger number of apps to generalize our results.

We apply a PSt approach with an SVM model because the combination of PSt and SVM exhibits the best performance (see RQ2).

Cross-validation has been shown to not be entirely reliable as an estimate (Esbensen et al., 2002). Therefore, to better understand the performance of our multi-labelling approach on the 601,221 Google Play store user reviews, we randomly select a statistical sample of these reviews with a confidence level of 95% and a confidence interval of 5%. We select 384 user reviews. An undergraduate student verified all the automatically labelled user reviews. The undergraduate student was a volunteer. The results of the undergraduate student were verified by myself to reduce any bias. If we disagreed we came to a consensus. We disagreed on 7% of the user reviews. We build a model on the previously labelled data and test the model on the random sample. We achieve an accuracy, exact match, precision micro, recall micro, f-measure micro, f-measure macro (by label) of 45%, 32%, 46%, 59%, 51%, 50% respectively as Table 5.11 demonstrates.

The drop in the performance of the model on the random sample compared to the previous performance of the model using cross-validation is expected. There are apps involved that may have specific issues that were not encountered in the training data. Increasing the size of the training data would improve the performance. This step is left for future work. Additionally, similar drops are reported in papers that contain a training set, a test set, and a validation set (Brameier and Banzhaf, 2001; Michielan et al., 2009). The common practice in machine learning is to train models using the training set and adjust the parameters and types of machine learning techniques to produce a model that performs best on the test set. The model's performance is then tested on a validation set that was not part of the model

Table 5.11: Performance of our multi-label approach on a random sample of the Google Play Store user reviews

Evaluation Measure	Pruned Sets with threshold extension - SVM
Precision	50%
Recall	62%
F-measure micro	55%
F-measure Macro (by label)	52%
Accuracy	49%
Exact Match	35%

selection process. We follow this approach by selecting the best model out of a possible nine models that performs best against the 10-fold cross validation. The validation set in our case is the Google Play Store data that we used to apply our three analytics use case scenarios on.

The performance of our approach is suitable for the following three scenarios because for each scenario, we concentrate on spikes in data. We are not concerned with small differences between issues in apps. We draw our results from significant differences in issues in comparison to other apps.

5.4.1 Scenario 1 - App Comparison

Motivation

Developers and users benefit from the ability to compare apps. Developers can conduct competitive analysis and users can conduct comparative analysis (Annie, 2014; Flurry, 2014).

Competitive analysis is the comparison of competing apps with similar features and target user base. Competitive analysis is useful for developers. For example, if a developer already has a weather app, competitive analysis would enable him to track the competition, to observe what users are asking for in competing apps. Furthermore, if a mobile app

developer wants to develop a new weather app, he could observe the problems faced by similar apps and attempt to avoid them e.g., what features should be added, avoid privacy violations.

Comparative analysis is the comparison of the available information about a product before making a purchase. For example, users may wish to download a weather app, but the app store only offers rudimentary tools to compare the many available weather apps (the store only shows raw ratings and user reviews). If two weather apps have the same rating, e.g., 4.0, a user would turn to the user reviews to help inform their choice. However, a user would be forced to read many user reviews to gain an understanding of the issue types that users are having. If an app store has graphs of issue distributions then the issue distribution would be helpful for a user to make an informed decision when purchasing/downloading an app e.g., one app is reported by users to crash much more than the other. Graphs of issue distributions are not currently available on app stores.

Data Collection

We collected 6,532 user reviews from three apps from the Google Play Store in the weather category. We preprocess and retain only one-star and two-star user reviews for a total of 536. We use an open source Google Play Store crawler to extract the apps' information and user reviews (Akdeniz, 2013). Each run collects the 500 newest submitted user reviews. We ran the crawler four times between a period of about three weeks (August 20th 2013 to September 9th 2013) to allow time for the submission of new user reviews.

Approach

We select the three most popular apps from the weather category with similar ratings between 4.1 and 4.4 and over 1 million downloads. We choose the weather category because most of the apps have a similar feature set and purpose i.e., to show the current weather and future weather forecasts. In total, there are 210, 89, and 237 one-star and two-star user reviews for the ‘AccuWeather’, ‘WeatherBug’, and ‘The Weather Channel’ apps respectively. We run our multi-labelling approach on the reviews of these selected apps.

Results

We find spikes in the frequency of occurrences of certain issue types between the apps. Figure 5.6 gives a more detailed overview of all three weather apps. The spike in ‘feature requests’ for ‘The Weather Channel’, compared to the other two apps, is due to the feature request for the ability to turn off the notifications (e.g., “They added a notification to show the current temp. Nice if you like it but you can’t turn it off and it stays there when if you exit the app.”), which is not present in the other two apps. The spike in ‘network problem’ for ‘AccuWeather’ is due to the time that it takes to refresh the information in the app. An example of a review is “This app stopped refreshing and would not upset weather info when refreshing manually”. ‘AccuWeather’ also crashes at a higher frequency (e.g., “i like the App info but beware, it crashes a lot!!!!”).

The most important issue types for users, measured by frequency, are different amongst the three apps. Users of ‘The Weather Channel’ complain about (from most to least): functional complaint, feature request and update issues. For ‘WeatherBug’, the issue types are update issue, functional complaint and crashing. For ‘AccuWeather’, the issue types are functional complaint, crashing and network problem. A developer could focus on the

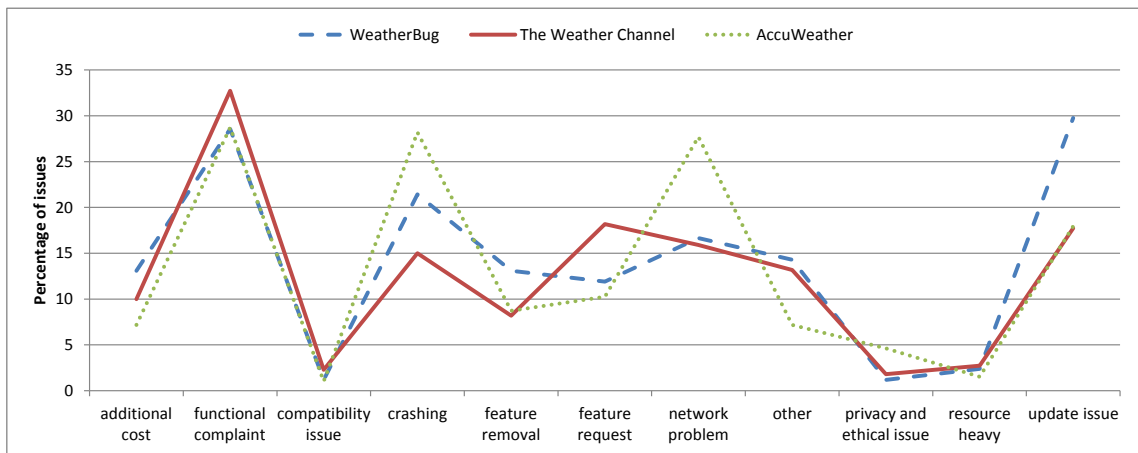


Figure 5.6: Comparison of 3 similar rated popular apps with similar ratings in the Weather category

issue types most pressing to their users as demonstrated in Figure 5.6.

There are not many ‘resource heavy’ issues. There are not many compatibility issues either, the lack of compatibility issues means that the developers might not have to worry as much about doing extensive testing across different phones and different operating system versions (Khalid et al., 2014b).

Overall, a user might wish to avoid downloading the ‘AccuWeather’ app because it has several spikes in ‘crashing’ and ‘network problem’ issues compared to the other two weather apps. However, if one were to choose between ‘The Weather Channel’ and ‘WeatherBug’, we observe that ‘WeatherBug’ crashes more but has many less feature requests suggesting that the app is unstable but it might contain most of the features that users want. Our manual analysis of ‘The Weather Channel’ reviews indicates that most of the feature requests are about the notification bar such as “There is no option to turn off the persistent notification.”.

A user may prefer a more stable app to an app that is missing desirable features, while another user may prefer the latest features and accept the instability. The knowledge of the

trade-off allows users to make an informed decision.

A developer of ‘The Weather Channel’ app can take action based on the knowledge that the app has more feature request issues than its competitors. Also, the developer of the ‘AccuWeather’ app could prioritize fixes and stability issues because of the spikes in the labelled network problems and crashing reviews.

5.4.2 Scenario 2 - App Store Overview

Motivation

Observing the distribution of issue types for an app store is beneficial to the owners of the app stores. Owners would be able to observe the issue types with the most user complaints. Owners could provide support in the form of Frequently Asked Questions (FAQs), tools, guidelines and policy changes in proportion to the issue distributions occurring in their app store. The app store owner could investigate why an issue type occurred so frequently relative to other issue types. The issues may be the fault of the app developers but if they occur for many developers, the app store owner could take action in the form of software updates and API changes.

Developers would benefit from knowing if the user reviews of their app have differences in the frequency of issue types between two different app stores (e.g., Apple App Store versus Google Play Store). The ability to perform such cross-store comparisons is of great value to developers, since the median number of platforms (stores) that developers develop for is two (mobile, 2014). For example, if an app on store A has more compatibility issues reported by users than on store B, the app’s developer could investigate why their app had more compatibility issues in store A and perform more compatibility testing in the future for store A.

Data Collection

We collected 3.7 million user reviews from 12,000 apps from the Google Play Store. We selected the top 400 apps in the USA across the 30 different categories (e.g., Photography, Sports and Education) based on Distimo's ranking of apps. Distimo is an app analytic company (Distimo, 2013). We again use the crawler mentioned in Section 5.4.1 to download 500 reviews of the top 400 apps across the 30 different categories of the Google Play Store. We ran the crawler twice, several weeks apart, to allow time for new user reviews to appear. Also some apps did not have 500 new reviews even after the second crawler run. Hence, the actual amount of user reviews that we crawled is lower than the expected amount of user reviews i.e., 6 million (500 reviews for each of the 12,000 studied apps).

We follow the same preprocessing steps as RQ2 (Section 5.3.3). From the initial 3.7 million user reviews, our selection of one-star and two-star reviews and preprocessing steps reduce the amount of reviews to 601,221 user reviews.

Approach

We first run our multi-labelling algorithm on the 601,221 user reviews to determine the distribution of issue types across the store. We then separate the data into the 30 categories from the app store e.g., social, finance, and education.

Lastly, to observe if there are differences between the same app across different stores, we compare two apps across both the Apple App Store and the Google Play Store. We select all the one-star and two-star user reviews for the Facebook and Kindle apps in our dataset for a total of 534 Facebook and 355 Kindle user reviews.

Results

Figure 5.7 shows that the dominant issue types for the Google Play store are functional complaints, feature requests, and network problems. We also observed the issue distributions per category, e.g., business, comics, sports etc. Table 5.12 displays issue type percentages that are more or less than two standard deviations above the mean for each category in one-star and two-star user reviews in the Google Play Store. We investigated several of the categories. The finance category has higher ‘functional complaint’ issues than others, e.g., mobile cheque processing does not work, login errors and unprocessed requests. The Shopping category had a higher number of ‘feature requests’ which included requests for better payment options and improved ease of use. The Shopping category also had higher ‘privacy and ethical issues’ which included complaints relating to unredeemed coupons or being asked to enter personal details before being able to claim a deal. The Cards category contained higher ‘additional cost’ issues as many employ a free trial version, known as a freemium model, that may disappoint users (Niculescu and Wu, 2011). The Brain and Personalization categories contained user reviews that complain about ads and notifications. The Weather and Productivity categories contained user reviews that mention unnecessary background processes that drained the battery. The Tools and ‘Libraries and Demo’ categories contained higher compatibility issues. The reviews reported lack of compatibility with specific phones and versions of the Android operating system.

We did not find any major deviations between the Facebook app of the Apple App Store and Google Play Store. However, there is a large spike for feature requests in the Kindle app on the Apple App Store as Figure 5.8 demonstrates. We investigate the reason and find that due to Apple’s store regulations, Amazon, the developer of the Kindle App, had to remove the ability to buy books from within the Kindle app. The removal of this feature

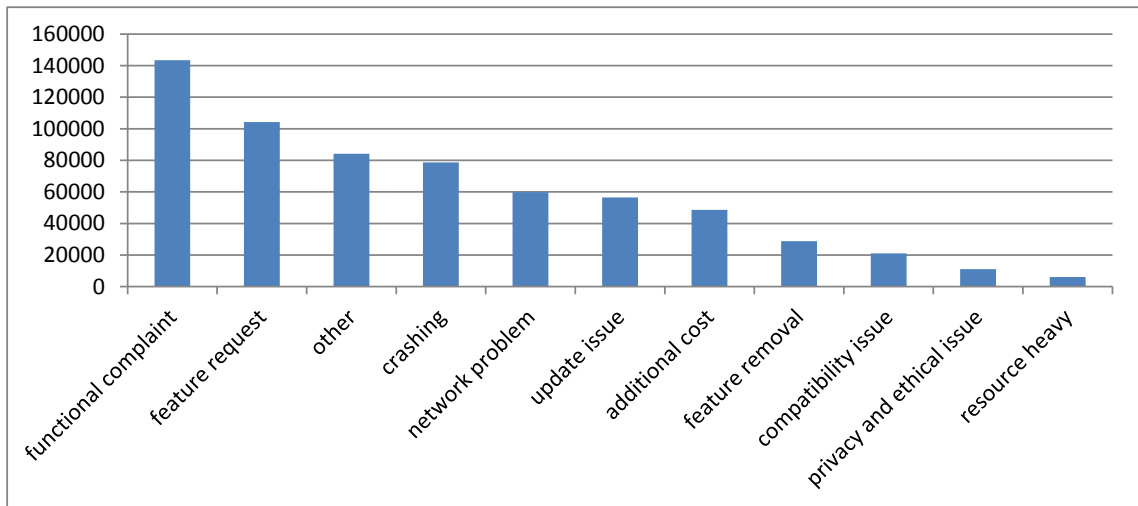


Figure 5.7: Distribution of issue types for 1 and 2 star user reviews in the Google Play Store dataset

Table 5.12: Store categories that have specific issue types above or below two standard deviations of the mean in our Google Play Store data set

Issue Type	Two Standard Deviations Above Mean	Two Standard Deviations Below Mean
Functional Complaint	Finance	Racing
Feature Request	Shopping	
Other	Racing	
Crashing	Sports, News and Magazines	
Network Problem		Personalization
Update Issue	News and Magazines	
Additional Cost	Cards	
Feature Removal	Brain, Personalization	
Compatibility Issue	Libraries and Demo, Tools	
Privacy and Ethical Issue	Shopping	
Resource Heavy	Productivity, Weather	

was a major issue for many users. The Google Play Store hadn't taken that step, hence the Google Play version of the app didn't have those feature requests.

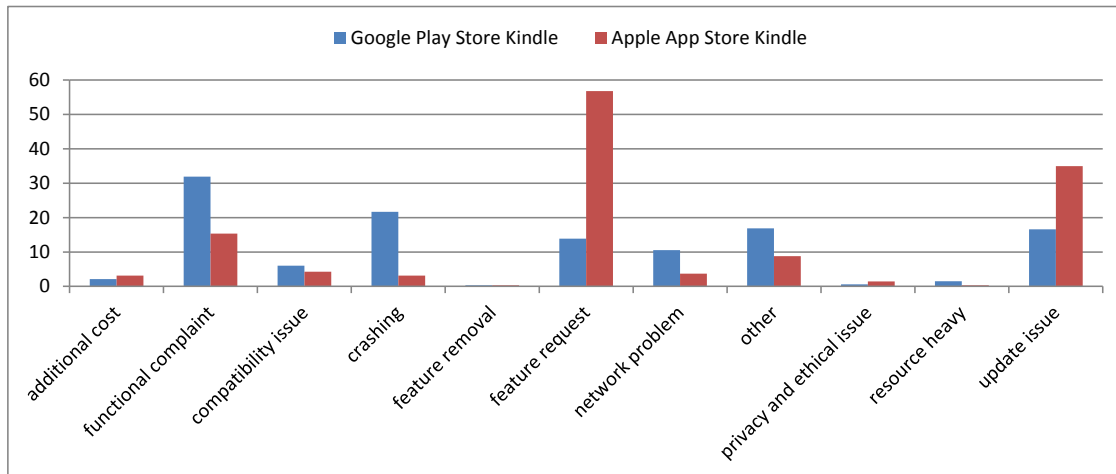


Figure 5.8: Comparison of the Kindle app for the Google Play and Apple App Stores

5.4.3 Scenario 3 - Anomaly Detection

Motivation

App store owners have specific developer content policies (Google, 2013a) and core app quality guidelines. For example, the Google Play Store has quality criteria for apps (Google, 2013b) that developers must follow. App developers are strongly recommended to follow these guidelines. These policies and guidelines are in place to ensure a certain baseline quality experience for the users. If a developer does not follow the guidelines then an app store may remove his app from the store. This is comparable to how grocery stores set a minimum quality standard for their products e.g., removing spoiled products (e.g., rotten fruits) from the customer's potential choices. One solution is to read the user reviews in order to find apps that might violate the quality criteria of a store. However, in order for an app store to perform this task, the store needs to analyze and read through millions of reviews across thousands of apps to find troublesome apps. Reading through the user reviews manually is a time consuming task. Therefore, an automated approach to analyze the

millions of reviews would be helpful to app store owners.

The app store owner could analyze the issue distribution of all apps within their store and automatically flag apps where one or more issue types occur above a threshold relative to all other apps in the store. The app store owner could warn the developer or remove the offending app. The app store owner could support their decision through evidence gathered from user reviews.

Data Collection

We use the same data collected for scenario 2 in Section 5.4.2.

Approach

We analyze the issue distribution of our sample of the Google Play Store data set in order to demonstrate the effectiveness of our anomaly detection approach for user reviews. We use control chart theory, which was invented to monitor manufacturing processes and detect deviations (Shewhart, 1931). Control chart theory has been applied to software engineering problems in other contexts (Abubakar and Jawawi, 2013; Ghaith et al., 2013; Nguyen et al., 2012).

We calculate the average and standard deviation for each issue type across all the apps (we exclude apps with less than 15 one-star and two-star user reviews to prevent apps with few reviews from being detected as outliers). If an issue type L_j occurs in the user reviews of an app with a frequency greater than x standard deviations of the average for all apps, we flag the app as anomalous. Traditionally, in control chart theory three standard deviations are used as the threshold for anomaly detection. We perform experiments on one, two and three standard deviations and compare the results. The actual deviation level can be

Table 5.13: Percentage of anomalous apps that are inaccessible in the store (1 year later) and anomalous apps flagged by our approach for 1, 2 and 3 standard deviations. N/A denotes there were no apps at that range

Issue Type	1-std dev		2-std dev		3-std dev	
	Inaccessible	Total Flagged	Inaccessible	Total Flagged	Inaccessible	Total Flagged
Privacy and Ethical Issues	15%	93	13%	31	20%	5
Feature Removal	23%	44	50%	8	100%	1
Resource Heavy	N/A	N/A	N/A	N/A	33%	3
Crashing	9%	192	15%	61	0%	12

determined by practitioners based on the amount of effort and time that they might wish to spend on this type of analysis.

For example, if app A had 80% of its user reviews labelled as crashing while the average amount of user reviews labelled as crashing was 30% and the standard deviation was 15% for all apps across the store, then app A is above the average + two standard deviations and will be flagged as anomalous.

$$flag(L_j) = \begin{cases} 0 & \text{if } L_j \leq 2 \times std_dev \\ 1 & \text{if } L_j > 2 \times std_dev \end{cases}$$

Results

Our initial results are displayed in Table 5.13. The higher the standard deviation, the more issues the apps have of a certain category. There are many more crashing apps than resource heavy apps as flagged by our approach.

We select two standard deviations as our threshold to further analyze the apps. Our

choice of two standard deviations is arbitrary but can be adjusted by an analyst based on their needs (e.g., the number of apps they can manually inspect, the severity of each issue type). Our reasoning was that amounts of issues that occurred with 2 standard deviations outside the mean were statistically rare and worth investigating.

Our approach flags 9 apps as anomalous for the ‘feature removal’ issue type, 36 for ‘privacy and ethical issues’, 3 for ‘resource heavy’, and 73 for ‘crashing’. Table 5.14 provides examples of anomalous apps for each issue type. We manually analyzed the reviews of the most anomalous apps for feature removal, privacy and ethical issue, resource heavy and crashing and highlight several example apps. Figure 5.9 demonstrates the number of apps that exceed our threshold. We note that an app store owner may define their own thresholds and it is entirely dependent on how sensitive the app store owners are to each issue. We investigate the user reviews of apps and find reports of direct violations of the policies and guidelines of the Google Play Store.

The 9 apps that were anomalous for feature removal issues had user reviews that contained many issues about ads. Users specifically mentioned frequent and obtrusive pop-up ads and ads in the notification bar. The app ‘Pic Stitch’ had many reports of ads in the notification bar and adding shortcuts and programs unknowingly to the user’s phone. This reported behavior is not permitted under Google’s developer policy as it states “Apps and their ads must not add homescreen shortcuts, browser bookmarks, or icons on the user’s device as a service to third parties or for advertising purposes.” (Google, 2013a) Additionally, “Apps and their ads must not display advertisements through system level notifications on the user’s device.” (Google, 2013a). The app ‘Pic Stitch’ could be removed based on its violation of Google’s policies if true. In fact, it no longer exists in the Google Play Store as of July 2014.

Table 5.14: Most anomalous apps from the four categories that were flagged in our analysis of the Google Play Store dataset

App Name	Issue	Comments
ESPN Bracket Bound 2013	Crashing	<p>“Constantly crashes and won’t let me look at my brackets. Thanks for putting out another crappy app ESPN”</p> <p>“This app crashes and has to restart literally every time I open it. This is a worthless app. Sucks”</p> <p>“Crashes about 10 times a day. Extremely annoying”</p> <p>“This would be a great app if it didn’t crash every time I use it”</p>
Pic Stitch	Feature Removal	<p>“It sends ads as notifications to my phone. And as others stated...too many in app adds. Just downloaded last night and am now deleting.”</p> <p>“1 minute in and my notifications is full of ads.... NO THANK YOU”</p> <p>“THERE IS WAY TOO MUCH SPAM AND ADS TIED TO THIS APP.”</p> <p>“Ads all over the place AND spam notifications on my phone...DELETE!!”</p>
Cabela’s	Resource Heavy	<p>“This app will run in your background and absolutely destroy your battery life”</p> <p>“This app uses the gps to know your location. It exhausted up my battery in less than half a day. After removal of the battery usage was fine again”</p> <p>“nice when shopping but eats up battery quick and you can’t stop it”</p> <p>“Used 31% of my battery without even opening the app. Not worth having in case I catch a tagged fish!”</p>
Quote Rocket Insurance	Privacy and Ethical Issue	<p>“This app is a piece of crap they spam you all the time and telemarketers call you all the time and it steals your information its called Quote Rocket it didn’t give my quote this stupid app wastes your time it is just a horrible piece of crap don’t get it I am warning you”</p> <p>“Too much information needed to give out.”</p> <p>“It just steals your information. You end up getting calls from stupid telemarketers.”</p> <p>“I go through everything & don’t get my crystals for my game & you can’t help me with Insurance for my car”</p> <p>“I did everything and filled it out correctly. Got an email with my quote and a phone call still no coins. :(“</p>

The app ‘Quote Rocket Insurance’ flagged as anomalous for privacy and ethical issues stated that if the user downloads the app and enters their contact information (for telemarketing purposes) then the user would be rewarded with an in-game currency for a videogame app. However, user reviews contained reports that users did not receive the in-game currency and that they were still contacted by telemarketers. The second app advertised that users would be given a \$20 gift card for free if they entered their information. Users reported in their reviews that the gift cards turned out to be for two very expensive women’s clothing stores and that they were later targeted by telemarketers. Google has a policy that apps cannot exist “where the primary functionality is to drive affiliate traffic to a website” (Google, 2013a). Additionally “forcing the user to click on ads or submit personal

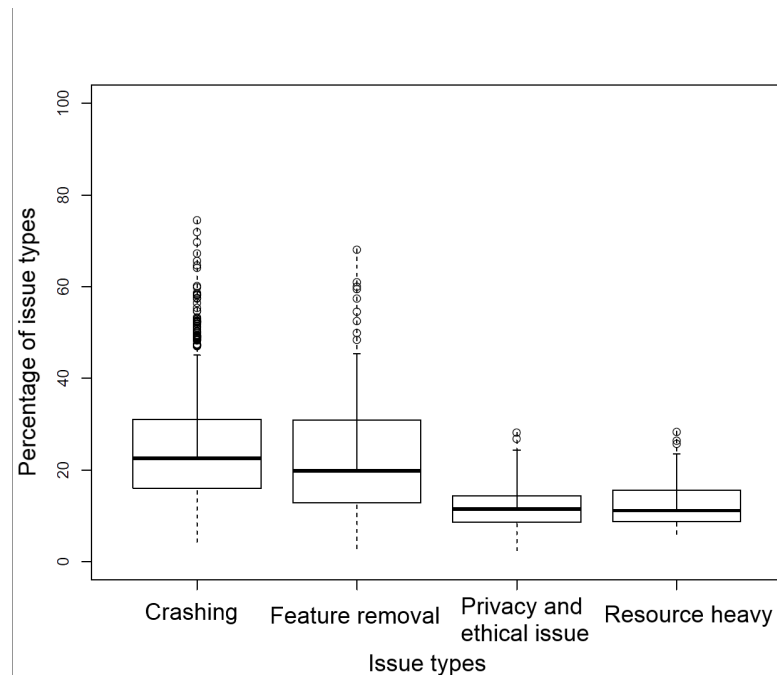


Figure 5.9: Box-plots with the anomalous apps as points in the graph above two standard deviations of the mean

information for advertising purposes in order to fully use an app is prohibited.”(Google, 2013a).

‘Cabelas’, the anomalous app for the resource heavy issues had issues about the battery life. Users reported that the app drained their battery while running as a background process, even when not in use. Google states in their core app quality guidelines that apps should not run background processes unless necessary to the function of the app.

Finally the user reviews on the anomalous apps associated with the crashing issues contained issues about crashing and freezing. Crashing and freezing are basic criteria of the core app quality guidelines.

In July 2014 (one year after our initial crawl) we test if the apps are still available in the Google Play Store by accessing the store page of each of the anomalous apps. If an app is no longer accessible (i.e., removed from the store possibly due to the flagged issues) through

its store page we mark the app as inaccessible. We find that the percentage of inaccessible apps increases or remains the same for apps at higher standard deviations for three out of the four anomalous issue types. Table 5.13 demonstrates the percentage of inaccessible apps for four issue types that were one, two and three standard deviations away from the mean for that issue type. In the table, if an app is anomalous at three standard deviations, it will not be part of the apps at one or two standard deviations. Likewise, an app at two standard deviations will not be present at one standard deviations.

We demonstrate that with our approach, app store owners can detect anomalous apps and choose to pro-actively warn developers or remove such anomalous apps from their app store.

We demonstrate through three scenarios that our multi-labelling approach assists all three stakeholders of mobile apps in the following three scenarios: app comparison, market overview and anomaly detection.

5.5 Discussion

In this section, we justify our decision to combine the Apple App Store and Google Play Store user reviews into one dataset. We also discuss the implications of the number of labels to predict in our approach. We investigate poor performing issue types, and present alternative approaches.

Table 5.15: Comparison of training on Google Play Store and testing on Apple App Store and vice versa

	Train	Test	Train	Test
	Google Play Store	Apple App Store	Apple App Store	Google Play Store
Accuracy	34		34	
Exact Match	20		20	
Precision	40		44	
Recall	43		40	
F-1 Micro	41		42	
F-1 Macro L	27		30	

5.5.1 Combining the Apple App Store and Google Play Store Data

In our multi-label approach, we wish to have as much labelled data as possible. However, the labelled data came from 2 different stores since the data was from a previous study and it wasn't expected that the data is to be used as training data for our study. Our solution was to combine the data and assume that they could be used as a single training dataset. However, we need to ensure that the two datasets did not have major differences. Preliminary results suggested they were similar as we found in Chapter 3 similar average length of reviews for the Apple App Store and Google Play Store (61 and 64 characters). A more important factor for our purposes is to measure their similarity between models built with each dataset. We judge their similarity by testing their respective prediction power on one another.

We randomly selected 1,066 user reviews from Apple App Store and an equal amount of Google Play Store user reviews from our manually labelled data (see Section 5.3.2). We built a model (we followed the approach in Section 5.3.3) on only the Apple App Store data and tested our model on the Google Play Store and vice versa.

As Table 5.15 shows, the prediction power of both models remains the same. Thus, we conclude that the user reviews are similar enough to be combined into a single data set.

5.5.2 Impact of Reducing the Number of Labels

We show in our previous single-labelled classification (see Section 5.3.3 where we can see individual f-measures) that the classification of certain issue types perform poorly compared to others. We also see from merging these issue types into the ‘other’ issue that the overall f-measure improves. Merging issue types demonstrates that as you decrease the number of labels the performance is increased and if you remove the bad performing labels the performance will increase even more.

Iacob and Harrison have proposed approaches to extract feature requests from the user reviews of mobile apps (Iacob and Harrison, 2013). Their proposed approach used a binary classifier. In other words, they only cared if a review was or was not a feature request. The proposed approach had 96% precision and 86% recall respectively. Their results support our aforementioned observation that a smaller number of labels lead to improved classification performance.

The choice of issue types should be decided based on the interest of each stakeholder and the specific analysis to ensure the best performing automated classification.

5.5.3 Poor performing issue types

We investigate the reason behind the poor automated classification performance of some of the issue types. As Table 5.9 shows, certain issue types are more difficult to automatically classify. In particular, response time, uninteresting content, and user interface performed below a 50% f-measure.

The goal of this subsection is not to improve the classification accuracy but instead to understand the reason for the low performance of some of the labels. Hence, we use Labelled Latent Dirichlet allocation (LLDA) which showed that these labels are ambiguous

(i.e., have a common set of words).

LLDA is a supervised learning version of the unsupervised approach *Latent Dirichlet Allocation* (LDA) (Ramage et al., 2009). LDA is a popular statistical topic model (Blei et al., 2003). It generates topics from the corpus of documents and then represents the documents as a set of topics with associated probabilities. A topic is a mixture of terms drawn from the documents with associated probabilities for that topic. To generate the topic and term distributions, a generative process is applied to learn the correct distributions such as Gibbs Sampling or Collapsed Variation Bayes. A weakness of LDA is that the number of topics must be defined beforehand as LDA makes no assumptions on the number of topics per document.

LLDA differs from LDA in that it requires documents to be labelled. LLDA constrains the topics to only the labels assigned to the training documents. Hence, the labelled topics correspond directly to the issue types. For example, LLDA will generate a list of words that are most associated with each issue type and infer which documents are most associated with each issue type. LLDA can handle multi-labelled problems inherently (Ramage et al., 2009). The output of LLDA is a probability distribution for each label and for each document. The output of LLDA can be used as a model to predict the label of user reviews. In order to make predictions a threshold value is required to be optimized.

We build an LLDA model with the Stanford Topic Modeling Tool¹. We use the same settings as (Ramage and Rosen, 2011). The topic and term smoothing parameters are set at 0.1 and 0.1 respectively. The number of sampling iterations are 1,500. Document similarity is computed using the cosine similarity measure. To infer the distributions of words and topics, we use collapsed variational bayes (CVB) inference.

¹<http://nlp.stanford.edu/software/tmt/tmt-0.4/>

We investigate the three ambiguous issue types e.g., response time, uninteresting content, user interface, that we removed because of their poor performance for automated classification. We find that the most occurring words in response time are similar to network problem e.g., ‘slow’, ‘takes’, ‘very’. We also find that user interface and uninteresting content share similar words with functional complaint and feature request e.g., ‘out’, ‘please’, ‘like’.

The small number of manually labelled user reviews for a particular issue type didn’t factor into the classification performance of individual issue types. The resource heavy issue had comparably little training data (just 0.4 and 6 percent of the Apple App Store and Google Play Store data respectively) yet its f-measure was near the average. We find the most probable word associated with the resource heavy issue is the word ‘battery’ which did not appear in any of the lists of most probable words associated with other issue types.

5.5.4 Alternative approaches

In this section we present and discuss unsupervised alternatives to our multi-labelling approach for labelling reviews and for performing anomaly detection. We first present an overview of supervised learning vs. unsupervised learning, followed by the results of the unsupervised alternatives.

Our approach uses supervised learning. Supervised learning builds a model that learns the correlations between training examples and the example’s labels to predict future example’s labels. An unsupervised approach is an alternative to a supervised approach. There are no pre-specified labels. An unsupervised model infers correlations between the data without any labelled training data. The benefit of an unsupervised approach is that it does not require any labelling (a potentially time-consuming or otherwise impossible task). The

drawback of unsupervised learning is that the model may infer connections that otherwise are unimportant or unwanted. Hence, is a trade-off between supervised and unsupervised learning depending on the goal.

Unsupervised labelling

We first present an unsupervised approach to our multi-labelling approach. The approach we use is topic modelling. Topic modelling associates topics to documents based on words in the documents. This association is done in an unsupervised (i.e., discovery/exploration-oriented fashion). As such, the topics are related by frequency and commonality instead of the topics of interest for a particular research problem. In our case, we had a specific research problem at hand (the concept of quality of an app). Hence, we did not consider topic modelling, instead we chose a supervised approach. A manual process was followed to identify issues and topics with our particular research problem in mind. The identified problem-relevant topics were then fed into an automated classifier in order to locate other occurrences of these topics.

To study the use of unsupervised learning, we perform a new experiment. In the new experiment, we ran topic modelling (i.e., LDA) on our manually labelled data in order to uncover unsupervised topics. We did two runs with the number of topics (K) equal to 14 and 150 topics. We chose 14 since that is the number of topics that were uncovered by our manual supervised approach and we chose 150 topics to get a feel of how well LDA would perform on a much large number of topics (Galvis Carreño and Winbladh, 2013).

The topics recovered using $K=150$ are fine-grained app-specific topics instead of being topics that talk about quality-related concerns (the focus of our research). Such examples included topics on GPS or shopping lists as Table 5.16 shows. The topics created by the

Table 5.16: Examples of general topics about photo and videogames with LDA (14 topics) and specific topics about gps and requesting help with LDA (150 topics)

Example topics in LDA			
topic size: 14		topic size: 150	
1	photo, app, picture, better, edit, adobe, use, good, upload, pic, does, photoshop, quality, feature, like, option, thing, useless, image, need	1	list, shop, search, recipe, result, disappear, click, button, view, great, title, ingredient, does, icon, feature, box, bug, groceries, menu, differ
2	game, play, video, watch, movie, good, need, like, great, money, sync, just, upgrade, time, star, coin, farmville, fun, really, spend	2	locate, await, place, mile, area, say, show, find, far, map, nearby, right, accuracy, check, state, try, close, horrible, gp, tip

14 topic run are too general to be useful. For example, one of the topics is about media e.g., game, play, video, watch and movie and another topic is about photos e.g., photo, app, picture, better and edit. In Table 5.16 we see two examples of such topics.

The second unsupervised approach we perform is keyword search. Keyword search is very simple. The approach searches for related words of a label in the review and if the search words match, the review is considered a part of the label. The drawbacks of this approach is that it requires extensive knowledge of how users report issues and an exhaustive list of potential search terms. We select the label words as keywords and search within our already labelled training data. If the label words are contained in a review then the approach predicts the review to contain the label. As Table 5.17 shows, most labels perform poorly with the exception of ‘crashing’ and ‘update issue’. Reviews with ‘crashing’ and ‘update issue’ are likely to contain words like ‘crashing’ or ‘update’ in the review itself. ‘Privacy and ethical issues’ contain more varied language to describe an issue.

Unsupervised anomaly detection

An unsupervised alternative to detecting anomalous apps with user reviews is a simple bag of words approach. To study the applicability of such a simple approach, we perform the following experiment. We find the average occurrence of every word that occurred

Table 5.17: Results for keyword search

Label	Search Terms	Accuracy Percentage
Network problem	network	0
Compatibility issue	compatibility	0.65
Resource heavy	resource, heavy	4.54
Update issue	update	62.9
Bug fix	bug	6.81
Additional cost	additional, cost	7.1
Privacy and ethical issue	privacy, ethical	3.1
Feature request	feature, request	9.5
Feature removal	remove	4.3
Crashing	crash, crashing	79.9

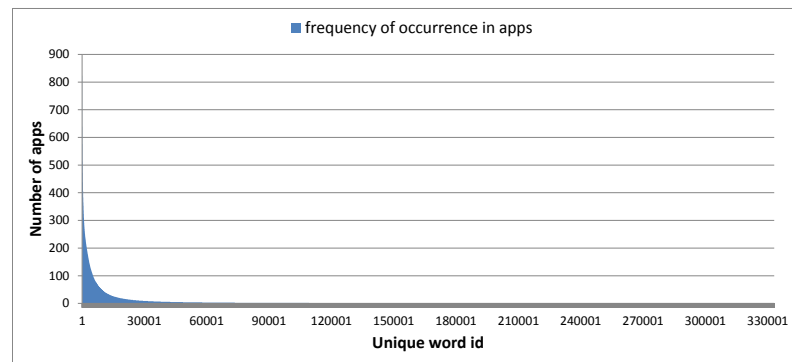


Figure 5.10: Distribution of the frequency of occurrence of anomalous words across all apps (sorted by frequency).

in any of the studied reviews. Next, find the 1,2 and 3 standard deviation of each word occurrence. We then flag words that occur outside x standard deviations. We find that the words occurring are not quality-related words. Instead they are simply words that appear in an app because they are specific to an app such as “asl” for a sign language app. The top 20 words are shown in Table 5.18. Furthermore, we find that the majority of anomalous words do not occur across all apps and are focused in a small selection of apps as Figure 5.10 shows. The limited coverage of the anomalous words would not be useful for our purposes of differentiating the majority of apps.

Table 5.18: Top 20 most frequent anomalous (3 standard deviations) words across apps

Words as they appeared in reviews
isnt, thus, means, wouldnt, couldnt wonder, awhile, mostly, provide, deserve, besides, ty, recomend, pleased, opinion, havent, putting, serious, appears, plain

5.6 Threats to Validity

Some threats could potentially limit the validity of our experiments. We now discuss potential threats and how we control or mitigate them.

5.6.1 Threats to Construct Validity

Since, we manually labelled our gold dataset of reviews with the different issue types, some reviews may have been incorrectly labelled. To mitigate this threat, we performed this labelling in an iterative manner and went over each review multiple times to ensure correct labelling of the reviews. To avoid any bias in the labelling process, one PhD student and a post-doctoral fellow, verified the labelling again. If they disagreed on a label they took the majority opinion, i.e., in total three votes, one from the person who labelled the data and two from persons who verified the data. If the vote was a three way tie they came to a consensus.

The labelled dataset was originally labelled for another purpose (Khalid et al., 2014a). The study of multi-labelled data in user reviews was not the original intention therefore there was no conflict of interest when the labelling occurred to try and show multiple labels.

5.6.2 Threats to Internal Validity

We have several parameters that need to be set in our study. The parameters for the SVM tool were chosen based on the default values that experts in the field had determined (Hall

et al., 2009b). We elected to follow their recommendations. The selection of thresholds in our multi-labelling approach could directly impact our results. To mitigate this threat, we selected the proportional cut threshold algorithm to automatically determine the threshold. Proportional cut thresholds for a classifier are determined automatically using the label distributions in the training data. There are other threshold techniques however proportional cut algorithm is shown to be efficient and performs as well as other more complex threshold algorithms (Read, 2010).

The selection of one-star and two-star ratings could bias the results. We assumed that one-star and two-star ratings contain negative issues. To mitigate this threat, we performed sentiment analysis to confirm our assumptions. The results of sentiment analysis showed that in majority of the cases, one-star and two-star star rated reviews are negative. We recognize that three, four and five star reviews may contain an issue, e.g., feature requests, in an overall positive review. We accept that we could have missed some more issues by eliminating all the three, four, and five star ratings. However, our multi-labelling approach is rating independent. Thus, adding more reviews and/or labels could produce similar results. However, more experiments are needed to generalize our findings on all the star ratings. Also lower ratings are of a much greater concern to developers since they decrease the overall rating of their app in the app store and addressing the concerns of those users is of a greater priority to developers than three, four and five star reviews (Khalid et al., 2014a).

The selection of multi-labelling approach, i.e., PSt, could directly impact our findings. Our choice of PSt is based on its performance which we found to be superior to BR and CC.

5.6.3 Threats to External Validity

Our training set of labelled reviews data is composed of 20 mobile applications. Hence our results may not generalize to all mobile applications. To mitigate this threat, we selected apps from a diverse set of categories, by selecting apps from high and low-rated apps and from the 2 most popular app stores. We employed a random sampling process for each app to collect a representative sample with 95% confidence and 5% confidence interval. The exact number of collected user reviews varies between apps as the total number of user reviews varies per app.

5.7 Conclusion

User reviews are an important indication of quality of an app. Labelling user reviews is beneficial to stakeholders. However, user reviews are difficult to label considering the unstructured noisy multi-labelled nature of the data. We demonstrate that even with such difficulties, we can effectively and automatically label the reviews to address real world problems of stakeholders. Our work is another step in moving towards leveraging user reviews to improve the quality of mobile apps and leverage the benefits of the unique mechanisms of app stores.

For future work we would like to improve the prediction performance for our multi-labelling approach especially for the ambiguous issue types. We would also like to perform user surveys of the major stakeholders to better understand how they would use the analytics use case scenarios that we demonstrated in this chapter.

Chapter 6

Conclusions and Future Work

This chapter summarizes our findings from previous chapters. We follow the summary with a presentation of the limitations of our results and outline potential future avenues of research.

6.1 Summary

Mobile app store distribution and feedback mechanisms differ from traditional software. The rapid release cycles and heavy influx of reviews presents unique challenges to stakeholders in the app marketplace. Little research has been conducted on these mechanisms of app stores prior to this thesis. Hence, we perform two empirical studies and propose one approach to leverage the rich information available through such mechanisms:

Mobile app stores have unique distribution and feedback mechanisms that do not exist in traditional software development. Developers, users, and app store owners must take into consideration these unique mechanisms that allow for rapid new releases and an influx of user feedback. More empirical studies should be undertaken to understand how developers and users interact with these mechanisms and new approaches should be developed to manage the challenges that arise and leverage the rich information that such mechanisms provide.

Chapter 3 presents a study on frequently-releasing apps. We find the distribution of release frequency is highly skewed. There is a very small subset of apps (~100 apps) that release very frequently at least once per week and many apps release less than bi-weekly. Almost half of frequent-releasing apps do not report what has changed between releases and often only make minor changes (bug fixes). Such a release pattern differs from traditional software with feature-rich infrequent releases.

Chapter 4 presents a study on the heavy influx of reviews in the Google Play Store. We find that there is a small subset of apps (0.19%) that receive an overwhelming number of reviews on a daily basis. Whereas the vast majority of apps receive 20 reviews or less per day. The topic of the reviews vary however, certain issues occur more often. Finally, we find that responding to reviews can increase the rating of the review positively. This type of overwhelming feedback pales in comparison to traditional software feedback methods of email or bug reporting.

Both Chapter 3 and 4 motivate Chapter 5's approach to deal with the consequences of the distribution and feedback mechanisms. Chapter 5 presents an approach to automatically label user reviews with a pre-defined set of labels. We find that we have moderate accuracy

and we demonstrate its usefulness on three separate analytics use case scenarios, comparative analysis, app store overview and anomaly detection. We compare different apps' issues using our approach which is useful for developers and users to compare the quality of similar apps. We present an overview of the app store and found that functional complaints are the most mentioned issue for users. Finally, we flag apps which have specific issues that occur at an anomalous rate in an app's reviews.

6.2 Limitations and Future Work

The limitations of each of our three studies were presented in their respective chapters. However, there is one limitation that is at the center of all three chapters and threatens the validity of our thesis. We address this threat in this section. We conclude with a discussion of future work.

We acknowledge that the selection of our apps may bias our results. We list possible threats to validity concerning our selection of apps and how we mitigated these threats.

We use data from an app analytics company to select the studied apps. Distimo may not be an accurate measure of popularity and could bias our results. The popular apps could have a different characteristics than less popular apps. Distimo uses a combination of ratings, downloads and their own proprietary method for determining popularity. However, there is no other comprehensive tool to determine popularity as Google does not publish these statistics. Our results may only generalize to popular mobile apps. Given the large number of unsuccessful and spam apps in the store (Ashraf, 2012), we feel that our study of popular apps is warranted instead of blindly studying all apps.

The selection of our studied apps may not generalize to all mobile apps. To mitigate

this threat, we selected apps from all the different categories of the market to be as representative as possible. There are a total of 30 categories from which we draw our data from.

We only study apps that are free. We define a free app as an app that is free-to-download. Paid apps may exhibit different characteristics in comparison to free apps. However, many paid apps have free versions available to download and there are considerably more free apps than paid apps in the Google Play Store. Additionally, in-app purchases can be made within a app. So developers can still make money from users without requiring the purchase of the app. We found that 25% of all our studied apps have in-app purchases by checking if the apps have the “billing” permission. We find that 42% of the frequently-reviewed apps have in-app purchases and 61% of the most reviewed apps have in-app purchases. These findings demonstrate that there is a mix of revenue generating models and that the incentives of profit are present in our studied apps. Also, some of our analysis in Section 3.4.3 required the analysis of app binaries which required to purchase the app first which we could not afford.

There remains future work that can be done to further the initial findings presented in this thesis. We present several future research questions. In Chapter 3, we ask what types of code changes are occurring in frequently-releasing apps? What are the percentage of corrective (i.e., bug fix) changes versus the percentage of new features? What types of code changes that correspond to particular release note types (i.e., bug fix, new feature).

For our second empirical study in Chapter 4, we ask if there are differences in paid versus free apps? Are there differences in the reviews between stores?

The approach that we have proposed in Chapter 5 is only a start and there remains work to be done to improve the accuracy. How can we further improve the accuracy of our

approach? What other scenarios would this approach be useful for?

Bibliography

Stanford topic modeling toolbox, November 2012. URL <http://nlp.stanford.edu/software/tmt/tmt-0.4/>.

Alhassan Muhammad Abubakar and Dayang NA Jawawi. A study on code peer review process monitoring using statistical process control. In *e-Proceeding of Software Engineering Postgraduates Workshop (SEPoW)*, page 136, 2013.

Adobe. Mobile analytics, May 2014. URL <http://www.adobe.com/ca/solutions/digital-analytics/mobile-analytics.html>.

Syed Nadeem Ahsan, Javed Ferzund, and Franz Wotawa. Automatic classification of software change request using multi-label machine learning methods. In *Software Engineering Workshop (SEW), 2009 33rd Annual IEEE*, pages 79–86. IEEE, 2009.

Akdeniz. Google play crawler, September 2013. URL <https://github.com/Akdeniz/google-play-crawler>.

App Annie. App annie, May 2014. URL <http://www.appannie.com/app-store-analytics/>.

Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. Is it a bug or an enhancement?: a text-based approach to classify change

- requests. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, CASCON '08, pages 23:304–23:318. ACM, 2008.
- Apple. Apple - job creation, July 2014a. URL <http://www.apple.com/about/job-creation/>.
- Apple. Viewing and changing your apps status and availability. https://developer.apple.com/library/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/Chapters/ChangingAppStatus.html, June 2014b.
- Apple. Apple app store, July 2014c. URL itunes.apple.com.
- Ashraf. Google bans notification bar ads, such as airpush, from play store apps, August 2012. URL <http://goo.gl/zJpdUZ>.
- Richard Baskerville and Jan Pries-Heje. Short cycle time systems development. *Information Systems Journal*, 14(3):237–264, 2004. ISSN 1365-2575. doi: 10.1111/j.1365-2575.2004.00171.x. URL <http://dx.doi.org/10.1111/j.1365-2575.2004.00171.x>.
- BlackBerry. Blackberry world, July 2014. URL <http://appworld.blackberry.com/>.
- David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

Kevin Bostic. Google play takes top spot in downloads, but apple's app store still tops revenue, July 2013. URL <http://appleinsider.com/articles/13/07/31/google-play-takes-top-spot-in-downloads-but-apples-app-store-still-tops-revenue>

M. Brameier and W. Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *Evolutionary Computation, IEEE Transactions on*, 5(1):17–26, Feb 2001. ISSN 1089-778X. doi: 10.1109/4235.910462.

Brut.alll and Connor Tumbleson. Apk tool. <https://code.google.com/p/android-apktool/>, January 2014. Accessed January 2014.

Margaret Butler. Android: Changing the mobile landscape. *Pervasive Computing, IEEE*, 10(1):4–7, 2011.

Rishi Chandy and Haijie Gu. Identifying spam in the ios app store. In *Proceedings of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality*, pages 56–59. ACM, 2012.

Brian X. Chen. Samsung galaxy phone is no. 1 for now. <http://goo.gl/iqTkcf>, November 2012. Accessed March 2014.

David Yu-Chung Chen. *Essays on mobile advertising and commerce*. PhD thesis, Harvard University Cambridge, Massachusetts, 2009.

Ning Chen, Jialiu Lin, Steven CH Hoi, Xiaokui Xiao, and Boshen Zhang. Ar-miner: Mining informative reviews for developers from mobile app marketplace. 2014.

Pern Hui Chia, Yusuke Yamamoto, and N Asokan. Is this app safe?: a large scale study on application permissions and risk signals. In *Proceedings of the 21st international conference on World Wide Web*, pages 311–320. ACM, 2012.

Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Comput. Surv.*, 44(2):7:1–7:35, March 2008. ISSN 0360-0300. doi: 10.1145/2089125.2089127. URL <http://doi.acm.org/10.1145/2089125.2089127>.

Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, 44(2):7, 2012.

Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

Gianluca Dini, Pierfrancesco Foglia, C Antonio Prete, and Michele Zanda. A social-feedback enriched interface for software download. *Journal of Organizational and End User Computing (JOEUC)*, 25(1):24–42, 2012.

Distimo. Google play store, united states, top overall, free, week 35 2013, September 2013. URL <http://www.distimo.com/leaderboards/google-play-store/united-states/top-overall/free>.

Eelco Dolstra, Merijn De Jonge, and Eelco Visser. Nix: A safe and policy-free system for software deployment. In *LISA*, volume 4, pages 79–92, 2004.

Kim H Esbensen, Dominique Guyot, Frank Westad, and Lars P Houmoller. *Multivariate data analysis: in practice: an introduction to multivariate data analysis and experimental design*. Multivariate Data Analysis, 2002.

Rong-En Fan and Chih-Jen Lin. A study on threshold selection for multi-label classification. *Department of Computer Science, National Taiwan University*, 2007.

Flurry. Flurry, May 2014. URL <http://www.flurry.com/solutions/analytics>.

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 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1276–1284, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2174-7. doi: 10.1145/2487575.2488202. URL <http://doi.acm.org/10.1145/2487575.2488202>.

Laura V. Galvis Carreño and Kristina Winbladh. Analysis of user comments: an approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 582–591, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-3076-3.

Kavita Ganesan, ChengXiang Zhai, and Evelyne Viegas. Micropinion generation: an unsupervised approach to generating ultra-concise summaries of opinions. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 869–878, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1229-5.

Gartner. Gartner says mobile app stores will see annual downloads reach 102 billion in 2013. <http://www.gartner.com/newsroom/id/2592315>, September 2013. Accessed January 2014.

Shadi Ghaith, Miao Wang, Philip Perry, and John Murphy. Profile-based, load-independent

- anomaly detection and analysis in performance regression testing of software systems. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 379–383. IEEE, 2013.
- Google. Bytecode for the dalvik vm. <http://goo.gl/5TKib9>. Accessed February 2014.
- Google. Google play developer program policies, September 2013a. URL <https://play.google.com/about/developer-content-policy.html>.
- Google. Core app quality guidelines, September 2013b. URL <http://developer.android.com/distribute/googleplay/quality/core.html>.
- Google. Google analytics, May 2014a. URL <http://www.google.ca/analytics/mobile/>.
- Google. Google play store, July 2014b. URL play.google.com.
- Google. Update your apps, July 2014c. URL <https://support.google.com/googleplay/android-developer/answer/113476>.
- Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009a. ISSN 1931-0145. doi: 10.1145/1656274.1656278. URL <http://doi.acm.org/10.1145/1656274.1656278>.
- Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009b.

- Dan Han, Chenlei Zhang, Xiaochao Fan, Abram Hindle, Kenny Wong, and Eleni Stroulia. Understanding android fragmentation with topic analysis of vendor-specific bugs. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pages 83–92. IEEE, 2012.
- Mark Harman, Yue Jia, and Yuanyuan Zhang. App store mining and analysis: MSR for app stores. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 108–111. IEEE, 2012.
- Frank E Harrell. *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*. Springer, 2001.
- Ahmed E Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*, pages 78–88. IEEE Computer Society, 2009.
- Kim Herzig, Sascha Just, and Andreas Zeller. It’s not a bug, it’s a feature: how misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 392–401. IEEE Press, 2013.
- Paul Hodgetts and Denise Phillips. Extreme adoption experiences of a b2b start-up. *Extreme Programming Perspectives*, pages 355–362, 2002.
- Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.
- Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

- Claudia Iacob and Rachel Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the Tenth International Workshop on Mining Software Repositories*, pages 41–44. IEEE Press, 2013.
- Nathan Ingraham. Apple announces 1 million apps in the app store, more than 1 billion songs played on itunes radio.
- J Jenkins. Velocity culture (the unmet challenge in ops). In *Presentation at O'Reilly Velocity Conference*, 2011.
- N. Jindal and B. Liu. Review spam detection. In *Proceedings of the 16th international conference on World Wide Web*, pages 1189–1190. ACM, 2007.
- Vigdis By Kampenes, Tore Dybå, Jo E Hannay, and Dag IK Sjøberg. A systematic review of effect size in software engineering experiments. *Information and Software Technology*, 49(11):1073–1086, 2007.
- H. Khalid, E. Shihab, M. Nagappan, and A. Hassan. What do mobile app users complain about? a study on free ios apps, 2014a. ISSN 0740-7459.
- Hammad Khalid. On identifying user complaints of ios apps. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013.
- 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 *22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014)*. ACM, 2014b.
- Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E. Hassan. What do

- mobile app users complain about? a study on free ios apps. In *IEEE Software*. IEEE Press, 2014c.
- F. Khomh, T. Dhaliwal, Ying Zou, and B. Adams. Do faster releases improve software quality? an empirical case study of mozilla firefox. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 179–188, 2012. doi: 10.1109/MSR.2012.6224279.
- Hee-Woong Kim, Hyun Lyung Lee, and Jung Eun Son. An exploratory study on the determinants of smartphone app purchase. In *The 11th International DSI and the 16th APDSI Joint Meeting*, Taipei, Taiwan, July 2011.
- S Kuppuswami, Kalimuthu Vivekanandan, Prakash Ramaswamy, and Paul Rodrigues. The effects of individual xp practices on software development effort. *ACM SIGSOFT Software Engineering Notes*, 28(6):6–6, 2003.
- Soo Ling Lim and P.J. Bentley. Investigating app store ranking algorithms using a simulation of mobile app ecosystems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 2672–2679, June 2013. doi: 10.1109/CEC.2013.6557892.
- 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 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013*, pages 477–487, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2237-9. doi: 10.1145/2491411.2491428. URL <http://doi.acm.org/10.1145/2491411.2491428>.
- Mario Linares-Vásquez, Bogdan Dit, and Denys Poshyvanyk. An exploratory analysis of

- mobile development issues using stack overflow. In *Proceedings of the Tenth International Workshop on Mining Software Repositories*, pages 93–96. IEEE Press, 2013.
- John Lynch. App store optimization: 8 tips for higher rankings. <http://goo.gl/htvSNL>, October 2012.
- Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 142–150. Association for Computational Linguistics, 2011.
- Matthias Marschall. Transforming a six month release cycle to continuous flow. In *Agile Conference (AGILE), 2007*, pages 395–400. IEEE, 2007.
- P. Melville, W. Gryc, and R.D. Lawrence. Sentiment analysis of blogs by combining lexical knowledge with text classification. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1275–1284. ACM, 2009.
- Lisa Michielan, Lothar Terfloth, Johann Gasteiger, and Stefano Moro. Comparison of multilabel and single-label classification applied to the prediction of the isoform specificity of cytochrome p450 substrates. *Journal of Chemical Information and Modeling*, 49(11): 2588–2605, 2009. doi: 10.1021/ci900299a. URL <http://pubs.acs.org/doi/abs/10.1021/ci900299a>. PMID: 19883102.
- Martin Michlmayr, Francis Hunt, and David Probert. Release management in free software projects: Practices and problems. In Joseph Feller, Brian Fitzgerald, Walt Scacchi, and

- Alberto Sillitti, editors, *Open Source Development, Adoption and Innovation*, volume 234 of *IFIP The International Federation for Information Processing*, pages 295–300. Springer US, 2007. ISBN 978-0-387-72485-0.
- Microsoft. Windows phone, July 2014. URL <http://www.windowsphone.com/en-ca/store>.
- Roberto Minelli and Michele Lanza. Software analytics for mobile applications—insights & lessons learned. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 144–153. IEEE, 2013.
- Mika V. Mntyl, Foutse Khomh, Bram Adams, Emelie Engstrm, and Kai Petersen. On rapid releases and software testing. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance, ICSM '13*, pages 20–29, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-0-7695-4981-1. doi: 10.1109/ICSM.2013.13. URL <http://dx.doi.org/10.1109/ICSM.2013.13>.
- Vision mobile. Developer Economics Q1 2014: State of the Developer Nation. Technical report, 05 2014.
- Susan M Mudambi and David Schuff. What makes a helpful online review? a study of customer reviews on amazon.com. *MIS Quarterly*, 34(1):185–200, 2010.
- Thanh HD Nguyen, Bram Adams, Zhen Ming Jiang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. Automated detection of performance regressions using statistical process control techniques. In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*, pages 299–310. ACM, 2012.

- Marius F Niculescu and Dong Jun Wu. When should software firms commercialize new products via freemium business models. *Under Review*, 2011.
- Tobias Otte, Robert Moreton, and Heinz D Knoell. Applied quality assurance methods under the open source development model. In *32nd Annual IEEE International Computer Software and Applications.*, pages 1247–1252. IEEE, 2008.
- Dennis Pagano and Bernd Bruegge. User involvement in software evolution practice: a case study. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 953–962. IEEE Press, 2013.
- Dennis Pagano and Walid Maalej. In *Proceedings of the 21st. IEEE International Requirements Engineering Conference*. IEEE, 2013. URL <http://mobis.informatik.uni-hamburg.de/wp-content/uploads/2013/07/RE2013PaganoMaalej.pdf>.
- Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *LREC*, 2010.
- Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. Whyper: towards automating risk assessment of mobile applications. In *Presented as part of the 22nd USENIX Security Symposium*, pages 527–542. USENIX, 2013.
- Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

- Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.
- Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.
- Adam Porter, Cemal Yilmaz, Atif M Memon, Arvind S Krishna, Douglas C Schmidt, and Aniruddha Gokhale. Techniques and processes for improving the quality and performance of open-source software. *Software Process: Improvement and Practice*, 11(2): 163–176, 2006.
- Martin F Porter. Snowball: A language for stemming algorithms.
- Martin F Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.
- Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2012.
- D Ramage and E Rosen. Stanford topic modeling toolbox, 2011.
- Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D Manning. Labeled lda: A supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 248–256. Association for Computational Linguistics, 2009.
- Jesse Read. *Scalable multi-label classification*. PhD thesis, University of Waikato, 2010.

- Jesse Read. Meka: A multi-label extension to weka, September 2013. URL <http://meka.sourceforge.net/>.
- Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. In *Machine Learning and Knowledge Discovery in Databases*, pages 254–269. Springer, 2009.
- Israel J. Mojica Ruiz, Meiyappan Nagappan, Bram Adams, and Ahmed E. Hassan. Understanding reuse in the android market. In *IEEE International Conference on Program Comprehension (ICPC)*, page To appear, June 2012.
- Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.
- Walter Andrew Shewhart. *Economic control of quality of manufactured product*, volume 509. ASQ Quality Press, 1931.
- Statista. Google play: number of android app downloads as of july 2013, July 2014. URL <http://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/>.
- Mark Syer. *Empirical Studies of Mobile Apps and Their Dependence on Mobile Platforms*. PhD thesis, Queen’s University, 2013.
- 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. *Int’l Working Conference on Source Code Analysis and Manipulation*, pages 55–64, 2011a.
- 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 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation, SCAM '11*, pages 55–64, 2011b.
- Mark D Syer, Meiyappan Nagappan, B Adams, and AE Hassan. Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open source android apps. In *Proc. of the IBM CASCON Conf.(CASCON'13)*, to appear, 2013.
- Mike Thelwall, Kevan Buckley, and Georgios Paltoglou. Sentiment strength detection for the social web. *J. Am. Soc. Inf. Sci. Technol.*, 63(1):163–173, January 2012. ISSN 1532-2882. doi: 10.1002/asi.21662. URL <http://dx.doi.org/10.1002/asi.21662>.
- Kai Tian, Meghan Revelle, and Denys Poshyvanyk. Using latent dirichlet allocation for automatic categorization of software. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*, pages 163–166. IEEE, 2009.
- Ivan Titov and Ryan McDonald. Modeling online reviews with multi-grain topic models. In *Proceedings of the 17th international conference on World Wide Web*, pages 111–120. ACM, 2008.
- Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.
- Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. In *Data mining and knowledge discovery handbook*, pages 667–685. Springer, 2010.
- Priya Viswanathan. Android os vs. apple ios which is better for developers? <http://goo.gl/ApQCb6>, May 2011. Accessed January 2014.

Karl Whitfield. Portio research, July 2014. URL
[http://www.portioresearch.com/en/blog/2013/
fast-growth-of-apps-user-base-in-booming-asia-pacific-market.
aspx](http://www.portioresearch.com/en/blog/2013/fast-growth-of-apps-user-base-in-booming-asia-pacific-market.aspx).

Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan, and Xiaoming Li. Comparing twitter and traditional media using topic models. In *Advances in Information Retrieval*, pages 338–349. Springer, 2011.